فهرست مطالب

فصل اول: مقدمه

فصل دوم: بهینه‌سازی

فصل سوم: الگوریتم‌های ژنتیک

فصل چهارم: مدل‌های ریاضی برای الگوریتم‌های ژنتیک

فصل پنجم: برنامه‌نویسی تکاملی

فصل ششم: استراتژی‌های تکاملی

فصل هفتم: برنامه‌نویسی ژنتیک

فصل هشتم: مطالب گوناگون مرتبط با الگوریتم‌های تکاملی

فصل نهم: ذوب فلزات

فصل دهم: بهینه‌سازی کلونی مورچگان

فصل یازدهم: بهینه‌سازی تجمع ذرات

فصل دوازدهم: تکامل تفاضلی (دیفرانسیلی)

فصل سیزدهم: الگوریتم‌های تخمین توزیع

فصل چهاردهم: بهینه‌سازی زیست‌جغرافی-محور

فصل پانزدهم: الگوریتم‌های فرهنگی

فصل شانزدهم: یادگیری مقابله محور

فصل هفدهم: دیگر الگئریتم‌های تکاملی

فصل هیجدهم: بهینه‌سازی ترکیبی

فصل نوزدهم: بهینه‌سازی مقید

فصل بیستم: بهینه‌سازی چند هدفه

فصل بیست و یکم: توابع برازندگی گران، شلوغ و پویا

ضمائم

فصل اول

مقدمه

محور اصلی این کتاب بحث در مورد روش‌های موجود برای حل مسائل بهینه‌سازی، به ویژه الگوریتم‌های تکاملی[1] است. به طور خاص ما[2] در این کتاب به بحث در مورد الگوریتم‌های تکاملی جهت بهینه‌سازی خواهیم پرداخت. اگر چه کتاب شامل تعدادی نظریه‌های ریاضی است، اما نباید آن را یک کتاب ریاضی در نظر گرفت، چرا که در اصل این کتاب یک کتاب مهندسی و یا علوم کامپیوتری کاربردی است. تمامی الگوریتم‌های بهینه‌سازی موجود در این کتاب با هدف نهایی پیاده‌سازی نرم‌افزاری ارائه شده‌اند. هدف این کتاب این است که الگوریتم‌های بهینه‌سازی تکاملی را به واضح‌ترین و دقیق‌ترین شکل ممکن ارائه داده و همچنین با فراهم آوردن مسائل و مراجع پیشرفته به مقدار کافی به خوانندگان آمادگی مشارکت در مسائل جدید در این زمینه را بدهد.

نگاهی کلی به فصل

این فصل با بخش ۱-۱ و با درآمدی بر نشان‌گذاری‌های ریاضی که در این کتاب به کار خواهیم برد شروع می‌شود. فهرست اختصارات نیز می‌تواند برای خواننده مفید واقع شود. در بخش ۱-۲ دلایلی برای پاسخ پرسش‌هایی فراهم شده است، پرسش‌هایی همچون: چرا تصمیم به نوشتن این کتاب گرفتم؟ هدفم از نوشتن این کتاب چیست؟ و چرا این کتاب از سایر کتاب‌های بی‌نظیر در زمینه الگوریتم‌های تکاملی متمایز است؟ بخش ۱-۳ به بیان پیش‌نیازهایی می‌پردازد که از خواننده انتظار می‌رود. بخش ۱-٤ به فلسفه تکالیف و چرایی وجود آن‌ها در این کتاب و همچنین به قابلیت دسترسی حل‌المسائل می‌پردازد. بخش ۱-۵ خلاصه‌ای از نشان‌گذاری‌های ریاضی به کار رفته در این کتاب را به دست می‌دهد. به خوانندگان قویاً توصیه می‌شود در هنگام مواجه با نشان‌گذاری‌های ناآشنا مرتباً به این بخش رجوع کرده و همچنین سعی کنند در تحقیقات و تکالیفشان از این نشان‌گذاری‌ها استفاده نمایند. بخش ۱-٦ طرح و نمای کلی از کتاب را به دست می‌دهد. در نهایت و در بخش ۱-۷ نکاتی مهم جهت تدریس این کتاب و همچنین توصیه‌هایی در مورد میزان اهمیت فصل‌ها نسبت به یکدیگر به مدرس ارائه شده است.

---

[1] Evolutionary Algorithms

[2] در این کتاب از روشی معمول برای اشاره به اشخاص استفاده شده است و آن استفاده از کلمه‌ی ما است. بعضی مواقع کتاب از کلمه ما برای اشاره به خواننده یا نویسنده استفاده کرده است. در مواردی نیز نویسنده از کلمه ما استفاده کرده است تا نشان دهد سخنی را به نمایندگی از جمعی از استادان و محققان فعال در زمینه الگوریتم‌های تکاملی و بهینه‌سازی بیان می‌کند. این تفاوت در متن مشهود خواهد بود. چندان به نحوه کاربرد کلمه ما دقیق نشوید چرا که این مسئله تنها به شیوه نوشتار برمی‌گردد، تا به منظوری خاص که نویسنده به دنبال آن باشد.

۱-۱ عبارت شناسی

برخی نویسندگان برای اشاره به الگوریتم‌های تکاملی از عبارت محاسبات تکاملی استفاده می‌نمایند. این نکته بیانگر این مطلب است که الگوریتم‌های تکاملی بر روی کامپیوتر پیاده‌سازی می‌شوند. با این حال عبارت محاسبات تکاملی می‌تواند بر الگوریتم‌هایی که برای بهینه‌سازی به کار نمی‌روند هم دلالت داشته باشد. برای مثال، الگوریتم‌های ژنتیک اولیه فی‌نفسه به‌منظور بهینه‌سازی مورد استفاده قرار نگرفتند بلکه برای مطالعه فرآیند انتخاب طبیعی در نظر گرفته شده بودند (فصل ۳ را ببینید). این کتاب در راستای الگوریتم‌های بهینه‌سازی تکاملی مدون شده است که نسبت به محاسبات تکاملی مبحثی اختصاصی‌تر است.

دیگران از عبارت بهینه‌سازی جمعیتی برای اشاره به الگوریتم‌های تکاملی استفاده می‌نمایند. این موضوع بیانگر آن است که به طور کلی الگوریتم‌های تکاملی متشکل از جمعیتی از راه‌حل‌های نامزد برای برخی مسائل است و با گذر زمان این جمعیت به سوی راه‌حلی بهتر حرکت می‌نماید. با این حال، بسیاری از الگوریتم‌های تکاملی تنها از یک راه‌حل نامزد در هر دوره تشکیل شده‌اند (برای مثال الگوریتم تپه‌نوردی[1] و استراتژی‌های تکامل). الگوریتم‌های تکاملی بسیار جامع‌تر از بهینه‌سازی جمعیتی هستند چرا که الگوریتم‌های تکاملی شامل الگوریتم‌های تک-ذره‌ای نیز هستند.

برخی نویسندگان از عبارت هوش کامپیوتری یا هوش محاسباتی برای اشاره به الگوریتم‌های تکاملی استفاده می‌کنند. این کار معمولاً برای تشخیص و تمایز الگوریتم‌های تکاملی از سیستم‌های خبره، که به طور سنتی با عنوان هوش مصنوعی شناخته می‌شوند، صورت می‌پذیرد. سیستم‌های خبره از استدلال استقرایی بهره می‌برند حال آنکه الگوریتم‌های تکاملی از استدلال استنتاجی استفاده می‌کنند. با این حال، گاهاً الگوریتم‌های تکاملی به‌عنوان نوعی هوش مصنوعی در نظر گرفته می‌شوند. هوش کامپیوتری عبارت جامع‌تری نسبت به الگوریتم‌های تکاملی است و شامل تکنولوژی‌هایی چون شبکه‌های عصبی، سیستم‌های فازی[2] و زندگی مصنوعی می‌شود. این تکنولوژی‌ها می‌توانند برای کاربردهایی غیر از بهینه‌سازی مورد استفاده قرار گیرند. به همین جهت، مبحث الگوریتم‌های تکاملی ممکن است نسبت به هوش کامپیوتری مطلبی خاص‌تر و ویژه‌تر به نظر برسد.

محاسبات نرم[3] عبارت دیگری مرتبط با الگوریتم‌های تکاملی است. محاسبات نرم نقطه مقابل محاسبات سخت[4] است. محاسبات سخت به محاسباتی بسیار دقیق، کامل و پیچیده عددی اشاره دارد. محاسبات نرم بر

[1] Hill Climbing
[2] Fuzzy Systems
[3] Soft Computing
[4] Hard Computing

محاسباتی با دقت کمتر دلالت می‌کند مانند آنچه که ما انسان‌ها در زندگی روزمره خود انجام می‌دهیم. الگوریتم‌های محاسبه نرم راه‌حل‌هایی خوب (اما غیر دقیق) برای مسائل سخت، نویزی، چند پیمانه‌ای[1] و چندهدفه ارائه می‌دهند. به همین علت، الگوریتم‌های تکاملی زیر مجموعه‌ای از محاسبات نرم در نظر گرفته می‌شوند.

سایر نویسندگان از عبارت "محاسبات الهام گرفته شده از طبیعت" و یا "محاسبات الهام گرفته شده از زیست" برای اشاره به الگوریتم‌های تکاملی استفاده می‌کنند. با این حال، برخی از الگوریتم‌های تکاملی، مانند تکامل تفاضلی و الگوریتم‌های تخمین توزیع، از طبیعت برانگیخته نشده‌اند. دیگر الگوریتم‌های تکاملی مانند استراتژی‌های تکاملی و یادگیری مقابله-محور ارتباط بسیار ضعیفی با فرآیندهای طبیعی دارند. الگوریتم‌های تکاملی نسبت به الگوریتم‌های الهام گرفته شده از طبیعت جامع‌ترند چرا که الگوریتم‌های تکاملی شامل الگوریتم‌هایی نیز می‌شوند که از زیست‌شناسی برانگیخته نشده‌اند.

یکی دیگر از عبارت‌هایی که به الگوریتم‌های تکاملی نسبت داده می‌شود، "یادگیری ماشینی" است. یادگیری ماشین مطالعه الگوریتم‌های کامپیوتری است که از تجارب خود یاد می‌گیرند. با این حال، این زمینه معمولاً شامل الگوریتم‌هایی غیر از الگوریتم‌های تکاملی نیز می‌شود. یادگیری ماشین معمولاً بسیار گسترده‌تر از الگوریتم‌های تکاملی در نظر گرفته شده و شامل مباحثی چون یادگیری تقویتی شبکه‌های عصبی، خوشه‌بندی، ماشین‌های بردار پشتیبان و غیره می‌شود.

برخی نویسندگان تمایل دارند از عبارت الگوریتم‌های اکتشافی برای اشاره به الگوریتم‌های تکاملی استفاده کنند[2]. الگوریتم‌های اکتشافی روش‌هایی هستند که از قوانین رویکردهای حس مشترک برای حل مسائل استفاده می‌کنند. معمولاً از الگوریتم‌های اکتشافی انتظار نمی‌رود که بهترین جواب را برای مسئله پیدا کند، بلکه هدف آن‌ها پیدا کردن جواب‌هایی است که به حد کافی به جواب بهینه نزدیک هستند. عبارت فرا اکتشافی برای توصیف گروهی از الگوریتم‌های اکتشافی به کار می‌رود. شاید نه همه اما بیشتر الگوریتم‌های تکاملی که در این کتاب درباره‌شان سخن رفته است با روش‌ها و امکانات و پارامترهای مختلفی قابل پیاده‌سازی هستند. به همین علت، همه آن‌ها را می‌توان فرا اکتشافی[3] نامید. برای مثال، خانواده تمام الگوریتم‌های بهینه‌سازی کلونی مورچگان را می‌توان فرا اکتشاف کلونی مورچگان نامید.

---

[1] Multimodal

[2] کلمه‌ی اکتشافی ترجمه واژه‌ی Heuristic است. Heuristic خود ریشه‌ای یونانی دارد و ترجمه‌ی کلمه‌ی Eurisko می باشد. این کلمه به معنای پیدا کردن و اکتشاف است. این کلمه همچنین ریشه‌ی کلمه‌ی Eureka نیز می‌باشد که این کلمه برای ابراز پیروزی بعد از کشف جواب یک مسئله به کار می‌رود (Eureka به معنی یافتم! است، عبارت معروفی که ارشمیدس بعد از رسیدن به جواب مسئله‌ی معروفش به کار برد.)

[3] Metaheuristic

بسیاری از نویسندگان الگوریتم‌های تکاملی را از هوش گروهی (تجمعی) جدا می‌پندارند. یک الگوریتم هوش گروهی الگوریتمی است که بر پایه گروه‌هایی قرار دارد که در طبیعت وجود دارند (برای مثال گروه‌های مورچه‌ها و یا پرندگان). بهینه‌سازی کلونی مورچگان (فصل ۱۰) و بهینه‌سازی تجمع ذرات (فصل ۱۱) دو الگوریتم مهم از الگوریتم‌های گروهی هستند و بسیاری از محققان اصرار دارند که این دو نباید به‌عنوان الگوریتم‌های تکاملی در نظر گرفته شوند. با این حال، برخی نویسندگان معتقدند که هوش گروهی زیر مجموعه‌ای از الگوریتم‌های تکاملی است. برای مثال یکی از به وجود آورندگان بهینه‌سازی تجمع ذرات از آن به‌عنوان یک الگوریتم تکاملی یاد می‌کند (شی[1] و ابرهارت[2]، ۱۹۹۹). از آنجا که الگوریتم‌های هوش گروهی به روش مشابهی همچون الگوریتم‌های تکاملی اتفاق می‌افتند (و این روش همان حرکت جمعیتی از راه‌حل‌های نامزد به سوی جواب بهتر در هر دوره است)، بنابراین ما هوش گروهی را الگوریتم تکاملی در نظر می‌گیریم.

هرچند که عبارت‌شناسی کاملاً دقیق نبوده و به متن کتاب وابسته است، اما ما قرار را بر این می‌گذاریم که از عبارت الگوریتم تکاملی برای اشاره به الگوریتمی استفاده کنیم که راه‌حل مسئله را طی دوره‌های زیادی به سمت جواب بهینه سوق می‌دهد. به‌طور معمول یک دوره الگوریتم تکاملی (با توجه به پایه‌ی بیولوژیکی این الگوریتم‌ها) یک "نسل" خوانده می‌شود. با این حال، این تعریف ساده الگوریتم تکاملی بی‌نقص نیست چرا که، برای مثال، این تعریف بیان می‌دارد که گرادیان یک الگوریتم تکاملی نزولی است و این موضوعی است که برای هیچ کس قابل قبول نیست. ما از تعریف غیررسمی استفاده می‌کنیم و آن این است که الگوریتمی الگوریتم تکاملی است که به‌طور معمول و کلی الگوریتم تکاملی در نظر گرفته شود. این تعریف دایره‌وار در ابتدا کمی آزاردهنده به نظر می‌رسد اما کسانی که در این زمینه کار می‌کنند به‌تدریج به آن عادت می‌کنند. در نهایت، انتخاب طبیعی به‌عنوان بقای برازنده‌ترین[3] تعریف شده است و برازندگی نیز خود در شأن کسانی تعریف می‌شود که احتمال بقای آن‌ها بیشتر از سایرین است.

۱–۲ چرا کتابی دیگر درباره الگوریتم‌های تکاملی؟

کتاب‌های خوب بسیاری در مورد الگوریتم‌های تکاملی وجود دارد که این خود باعث به وجود آمدن این سؤال می‌شود: چرا یک کتاب دیگر با موضوع الگوریتم‌های تکاملی؟ دلیل نوشته شدن این کتاب ارائه یک

[1] Shi
[2] Eberhart
[3] Survival of The Fittest

رویکرد آموزشی، چشم انداز و محتوایی است که در هیچ کتاب دیگری در دسترس نیست. بالأخص امید است تا موارد زیر دراین کتاب محقق شوند:

- در کتاب رویکردی سرراست ارائه شده باشد که به خواننده کمک کند درکی واضح و در عین حال دقیق از الگوریتم‌های تکاملی کسب کند. بسیاری از کتاب‌ها گستره‌ی وسیعی از این الگوریتم‌ها را بدون هیچ‌گونه پیش زمینه نظری و به مثابه کتاب آشپزی ارائه داده‌اند. برخی دیگر از این کتاب‌ها بیشتر مانند یک رساله‌ی پژوهشی‌اند تا یک کتاب درسی و برای بسیاری از دانشجویان متوسط مهندسی ناملموس هستند. کتاب حاضر با ارائه الگوریتم‌ها و با پیاده‌سازی آسان سعی دارد به یک تعادل قابل قبول برسد.
- در کتاب برخی مثال‌های ساده جهت ایجاد درکی بصری از ریاضیات، معادلات و نظریه‌ی الگوریتم‌های تکاملی تدارک دیده شده‌اند. بسیاری از کتاب‌ها نظریه‌ی این الگوریتم‌ها را توضیح داده و سپس مسائل و یا مثال‌هایی را ارائه می‌کنند که درکی بصری به دست نمی‌دهند. با این حال، می‌توان مسائل و مثال‌های ساده‌ای را ارائه داد که برای حلشان تنها به یک قلم و کاغذ نیاز باشد. این مسائل ساده به دانشجویان کمک می‌کند تا نحوه عملی شدن نظریات را بهتر دیده و درک کنند.
- کدهای MATLAB تمامی مثال‌ها در سایت نویسنده قابل دسترس است[1]. بسیاری از کتاب‌های دیگر نیز دارای چنین کدهایی هستند اما غالباً این کدها یا کامل نبوده و یا به روز نیستند که این موضوع ممکن است برای خواننده کسالت‌آور باشد. آدرس پست الکترونیکی نویسنده نیز در این وب سایت در دسترس است. من مشتاقانه از هرگونه نظر، پیشنهاد استقبال می‌کنم. صد البته که آدرس‌های وب ممکن است منسوخ شوند، اما این کتاب شامل فهرست‌نویسی‌هایی از شبه‌کدهای رده بالا و الگوریتم‌واری است که بسیار مهم‌تر از فهرست‌نویسی یک برنامه خاص است. توجه داشته باشید که مثال‌ها و کدهای MATLAB به‌عنوان بهترین الگوریتم بهینه‌سازی در نظر گرفته نشده‌اند، بلکه آن‌ها برای فراهم آوردن درکی پایه‌ای از مفاهیم اساسی ارائه شده‌اند. هرگونه تحقیق و یا کاربرد جدی باید به کدهای نمونه تنها به‌عنوان نقطه‌ای اولیه برای شروع اتکا کنند.
- این کتاب شامل نظریه و جدیدترین الگوریتم‌های تکاملی‌ای است که در بیشتر کتاب‌های درسی دیگر غیر قابل دسترس‌اند. این موضوعات شامل تئوری مارکوفف، مدل‌های الگوریتم تکاملی، مدل سیستم پویای الگوریتم‌های تکاملی، الگوریتم‌های کلونی مصنوعی زنبورها، بهینه‌سازی زیست–جغرافی محور، یادگیری مقابل–محور، الگوریتم‌های تجمع مصنوعی ماهی‌ها، جهش قورباغه‌ای، بهینه‌سازی

[1] http://academic.csuohio.edu/simond/evolutionaryoptimization

باکتریال و بسیاری دیگر هستند. این موضوعات به تازگی به مبحث الگوریتم‌های تکاملی افزوده شده‌اند و این مباحث آنچنان که در این کتاب پوشش داده شده‌اند در کتاب دیگری مورد پوشش قرار نگرفته‌اند. با این حال، این کتاب در نظر ندارد تا زمینه‌ای خاص از الگوریتم‌های تکاملی را مورد بررسی قرار دهد، بلکه می‌کوشد مروری با سطح بالا را از بسیاری از زمینه‌های تحقیقاتی این الگوریتم‌ها فراهم کند تا خواننده بتواند درکی وسیع از این الگوریتم‌ها پیدا کرده و برای ادامه مطالعات خود در این زمینه در مسیری درست قرار بگیرد.

۱–۳ پیش‌نیازها

به‌طور کلی، یک دانشجو بدون نوشتن نرم‌افزار خود در زمینه الگوریتم تکاملی نمی‌تواند از این درس چیزی بیاموزد. به همین خاطر، سر رشته داشتن در برنامه‌نویسی را می‌توان در فهرست پیش نیازهای این درس قرار داد. در دانشگاهی که من این درس را به دانشجویان مهندسی برق و مهندسی کامپیوتر تدریس می‌کنم هیچ درس پیش نیاز خاصی وجود ندارد، تنها پیش‌نیاز برای دانشجویان لیسانس آن است که باید در حال سپری کردن سال آخر تحصیل خود باشند و برای دانشجویان فوق لیسانس هیچ پیش‌نیاز ی وجود ندارد. با این حال، فرض من بر آن است که دانشجویان سال آخر لیسانس و همچنین دانشجویان فوق لیسانس برنامه‌نویسان قابلی هستند.

نشان‌گذاری به کار رفته در این کتاب بر این فرض است که خواننده با نشان‌گذاری‌های ریاضی استاندارد که در جبر، هندسه، نظریه مجموعه‌ها و حسابان وجود دارند، آشنایی دارد؛ بنابراین یک پیش‌نیاز دیگر برای درک این کتاب نوعی بلوغ ریاضی است که اکثر قریب به اتفاق داشجویان سال آخر لیسانس از آن بهره‌مند هستند. نشان‌گذاری ریاضی در بخش ۱–۵ توضیح داده شده است. اگر خواننده نشان‌گذاری‌های توضیح داده شده در این بخش را بفهمد، به احتمال زیاد قادر خواهد بود سایر مباحث کتاب را دنبال کند.

ریاضی موجود در قسمت نظری کتاب (فصل ٤ قسمت ٦ و ۷، بیشتر مباحث فصل ۱۳ و چند بخش پراکنده دیگر) نیاز به دانشی در زمینه احتمالات و نظریه سیستم‌های خطی دارد. اگر دانشجو درس‌های مرتبط با این دو موضوع را نگذرانده باشد، دنبال کردن این موارد برایش دشوار خواهد بود. در صورتی که درسی بر پایه این کتاب برای دانشجویان لیسانس در نظر گرفته شود احتمالاً باید از این قسمت صرفنظر گردد.

۱–۴ مسائل و تمرین‌ها

مسائل انتهای هر فصل برای دادن قابلیت انعطاف‌پذیری به دانشجویان نوشته آورده شده‌اند، این تمرین‌ها به‌منظور تقویت درک دانشجو از نظریات، عمیق ساختن درک بصری وی از مفاهیم و بسط دادن مهارت‌های

تحلیلی دانشجو ارائه شده‌اند. مسائل کامپیوتری نیز به‌منظور کمک به گسترش توانایی تحقیقاتی دانشجویان و همچنین آموزش چگونگی استفاده از نظریات در مواجه با مسائلی که در صنعت با آن‌ها مواجه می‌شوند، ارائه شده‌اند. هر دو نوع این مسائل در کسب مهارت در زمینه الگوریتم‌های تکاملی از اهمیت زیادی برخوردارند. نمی‌توان تفاوتی محض میان تمرین‌های کامپیوتری و تمرین‌های نوشتاری قائل شد و آن‌ها را دو مقوله‌ی کاملاً جدا از هم پنداشت، چرا که برخی تمرین‌های نوشتاری نیاز به مقداری کار کامپیوتری داشته و در مقابل نیز برخی تمرین‌های کامپیوتری نیاز به تحلیل نوشتاری دارند. مدرس باید با توجه به شیوه‌ی تدریس خود تکالیفی مرتبط با الگوریتم‌های تکاملی در نظر بگیرد. تکالیفی که مانند پروژه بوده و برای طول ترم به دانشجو محول می‌شوند، برای موضوعاتی این چنینی آموزنده هستند. برای مثال، می‌توان از دانشجویان خواست تا برخی مسائل بهینه‌سازی کاربردی و واقعی را با استفاده از الگوریتم‌های تکاملی بحث شده در این کتاب حل کرده و کارایی آن‌ها را نسبت به هم سنجیده و تفاوت‌های آن‌ها را بیان کرده و نتیجه را در پایان ترم گزارش دهد.

یک حل‌المسائل برای تمام مسائل موجود در کتاب (چه مسائل کامپیوتری و چه مسائل نوشتاری)، از سوی ناشر برای مدرسان در نظر گرفته شده است. به مدرسان توصیه می‌شود برای کسب اطلاعات بیشتر در مورد چگونگی تهیه‌ی این حل‌المسائل با ناشر مربوطه تماس بگیرند. برای حفاظت از راستی و درستی تکالیف، حل‌المسائل تنها برای مدرسان در دسترس خواهد بود.

۱–۵ نشان‌گذاری

برخی نشان‌گذاری‌های ریاضی موجود در این کتاب در فهرست زیر توضیح داده شده‌اند.

- ✓ $y \to x$ یک نشان‌گذاری محاسباتی است که بیان می‌دارد y به متغیر x گماشته می‌شود. برای مثال الگوریتم زیر را در نظر بگیرید.

$$a = x^2 \text{ ضریب}$$

$$b = x^1 \text{ ضریب}$$

$$c = x^0 \text{ ضریب}$$

$$x^* \leftarrow \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

سه خط اول این الگوریتم جمله‌های گمارشی نیستند بلکه تنها به تعریف مقادیر a , b , c می‌پردازند. این سه پارامتر می‌توانند توسط کاربر یا یک الگوریتم دیگر مقداردهی شوند. آخرین خط اما، یک حکم نگارشی است که بیان می‌دارد مقدار سمت راست پیکان به x^* نگاشته می‌شود.

- ✓ $\frac{df(.)}{dx}$ مشتق کلی $f(.)$ نسبت به x است. برای مثال، فرض کنید $y = 2x$ و $f(x,y) = 2x + 3y$، بنابراین $f(x,y) = 8x$ و $\frac{df(.)}{dx} = 8$ خواهد بود.
- ✓ $f_x(.)$ که به صورت $\frac{\partial f(.)}{\partial x}$ نیز نمایش داده می‌شود، مشتق جزیی $f(.)$ نسبت به x است. برای مثال، بار دیگر فرض کنید $y = 2x$ و $f(x,y) = 2x + 3y$ باشد. آنگاه $f_x(x,y) = 2$ خواهد بود.
- ✓ $\{x: x \in S\}$ مجموعه‌ای از تمام x هایی است که متعلق به مجموعه‌ی S می‌باشد. یک نشان‌گذاری مشابه دیگر برای نشان دادن مقادیری از x که یک شرط خاص را برآورده می‌کنند وجود دارد. برای مثال، $\{x: x^2 = 4\}$ دقیقا همان $\{x: x \in \{-2,2\}\}$ است.
- ✓ $[a,b]$ یک بازه‌ی بسته بین a و b است که بیانگر مجموعه‌ی $\{x: a \leq x \leq b\}$ است. این مجموعه می‌تواند یک مجموعه از اعداد صحیح و یا حقیقی باشد.
- ✓ (a,b) یک بازه‌ی باز بین a و b است که بیانگر مجموعه‌ی $\{x: a < x < b\}$ است. این مجموعه نیز می‌تواند یک مجموعه از اعداد صحیح و یا حقیقی باشد.
- ✓ می‌توان از متن کتاب دریافت که $\{x_i\}, i \in S$ مختصرشده‌ی $\{x_i: i \in S\}$ است. برای مثال، اگر $i \in [1,N]$ آنگاه $\{x_i\} = \{x_1, x_2, \dots, x_N\}$.
- ✓ $S_1 \cup S_2$ مجموعه‌ای از تمام x هاست که x یا متعلق به مجموعه‌ی S_1 است یا S_2 و یا هردو. برای مثال، اگر $S_1 = \{1,2,3\}$ و $S_2 = \{7,8\}$ باشد. آنگاه $S_1 \cup S_2 = \{1,2,3,7,8\}$ خواهد بود.
- ✓ $|S|$ بیانگر تعداد عضوهای مجموعه S است. برای مثال، اگر $S = \{i: i \in [4,8]\}$ آنگاه $|S| = 5$. اگر $S = \{3,19,\pi,\sqrt{2}\}$ آنگاه $|S| = 4$. اگر $S = \{\alpha: 1 < \alpha < 3\}$ آنگاه $|S| = \infty$.
- ✓ $\emptyset$ مجموعه‌ی تهی است و $|\emptyset| = 0$.
- ✓ $x \bmod y$ باقیمانده تقسیم x بر y است. برای مثال $8 \bmod 3 = 2$.
- ✓ $\lceil x \rceil$ گرد شده‌ی عدد x به سمت بالاست و این مقدار برابر کوچکترین عدد صحیحی است که از x بزرگتر است. برای مثال، $\lceil 3.9 \rceil = 4$ و $\lceil 5 \rceil = 5$.
- ✓ $\lfloor x \rfloor$ گرد شده‌ی عدد x به سمت پایین است و این مقدار برابر بزرگترین عدد صحیحی است که از x کوچکتر است. برای مثال، $\lfloor 3.9 \rfloor = 3$ و $\lfloor 5 \rfloor = 5$.
- ✓ $min_x f(x)$ به مسئله‌ای اشاره دارد که هدف آن یافتن مقداری از x است که به ازای آن کوچکترین مقدار $f(x)$ به دست می‌آید. این عبارت همچنین معرف کوچکترین مقدار $f(x)$ است. برای مثال، فرض کنید $f(x) = (x-1)^2$. حال برای یافتن $min_x f(x)$ منحنی $f(x)$ را رسم کرده و مشاهده می‌کنیم در کدام نقطه $f(x)$ کمترین مقدار را خواهد داشت. همچنین می‌توان از مشتق‌گیری برای

پیدا کردن این مقدار استفاده کرد. در مورد این مثال خاص می‌توان دید که $min_x f(x) = 0$. تعریف مشابهی برای $max_x f(x)$ وجود دارد.

- ✓ $argmin_x f(x)$ مقداری از x است که به ازای آن $f(x)$ کمترین مقدار را دارد. برای مثال، دوباره فرض کنید $f(x) = (x-1)^2$. کمترین مقدار $f(x)$ برابر ۰ است که این مقدار به ازای $x = 1$ رخ می‌دهد، بنابراین $argmin_x f(x) = 1$. تعریفی مشابه آنچه گفته شد برای $argmax_x f(x)$ نیز وجود دارد.
- ✓ R^S مجموعه‌ای از تمامی بردارهای حقیقی با طول S است. این مجموعه می‌تواند هم بیانگر بردارهای ستونی و هم بردارهای سطری باشد.
- ✓ $R^{S \times P}$ مجموعه تمام ماتریس‌هایی با ابعاد $S \times P$ می‌باشد.
- ✓ $\{y_k\}_{k=L}^{U}$ مجموعه‌ای از تمام y_k هایی است که در آن k عددی صحیح بین U و L است. برای مثال، $\{y_k\}_{k=2}^{5} = \{y_2, y_3, y_4, y_5\}$.
- ✓ $\{y_k\}$ مجموعه‌ای از تمام y_k هایی است که مقدار k با توجه به متن کتاب مشخص می‌شود. برای مثال، فرض کنید که متن کتاب اشاره می‌دارد که سه مقدار از y یعنی y_1, y_2, y_3 موجودند، آنگاه $\{y_k\} = \{y_1, y_2, y_3\}$.
- ✓ $\exists$ به معنی "وجود دارد" و $\nexists$ به معنی "وجود ندارد" هستند. برای مثال، اگر $Y = \{1,6,9\}$، آنگاه $\exists\, y > 2 : y \in Y$. از سوی دیگر $\nexists\, y > 10 : y \in Y$.
- ✓ $A \to B$ یعنی عبارت B نتیجه‌ای است از عبارت A. برای مثال، $(x > 10) \to (x > 5)$.
- ✓ I ماتریس واحد است و ابعاد آن به متن کتاب بستگی دارد.

برای مشاهده‌ی نشان‌گذاری‌های بیشتر می‌توانید به لیست اختصارات مراجعه کنید.

۱–۶ طرح کلی کتاب

این کتاب به شش بخش تقسیم شده است:

۱. بخش I شامل فصل حاضر و یک فصل دیگر است که شامل موضوعات مقدماتی مرتبط با بهینه‌سازی است. آن انواع مختلف مسائل بهینه‌سازی و الگوریتم ساده در عین حال مؤثر تپه‌نوردی را معرفی کرده و در نهایت با بحث در مورد اینکه چه چیز یک الگوریتم را در زمره‌ی الگوریتم‌های هوشمند قرار می‌دهد سخن را به پایان خواهد برد.

۲. بخش II به بحث در مورد چهار الگوریتم تکاملی می‌پردازد که به‌طور معمول الگوریتم‌های کلاسیک در نظر گرفته می‌شوند: الگوریتم ژنتیک، برنامه‌نویسی تکاملی، استراتژی‌های تکاملی، برنامه‌نویسی ژنتیک.

بخش II همچنین شامل فصلی است که در مورد رویکردهای مختلف برای تحلیل ریاضی الگوریتم‌های ژنتیک صحبت خواهد کرد. این بخش با فصلی به پایان خواهد رسید که در مورد برخی تنوعات الگوریتم‌وار که می‌توان در این الگوریتم‌های کلاسیک به کار برد بحث خواهد کرد. این تنوعات در مورد الگوریتم‌های تکاملی مدرن نیز قابل استفاده هستند. الگوریتم‌های تکاملی مدرن در بخش بعدی پوشش داده شده‌اند.

۳. بخش III در مورد برخی الگوریتم‌های تکاملی مدرن بحث خواهد کرد. هرچند برخی از این الگوریتم‌ها چندان هم مدرن نبوده و تاریخچه‌ی آن‌ها به دهه‌ی ۱۹۸۰ بر می‌گردد، اما سایر آن‌ها مربوط به دهه‌ی اول قرن ۲۱ هستند.

٤. بخش IV در مورد انواع خاصی از مسائل بهینه‌سازی سخن خواهد گفت و نشان خواهد داد چگونه می‌توان با اصلاح الگوریتم‌های تکاملی معرفی شده در بخش‌های قبلی از آن‌ها برای حل مسائل یاری جست. این انواع خاص شامل موارد زیر هستند:
   - ✓ مسائل ترکیبی که دامنه‌شان اعداد صحیح است.
   - ✓ مسائل مقید (دارای محدودیت) که دامنه‌شان محدود به مجموعه‌ای خاص است.
   - ✓ مسائل چندهدفه که در آن‌ها بهینه کردن چند هدف به‌صورت همزمان مد نظر است.
   - ✓ مسائل با توابع برازندگی نویزی و پرهزینه که در آن‌ها محاسبه‌ی عملکرد راه‌حل‌های نامزد یا پرهزینه است یا سخت.

۵. بخش V شامل ضمایمی است که این ضمایم در مورد موضوعاتی مهم و یا جالب توجه بحث خواهند کرد.
   - ✓ ضمیمه الف شامل توصیه‌هایی کاربردی و گوناگون برای محققان و دانشجویان است.
   - ✓ ضمیمه ب به بیان قضیه No-Free-Lunch خواهد پرداخت و به ما خواهد گفت که روی هم رفته همه‌ی الگوریتم‌های تکاملی یکسان عمل می‌کنند. این ضمیمه همچنین چگونگی استفاده از مباحث آماری برای ارزیابی تفاوت میان الگوریتم‌های تکاملی را بیان خواهد نمود.
   - ✓ ضمیمه ج شامل برخی توابع استاندارد محک است که می‌توان از آن‌ها برای مقایسه‌ی عملکرد الگوریتم‌های تکاملی مختلف بهره برد.

۱–۷ درسی بر مبنای این کتاب

هرگونه درسی که بر پایه‌ی این کتاب باشد باید با فصل‌های ۱ و ۲، که شامل مروری بر مسائل بهینه‌سازی هستند، کار خود را آغاز کند. از آنجا به بعد، فصل‌های باقی‌مانده را می‌توان به هر ترتیبی مطالعه نمود. این موضوع به روش تدریس مدرس بستگی دارد. تنها استثنا واضح فصل‌های ۳ و ٤ هستند به این ترتیب که الگوریتم‌های ژنتیک (فصل ۳) باید قبل از مدل ریاضی‌شان (فصل ٤) مورد مطالعه و تدریس قرار بگیرند. همچنین حداقل یکی از فصل‌های بخش II یا III باید قبل از مطالعه‌ی فصلی از بخش IV به‌طور مفصل بررسی شود.

معمولاً دروس ارائه شده برای آنکه بتوانند زمینه‌ای مناسب درباره‌ی الگوریتم‌های تکاملی کلاسیک را به دانشجویان ارائه دهند، باید فصل ۳ و فصل‌های ۵ تا ۷ را مورد پوشش خود قرار دهند. اگر دانشجویان از مهارت ریاضی کافی برخوردار بوده و وقت کافی در دسترس باشد، فصل ٤ را نیز می‌توان در طول ترم در برنامه درسی گنجاند. فصل ٤ برای دانشجویان فوق لیسانس حائز اهمیت است چرا که در این فصل نشان داده خواهد شد که الگوریتم‌های تکاملی مبحثی صرفاً کیفی نیست بلکه می‌توان، و باید، پایه‌ی نظری نیز برای آن‌ها در نظر گرفت. امروزه بسیاری از تحقیقات در زمینه الگوریتم‌های تکاملی بر پایه‌ی اطلاعات و تنظیمات الگوریتم‌وار کوچک بوده و دارای اساسی ریاضی‌وار نیستند تا از آن‌ها حمایت کند. بسیاری از کسانی که از الگوریتم‌های تکاملی استفاده می‌کنند تنها به دنبال کسب نتیجه هستند که البته اشکالی هم ندارد؛ اما محققان علمی باید به همان اندازه که به استفاده از الگوریتم‌ها اهمیت می‌دهند، به اساس نظری آن‌ها نیز توجه داشته باشند. فصل‌های موجود در بخش III و IV می‌توانند بر اساس نظر شخصی مدرس مورد پوشش قرار بگیرند.

ضمایم موجود در انتهای کتاب جزو بخش‌های اصلی کتاب نیستند چرا که آن‌ها فی‌نفسه در مورد الگوریتم‌های تکاملی نیستند؛ اما با این حال اهمیت این ضمایم نباید دست کم گرفته شود. به‌طور خاص، مطالبی که در ضمایم ب و ج وجود دارد از اهمیت ویژه‌ای برخوردار بوده و باید در هر درسی که در مورد الگوریتم‌های تکاملی است مورد بحث قرار گیرند. من توصیه می‌کنم این دو ضمیمه بلافاصله بعد از تدریس اولین فصل در بخش‌های II و III مورد بحث و بررسی مفصل قرار گیرند.

با جمع بندی موارد فوق، طرح کلی برای درس فوق لیسانس در طول یک ترم به‌صورت زیر خواهد بود:

- ✓ فصل ۱ و ۲
- ✓ فصل ۳
- ✓ ضمایم ب و ج

- ✓ فصل‌های ٤–٨ پیشنهاد می‌شود برای دانشجویان لیسانس از فصل ٤ صرف نظر شود.
- ✓ برخی فصل‌های بخش III بنا به ترجیح مدرس. با وجود احتمال آغاز "جنگ الگوریتمی" بین من و خوانندگانم من با جرأت اعلام می‌دارم که ACO و PSO و DE از سایر الگوریتم‌های تکاملی مهم‌تر بوده و به همین خاطر معتقدم که مدرس باید حداقل فصل‌های ۱۰ تا ۱۲ را در تدریس خود مورد پوشش قرار دهد.
- ✓ برخی فصل‌های بخش IV بنا به ترجیح مدرس و مقدار زمان در دسترس.

فصل دوم

بهینه‌سازی

بهینه‌سازی کارهای ما را اشباع کرده و تقریبا در هر جنبه‌ای از مهندسی دخیل است.
(دنیس برنستاین[1]، ۲۰۰۶)

همان‌طور که جمله‌ی بالا نشان می‌دهد، قسمتی از همه‌ی کارهایی که ما انجام می‌دهیم شامل بهینه‌سازی است. برنامه‌ریزی‌های شخصی، روش‌های تدریس، سیستم‌های اقتصادی، استراتژی بازی‌ها، سیستم‌های بیولوژیکی و سیستم‌های درمانی-سلامتی، همگی نیاز به بهینه‌سازی دارند. بهینه‌سازی یک زمینه‌ی بسیار جذاب برای مطالعه و تحقیق است، نه تنها به خاطر محتوای نظری و الگوریتمیک آن، بلکه به خاطر قابلیت اجرایی گسترده‌ی آن.

مروری بر فصل

این فصل مروری خلاصه بر بهینه‌سازی خواهد داشت (بخش۲-۱) و شامل بهینه‌سازی مقید یا دارای محدودیت (بخش ۲-۲)، بهینه‌سازی با چندین هدف (بخش ۲-۳) و بهینه‌سازی‌های دارای چند راه‌حل (بخش ۲-٤) می‌شود. تمرکز اصلی ما در این کتاب بر مسائل بهینه‌سازی پیوسته خواهد بود، مسائلی که در آن‌ها متغیر مستقل می‌تواند به‌طور پیوسته تغییر نماید. با این حال، مسائلی که در آن‌ها متغیر مستقل به یک مجموعه‌ی متناهی محدود می‌شود، که به آن‌ها مسائل ترکیبی می‌گویند، نیز بسیار مورد توجه خواهند بود. در بخش ۲-۵ به معرفی این گونه مسائل خواهیم پرداخت. در بخش ۲-٦ به معرفی یک الگوریتم بهینه‌سازی ساده و کلی به نام الگوریتم تپه‌نوردی و همچنین برخی از انواع آن خواهیم پرداخت. در نهایت در بخش ۲-۷ با بحث در مورد برخی مفاهیم مرتبط با طبیعت هوش و چگونگی ارتباط آن‌ها با الگوریتم‌های تکاملی فصل حاضر را به پایان خواهیم برد.

۲-۱ بهینه‌سازی غیر مقید

بهینه‌سازی به‌صورت مجازی قابل اعمال به تمامی‌جنبه‌های زندگی است. الگوریتم‌های بهینه‌سازی را می‌توان به هر چیزی اعمال کرد. از تولید مثل کفتارها گرفته تا تحقیقات در زمینه‌ی لقاح و تخم بارورشده. تنها چیزی که می‌تواند دایره‌ی کاربردهای الگوریتم‌های تکاملی را محدود کند قدرت تخیل یک مهندس است. به همین دلیل است که طی چند دهه‌ی گذشته چنین تحقیقات گسترده‌ای بر روی الگوریتم‌های تکاملی انجام شده و چنین وسیع مورد استفاده واقع شده‌اند.

---

[1] Dennis Bernstein

برای مثال در مهندسی از الگوریتم‌های تکاملی برای یافتن بهترین مسیرهای یک روبات برای انجام یک کار معین استفاده می‌شود. فرض کنید شما در کارخانه‌ی خود یک روبات دارید و می‌خواهید این روبات کار خود را به سریع‌ترین شکل ممکن و یا با مصرف کمترین توان ممکن انجام دهد. چگونه می‌توانید بهترین مسیر را برای روبات بیابید؟ آنقدر تعداد مسیرهای ممکن زیاد است که پیدا کردن بهترین مسیر مانند آن است که بخواهیم در انبار کاه به دنبال سوزن بگردیم. اما الگوریتم‌های تکاملی اگر کار را آسان نکنند حداقل آن را کنترل‌پذیر کرده و اگر بهترین راه‌حل را به دست ندهند حداقل راه‌حل خوبی ارائه خواهند کرد. روبات‌ها بسیار غیر خطی‌اند، به همین علت فضای جستجو برای مسائل بهینه‌سازی روباتیک شامل قله‌ها و دره‌های زیادی است. مثالی که در ابتدای این فصل زده شد شاهدی بر این مدعاست. اما در مسائل روباتیکی وضعیت از این هم بدتر می‌شود چرا که در این مسائل، این قله‌ها و دره‌ها به جای آنکه در فضای سه بعدی معمول قرار بگیرند، در یک فضای چندبعدی واقع می‌شوند. این پیچیدگی در مسائل بهینه‌سازی روباتیک آن‌ها را به هدفی بالقوه برای الگوریتم‌های تکاملی مبدل می‌سازد. این ایده هم برای روبات‌های ایستا (مانند بازوی روباتیک) و هم برای روبات‌های متحرک به کار رفته است.

الگوریتم‌های تکاملی همچنین برای آموزش شبکه‌های عصبی[1] و سیستم‌های با منطق فازی[2] به کار گرفته شده‌اند.

در شبکه‌های عصبی باید ساختار شبکه و همچنین وزن نورون‌ها را برای دستیابی به بهترین عملکرد ممکن شبکه بیابیم. در اینجا نیز آنقدر تعداد راه‌حل‌های محتمل زیاد است که انجام این کار بسیار سخت و دشوار می‌نماید. اما در اینجا نیز می‌توان از الگوریتم‌های تکاملی برای یافتن بهترین ساختار و بهترین وزن‌ها بهره برد. موضوعی کاملا مشابه در مورد سیستم‌های با منطق فازی وجود دارد. از چه قوانینی باید استفاده کنیم؟ از چه تعداد تابع عضویت باید استفاده کرد؟ از چه نوع تابع عضویتی باید استفاده نمود؟ الگوریتم‌های ژنتیک می‌توانند به حل این مسائل بهینه‌سازی دشوار کمک نمایند.

الگوریتم‌های تکاملی حتی برای تشخیص‌های پزشکی نیز به کار رفته‌اند. برای مثال، بعد از بافت برداری، متخصصان چگونه تشخیص دهند کدام سلول‌ها سرطانی شده و کدامیک سرطانی نیستند؟ برای تشخیص سرطان باید به دنبال چه ویژگی‌ای باشند؟ چه ویژگی‌هایی مهم تلقی شده و کدامیک نامربوط خوانده می‌شوند؟ اگر بیمار متعلق به یک جامعه‌ی آماری خاص باشد، کدام ویژگی مهم خواهد بود؟ یک الگوریتم تکاملی می‌تواند به گرفتن چنین تصمیم‌هایی کمک کند. الگوریتم تکاملی همواره نیاز به یک متخصص دارد

[1] Neural Network
[2] Fuzzy Logic Systems

تا الگوریتم را آغاز کرده و به آن تعلیم دهد اما بعد از این الگوریتم تکاملی می‌تواند بهتر از معلم خود عمل کند. الگوریتم تکاملی نه تنها خسته و وامانده نمی‌شود، بلکه الگوهایی بسیار ظریف، که انسان قادر به شناسایی‌شان نیست را استخراج کند. تا کنون از الگوریتم‌های تکاملی برای تشخیص چندین نوع از سرطان‌ها استفاده شده است.

بعد از آن که بیماری تشخیص داده شد، سؤال بعدی در مورد چگونگی مدیریت بیماری خواهد بود. برای مثال، بعد از اینکه سرطان شناسایی شود، بهترین درمان برای بیمار چه خواهد بود؟ اگر راه درمان تشعشع است، چه نوع تشعشعی و برای چه مدت و با چه مقداری مورد نیاز است؟ اثرات جانبی را چه باید کرد؟ این نیز خود یک مسئله‌ی پیچیده‌ی بهینه‌سازی دیگر است. مضرات یک درمان اشتباه از فواید آن بیشتر خواهد بود. تعیین راه درمان مناسب یک تابع پیچیده از مواردی همچون نوع سرطان، محل سرطان، جامعه‌ی آماری بیمار، سلامتی کلی و فاکتورهای دیگر است؛ بنابراین از الگوریتم‌های ژنتیک نه تنها برای تشخیص بیماری بلکه برای تعیین درمان مناسب نیز استفاده می‌شود.

الگوریتم‌های تکاملی هرگاه که بخواهید مسئله‌ای دشوار را حل نمائید باید مد نظر قرار داده شوند. این بدان معنی نیست که الگوریتم‌های تکاملی همواره بهترین انتخاب هستند. ماشین حساب‌ها برای جمع اعداد از الگوریتم‌های تکاملی استفاده نمی‌کنند چرا که الگوریتم‌های بسیار ساده‌تر و مؤثرتری برای این کار وجود دارد! اما الگوریتم‌های تکاملی را باید حداقل برای حل مسائل پیچیده در نظر گرفت. اگر می‌خواهید یک پروژه‌ی مسکن و یا یک سیستم حمل و نقل طراحی کنید، الگوریتم‌های تکاملی جواب شما خواهند بود. اگر می‌خواهید یک مدار پیچیده‌ی الکتریکی و یا یک برنامه‌ی کامپیوتری طراحی کنید، می‌توانید از یک الگوریتم تکاملی برای انجام کارتان بهره ببرید.

یک مسئله‌ی بهینه‌سازی می‌تواند به‌صورت یک مسئله‌ی مینیمم‌سازی و یا ماکزیمم‌سازی مطرح شود. برخی اوقات ما به دنبال مینیمم کردن یک تابع هستیم و گاهی نیز به دنبال ماکزیمم کردن. این دو نوع مسئله به سادگی به یکدیگر تبدیل می‌شوند:

$$\begin{aligned} &min_x f(x) \leftrightarrow max_x |-f(x)| \\ &max_x f(x) \leftrightarrow min_x |-f(x)| \end{aligned} \qquad (۱-۲)$$

تابع $f(x)$ تابع هدف نام داشته و بردار x، متغیر مستقل و یا متغیر تصمیم خوانده می‌شود. توجه داشته باشید که با توجه به متن کتاب عبارت "متغیر مستقل" و یا "متغیر تصمیم" گاهی به کل بردار x اشاره داشته و گاهی نیز به عنصر خاصی در بردار x اشاره می‌کند. عناصر موجود در x گاها "ویژگی‌های راه‌حل" نیز خوانده می‌شوند. تعداد عناصر موجود در x نیز بعد یا دیمانسیون مسئله نامیده می‌شود. همان‌طور که از

معادله‌ی ۲-۱ پیداست، می‌توان از هر الگوریتمی که برای مینیمم کردن یک تابع طراحی شده است برای ماکزیمم کردن نیز بهره برد. عکس این مسئله نیز صادق است. هرگاه سعی کنیم تابعی را مینیمم کنیم، تابع را تابع هزینه نامیده و هرگاه بخواهیم تابعی را ماکزیمم کنیم، تابع را تابع برازندگی می‌نامیم.

$$\begin{aligned} &min_x f(x) \Rightarrow \text{تابع هزینه یا هدف نامیده می‌شود } f(x) \\ &max_x f(x) \Rightarrow \text{تابع برازندگی یا هدف نامیده می‌شود } f(x) \end{aligned} \qquad (۲-۲)$$

مثال ۲-۱

این مثال عبارات به کار رفته در این کتاب و مفاهیم آن‌ها را نشان می‌دهد. فرض کنید می‌خواهیم تابع زیر را مینیمم کنیم:

$$f(x,y,z) = (x-1)^2 + (y+2)^2 + (z-5)^2 + 3 \qquad (۳-۲)$$

متغیرهای x، y و z متغیرهای مستقل یا متغیرهای تصمیم یا ویژگی‌های راه‌حل نامیده می‌شوند. این مسئله یک مسئله‌ی سه بعدی است. $f(x,y,z)$ تابع هدف یا تابع هزینه خوانده می‌شود. با تعریف $g(x,y,z) = -f(x,y,z)$ می‌توان مسئله را به یک مسئله‌ی ماکزیمم‌سازی تبدیل نمود و تابع $g(x,y,z)$ را ماکزیمم نمود. تابع $g(x,y,z)$ تابع هدف یا برازندگی نام دارد. راه‌حل مسئله‌ی $min_x f(x,y,z)$ همان راه‌حل مسئله‌ی $max_x g(x,y,z)$ است و در این راه‌حل $x = 1$ و $y = -2$ و $z = 5$ خواهد بود. در کل جواب بهینه‌ی $f(x,y,z)$ منفی جواب بهینه‌ی $g(x,y,z)$ خواهد بود.

همان‌طور که در مثال بعد مشاهده خواهد شد، گاهاً مسئله‌ی بهینه‌سازی ساده بوده و با روش‌های تحلیلی قابل حل است.

مثال ۲-۲

مسئله‌ی زیر را در نظر بگیرید.

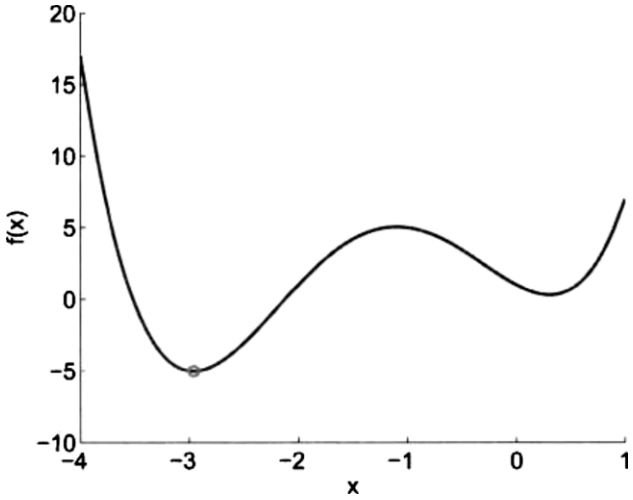
$$min_x \mathrm{f(x)} = ? \ , \qquad f(x) = x^4 + 5x^3 + 4x^2 - 4x + 1 \qquad (٤-۲)$$

منحنی تابع $f(x)$ در شکل ۲-۱ نشان داده شده است. از آنجا که $f(x)$ یک تابع چندجمله‌ای از درجه‌ی ٤ است، دارای سه نقطه‌ی ساکن می‌باشد. این سه نقطه در واقع مقادیری از x هستند که به ازای آن‌ها $f'(x) = 0$ خواهد بود. همان‌طور که از شکل ۲-۱ قابل مشاهده است، این سه مقدار $x = -2.96$ ،

$x = -1.10$ و $x = 0.31$ هستند. می‌دانیم مقدار $f'(x)$، که برابر $4x^3 + 15x^2 + 8x - 4$ است، در این سه نقطه برابر صفر خواهد بود. مقدار مشتق دوم تابع $f(x)$ در این سه نقطه برابرست با:

$$f''(x) = 12x^2 + 30x + 8 = \begin{cases} 24.33 & ; \quad x = -2.96 \\ -10.48 & ; \quad x = -1.10 \\ 18.45 & ; \quad x = 0.31 \end{cases} \qquad (۵-۲)$$

به خاطر آورید که مقدار مشتق دوم یک تابع در مینیمم محلی مثبت و در ماکزیمم محلی منفی است. مقادیر در نقاط ثابت نشان می‌دهد که $x = -2.96$ و $x = 0.31$ مینیمم محلی بوده و $x = -1.10$ ماکزیمم محلی است.



شکل ۲-۱ مثال ۲-۲: یک مثال ساده‌ی مینیمم‌سازی. تابع $f(x)$ دو مینیمم محلی و یک ماکزیمم محلی دارد.

تابع مثال ۲-۲ دو مینیمم محلی و یک مینیمم کلی دارد. توجه داشته باشید که مینیمم کلی خود یک مینیمم محلی است. برای برخی از توابع $min_x f(x)$ در بیش از یک مقدار از x اتفاق می‌افتد. در این صورت این توابع دارای چند مینیمم کلی خواهند بود. مینیمم محلی x^* را می‌توان به‌صورت زیر تعریف نمود:

$$f(x^*) < f(x) \quad \text{به طوری که برای تمامی } x \text{ها } \|x - x^*\| < \varepsilon \qquad (۶-۲)$$

در اینجا $\|.\|$ یک ماتریس فاصله بوده و $\varepsilon > 0$ یک همسایگی دلخواه است. در شکل ۲-۱ می‌توان دید که اگر برای مثال اندازه‌ی همسایگی $\varepsilon = 1$ باشد، $x = 0.31$ یک بهینه‌ی محلی است. اما اگر $\varepsilon = 4$ در نظر

گرفته شود، $x = 0.31$ یک بهینه‌ی محلی نخواهد بود. مینیمم کلی x^* را نیز می‌توان به‌صورت زیر تعریف کرد:

$$f(x^*) \leq f(x) \quad \text{به ازای تمامی} x\text{ها} \qquad (۷-۲)$$

۲–۲ بهینه‌سازی مقید (دارای محدودیت)[۱]

در بسیاری از مواقع مسئله‌ی بهینه‌سازی دارای قید و محدودیت است. این‌گونه مسائل شامل مینیمم کردن تابعی مانند $f(x)$ هستند به طوری که مقادیر مجاز x دارای محدودیت هستند. مثال زیر این مطلب را به روشنی توضیح می‌دهد.

مثال ۲–۳

تابع زیر را در نظر بگیرید.

$$f(x) = x^4 + 5x^3 + 4x^2 - 4x + 1 \ , \ x \geq -1.5 \ , \ min_x f(x) = ? \qquad (۸-۲)$$

این مسئله در واقع همان مسئله‌ی مثال ۲–۲ است تنها با این تفاوت که x در این مسئله دارای محدودیت است. منحنی $f(x)$ برای مقادیر مجاز x در شکل ۲–۲ نشان داده شده است. با کمی دقت در منحنی می‌توان مینیمم محدود را تشخیص داد. برای حل و تحلیل این مسئله ابتدا باید مانند مثال ۲–۲ نقاط ثابت را به دست آورده و سپس مقادیر غیر مجاز را از محاسبات خود حذف کنیم. با این کار در خواهیم یافت که دو مینیمم محلی در نقاط $x = 0.31$ و $x = -2.96$ اتفاق می‌افتند که از بین این دو تنها $x = 0.31$ در شرایط و محدودیت مسئله صدق می‌کند. در قدم بعدی باید مقدار $f(x)$ را در کران‌های بازه‌ی محدودیت محاسبه کرده و آن را با مینیمم‌های محلی مقایسه کرده تا مینیمم کلی به دست آید. بدین ترتیب خواهیم داشت:

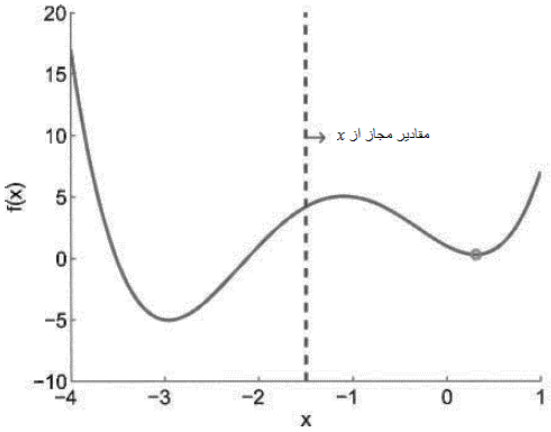
$$f(x) = \begin{cases} 4.19 \ , \ x = -1.5 \\ 0.30 \ , \ x = 0.31 \end{cases} \qquad (۹-۲)$$

می‌توان دید که $x = 0.31$ مقدار مینیمم‌ساز برای این مسئله است.

اگر کران پایین بازه‌ی محدودیت را بر روی محور حقیقی به سمت چپ حرکت دهیم، مقدار مینیمم‌ساز x ممکن است به جای آن که در مینیمم محلی $x = 0.31$ رخ دهد، در کران پایین بازه‌ی محدودیت اتفاق

[۱] توجه شود عبارت‌های مقید و دارای محدودیت و همچنین دو عبارت قید و محدودیت با یکدیگر هم ارز و برابرند.

بیافتد. اگر کران پایین بازه‌ی محدودیت از $x = -2.96$ کوچکتر شود، آنگاه مقدار مینیمم‌ساز x برای این مسئله با مثال ۲-۲ یکسان خواهد بود.



شکل ۲-۲ مثال ۲-۳: یک مسئله‌ی مینیمم‌سازی مقید ساده. مینیمم مقید در $x = 0.31$ اتفاق می‌افتد.

مسائل بهینه‌سازی موجود در دنیای واقعی تقریبا همیشه دارای قید و محدودیت هستند. در این مسائل، مقادیر بهینه‌ساز متغیرهای مستقل در بازه‌ای از محدودیت قرار می‌گیرند. این موضوع عجیب نیست چراکه ما به‌طورمعمول انتظار داریم بهترین طراحی مهندسی، بهترین تخصیص منابع و هر هدف دیگری را با مورد استفاده قرار دادن تمامی منابع موجود به دست آوریم (برنستاین، ۲۰۰۶). به همین علت، محدودیت‌ها تقریباً در تمامی مسائل بهینه‌سازی واقعی دارای اهمیت هستند. فصل ۱۹ در این باره به تفصیل بحث خواهد کرد.

۲-۳ بهینه‌سازی چندهدفه

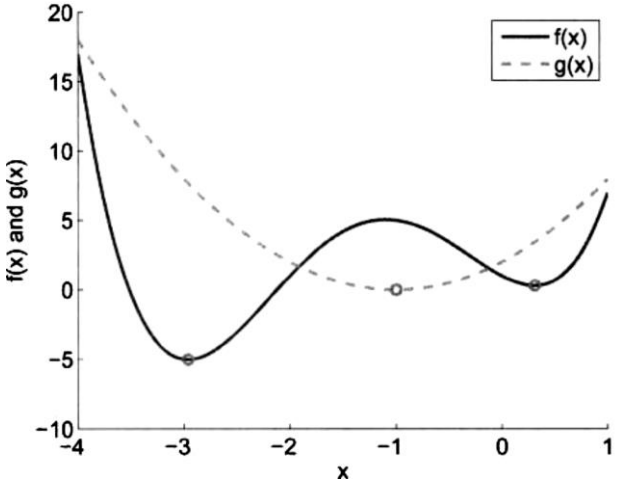
مسائل بهینه‌سازی عملی نه تنها دارای قید و محدودیت هستند بلکه چندهدفه نیز می‌باشند. این بدان معناست که ما علاقه داریم چند مقدار را به‌طورهمزمان مینیمم کنیم. برای مثال، در مسائل کنترل موتور ممکن است مینیمم‌سازی مصرف سوخت و همچنین مینیمم‌سازی خطای ردگیری هر دو مد نظر باشند. در این مسئله می‌توان خطای ردگیری را با مصرف سوخت زیاد بسیار کم کرد، حال آنکه برای مصرف سوخت کم باید خطای ردگیری زیاد را به جان خرید. می‌توان موتور را خاموش کرد تا مصرف سوخت به صفر برسد اما در این حالت خطای ردگیری جالب نخواهد بود.

مثال ۲-٤

مسئله‌ی زیر را در نظر بگیرید.

$$f(x) = x^4 + 5x^3 + 4x^2 - 4x + 1 \ , \ g(x) = 2(x-1)^2 \ , \ min_x[f(x), g(x)] = ? \qquad (۱۰-۲)$$

اولین تابع هدف برای بهینه‌سازی همان تابع مثال ۲-۲ است. اما در اینجا می‌خواهیم تابع $g(x)$ را نیز مینیمم کنیم. توابع $f(x)$ و $g(x)$ و مینیمم آن‌ها در شکل ۲-۳ نشان داده شده‌اند. با توجه در این منحنی‌ها متوجه می‌شویم که $x = -2.96$ تابع $f(x)$ را مینیمم می‌نماید در حالی که $g(x)$ در $x = -1$ مینیمم می‌شود. در این مسئله مقدار مطلوب x مشخص نیست چرا که دو هدف در تقابل با یکدیگر قرار دارند. با این حال، از شکل ۲-۳ مشخص است که مقادیری از x که کوچکتر از ۲.۹٦- و یا بزرگتر از ۰.۳۱ در هر حال مطلوب نخواهند بود، چراکه برای این مقادیر از x هر دو تابع $f(x)$ و $g(x)$ مقادیری بیشتر از مینیمم‌شان پیدا خواهند کرد و این مطلوب نیست.



شکل ۲-۳ مثال ۲-٤: یک مسئله‌ی مینیمم‌سازی چندهدفه‌ی ساده. $f(x)$دو مینیمم محلی و $g(x)$ یک مینیمم محلی دارد. دو هدف در تقابل با یکدیگر قرار دارند.

یک راه برای حل این مسئله آن است که $g(x)$ را به عنوان تابعی از $f(x)$ رسم نماییم. این مطلب در شکل ۲-٤ نشان داده شده است. در این شکل x بین ۳.٤- و ۰.۸ متغیر است. حال به بررسی هر قسمت از این منحنی می‌پردازیم.

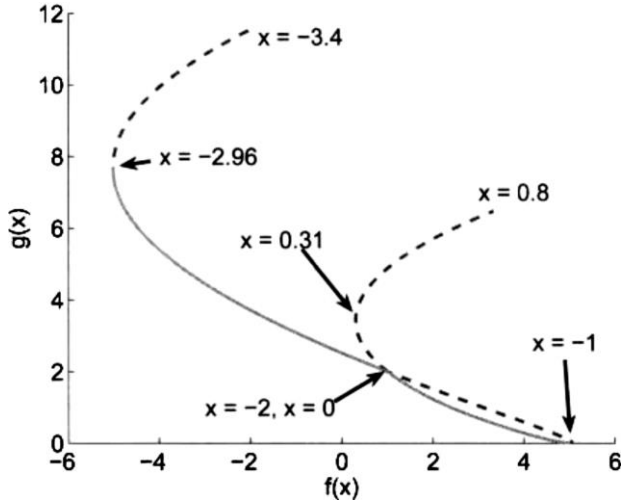
- $x \in [-3.4,2.96]$: با افزایش x از 3.4- تا 2.96- هر دو نابع $f(x)$ و $g(x)$ کاهش می‌یابند. بنابراین ما هیچگاه $x < -2.96$ را انتخاب نخواهیم کرد.
- $x \in [-2.96,-1]$: با افزایش x از 2.96- تا 1-، $g(x)$ کاهش یافته و $f(x)$ افزایش می‌یابد.
- $x \in [-1,0]$: با افزایش x از 1- تا 0، $g(x)$ افزایش یافته و $f(x)$ کاهش می‌یابد. با این حال، در این قسمت از منحنی با اینکه $g(x)$ افزایش می‌یابد اما افزایش آن کمتر از حالت $x \in [-2,-1]$ است. بنابراین $x \in [-1,0]$ بر $x \in [-2,-1]$ ارجحیت دارد.
- $x \in [0,0.31]$: با افزایش x از ۰ تا 0.31، $g(x)$ افزایش و $f(x)$ کاهش می‌یابد. با نگاه به منحنی می‌توان دید که مقدار $g(x)$ برای $x \in [0,0.31]$ از $x \in [-2.96,2]$ بزرگتر است. بنابراین، هیچگاه $x \in [0,0.31]$ انتخاب نخواهد شد.
- $x \in [0.31,0.8]$: در نهایت، با افزایش x از 0.31 تا 0.8 هم $f(x)$ و هم $g(x)$ افزایش می‌یابند. بنابراین $x > 0.31$ هیچگاه مطلوب نیست.

با جمع بندی نتایج فوق، مقادیر بالقوه مطلوب $f(x)$ و $g(x)$ را با خط توپر در شکل ۲-٤ رسم کرده‌ایم. برای مقادیری از x که بر روی خط توپر قرار دارند، نمی‌توان مقداری دیگر از x پیدا کرد که $f(x)$ و $g(x)$ را به‌طورهمزمان کاهش دهد. خط توپر ناحیه‌ی پَرِتو[1] نام دارد و مجموعه‌ی مقادیر مطلوب x نیز مجموعه‌ی پرتو نامیده می‌شود.

$$\begin{aligned} &\text{مجموعه‌ی پرتو} : x^* = \{x: x \in [-2.96,-2] \text{ یا } x \in [-1,0]\} \\ &\text{ناحیه‌ی پرتو} : \{[f(x), g(x)]: x \in x^*\} \end{aligned} \qquad (۲-۱۱)$$

بعد از به دست آوردن ناحیه‌ی پرتو، می‌توان بیشتر در موردمقادیر بهینه‌ساز xسخن گفت. . انتخاب نقطه‌ای از ناحیه‌ی پرتو به عنوان راه‌حل تنها به دید مهندسی حاکم بر مسئله بستگی دارد. ناحیه‌ی پرتو مجموعه‌ای از جواب‌های منطقی را به دست می‌دهد، اما انتخاب هر x از ناحیه‌ی پرتو متضمن تبادل و تعادل میان دو هدف مسئله می‌باشد.

[1] Pareto



شکل ۲-٤ مثال ۲-٤: منحنی $g(x)$ بر حسب $f(x)$ برای یک مسئله‌ی مینیمم‌سازی چندهدفه‌ی ساده به طوری که x میان -3.4 و 0.8 متغیر است. خط توپر نشان دهنده‌ی ناحیه‌ی پرتو می‌باشد.

مثال ۲-٤ یک مسئله‌ی بهینه‌سازی چندهدفه‌ی بسیار ساده و دارای تنها دو هدف است. یک مسئله‌ی بهینه‌سازی معمول در دنیای واقعی بسیار بیشتر از دو هدف را دنبال کرده و به همین علت به دست آوردن ناحیه‌ی پرتوی آن بسیار مشکل است. حتی اگر بتوانیم به نحوی ناحیه‌ی پرتو را به دست آوریم قادر نخواهیم بود آن را به تصویر بکشیم چرا که این ناحیه چندبعدی خواهد بود. فصل ۲۰ به‌طورمفصل به بهینه‌سازی چندهدفه خواهد پرداخت.

۲-۴ بهینه‌سازی چندپیمانه ای

یک مسئله‌ی بهینه‌سازی چندپیمانه‌ای مسئله‌ای است که بیش از یک مینیمم محلی دارد. شکل ۲-۱ معرف چنین مسئله‌ای بود اما در آنجا تنها دو مینیمم محلی وجود داشت. به همین علت مسئله به راحتی قابل حل بود. با این حال، برخی مسائل دیگر دارای چندین مینیمم محلی بوده و به همین دلیل پیدا کردن این که کدامیک از آن‌ها مینیمم کلی است کاری چالش برانگیز است.

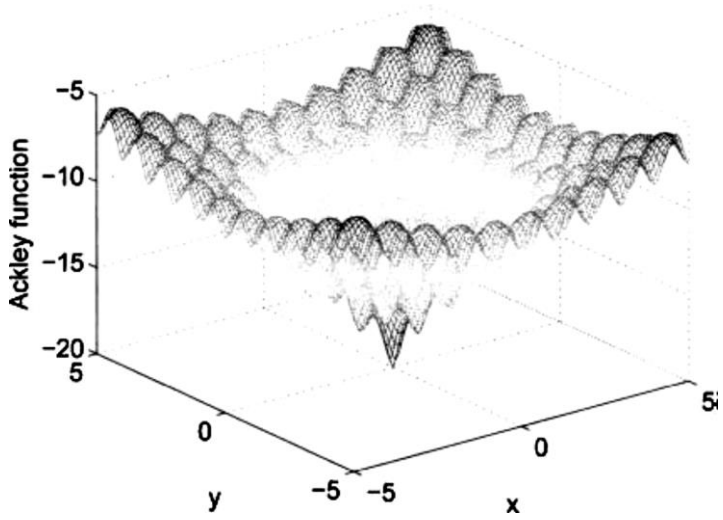
مثال ۲-۵

تابع زیر را در نظر بگیرید.

$$min_{x,y} f(x,y) = ? \qquad (۱۲-۲)$$

$$f(x,y) = e - 20\exp\left(-0.2\sqrt{\frac{x^2+y^2}{2}}\right) - \exp\left(\frac{\cos(2\pi x)+\cos(2\pi y)}{2}\right)$$

این یک تابع اکلی[1] دو بعدی است. این تابع در شکل ۲-۵ رسم شده است و در ضمیمه‌ی ۱-۲.ج توضیح داده شده است. یک منحنی مانند آنچه در شکل ۲-۵ قابل مشاهده است را معمولاً *دورنمای برازندگی*[2] می‌نامند چرا که به شکل هندسی نحوه‌ی تغییرات تابع برازندگی و یا تابع هزینه نسبت به متغیرهای مستقل را نشان می‌دهد. اگر تعداد ابعاد زیاد شوند قادر به رسم این منحنی نخواهیم بود. با این حال، حتی اگر مسئله بیش از دو بعد داشته باشد باز هم برازندگی و یا هزینه‌ای که تابعی از متغیرهای مستقل باشد را دورنمای برازندگی می‌نامند. شکل ۲-۵ نشان می‌دهد که حتی در دو بعد تابع اکلی دارای تعداد زیادی مینیمم محلی است. حال تصور کنید تعداد این مینیمم‌ها در ۲۰ و ۳۰ بعد چه قدر خواهد بود. برای حل این مسئله می‌توان مشتق $f(x,y)$ را نسبت به x و y حساب کرد و سپس با حل $f_x(x,y) = f_y(x,y) = 0$ مقادیر مینیمم محلی را پیدا نمود. اما به هر حال حل همزمان این معادلات کاری دشوار است.



شکل ۲-۵ مثال ۲-۵: تابع اکلی دوبعدی

مثال قبل نشان داد چرا الگوریتم‌های تکاملی مفید هستند. ما توانستیم مسائل شکل‌های ۲-۲ و ۲-۳ و ۲-٤ را با روش‌های هندسی و محاسباتی حل نماییم اما در دنیای واقعی بسیاری از مسائل مانند مثال ۲-۵

[1] Ackley
[2] Fitness Landscape

هستند تنها با این تفاوت که در این مسائل تعداد متغیرهای مستقل و تعداد هدف‌ها بیشتر بوده و همچنین محدودیت بیشتری وجود دارد. در این گونه مسائل روش‌های هندسی و محاسباتی به تنگ آمده و الگوریتم‌های تکاملی نتایج بهتری را به دست خواهند داد.

۲–۵ بهینه‌سازی ترکیبی

فرض کنید یک مدیر می‌خواهد به چهار شعبه از اداره‌اش سرکشی کند به طوری که از دفتر اصلی‌اش شروع کرده و در نهایت به آن بازگردد. دفتر اصلی واقع در شهر A بوده و سایر شعبات در شهرهای B و C و D قرار دارند. این مدیر می‌خواهد به‌گونه‌ای به این شعبات سرکشی کند که مجموع کل مسافت سفرش مینیمم شود. شش راه‌حل ممکن برای این مسئله وجود دارد که آن‌ها را با S_i نمایش می‌دهیم.

$$\begin{aligned} S_1&: \ A \to B \to C \to D \to A \\ S_2&: \ A \to B \to D \to C \to A \\ S_3&: \ A \to C \to B \to D \to A \\ S_4&: \ A \to C \to D \to B \to A \\ S_5&: \ A \to D \to B \to C \to A \\ S_6&: \ A \to D \to C \to B \to A \end{aligned} \qquad (۱۳–۲)$$

این مدیر می‌تواند به سادگی با محاسبه‌ی کل مسافت برای هر یک از راه‌حل‌ها این مسئله را حل نماید[۱]. مسئله‌ی مثال ۲–٦ به مسئله‌ی فروشنده‌ی دوره گرد (TSP)[۲] بسته مشهور است[۳]. برای یک مسئله‌ی TSP با چهار شهر می‌توان به راحتی تمام راه‌حل‌های ممکن را محاسبه نمود. محاسبه و جستجوی تمام راه‌حل‌های ممکن یک مسئله‌ی ترکیبی را جستجوی brute-force یا جستجوی جامع می‌نامند. اگر برای این کار وقت داشته باشید این راه بهترین راه برای حل مسائل ترکیبی خواهد بود چرا که رسیدن به بهترین جواب را تضمین می‌کند.

حال این سؤال پیش می‌آید: برای یک مسئله‌ی TSP با n شهر چند راه‌حل ممکن وجود دارد؟ با کمی تفکر جواب را خواهیم یافت: $(n-1)!$ راه‌حل وجود دارد. این عدد با افزایش n با سرعت بسیار زیادی افزایش می‌یابد و حتی برای مقادیر نسبتا کم n هم محاسبه‌ی همه‌ی راه‌حل‌ها غیر ممکن خواهد بود. فرض کنید این مدیر باید در هر یک از ۵۰ ایالت آمریکا به یک شهر سر بزند. تعداد راه‌حل‌های ممکن

---

[۱] در واقع این مسئله ساده‌تر از چیزی است که به نظر می‌رسد. S_1 مسیر برعکس S_6 است. به همین علت مسافت این دو راه‌حل با هم یکی است. این موضوع برای S_2 و S_4 و همچنین S_3 و S_5 نیز صادق است.

[2] Traveling Salesman Problem

[۳] مسئله‌ی فروشنده‌ی دوره‌گرد باز مسئله‌ای است که در آن از هر شهری دقیقا یک بار بازدید شده بدون آنکه به شهر اول بازگردیم.

$49! = 6.1 \times 10^{62}$ خواهد بود. آیا کامپیوترهای مدرن قادر به به محاسبه‌ی این مقدار از راه‌حل‌های ممکن هستند؟ جهان حدود ۱۵ میلیارد سال عمر دارد. این عدد برابر 4.7×10^{17} ثانیه است. حال فرض کنید یک تریلیون کامپیوتر از آغاز جهان در حال پردازش هستند. همچنین فرض کند هر کدام از این کامپیوترها قادرند در هر ثانیه مقدار یک تریلیون راه‌حل را محاسبه کنند. در این حالت، اگر از ابتدای به وجود آمدن کیهان در حال محاسبه می‌بودیم، تا به امروز قادر بوده‌ایم تنها 4.7×10^{41} راه‌حل را محاسبه کنیم. این یعنی به جواب حتی نزدیک هم نشده‌ایم!

جور دیگری هم می‌توان به مسئله‌ی TSP نگاه کرد. بر روی زمین بین 10^{20} تا 10^{24} دانه‌ی شن وجود دارد (ولند[1]، ۲۰۰۹). اگر هر یک از این دانه‌های شن را یک سیاره فرض کنیم که به اندازه‌ی زمین دانه‌ی شن دارند، باز هم تعداد راه‌حل‌های ممکن TSP با ۵۰ شهر بسیار بزرگتر از جمع تمام این ریز شن‌ها خواهد بود. کاملا مشهود است برای چنین مسئله‌ی بزرگی نمی‌توان از جستجوی brute-force استفاده نمود.

دیدیم که برخ مسائل آن قدر بزرگ‌اند که رویکرد brute-force در مورد آن‌ها عملی نیست. از سوی دیگر، مسائل ترکیبی مانند TSP دارای متغیرهای مستقل پیوسته نیستند و به همین علت نمی‌توان آن‌ها را با مشتق‌گیری حل نمود. تا هنگامی‌که تمام راه‌حل‌های ممکن را امتحان نکرده باشیم نمی‌توانیم از یافتن بهترین راه‌حل برای مسئله‌ی ترکیبی مطمئن باشیم. با این حال، الگوریتم‌های تکاملی وسیله‌ای قدرتمند برای یافتن راه‌حل‌های مناسب هستند. الگوریتم‌های تکاملی جادو نمی‌کنند، اما به ما کمک می‌کنند جواب‌هایی خوب و مناسب برای این نوع مسائل چندبعدی و بزرگ پیدا کنیم، حتی اگر این جواب‌ها بهترین راه‌حل ممکن نباشند. راه‌حل‌های بالقوه در یک الگوریتم تکاملی اطلاعاتشان را با یکدیگر به اشتراک گذاشته و در نهایت به اجماعی می‌رسند که این اجماع همان بهترین راه‌حل است. ما نمی‌توانیم اثبات کنیم که جواب نهایی الگوریتم بهترین راه‌حل است چرا که برای این کار باید تمامی راه‌حل‌های ممکن را بیازماییم. اما وقتی که راه‌حل‌های الگوریتم‌های تکاملی را با دیگر راه‌حل‌ها مقایسه می‌کنیم، می‌بینم که الگوریتم‌های تکاملی بسیار کارآمد هستند. فصل ۱۸ مسائل بهینه‌سازی تکاملی ترکیبی را با جزییات بیشتری بررسی خواهد نمود.

۲-۶ تپه‌نوردی

این بخش یک الگوریتم ساده‌ی بهینه‌سازی به نام تپه‌نوردی را معرفی می‌نماید. تپه‌نوردی در واقع به خانواده‌ای از الگوریتم‌ها با تنوع بسیار زیاد اطلاق می‌شود. برخی محققان تپه‌نوردی را یک الگوریتم تکاملی ساده در نظر گرفته در حالی که برخی دیگر آن را یک الگوریتم غیرتکاملی می‌دانند. ساده بودن، به طرز

[1] Welland

شگفت‌آوری مؤثر بودن، داشتن تنوعات ساده‌ی بسیار و فراهم آوردن محک‌های مناسب برای مقایسه با سایر الگوریتم‌های تکاملی، همگی از جمله دلایلی هستند که تپه‌نوردی را در هنگام مواجه با یک مسئله‌ی بهینه‌سازی جدید به اولین انتخاب مناسب تبدیل می‌کنند. مفهوم تپه‌نوردی به قدری سرراست است که احتمالاً به دفعات و در زمان‌های بسیار دور اختراع شده است. به همین علت تعیین کردن مبدأ و سرچشمه‌ی آن کاری بس دشوار است.

اگر می‌خواهید به مرتفع‌ترین نقطه‌ی یک چشم‌انداز برسید، یک راه منطقی آن است که قدمی‌در جهت شدیدترین شیب بردارید. بعد از قدم اول باید دوباره شیب تپه را ارزیابی کرده و قدم دوم را نیز در جهت شدیدترین شیب بردارید. این فرآیند آنقدر ادامه خواهد یافت تا دیگر هیچ جهتی شما را به نقطه‌ای مرتفع‌تر هدایت نکند و این یعنی شما به بالاترین نقطه‌ی تپه رسیده‌اید. این یک استراتژی جستجوی محلی است و تپه‌نوردی نام دارد.

یک استراتژی بهتر آن است که ابتدا نگاهی به اطراف انداخته و سپس مرتفع‌ترین نقطه و همچنین بهترین راه رسیدن به آن را تخمین بزنید. این کار مشکل زیگزاگ شدن مسیر و همچنین گیر افتادن بر روی قله‌ی تپه‌های کوچک که ارتفاعشان از ارتفاع تپه‌ی اصلی کمتر است را حل خواهد کرد. اما اگر میدان دید کم باشد آنگاه به احتمال زیاد استراتژی جستجوی محلی بهترین راه‌حل خواهد بود.

تپه‌نوردی بسته به شکل تپه، تعداد ماکزیمم‌های محلی و نقطه‌ی آغاز می‌تواند خوب و یا بد عمل کند. تپه‌نوردی به خودی خود می‌تواند به عنوان الگوریتم بهینه‌سازی مورد استفاده قرار گیرد. همچنین می‌توان آن را با یک الگوریتم تکاملی مخلوط کرد و به این ترتیب توانایی جستجوی کلی الگوریتم تکاملی را با توانایی جستجوی محلی تپه‌نوردی ترکیب نمود. استراتژی تپه‌نوردی انواع مختلفی دارد (میتچل[1]، ۱۹۹۸)، که در مورد برخی از آن‌ها در این بخش بحث خواهیم کرد.

شکل ۲-٦ الگوریتم تپه‌نوردی با شدیدترین شیب[2] را نشان می‌دهد. این الگوریتم روند محافظه‌کارانه‌ای دارد، به این ترتیب که هر بار تنها یک ویژگی راه‌حل را تغییر داده و راه‌حل بهینه‌ی حاضر را با راه‌حل بهینه‌ی به دست آمده از این تغییر جایگزین می‌نماید.

[1] Mitchell
[2] Steepest Ascent Hill Climbing

$x_0 \leftarrow$ ذره ی تولید شده ی اتفاقی

تا زمانی که شرایط قطع عملیات فراهم نشده

مقدار برازندگی $f(x_0)$ را برای x_0 محاسبه کن

برای هر ویژگی راه حل q=1,…,n

q امین ویژگی x_q را با یک جهش جایگزین کن

مقدار برازندگی $f(x_q)$ را محاسبه کن

ویژگی راه حل بعدی

$$x' \leftarrow argmax_{x_q}(f(x_q): q \in [0,n])$$

اگر $x_0=x'$ آنگاه

$x_0 \leftarrow$ ذره ی تولید شده ی اتفاقی

در غیر این صورت

$x_0 \leftarrow x'$

پایان اگر

نسل بعدی

شکل ۲-٦ شبه کد بالا طرح کلی الگوریتم تپه‌نوردی با شدیدترین شیب برای ماکزیمم‌سازی تابع بعدی $f(x)$ را نشان می‌دهد. توجه داشته باشید که x_q برابر x_0 است و تنها q امین ویژگی آن دستخوش تغییر شده است.

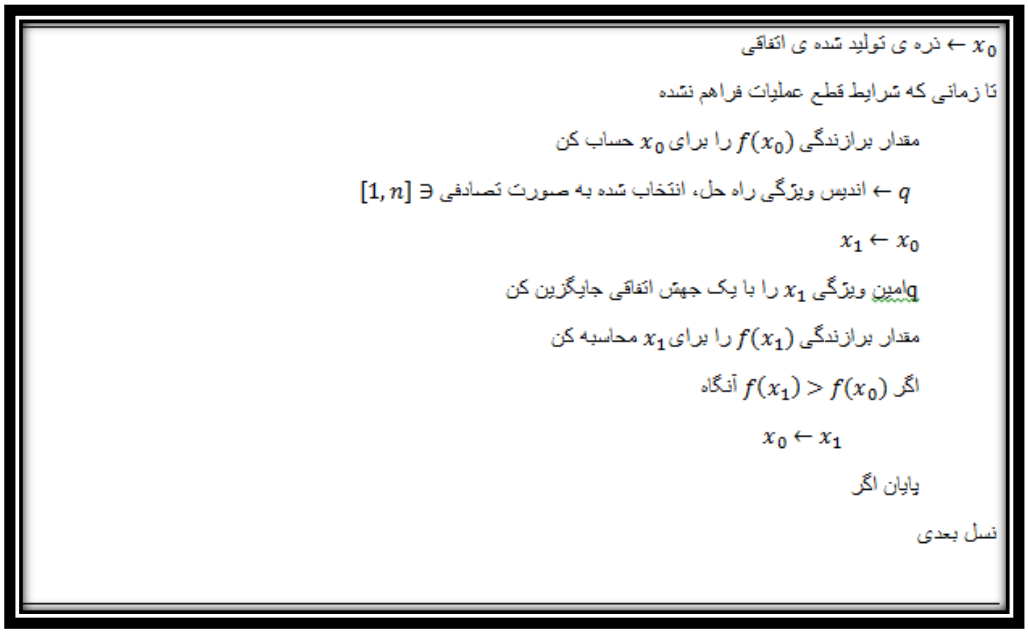
شکل ۲-۷ الگوریتم تپه‌نوردی با شیب جنبی را که تپه‌نوردی ساده نیز نامیده می‌شود نشان می‌دهد. این الگوریتم نیز همانند تپه‌نوردی با شدیدترین شیب، هربار تنها یک ویژگی راه‌حل را دچار جهش می‌کند. اما بهینه‌سازی با شیب جنبی کمی حریصانه‌تر عمل می‌کند چرا که به محض آنکه راه‌حلی بهتر پیدا می‌شود آن را جایگزین راه‌حل حاضر می‌کند.

دو الگوریتم تپه‌نوردی بعدی انتخاب ویژگی راه‌حل برای جهش را به‌صورت اتفاقی انجام می‌دهند و به همین دلیل در کلاس کلی تپه‌نوردی اتفاقی قرار می‌گیرند. شکل ۲-۸ الگوریتم تپه‌نوردی با جهش اتفاقی را نشان می‌دهد. این الگوریتم بسیار مشابه الگوریتم تپه‌نوردی شیب جنبی است تنها با این تفاوت که در این الگوریتم ویژگی راه‌حل جهش یافته به‌صورت اتفاقی انتخاب شده است. شکل ۲-۹ الگوریتم تپه‌نوردی تطبیقی را به تصویر می‌کشد. این الگوریتم مشابه الگوریتم تپه‌نوردی با جهش اتفاقی است با این تفاوت که قبل از آنکه راه‌حل جهش یافته با راه‌حل بهینه‌ی حاضر مقایسه شود، هر ویژگی راه‌حل با یک احتمالی دچار جهش می‌شود.

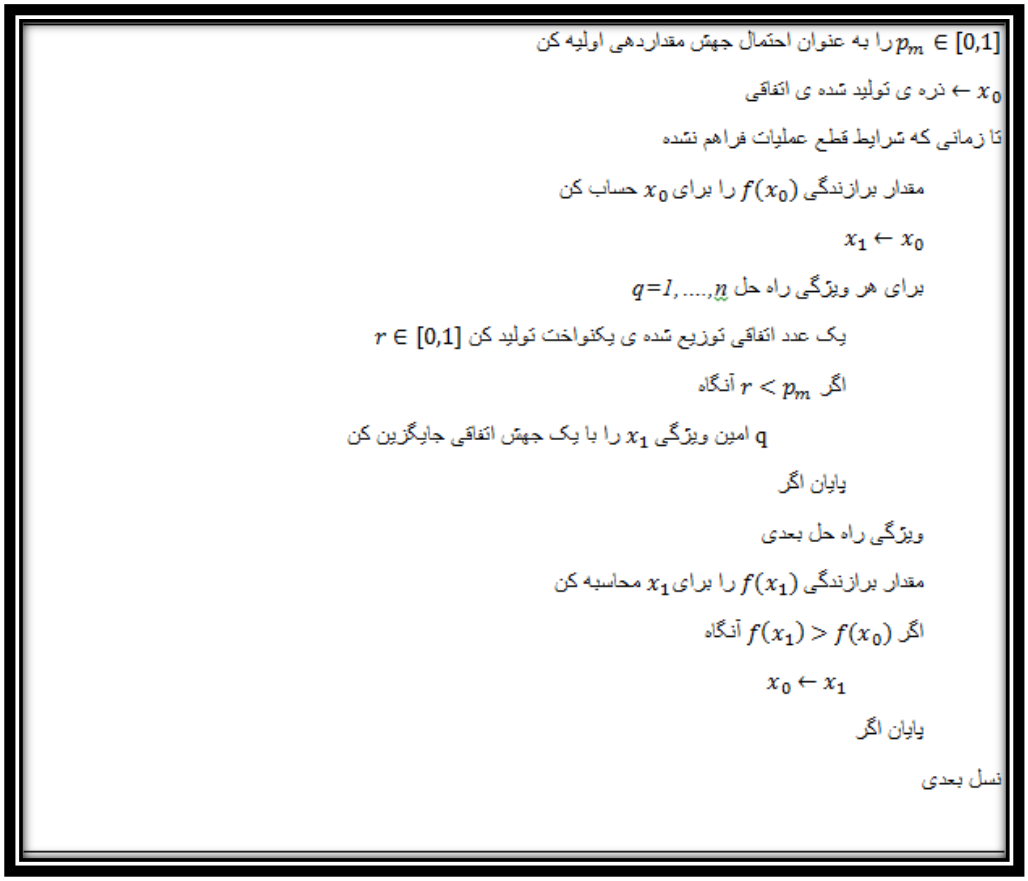
نتایج یک الگوریتم تپه‌نوردی (شکل ۲-٦ تا ۲-۹) قویا به شرایط اولیه‌ی x_0 بستگی دارد. به همین دلیل آزمودن تپه‌نوردی با چند نقطه‌ی آغازین اتفاقی مختلف کاری عقلانی به نظر می‌رسد. به این رویکرد که در آن الگوریتم تپه‌نوردی را داخل حلقه‌ای از شرایط اولیه قرار می‌دهند، بازآغاز نمودن اتفاقی تپه‌نوردی می‌گویند.

```
x0 ← ذره ی تولید شده ی اتفاقی
تا زمانی که شرایط قطع عملیات فراهم نشده
    مقدار برازندگی f(x0) را برای x0 حساب کن
    ReplaceFlag ← نادرست
    برای هر ویژگی راه حل q = 1, ...., n
        xq ← x0
        qامین ویژگی راه حل را با یک جهش اتفاقی جایگزین کن
        مقدار برازندگی f(xq) را برای xq محاسبه کن
        اگر f(xq) > f(x0) آنگاه
            x0 ← xq
            ReplaceFlag ← درست
        پایان اگر
    ویژگی راه حل بعدی
    اگر نه (ReplaceFlag)
        x0 ← ذره ی تولید شده ی اتفاقی
    پایان اگر
نسل بعدی
```

شکل ۲-۷ شبه کد بالا طرح کلی الگوریتم تپه‌نوردی جنبی برای ماکزیمم‌سازی تابع د بعدی $f(x)$ را نشان می‌دهد. توجه داشته باشید که x_q با x_0 برابر است با این تفاوت که qامین ویژگی آن دچار جهش شده است.



شکل ۲-۸ شبه کد بالا طرح کلی الگوریتم تپه‌نوردی با جهش اتفاق برای ماکزیمم‌سازی تابع n بعدی $f(x)$ را نمایش می‌دهد. توجه داشته باشید که x_1 با x_0 برابر است تنها با این تفاوت که ۱ ویژگی دچار دگرگونی شده است.



شکل ۲-۹ شبه کد بالا طرح کلی الگوریتم تپه‌نوردی تطبیقی برای ماکزیمم‌سازی تابع n بعدی $f(x)$ را نمایش می‌دهد. توجه داشته باشید که x_q با x_0 برابر است با این تفاوت که qامین ویژگی آن دچار جهش شده است.

مثال ۲-۷

در این مثال چهار الگوریتم تپه‌نوردی یاد شده را برای یک مجموعه مسائل محک ۲۰ بعدی شبیه‌سازی می‌کنیم (ضمیمه‌ی ج.۱ را ببینید). توجه داشته باشید که مسائل محک موجود در ضمیمه ج.۱ مسائل مینیمم‌سازی هستند، به همین علت الگوریتم‌های تپه‌نوردی را به شکلی سرراست تعدیل کرده تا به الگوریتم‌های دره‌نوردی[1] که نقطه‌ی مقابل تپه‌نوردی است، برسیم. هر الگوریتم را برای هر محک ۵۰ بار اجرا می‌کنیم و در هر بار اجرا از شرایط اولیه‌ای متفاوت استفاده می‌نماییم. برای تپه‌نوردی تطبیقی از

[1] Hill Descending

$p_m = 0.1$ استفاده می‌کنیم. همچنین هر یک از الگوریتم‌های تپه‌نوردی را بعد از ۱۰۰۰ بار ارزیابی تابع برازندگی خاتمه می‌دهیم.

جدول ۲–۱ نتایج را نشان می‌دهد. توجه داشته باشید که تپه‌نوردی با شدیدترین شیب (شکل ۲–٦) در هر نسل به حداقل n بار ارزیابی تابع برازندگی نیاز دارد (برای این مثال $n = 20$ است)، این در حالی است که تپه‌نوردی با جهش اتفاقی (شکل ۲–۸) به یک بار ارزیابی تابع برازندگی در هر نسل نیاز دارد. معمولاً قسمت اعظم محاسبات در الگوریتم‌های اکتشافی به ارزیابی تابع برازندگی اختصاص دارد (فصل ۲۱ را ببینید). به همین علت، برای انجام یک مقایسه‌ی عادلانه میان الگوریتم‌های بهینه‌سازی مختلف، باید تعداد ارزیابی‌ها از تابع برازندگی برای همگی یکسان باشد. اگر الگوریتم‌های بهینه‌ساز مختلف در هر نسل دارای تعداد ارزیابی‌های یکسان از تابع برازندگی باشند (برای مثال تپه‌نوردی با شدیدترین شیب در شکل ۲–٦ و تپه‌نوردی با شیب جنبی در شکل ۲–۷)، آنگاه مقایسه‌ی آنها بر حسب تعداد نسل‌ها عادلانه خواهد بود.

جدول ۲–۱ نشان می‌دهد که برای ۱۲ محک از ۱ محک، تپه‌نوردی با جهش اتفاقی بهترین عملکرد را داشته و به‌طور میانگین بسیار بهتر از سایر روش‌های تپه‌نوردی عمل می‌کند. تپه‌نوردی تطبیقی و تپه‌نوردی شدیدترین شیب هر دو تنها در یک محک و آن هم به مقدار جزیی، از تپه‌نوردی با جهش اتفاقی بهتر عمل می‌کنند. با این حال، عملکرد تپه‌نوردی تطبیقی می‌تواند به شدت به نرخ جهش بستگی داشته باشد (مسئله‌ی ۲–۱۱ را ببینید). به همین دلیل در این مثال سعی شده است از بهترین نرخ جهش استفاده شود.

جدول ۲-۱ مثال ۲-۷: عملکرد الگوریتم‌های تپه‌نوردی. جدول مقادیر مینیمم نرمالیزه شده توسط چهار الگوریتم تپه‌نوردی و همچنین میانگین آنها در طول ۵۰ شبیه سازی مونت کارلو نشان می‌دهد. از آنجا که تپه‌نوردی اتفاقی است، نتایج شما ممکن است متفاوت با این جدول باشد.

محک	شدیدترین شیب	شیب جنبی	جهش اتفاقی	تطبیقی
اکلی	۲,۲۷	۱,۸۲	۱,۰۰	۱,۷۰
فلچر[1]	۱,٦۲	۱,۸۷	۱,۰۰	۱,٦۸
گرینوانک[2]	۹,۵۸	٤,٤۱	۱,۰۰	۳,۸۱
مجازات شماره ۱	۲٦٦۲٤	۲۱٦۰	۱,۰۰	۲۸۱
مجازات شماره ۲	۹۹۳٤۷	۵٦۹۰	۱,۰۰	٤۱۷۸
درجه چهار	۱۳۳,۹٤	۲۹,۹۹	۱,۰۰	۲۵,٦۱
رستریجین	۳,۷٦	۲,۵۲	۱,۰۰	۲,۱۰
روزنبروک[3]	۲,٦۸	۱,۵۰	۱,۰۰	۱,۷۲
اشوفل۱,۲[4]	۱,٦۳	۱,۳۷	۱,۲٤	۱,۰۰
اشوفل ۲,۲۱	۱,۰۰	۱,۷۵	۱,۰۲	۱,۱۲
اشوفل ۲,۲۲	۳,٦۵	۲,۷۳	۱,۰۰	۲,۳۰
اشوفل ۲,۲٦	۵,۰۵	۳,٦۳	۱,۰۰	۲,۹۱
کروی[5]	۱۷,۹۷	۷,۳۲	۱,۰۰	٦,۰۹
Step	۱٦,۵۸	٦,۷۸	۱,۰۰	٦,۵۲

۲-۶-۱ الگوریتم‌های بهینه‌سازی بایاس شده

در این قسمت می‌خواهیم به یک پیش‌بینی احتیاطی مرتبط با توابع محک که باید در طول دوره‌ی مطالعه‌ی بهینه‌سازی به آن توجه داشت بپردازیم. این پیش‌بینی احتیاطی شامل دو جمله‌ی مرتبط با هم می‌شوند: اول

[1] Fletcher
[2] Grienwank
[3] Rosenbrock
[4] Schwefel
[5] Sphere

آنکه، مینیمم بسیاری از توابع هزینه‌ی محک در میانه‌ی دامنه‌ی جستجو قرار دارند و ثانیاً، بسیاری از الگوریتم‌های بهینه‌سازی به سمت میانه‌ی دامنه‌ی جستجو بایاس (گرایش) دارند[۱].

در مورد پدیده‌ی بایاس در ضمیمه‌ی ج.۷ به تفصیل سخن خواهیم گفت. به خواننده قویاً توصیه می‌شود قبل از انجام هرگونه تحقیق پیشرفته این ضمیمه را به دقت مطالعه کند. بسیاری از نتایج شبیه‌سازی موجود در این کتاب بر اساس توابع محکی هستند که مینیمم آن‌ها در مرکز دامنه‌ی جستجو قرار دارد. آن نتایج به دنبال به تصویر کشیدن دقیق کارایی الگوریتم بهینه‌سازی نیستند بلکه تنها برآنند تا کاربرد یک الگوریتم بهینه‌سازی خاص را نشان دهند. قبل از آنکه بتوانیم در مورد کارایی الگوریتم بهینه‌سازی نتیجه‌گیری مطمئنی انجام دهیم باید روش غیربایاس کردن موجود در ضمیمه ج.۷ را پیاده‌سازی نماییم.

۲-۶-۲ اهمیت شبیه‌سازی مونت کارلو

توجه داشته باشید که در مثال ۲-۷ ما نتایج ۵۰ شبیه‌سازی را برای رسم کارایی GA[۲] دودویی و GA پیوسته میانگین گرفتیم. نشان دادن نتایج یک شبیه‌سازی چیزی را اثبات نمی‌کند چرا که نتایج به یک مولد اعداد تصادفی وابسته‌اند. با یک بار شبیه‌سازی و یا یک بار آزمایش به سختی می‌توان به نتایج معتبری دست یافت. در این باره در ضمیمه‌ی ب بحث شده است، اما در اینجا نیز متذکر می‌شویم از نتایج چند شبیه‌سازی میانگین گرفته شده است.

شبیه‌سازی چندباره برای تحلیل کارایی، به‌طورمعمول شبیه‌سازی مونت کارلو نامیده می‌شود. این نام از سه دانشمند به نام‌های جان ون نیومن[۳]، استانیس لاو اولام[۴] و نیکولاس متروپولیس[۵] که در دهه‌ی ۱۹۴۰ بر روی سلاح‌های هسته‌ای کار می‌کردند گرفته شده است. بیشتر کار آن‌ها شامل تحلیل نتایج چندین آزمایش می‌شد و در این میان ارتباط واضحی میان تحلیل آماری آزمایش‌هایشان و ویژگی‌های آماری قمار وجود داشت. این ارتباط و همچنین این موضوع که عموی اولام یک قمارباز بدنام در کازینوی مونت کارلو در موناکو بود، آن‌ها را به سمت نام "شبیه‌سازی مونت کارلو" هدایت کرد (متروپولیس، ۱۹۸۷).

---

[۱] این موضوع تنها برای الگوریتم‌های تپه‌نوردی موجود در این قسمت صادق نیست بلکه برای بسیاری از الگوریتم‌های تکاملی که بعداً درباره‌شان بحث خواهیم کرد نیز صادق است.

[2] Genetic Algorithm
[3] Jan Van Neumann
[4] Stanislaw Ulam
[5] Nicholas Metropolis

۲-۷ هوش

در روزهای ابتدایی رشته‌ی کامپیوتر، محققان متوجه شدند که کامپیوترها در انجام کارهایی که انسان‌ها در انجامشان ضعیف عمل می‌کنند، مانند محاسبه‌ی مسیر موشک‌های بالستیک، عملکرد خوبی دارند. اما کامپیوترها در انجام کارهایی که انسان‌ها به خوبی انجام می‌دهند، مانند تشخیص چهره، چندان مؤثر واقع نشده و نمی‌شوند. این موضوع باعث شد با تقلید از رفتار بیولوژیکی، برای بهتر ساختن عملکرد کامپیوترها در چنین زمینه‌هایی تلاش‌های صورت گیرد. این تلاش‌ها در نهایت به تکنولوژی‌هایی چون سیستم‌های فازی، شبکه‌های عصبی، الگوریتم‌های ژنتیک و سایر الگوریتم‌های تکاملی منجر شد. به همین علت است که الگوریتم‌های تکاملی به عنوان جزئی از دسته‌بندی هوش کامپیوتری در نظر گرفته می‌شوند. ما تلاش خواهیم کرد الگوریتم‌های تکاملی را با ایجاد الگوریتم‌های هوشمند بسط دهیم. اما هوشمند بودن به چه معناست؟ آیا بدین معناست که الگوریتم‌های تکاملی ما در تست IQ نمره‌ی خوبی کسب می‌کنند؟ این بخش در مورد هوش و برخی مشخصات آن مانند تطبیق، اتفاقی بودن، ارتباط، فیدبک، اکتشاف و بهره‌برداری یا انتفاع بحث خواهد کرد. این‌ها مشخصه‌هایی هستند که ما در هنگام جستجوی الگوریتم‌های هوشمند در الگوریتم‌های تکاملی پیاده‌سازی خواهیم کرد.

۲-۷-۱ تطبیق

ما به‌طورمعمول تطبیق با محیط متغیر را از ویژگی‌ها هوش در نظر می‌گیریم. فرض کنید شما سر هم نمودن یک قطعه‌ی فرضی را یاد می‌گیرید، مدیرتان از شما می‌خواهد تا یک قسمت از یک ماشین را که شما تا به حال ندیده‌اید، سر هم کنید. اگر شما هوشمند باشید خواهید توانست آنچه را که در مورد قطعه‌ی فرضی می‌دانید به آن قسمت از ماشین تعمیم داده و در نهایت آن را سرهم کنید. اما اگر شما چندان هوشمند نباشید مجبور خواهید بود در مورد آن قسمت از ماشین تمام جزئیات را بدانید تا بتوانید آن را سرهم کنید. با این حال تطبیق تنها شرط لازم برای هوشمندی نیست چرا که برای مثال کنترل‌کننده‌های تطبیقی (آستروم[1] و ویتنمارک[2]، ۲۰۰۸) هوشمند در نظر گرفته نمی‌شوند. یک ویروس که تاب تحمل شرایط محیطی بسیار بد را دارد هوشمند خوانده نمی‌شود. بنابراین نتیجه می‌گیریم که تطبیق شرط لازم ولی ناکافی برای هوشمندی است. تلاش ما بر آن است تا الگوریتم‌های تکاملی‌ای طراحی کنیم که به بسیاری از مسائل قابل تطبیق باشند. قابلیت تطبیق تنها یکی از چندین معیار یک الگوریتم تکاملی موفق است.

---

[1] Astrom
[2] Wittenmark

۲-۷-۲ اتفاقی بودن

ما به‌طور معمول به اتفاقی بودن به عنوان جمله‌ای منفی و نامطلوب نگاه می‌کنیم. هیچ یک از ما از اتفاقات غیر قابل پیش‌بینی در زندگی‌هایمان استقبال نمی‌کنیم و همواره سعی در کنترل محیطمان داریم. با این حال، اتفاقی بودن تا یک حدی از عناصر لازم هوش است. یک گورخر را در نظر بگیرید که در حال فرار از یک شیر است. اگر این گورخر در جهت مستقیم و با سرعت ثابتی حرکت کند به راحتی شکار خواهد شد. اما یک گورخر هوشمند در مسیری زیگزاگ و غیر قابل پیش‌بینی حرکت خواهد کرد تا از دست شکارچی در امان باشد. حال فرض کنید که یک شیر می‌خواهد یک گورخر را شکار کند. شیر هر روز در کمین گله‌ی گورخرها می‌نشیند. اگر شیر هر روز در یک زمان و مکان ثابت در کمین بنشیند، گورخرها به راحتی می‌توانند از مواجه با شیر اجتناب کنند. اما یک شیر هوشمند در زمان‌ها و مکان‌های متفاوت و به‌گونه‌ای غیر قابل پیش‌بینی حمله می‌کند. به همین جهت است که می‌گوییم اتفاقی بودن از ویژگی‌های هوش است.

از سوی دیگر اتفاقی بودن به مقدار زیاد مخرب خواهد بود. اگر گورخر به‌طوراتفاقی تصمیم بگیرد هنگام فرار بر روی زمین دراز بکشد، می‌توان هوشش را زیر سؤال برد. اگر یک شیر هنگام جستجو برای یافتن یک گورخر به‌طوراتفاقی تصمیم به کندن چاله بگیرد، آنگاه ما حق خواهیم داشت در هوشمندی‌اش شک کنیم. بنابراین اتفاقی بودن یکی از خصیصه‌های هوشمندی است اما محدودیت‌هایی نیز دارد.

الگوریتم‌های تکاملی‌ای که ما طراحی می‌کنیم برخی عناصر اتفاقی بودن را دارا هستند. اگر اتفاقی بودن را از طراحی‌هایمان حذف کنیم، الگوریتم‌های تکاملی‌مان عملکرد خوبی نخواهند داشت. از سوی دیگر استفاده‌ی زیادی از اتفاقی بودن نیز بر عملکرد الگوریتم‌ها تأثیر منفی خواهد گذاشت. ما باید از مقدار مناسبی اتفاقی بودن در طراحی‌هایمان استفاده کنیم. صد البته، همان‌طور که پیش از این نیز گفته شد، الگوریتم‌های تکاملی قابلیت تطبیق دارند. به همین علت الگوریتم‌های تکاملی خوب برای مقادیری از اتفاقی بودن خوب عمل می‌کنند. نمی‌توان انتظار داشت که الگوریتم‌های تکاملی به قدری تطبیق‌پذیر باشند که بتوان از هر مقدار اتفاقی بودن استفاده نمود، اما آنقدر تطبیق‌پذیر خواهند بود که مقدار دقیق میزان اتفاقی بودن چندان مهم و بحرانی نباشد.

۲-۷-۳ ارتباط

ارتباط یک ویژگی هوش است. فرض کنید یک فرد نابغه می‌خواهد تست IQ بدهد ولی قادر به برقراری ارتباط نیست. او در این آزمون رد خواهد شد هرچند که یک نابغه است. بسیاری از افراد کر و لال و متوهم در آزمون IQ رد می‌شوند هرچند که بسیار هوشمندند. بچه‌هایی که به دور از فعل و انفعالات انسانی بزرگ

می‌شوند غیرخلاق‌اند، هوشمند نیستند و خوشحال نخواهند بود (نیوتون[1]، ۲۰۰٤). عدم ارتباط با دیگران در طول سال‌های شکل‌گیری آن‌ها باعث می‌شود نتوانند ظرفیت ذهنی و فکری خود را توسعه دهند. این سال‌های انزوا غیرقابل جبران‌اند و آن‌ها قادر نخواهند بود نحوه‌ی برقراری ارتباط را بیاموزند و به همین علت هم نمی‌توانند خود را با جامعه وفق دهند.

هوش نه تنها شامل ارتباط می‌شود بلکه خود معلول و مشمول است. به عبارت دیگر، هوشمندی از جمعیتی از افراد بر می‌آید. یک فرد نمی‌تواند به تنهایی هوشمند باشد. می‌توان گفت که در جهان افراد هوشمند بسیاری وجود دارند و حتی اگر یکی از این افراد منزوی شود باز هم هوشمند خواهد بود. با این حال، این افراد هوشمندی خود را از ارتباط با دیگران کسب کرده‌اند. یک مورچه‌ی تنها بی‌هدف و سرگردان خواهد بود و کاری انجام نمی‌دهد اما یک کلونی از مورچه‌ها می‌توانند کوتاه‌ترین مسیر برای یافتن غذا را یافته، شبکه‌ای استادانه از تونل‌ها را به وجود آورده و خودشان را به عنوان یک اجتماع خودمختار مدیریت کنند. به همین ترتیب، یک فرد بدون ارتباط با یک اجتماع نمی‌تواند کاری از پیش ببرد. یک اجتماع می‌تواند انسان را به ماه بفرستد، میلیون‌ها نفر را از طریق اینترنت به یکدیگر متصل کند و در بیابان سیستم‌های منابع آب و غذا بسازد.

می‌توان دید که هوش و ارتباط تشکیل یک حلقه‌ی فیدبک مثبت را می‌دهند. برای گسترش هوش نیاز به ایجاد ارتباط است و برای ایجاد ارتباط نیاز به هوشمندی است. اما نکته‌ی اصلی در اینجا این است که ارتباط یک ویژگی هوش است. به همین علت است که بیشتر الگوریتم‌های تکاملی شامل جمعیتی از راه‌حل‌های نامزد برای مسائل هستند. این راه‌حل‌های نامزد، که ما آن‌ها را ذره می‌نامیم، با یکدیگر ارتباط برقرار کرده و از شکست‌ها و پیروزی‌های یکدیگر تجربه کسب می‌کنند. با گذر زمان، جمعیت ذره‌ها به سمت راه‌حلی خوب و مناسب برای مسئله حرکت خواهد کرد.

۲-۷-۴ فیدبک

فیدبک یک ویژگی بنیادین هوش است. این موضوع شامل تطبیق، که در مورد آن بحث شد، نیز می‌شود. یک سیستم اگر نتواند محیط خود را حس کرده و به آن واکنش نشان دهد، نمی‌تواند با محیط تطبیق پیدا کند. با این حال، فیدبک شامل چیزی بیش از انطباق، یعنی یادگیری نیز می‌شود. وقتی ما اشتباهی را مرتکب می‌شویم، از این اشتباه درس گرفته و دیگر آن را تکرار نمی‌کنیم[2]. مهمتر از این، وقتی دیگران مرتکب اشتباه

[1] Newton

[2] آلبرت انیشتین دیوانگی را انجام چندین باره‌ی یک کار و انتظار نتیجه‌های متفاوت تعریف می‌کند.

می‌شوند ما از اشتباه آن‌ها درس گرفته و رفتارمان را به‌گونه‌ای تغییر می‌دهیم تا اشتباهات آنان را تکرار نکنیم. شکست تولید فیدبک منفی کرده و در نقطه‌ی مقابل، پیروزی تولید فیدبک مثبت می‌کند و ما را به سمت اتخاذ رفتارهایی که منجر به پیروزی می‌شوند سوق می‌دهد. گاهاً مشاهده می‌کنیم که برخی افراد از اشتباهات درس نمی‌گیرند و رفتارهایی را که اثبات شده منجر به پیروزی می‌شوند را اتخاذ نمی‌کنند. ما این افراد را هوشمند در نظر نمی‌گیریم.

فیدبک همچنین پایه‌ی بسیاری از پدیده‌های طبیعی است. چرخه‌ی آب شامل توالی بینهایت از باران و تبخیر است. باران بیشتر منجر به تبخیر بیشتر و تبخیر بیشتر منجر به باران بیشتر می‌شود. از آنجا که این فرایند شامل مقدار ثابتی از آب است، چرخه‌ی آب میزان پایداری از رطوبت بر روی زمین و در آسمان را منجر می‌شود. اگر به طریقی این مکانیزم فیدبک دچار اختلال شود، مشکلات زیادی از جمله سیل و خشکسالی رخ خواهد داد.

تعادل میان قند و انسولین در بدن انسان نیز یک مکانیزم فیدبک است. هرچه بیشتر قند مصرف کنیم، پانکراسمان بیشتر انسولین تولید خواهد کرد. هرچه پانکراسمان بیشتر انسولین تولید کند، قند بیشتری از خون جذب خواهد شد. قند زیاد باعث بروز ازدیاد قند خون شده و مقدار بسیار کم قند در خون باعث بروز Poglycemia می‌شود. دیابت از نتایج بروز اختلال در مکانیزم فیدبک قند/انسولین است و می‌تواند باعث بروز مشکلات جدی و طولانی مدت برای سلامتی انسان شود.

این توصیف از هوش به عنوان علامت مشخصه‌ی هوش اغلب در نظریه کنترل هوشمند رسمیت پیدا می‌کند. فیدبک یک شرط کافی برای هوش نیست. هیچ کس کنترل‌کننده‌ی نسبی را هوشمند نخوانده و هیچ کس یک ترموستات مکانیکی را هوشمند نمی‌داند. فیدبک یک شرط لازم اما ناکافی برای هوش است.

طراحی‌های ما از الگوریتم‌های تکاملی فیدبک مثبت و منفی را با هم ترکیب می‌کند. یک الگوریتم تکاملی بدون فیدبک چندان مؤثر واقع نخواهد شد اما یک الگوریتم تکاملی با فیدبک، این شرط لازم را برآورده خواهد کرد.

۲-۷-۵ اکتشاف و انتفاع (بهره‌برداری)

اکتشاف جستجوی ایده‌ها و استراتژی‌های جدید است. انتفاع، استفاده از ایده‌ها و استراتژی‌های موجود است که موفقیتشان قبلاً اثبات شده است. اکتشاف فرایندی با ریسک بالاست چرا که بسیاری از ایده‌های جدید زمان را هدر داده و به بن بست می‌رسند. با این حال، اکتشاف می‌تواند بسیار پرثمر باشد، چرا که بسیاری از ایده‌های جدید چنان جواب‌هایی به دست خواهند داد که شاید فکرش را هم نتوانیم بکنیم. انتفاع

رابطه‌ای نزدیک با استراتژی‌های فیدبک بحث شده دارد. کسی که هوشمند است از داشته‌ها و دانسته‌های خود استفاده می‌کند و به دنبال اختراع دوباره‌ی چرخ نمی‌رود! از سوی دیگر همین فرد هوشمند از ایده‌های جدید استقبال کرده و حاضر است به نحوی حساب شده ریسک کند. هوش شامل تعادلی مناسب میان اکتشاف و انتفاع است. این تعادل مناسب به میزان نظم محیطمان بستگی دارد. اگر محیطمان به سرعت در حال تغییر است دانشمان به سرعت منسوخ شده و نمی‌توانیم خیلی به انتفاع تکیه کنیم. از سوی دیگر، اگر محیطمان به مقدار زیادی ثابت و بدون تغییر است آن گاه دانشمان قابل اتکا بوده و امتحان کردن ایده‌های جدید کار عقلانی نخواهد بود.

طراحی‌های ما به یک تعادل مناسب میان اکتشاف و انتفاع برای موفقیت نیاز دارد. اکتشاف زیاد به منزله‌ی اتفاقی بودن زیاد است و به احتمال زیاد نتایج بهینه‌سازی خوبی را به دنبال نخواهد داشت. از سوی دیگر، انتفاع زیاد به معنای مقدار کم اتفاقی بودن است. تعادل مناسب میان اکتشاف و انتفاع در الگوریتم‌های تکاملی توسط جان هالند[1]، یکی از پیشروان الگوریتم ژنتیک، "اختصاص بهینه‌ی آزمایش‌ها" نامیده شده است.

۲–۸ نتیجه‌گیری

نکته‌ی کلیدی این فصل آن است که بهینه‌سازی جنبه‌ای بنیادین از مهندسی است. وقتی ما سعی بر بهینه کردن تابعی داریم از این تابع با عنوان تابع هدف یاد می‌کنیم. هنگامی‌که می‌خواهیم تابعی را مینیمم کنیم از تابع با عنوان تابع هزینه یاد می‌کنیم. هنگامی هم که می‌خواهیم تابعی را ماکزیمم کنیم از واژه‌ی تابع برازندگی استفاده می‌کنیم. هر مسئله‌ی بهینه‌سازی را می‌توان به راحتی از یک مسئله‌ی ماکزیمم‌سازی به یک مسئله‌ی مینیمم‌سازی و به عکس تبدیل کرد. برخی مسائل خاص که در این فصل معرفی کردیم مسائل مقید، چندهدفه و چندپیمانه‌ای می‌باشند. تقریباً تمام مسائلی که در دنیای واقعی وجود دارند همگی مقید، چندهدفه و چندپیمانه‌ای می‌باشند. یک گونه‌ی خاص دیگر از مسائل، مسائل ترکیبی می‌باشند که در آن‌ها متغیر مستقل به مجموعه‌ای متناهی تعلق دارد.

ما در این فصل تپه‌نوردی را به عنوان الگوریتمی‌ساده اما مؤثر معرفی کردیم. این الگوریتم انواع مختلفی دارد. اگرچه اکثر آن‌ها بسیار ساده هستند اما نسبت به بسیاری از الگوریتم‌های پیچیده‌تر محک‌های بسیار خوبی را فراهم می‌کنند. در نهایت، برخی ویژگی‌های هوش طبیعی را متذکر شدیم و همچنین در مورد چگونگی پیاده‌سازی این ویژگی‌ها در الگوریتم‌های تکاملی‌مان به بحث و گفتگو پرداختیم.

[1] John Holland

مسائل نوشتاری

۲-۱ می‌خواهیم تابع زیر را مینیمم کنیم.

$$f(x) = 40 + \sum_{i=1}^{4} x_i^2 - 10\cos(2\pi x_i)$$

می‌خواهیم این تابع را مینیمم کنیم. توجه داشته باشید که این تابع یک تابع *رستریجین* است – بخش ج.۱۱-۱ را ببینید.

- متغیرهای مستقل $f(x)$ چه هستند؟ متغیرهای تصمیم $f(x)$ کدامند؟ ویژگی‌های راه‌حل $f(x)$ کدامند؟
- بعد این مسئله چند است؟
- راه‌حل این مسئله چیست؟
- این مسئله را به‌عنوان یک مسئله‌ی ماکزیم‌سازی بازنویسی کنید.

۲-۲ تابع $f(x) = sin\, x$ را در نظر بگیرید.

- $f(x)$ چند مینیمم محلی دارد؟ مقادیر تابع در این مینیمم‌ها و مقدار مینیمم‌ساز محلی x چه قدر است؟
- $f(x)$ چند مینیمم کلی دارد؟ مقدار تابع در مینیمم کلی و مقدار مینیمم‌ساز کلی x چه قدر است؟

۲-۳ تابع $f(x) = x^3 + 4x^2 - 4x + 1$ را در نظر بگیرید.

- این تابع دارای چند مینیمم محلی است؟ مقدار تابع در این مینیمم‌ها و مقدار مینیمم‌ساز محلی x چه قدر است؟
- این تابع دارای چند ماکزیمم محلی است؟ مقدار تابع در این ماکزیمم‌ها و مقدار ماکزیمم‌ساز محلی x چه قدر است؟
- این تابع چند مینیمم کلی دارد؟
- این تابع چند ماکزیمم کلی دارد؟

۲-٤ همان تابع مثال ۲-۳ را با محدودیت $x \in [-5,3]$ در نظر بگیرید.

- $f(x)$ چند مینیمم محلی دارد؟ مقادیر تابع در این مینیمم‌ها و مقدار مینیمم‌ساز محلی x چه قدر است؟

- $f(x)$ چند ماکزیمم محلی دارد؟ مقدار تابع در این ماکزیمم‌ها و مقدار ماکزیمم‌ساز محلی x چه قدر است؟
- $f(x)$ چند مینیمم کلی دارد؟ مقدار تابع در مینیمم کلی و مقدار مینیمم‌ساز کلی x چه قدر است؟
- $f(x)$ چند ماکزیمم کلی دارد؟ مقدار تابع در این ماکزیمم‌ها و مقدار ماکزیمم‌ساز کلی x چه قدر است؟

۲-۵ به یاد بیاورید که شکل ۲-٤ ناحیه‌ی پرتو را برای یک مسئله دو هدفه نشان می‌دهد.

- مجموعه‌ای از نقاط محتمل را در صفحه‌ی $f-g$ رسم کنید با این فرض که هدف ماکزیمم کردن f و مینیمم کردن g باشد.
- مجموعه‌ای از نقاط محتمل را در صفحه‌ی $f-g$ رسم کنید با این فرض که هدف مینیمم کردن f و ماکزیمم کردن g باشد.
- مجموعه‌ای از نقاط محتمل را در صفحه‌ی $f-g$ رسم کنید با این فرض که هدف ماکزیمم‌سازی f و g باشد.

۲-٦ چند مسیر بسته‌ی یکتا میان n شهر وجود دارد؟ مقصود از یکتایی آن است که شهر آغازین و مسیر سفر اهمیتی ندارند. برای مثال، در یک مسئله‌ی با چهار شهر A، B، C و D مسیرهای $A \to B \to C \to D \to A$ با مسیرهای $D \to A \to B \to C \to D$ و $B \to C \to D \to A \to B$ یکسان در نظر گرفته می‌شود.

۲-۷ یک مسئله‌ی TSP با شهرهای جدول ۲ در نظر بگیرید.

جدول ۲-۲ مختصات شهرهای مسئله‌ی مثال ۲-۷

شهر	x	y
A	5	9
B	9	8
C	-6	-8
D	9	-2
E	-5	9
F	4	-7
G	-9	1

- چند مسیر بسته بین این هفت شهر وجود دارد؟
- آیا با نگاه کردن به مختصات موجود در جدول ۲-۲ می‌توان به راحتی راه‌حل را پیدا کرد.

- مختصات را رسم کنید. آیا با نگاه کردن به منحنی می‌توان به راحتی راه‌حل را ترسیم نمود؟ بهترین راه‌حل چیست؟ این مسئله به ما نشان می‌دهد که گاه با نگاهی متفاوت می‌توان جواب یک مسئله را پیدا نمود.

۲-۸ با فرض یک مسئله‌ی ماکزیمم‌سازی دلخواه از تابع $f(x)$ و راه‌حل نامزد اتفاقی x_0، با چه احتمالی الگوریتم تپه‌نوردی شدیدترین شیب بعد از نسل اول مقداری مانند x' خواهد یافت به طوری که $f(x') > f(x)$ باشد.

مسائل کامپیوتری

۲-۹ تابع مسئله‌ی ۲-٤ را رسم کرده و مینیمم‌های محلی و کلی را بر روی آن به وضوح مشخص کنید.

۲-۱۰ مسئله‌ی بهینه‌سازی چندهدفه‌ی $\min\{f_1, f_1\}$ را در نظر بگیرید.

$$f_1(x_1, x_2) = x_1^2 + x_2 \quad ; \quad f_2(x_1, x_2) = x_1 + x_2^2$$

x_1 و x_2 هر دو در بازه‌ی [-10,10] قرار دارند.

- مقادیر f_1 و f_2 را برای تمامی‌مقادیر صحیح مجاز x_1 و x_2 محاسبه کرده و نقاط را در فضای $f_1 - f_2$ رسم کنید.
- با توجه به بند قبل، توصیفی ریاضی از مجموعه‌ی پرتو ارائه دهید. مجموعه‌ی پرتو را در فضای $x_1 - x_2$ رسم کنید.

۲-۱۱ تپه‌نوردی تطبیقی.

- ۲۰ شبیه‌سازی مونت کارلو برای الگوریتم تپه‌نوردی تطبیقی اجرا کنید. هر شبیه سازی شامل ۱۰۰۰ نسل باشد و آن را برای تابع دو بعدی اکلی به کار ببرید. مینیمم به دست آمده از هر شبیه‌سازی را ثبت کرده و میانگین آن‌ها را محاسبه کنید. این کار را برای ده نرخ جهش انجام دهید، $p_m = {}^{k}/_{10}$ ، $k \in [1,10]$، و نتایج خود را در جدول ۲-۳ یادداشت کنید. کدام نرخ جهش از بقیه بهتر است؟
- بند قبل را تکرار کنید. آیا نتایج مشابهی می‌گیرید؟ در مورد تعداد شبیه‌سازی‌های مورد نیاز برای دستیابی به نتایج تجدیدپذیر برای این مسئله چه نتیجه‌ای می‌گیرید؟

بند اول را برای تابع ۱۰ بعدی اکلی دوباره اجرا کنید. در مورد ارتباط بین نرخ جهش بهینه و بعد مسئله چه نتیجه‌ای می‌گیرید؟

جدول ۲-۳ جدول را برای مسئله‌ی ۲-۱۱ کامل کنید.

P_m	میانگین نتایج
۰,۱	
۰,۲	
۰۳	
۰,٤	
۰,٥	
۰,٦	
۰,۷	
۰,۸	
۰,۹	
۱	

فصل سوم

الگوریتم‌های ژنتیک

الگوریتم‌های ژنتیک بهینه‌ساز توابع نیستند.

کنث د جونگ[1] (د جونگ، ۱۹۹۲)

(الگوریتم‌های ژنتیک (GAs)، ابتدایی‌ترین، شناخته‌شده‌ترین و پراستفاده‌ترین الگوریم‌های تکاملی هستند. GAها شبیه‌سازی انتخاب طبیعی هستند که می‌توانند مسائل بهینه‌سازی را حل کنند. علیرغم نقل قول بالا توسط کنث د جونگ، اغلب از GAها به‌عنوان ابزاری قدرتمند برای بهینه‌سازی استفاده می‌شود. سخن دِ جونگ بر این نکته تأکید می‌کند که GAها در اصل برای مطالعه‌ی سیستم‌های تطبیقی ایجاد شده‌اند تا بهینه کردن توابع. GAها دسته‌بندی بسیار وسیع‌تری از سیستم‌ها را نسبت به بهینه‌سازهای توابع شامل می‌شوند. ما می‌توانیم از GAها برای مطالعه‌ی پویایی سیستم‌های تطبیقی (میچل، ۱۹۹۸، فصل ٤)، برای فراهم آوردن توصیه‌هایی برای طراحان مد (کیم[2] و چو[3]، ۲۰۰۰)، جهت فراهم نمودن تعادل برای طراحان پل (فروتا[4]و همکاران، ۱۹۹۵) و بسیاری کاربردهای دیگر به غیر از بهینه‌سازی استفاده کنیم. برخی اوقات مرز میان یک الگوریتم بهینه‌سازی و یک الگوریتم غیر بهینه‌سازی چندان مشخص نیست چرا که تمام الگوریتم‌ها برآنند تا بهترین عملکرد خود را ارائه دهند. به هر حال، توجه عمدهی ما در این کتاب به کاربرد بهینه‌سازی GAها معطوف است. برای آغاز مطالعه‌ی GAها ابتدا برخی ویژگی‌های اساسی انتخاب طبیعی را مشاهده می‌کنیم.

۱. یک سیستم زیستی شامل جمعیتی از ذرات است که بیشترشان قابلیت بازتولید دارند.

۲. ذره‌ها دوره‌ی زندگی محدودی دارند.

۳. در جمعیت تنوع وجود دارد.

۴. قابلیت زنده ماندن مطلقا با قابلیت بازتولید همبستگی دارد.

الگوریتم‌های ژنتیک تمامی ویژگی‌های انتخاب طبیعی را شبیه‌سازی می‌کنند. وقتی با یک مسئله‌ی بهینه‌سازی مواجه می‌شویم، جمعیتی از راه‌حل‌های نامزد را، که به آن‌ها ذره می‌گوییم، تولید می‌نماییم. برخی از این راه‌حل‌ها خوب‌اند و برخی چندان خوب نیستند. ذرات خوب شانس زیادی برای بازتولید دارند در حالی که ذرات ضعیف از شانس کمی برای این کار برخوردار هستند. والدین فرزندان را ایجاد نموده و سپس از جمعیت خارج شده تا برای اولاد خود جا باز کنند. با آمد و رفت نسل‌ها، جمعیت برازنده‌تر می‌شود. گاهی یک یا چند "سوپرمن" به وجود می‌آیند که ذره‌هایی بسیار برازنده هستند و می‌توانند راه‌حل‌هایی نزدیک به بهینه برای مسئله‌ی مهندسی ما ارائه نمایند.

---

[1] Kenneth De Jong
[2] Kim
[3] Cho
[4] Furuta

مروری بر فصل

این فصل مروری بر الگوریتم‌های ژنتیک طبیعی و مصنوعی برای مسائل بهینه‌سازی خواهد کرد. از آنجا که در این فصل مبحث الگوریتم‌های تکاملی را تازه شروع می‌کنیم، نسبت به فصل‌های دیگر زمان بیشتری را به زیرساخت‌های تاریخی و بیولوژیکی اختصاص خواهیم داد. خوانندگانی که مایل‌اند مستقیماً به سراغ مطالعه‌ی GA بروند می‌توانند از سه بخش اول صرف نظر کنند، بی‌آنکه درک خود از GA را به طور جدی به خطر بیاندازند. بخش ۳-۱ با تمرکز بر کارهای چارلز داروین[1] و گرگور مندل[2] در قرن ۱۹، مختصراً در مورد تاریخچه‌ی علم ژنتیک بحث خواهد نمود. بخش ۳-۲ علم ژنتیک را مرور کرده و بدین ترتیب اساس و شالوده‌ی GAها شکل خواهد گرفت. بخش ۳-۳ تاریخچه‌ی شبیه‌سازی‌های کامپیوتری از ژنتیک را فراهم می‌کند. این کار در دهه‌ی ۱۹٤۰ توسط زیست‌شناس‌هایی که به مطالعه‌ی انتخاب طبیعی علاقه‌مند بودند آغاز شد و با انفجار تحقیقات در زمینه‌ی GA در دهه‌ی ۱۹۷۰ و ۱۹۸۰ به پایان رسید.

بخش ۳-٤ یک GA ساده‌ی دودویی را به صورتی منظم و قدم به قدم ارائه خواهد نمود. GA بر پایه‌ی ژنتیک طبیعی است و راه‌حل‌هایی برای مسائل بهینه‌سازی ارائه می‌دهد که در آن‌ها کروموزوم‌ها ژن[3]های دودویی دارند. GA دودویی به طور طبیعی برای مسائلی مناسب است که دامنه‌شان شامل فضای جستجوی n بعدی دودویی بوده و یا حداقل دامنه‌شان گسسته است.

ما می‌توانیم برای مسائل بهینه‌سازی با دامنه‌ی پیوسته، از رشته بیت‌ها برای ارائه‌ی راه‌حل‌های نامزد استفاده کنیم. برای این کار باید از تعداد بیت‌های کافی برای رسیدن به کیفیت و وضوح لازم استفاده نمود. اما به‌طور طبیعی راه‌حل‌های نامزد برای مسائل با دامنه‌ی پیوسته به‌صورت برداری از اعداد حقیقی ارائه می‌شوند. به همین علت بخش ۳-۵ GAها را به مسائل با دامنه‌ی پیوسته تعمیم خواهد داد.

۳-۱ تاریخچه‌ی ژنتیک

ژنتیک علم مطالعه‌ی وراثت و تنوع در ارگانیسم‌های زنده است. این بخش با تمرکز بر کارهای چارلز داروین، پدر علم تکامل، و گرگور مندل، پدر علم ژنتیک، تاریخچه‌ای مختصر از ایجاد و گسترش ژنتیک مدرن را ارائه خواهد داد.

---

[1] Charles Darwin
[2] Gregor Mendel
[3] Allele

۳-۱-۱ چارلز داروین

چارلز داروین، پسر پزشک ثروتمند رابرت[1] داروین، در سال ۱۸۰۹ در انگلستان به دنیا آمد. جایگاه ویژه‌ی چارلز در زندگی به وی این اجازه را می‌داد تا به‌عنوان یک مرد جوان از علاقه‌ای به علاقه‌ی دیگر و از شاخه‌ای به شاخه‌ی دیگر بپرد. به گونه‌ای که به نظر می‌رسید برایش مقدر شده تا زندگی‌اش را در این شاخه به شاخه پریدن‌ها، تنبل‌وار به هدر دهد. پدر او مردی کوشا و تلاشگر بود اما همان‌گونه که می‌توان انتظار داشت، پدر کوشا منجر به پسری تنبل شد. روزی رابرت به پسرش چارلز گفت: "تو به هیچ چیز جز شکار، سگ‌ها و گرفتن موش‌ها اهمیت نمی‌دهی و روزی مایه‌ی شرمساری خود و تمام خانواده‌ات خواهی شد." (داروین و همکاران، ۲۰۰۲، صفحه‌ی ۱۵).

رابرت سعی می‌کرد پسرش را در کارهای پزشکی خود درگیر کند اما چارلز علاقه‌ای به این کار نداشت و علاوه بر این از دیدن خون تنفر داشت. به همین دلیل رابرت پسرش را به دانشگاه کمبریج فرستاد تا برای وزارت تحصیل کند.

چارلز علاقه‌ی چندانی به درس‌های دانشگاه‌اش نداشت. چیزی که او واقعاً به آن علاقه‌مند بود، طبیعت بود. او تمام وقتش را به کاوش و مطالعه‌ی طبیعت، خواندن کتاب‌های طبیعت‌شناسان بزرگ و جمع‌آوری سوسک‌ها سپری می‌کرد. زندگی او با ماهرتر شدن وی به‌عنوان یک طبیعت‌شناس تمرکز بیشتری پیدا می‌کرد. چارلز شروع به ملاقات اساتید و دانشجویانی کرد که مانند او به طبیعت علاقه‌مند بودند. وی کم‌کم برنامه‌ای برای ترک تحصیلات وزارت و دنبال نمودن علاقه‌ی اصلی‌اش ریخت. بالاخره جاه‌طلبی به او راه پیدا کرده بود.

در سال ۱۸۳۱ و در سن ۲۲ سالگی، چارلز برای یک موقعیت کاری در بیگل[2] درخواست فرستاد. بیگل یک کشتی بود که از طرف دولت انگلستان مأمور شده بود تا کرانه‌ی جنوبی آمریکای جنوبی را بکاود. چارلز برای این کار، با شرط آنکه تمام هزینه‌های خود را متقبل شود، قبول شد.

اگر شما پدر چارلز داروین بودید چه می‌کردید؟ پسرتان را به دانشگاه فرستاده‌اید و خرج و مخارج تحصیل‌اش را برای سه سال پرداخت کرده‌اید و حالا پسرتان پیش شما می‌آید و از شما درخواست پول برای یک سفر دریایی پنج ساله می‌کند. صد البته که جواب شما نه خواهد بود و این دقیقاً حرفی بود که رابرت در ابتدا به پسر خود زد. خوشبختانه، رابرت متوجه شد که پسرش در حال تبدیل شدن به یک مرد کامل است

[1] Robert
[2] Beagle

و علاقه‌ی اصلی‌اش در زندگی را پیدا کرده است. رابرت در نهایت متقاعد شد و هزینه‌ی این سفر را تقبل نمود.

در طول سفر پنج ساله‌ی بیگل، چارلز همراه با ۷۰ دریانورد دیگر در یک کشتی با طول ۹۰ فوت و عرض ۲۵ فوت زندگی کرد. کشتی دارای وسایل و ابزار پیمایش و آذوقه‌ی کافی بود به‌طوری که می‌توانست تا چند ماه در دریا سفر کند. قطعاً این سفر برای چارلز گذرانی سخت از زندگی راحتش بوده است اما وی از آن سربلند بیرون آمد. او بیشتر زمانش را در دریا به مطالعه گذراند. هنگامی که کشتی در جزایر و یا سرزمین‌های اصلی آمریکای جنوبی توقف می‌کرد، چارلز نمونه‌هایی از حیوانات مختلف را جمع‌آوری می‌کرد و با اولین کشتی در دسترس به انگلستان می‌فرستاد. او انواع زیادی از گونه‌ها را در طول سفرش جمع‌آوری نمود. گونه‌های مشابه از جزایر همسایه به قدری متفاوت از هم بودند که گویی هر یک از آن‌ها به محیط مخصوص خود انطباق پیدا کرده‌اند.

چارلز در سال ۱۸۳٦ و در سن ۲۷ سالگی به خانه‌ی خود در انگلستان بازگشت. تقریباً بلافاصله پس از بازگشتش کار بر روی کتابش، اصالت گونه‌ها، را آغاز نمود (داروین، ۱۸۵۹)، کتابی که نگارش آن دهه‌ها به طول انجامید. وی همچنین در نوشتن مقاله برای ژورنال‌ها و صحبت در کنفرانس‌ها بسیار فعال بود. وی در طول زمانی که در حال پرداختن نظریه‌ی منسجم انتخاب طبیعی بود به مطالعه و یادگیری خود ادامه داد.

انتخاب طبیعی بیان می‌دارد تنها ذره‌ها (یا افراد)ی که برازنده‌تر از سایرین هستند می‌توانند ویژگی‌های خود را به فرزندانشان انتقال دهند. بدین طریق است که انطباق صورت می‌پذیرد، از طریق "بقای برازنده‌ترین".

هنگامی که چارلز در حال کار بر روی کتابش بود، در مورد عمومی کردن نظریه‌ی انتخاب طبیعی مردد بود. از آنجا که وی سه سال برای وزارت تحصیل کرده بود می‌دانست که این نظریه به دلیل تقابل احتمالی‌اش با انجیل می‌تواند طوفانی از بحث و مجادله به پا کند. او می‌خواست قبل از آنکه نتایجش را منتشر کند یک اثر بزرگ و بی‌نقص ایجاد کرده باشد. به هر حال، وی در سال ۱۸۵۸ مقاله‌ای از جانب آلفرد والاس[1]، طبیعت‌شناسی که در حال سفر در اقیانوس آرام جنوبی بود، دریافت کرد. والاس مستقل از داروین به بسیاری از ایده‌های داروین رسیده بود و مقاله‌اش را برای داروین فرستاده بود و از او خواسته بود تا در انتشار آن به وی کمک کند.

[1] Alfred Wallac

داروین در یک مخمصه افتاده بود. او باید تصمیمش را می‌گرفت. وی می‌توانست نامه‌ی والاس را از بین ببرد[۱]، نتایج خود را منتشر کند و امتیاز نظریه‌ی تکاملش را مطالبه کند و یا می‌توانست مقاله‌ی والاس را منتشر کرده و به وی اجازه دهد این امتیاز را از آن خود نماید. با تکیه بر اعتبارش، داروین تصمیم به تلاش برای رسیدن به یک تعادل گرفت. داروین به سرعت مقاله‌ی خود را نوشت و در کنفرانس بعدی هم مقاله‌ی خود و هم مقاله‌ی والاس را ارائه کرد. او سپس ملاحظات نهایی را به کتابش، که به دلیل عجله‌ی داروین برای محکم کردن ادعایش در مورد امتیاز و اعتباری که حقش بود بسیار کوتاه‌تر از نسخه‌ی اصلی‌اش شده بود[۲]، اضافه کرد. "اصالت گونه‌ها" در سال ۱۸۵۹ منتشر شد و تمام ۱۲۵۰ نسخه‌ی چاپ اول آن در روز اول به فروش رسید. داروین به سرعت در حال تبدیل شدن به مشهورترین و پرمجادله‌ترین دانشمند نسل خود بود.

اگرچه نظریه‌ی تکامل داروین به سرعت اعتبار علمی کسب کرد اما مانند سایر نظریه‌های جدید بدون معارض هم نبود. اولاً، به نظر می‌رسید که این نظریه در تعارض با آموزش‌های انجیل در زمینه‌ی آفرینش ویژه‌ی تمام گونه‌ها قرار دارد و به همین علت مستعد حملات رهبران مذهبی بود. ثانیاً داروین در مورد این که خصایص چگونه از والدین به فرزندان انتقال پیدا می‌کند توضیحی نداده بود. به نوعی، این که نظریه‌ی داروین با توجه به عدم توضیح در مورد موضوع وراثت چنین سریع مورد قبول واقع شد، جای شگفتی دارد. او شاهد وقوع این پدیده بود و به همین علت انتخاب طبیعی را بدیهی شمرد اما نمی‌دانست که این پدیده چگونه اتفاق می‌افتد.

داروین همراه با سایر دانشمندان عصر خود تصوری غلط از وراثت داشت. اولاً آنکه، وی معتقد بود خصایص والدین می‌توانند در فرزندان مخلوط شوند. برای مثال، فرزند یک موش سیاه و یک موش سفید می‌توانست خاکستری رنگ باشد. ثانیاً، وی باور داشت که ویژگی‌های اکتسابی می‌توانند به فرزندان انتقال داده شوند. برای مثال، مردی که با وزنه زدن قوی می‌شود، فرزندانی قوی خواهد داشت.

داروین، فرزندی از یک خانواده‌ی توانگر، نظریه‌ی انتخاب طبیعی را ایجاد کرد اما اثبات آن به گرگور مندل، فرزندی از یک خانواده‌ی فقیر، محول شد.

---

[۱] چه کسی به ادعای داروین مبنی بر نرسیدن نامه ای از آرام جنوب به انگلستان شک می‌کرد؟

[۲] این موضوع در طولانی مدت نتایج خوبی به دنبال داشت کتاب داروین حدود ۵۰۰ صفحه بود و اگر آن‌طور که داروین قصد داشت به پایان می‌رسید چند صد صفحه‌ی دیگر به آن اضافه می‌شد و این امر ممکن بود خوانندگان بالقوه را بترساند. با این خلاصه‌سازی داروین به تعداد خوانندگان کتابش افزود و موفقیت بیشتری کسب کرد.

۳-۱-۲ گرگور مندل

گرگور مندل اولین کسی بود که توانست چگونگی وقوع وراثت را بفهمد و توضیح دهد. وی در سال ۱۸۲۲ در یک خانواده‌ی فقیر در چکوسلوواکی زاده شد. خانواده‌اش نام جوهان[1] بر وی نهادند (بانکستون[2]، ۲۰۰۵). پدرش به کمک او برای کار در مزرعه نیاز داشت اما جوهان جوان خود را بیشتر مناسب کار آکادمیک می‌دید تا کار فیزیکی و عملی. خانواده‌اش به سختی می‌توانستند از عهده‌ی هزینه‌های تحصیل وی برآیند. با این حال، آن‌ها وی را به مدرسه فرستادند تا بتواند فرصت‌هایی که خودشان از آن‌ها محروم بودند را دنبال کند و به دست آورد. با وجود حمایت خانواده و کار پاره وقت، وی به سختی می‌توانست به‌عنوان یک دانش‌آموز از لحاظ اقتصادی دوام بیاورد. موقعیت اقتصادی وی در آن زمان درست مانند وضعیت اقتصادی بسیاری از دانشجویان فوق لیسانس امروزی بود تنها با این تفاوت که در آن زمان فرصت‌های کمتری برای دریافت کمک‌های مالی وجود داشت.

در سن ۲۱ سالگی مندل از وجود یک صومعه با خبر شد، جایی که می‌توانست تحصیلاتش را بدون نگرانی مالی ادامه دهد. وی باید سوگند تجرد و فقر یاد می‌کرد؛ اما منافع مالی آن بهتر از آن بود که بتواند دست رد بر سینه‌ی آن بزند. مندل فردی مذهبی نبود اما این فرصت را مغتنم شمرد و به عضویت فرقه‌ی آگوستین[3] در صومعه‌ی توماس مقدس[4] در آمد.

آگوستین یکی از بزرگترین رهبران روشنفکر تاریخ مسیحیت است که حدود ٤٠٠ سال بعد از میلاد مسیح در امپراطوری روم می‌زیسته است. الهیات وی مؤکد توانایی خداوند در استفاده از دانش دنیوی برای ایجاد ارتباط با انسان‌ها بود. صومعه‌ای که مندل به آن پیوسته بود، همان‌طور که از اسم آن نیز پیداست، به یادگیری در مورد همه‌ی جنبه‌های زندگی تشویق می‌کرد و به همین علت مکانی مناسب برای مندل بود. این صومعه از دید دنیوی با بسیاری از صومعه‌های دیگر متفاوت بود. در این صومعه نیازی نبود راهبان خود را تنبیه کنند و یا تمام روز خود را صرف دعا کردن و نماز خواندن کنند و یا روزه‌ی سکوت بگیرند. آن‌ها فقط باید به مطالعه می‌پرداختند و باور می‌داشتند خداوند از طریق آموزه‌هایشان با آن‌ها صحبت می‌کند. بر طبق سنت، جوهان مندل پس از آنکه به صومعه‌ی توماس مقدس پیوست، نامی جدید برای خود برگزید. از این پس نام او گرگور مندل بود.

[1] Johan
[2] Bankston
[3] Augustin
[4] St. Thomas

مندل به‌عنوان یک راهب به شرکت در کلاس‌های دانشگاه‌ها ادامه داد، در مدارس به تدریس علوم پرداخت و تحقیقات خود را در صومعه آغاز کرد. تحقیقاتی که وی دنبال می‌کرد شامل تولید مثل گیاهان، بالاخص نخود فرنگی می‌شد. به دلیل پس زمینه‌ای که وی از مزرعه‌ی پدرش داشت، این زمینه زمینه‌ای بسیار مناسب بود.

در حین آزمایشاتش بر روی نخود فرنگی‌ها، مندل دریافت که نخود فرنگی گونه‌های مختلفی دارد. برخی از آن‌ها نرم بودند در حالی که برخی دیگر سفت بودند. برخی از آن‌ها رنگ سبز داشتند در حالی که برخی دیگر رنگشان به زردی می‌زد. تعدادی از آن‌ها در یک نقطه جوانه می‌زدند و تعدادی دیگر جوانه‌شان در نقطه‌ای متفاوت می‌رویید. با انجام آزمایشات بیشتر مندل دریافت که گونه‌ها توسط تعدادی واحدهای وراثت، که غیر قابل رؤیت‌اند، کنترل می‌شوند. وی نام آن‌ها را عناصر گذاشت. برخی از این عناصر قوی بوده و کنترل بیشتری بر روی گونه‌ی نخود فرنگی دارند. سایر عناصر ضعیف بوده و کنترل کمی بر گونه‌ی نخود فرنگی دارند. امروزه ما از کلمه‌ی ژن به جای کلمه‌ی عنصر استفاده می‌کنیم و به جای آنکه از کلمات قوی و ضعیف استفاده کنیم می‌گوییم ژن‌ها یا غالب‌اند یا مغلوب. اما این مندل بود که برای اولین بار مفاهیم ژنتیک، وراثت و غلبه را درک کرد. اثر مندل حلقه‌ی گمشده‌ی نظریه‌ی داروین بود و چگونگی انتخاب طبیعی را توضیح می‌داد.

مندل یافته‌های خود را در سال ۱۸٦۵ در کنفرانسی ارائه داد. این اتفاق تنها شش سال پس از انتشار کتاب "اصالت گونه‌ها"ی داروین رخ داد اما بنا به دلایلی بزرگی کار مندل در آن زمان درک نشد. پذیرش کار بزرگ علمی مندل نمی‌توانست بیش از این نسبت به پذیرش کار داروین متفاوت باشد. داروین به سرعت مشهور شد در حالی که کار مندل نادیده گرفته شد. او کار خود را در گمنامی ادامه داد و تعدادی مقاله نیز این جا و آن جا منتشر کرد که همه‌ی آن‌ها توسط جامعه‌ی علمی نادیده گرفته شدند.

مندل در سال ۱۸٦۸ در صومعه‌ی توماس مقدس رهبر اجرایی شد و پس از آن دیگر وقت چندانی برای پرداختن به علم نداشت[1]. مندل در سال ۱۸۸٤ دار فانی را وداع گفت. کار وی در زمینه‌ی ژنتیک حدود سال ۱۹۰۰ توسط زیست‌شناس هلندی هوگو د وریس[2]، گیاه‌شناس آلمانی کارل کورنز[3] و بذرشناس اتریشی اریک ون شرماک[4] دوباره احیا شد.

---

[1] همان‌طور که امروزه هم متداول است، یک شغل رو به پیشرفت در تحقیقات علمی به طرز حزن انگیزی توسط یک ترفیع به کارهای اجرایی از بین رفت.

[2] Hugo De Vris
[3] Carl Correns
[4] Erich Von Tschemark

۳-۲ علم ژنتیک

هر یک از گونه‌ها توسط یک جفت ژن کنترل می‌شوند. ژنتیک انسان‌ها دیپلوید[1] خوانده می‌شود تا نشان دهد که ژن‌های هرگونه در دسته‌های دوتایی قرار دارند. برخی حیوانات و گیاهان هاپلوید[2] هستند، بدین معنی که هرگونه توسط مجموعه‌ای از ژن‌های منفرد تعیین می‌شوند. سایر ارگانیسم‌ها پولیپلوید[3] هستند بدین معنا که هرگونه توسط تعدادی بیشتر از دو مجموعه از ژن‌ها تعیین می‌شود.

در ژن‌های دیپلوید برخی صفت‌های ژنتیکی غالب‌اند در حالی که برخی دیگر مغلوب‌اند. اگر ژن غالب و مغلوب هر دو در یک فرد ظاهر شوند آن‌گاه ژن غالب تعیین‌کننده‌ی گونه‌ی آن فرد خواهد بود. ژن مغلوب تنها زمانی تعیین‌کننده‌ی گونه خواهد بود که هر دو ژن یکی باشند.

مثال ۳-۱

سه فرد را در نظر بگیرید. کریس[4] دو ژن چشم-قهوه‌ای دارد، کیم دو ژن چشم-سبز دارد و تری[5] یک ژن چشم-قهوه‌ای و یک ژن چشم-سبز دارد. از آنجا که کریس دو ژن چشم-قهوه‌ای دارد بنابراین چشم‌هایش قهوه‌ای خواهد بود. کیم نیز به علتی مشابه چشمانی سبز خواهد داشت حال آنکه تری یک ژن چشم-قهوه‌ای و یک ژن چشم-سبز دارد و ژن چشم-قهوه‌ای غالب است بنابراین چشمان تری قهوه‌ای خواهد بود.

- کریس: قهوه‌ای/قهوه‌ای ← چشمان قهوه‌ای
- کیم: سبز/سبز ← چشمان سبز
- تری: قهوه‌ای/سبز ← چشمان قهوه‌ای

اگر کریس و تری ازدواج کنند، هر یک از آن‌ها یک ژن رنگ چشم به فرزندشان انتقال می‌دهند. بنابراین فرزند آن‌ها می‌تواند یا دو ژن چشم-قهوه‌ای داشته و یا یک ژن چشم-قهوه‌ای و یک ژن چشم-سبز داشته باشد. با این حال همه‌ی فرزندان آن‌ها چشمانی قهوه‌ای خواهند داشت چراکه ژن چشم-قهوه‌ای غالب است.

اگر کریس و کیم ازدواج کنند، فرزند آن‌ها یک ژن چشم-سبز از کیم و یک ژن چشم-قهوه‌ای از کریس به ارث خواهد برد. از آن جا که قهوه‌ای غالب است پس تمام فرزندانشان چشمانی قهوه‌ای خواهند داشت.

اگر تری و کیم ازدواج کنند، فرزند آن‌ها یا دو ژن چشم-سبز خواهد داشت و یا یک ژن چشم-سبز و یک ژن چشم-قهوه‌ای به ارث خواهد برد. فرزند آن‌ها می‌تواند چشمانی سبز یا قهوه‌ای داشته باشد.

---

[1] Diploid
[2] Haploid
[3] Polyploid
[4] Chris
[5] Terry

حال فرض کنید یک منفعت تکاملی در داشتن چشمان سبز وجود داشته باشد. برای مثال، مردانی که چشم سبز دارند برای زنان بسیار جذاب‌اند. یا شاید رنگ سبز چشم باعث شود چشم نسبت به برخی فرکانس‌های نور مرئی حساس‌تر بوده و به همین دلیل افرادی که چشمان سبز دارند افراد موفق‌تری در شکار باشند. در چنین شرایطی، شانس زنده ماندن افرادی که چشمان سبز دارند بیشتر از کسانی است که چشم قهوه‌ای دارند. به علاوه آنکه چشم سبزها از چشم قهوه‌ای‌ها قوی‌تر بوده و این موضوع فرصت زاد و ولد بیشتری را در اختیار آن‌ها قرار می‌دهد. این موضوع با افزایش ژن‌های چشم–سبز و کاهش ژن‌های چشم–قهوه‌ای، ژن‌های انسان‌ها را تحت تأثیر قرار می‌دهد. این پدیده انتخاب طبیعی یا بقای برازنده‌ترین نام دارد و در واقع همان چیزی است که داروین در حین سفرش استنباط کرده بود.

گاهاً یک جهش بر فرزندان اثر می‌گذارد. در چنین مواردی ژن‌ها دست نخورده از والدین به فرزندان انتقال پیدا نمی‌کنند بلکه دستخوش تغییر می‌شوند. جهش‌ها به دلیل نقص بنیادین زندگی و فرایندهای زیستی از جمله تشعشع یا بیماری اتفاق می‌افتند.

اکثریت گسترده‌ای از جهش‌ها خنثی هستند بدین معنی که به دلیل قابلیت فزونی و ارتجاعی زیست، این جهش‌ها یا تأثیری بر فرزندان نداشته و یا این تأثیرات بسیار ناچیزاند. این جهش‌های خنثی در جستجوی زیست‌شناسی برای یافتن برازندگی بهبود یافته مهم هستند و ما نیز جلوتر در این کتاب مشاهده خواهیم کرد که در یک الگوریتم تکاملی نیز این جهش‌های خنثی بسیار حایز اهمیت هستند. با این حال بیشتر جهش‌هایی که فرزندان را تحت تأثیر قرار می‌دهند مضر هستند. بیش از ٦٠٠٠ جهش تک ژنی معمول وجود دارد که در هر ۲۰۰ بار زایش یک بار اتفاق می‌افتند. برخی ناهنجاری‌های ژنتیکی در هنگام تولد بروز می‌کنند و برخی نیز چندین سال بعد از تولد علائم خود را نشان می‌دهند. برای مثال، بسیاری از انواع سرطان‌ها ریشه‌ای ژنتیکی دارند.

با وجود این، هر از چند گاهی جهشی اتفاق می‌افتد که مفید است. برای مثال فکر کنید در مثال ۳–۱ یکی از فرزندان تری و کیم دچار جهش ژنتیکی شده و رنگ چشمانش بنفش شود. همچنین فرض کنید که این فرزند به دلیل وجود همبستگی میان قابلیت تطبیق قرنیه و عنبیه‌ی بنفش، دید تیزتری پیدا کند. این موضوع باعث خواهد شد جهش یافته‌هایی که چشمان بنفش دارند در شکار موفق‌تر باشند و فرصت‌های بیشتری برای جفت‌گیری داشته باشند. اگر ژن چشم–بنفش وی غالب باشد آن گاه تمام فرزندان وی چشمانی بنفش داشته و این جهش در سراسر گونه پخش خواهد شد. اما اگر ژنش مغلوب باشد آن گاه در صورتی فرزندانی با چشم بنفش خواهد داشت که با کسی مانند خودش جفت‌گیری کند. جهش به همراه انتخاب طبیعی قابلیت

بقای گونه‌ها را بهبود می‌بخشد. بدون جهش، گونه‌ها ایستا و راکد خواهند بود. جهش به‌طور کلی برای افراد زیان‌بار است اما می‌تواند به طرز قابل ملاحظه‌ای برای گونه‌ها مفید واقع شود.

۳-۳ تاریخچه‌ی الگوریتم‌های ژنتیک

سال ۱۹۰۳ سال خوبی برای تکنولوژی بود. کمپانی مارکوفنی[1] اولین پخش رادیویی بین قاره‌ای را آغاز کرد، برادران رایت[2] اولین پرواز هواپیمایشان را با موفقیت به انجام رساندند و نیومان جانوس[3] در بوداپست مجارستان به دنیا آمد. نبوغ نیومان خود را در مطالعات حریصانه و استعداد ریاضی وی نشان داد. پدر و مادر وی، که هر دو از طبقه‌ی تحصیل کرده‌ی جامعه بودند، به زودی دریافتند که وی یک اعجوبه است اما مراقب بودند تا خیلی به وی سخت نگیرند. هنگامی که وی به سن ۲۳ سالگی رسید، یک مدرک لیسانس در رشته‌ی مهندسی شیمی و یک مدرک دکترا در رشته‌ی ریاضی داشت. او به‌عنوان یک فرد آکادمیک به فعالیت‌های خود ادامه داد و در سال ۱۹۲۹ به‌عنوان عضوی از هیات علمی در دانشگاه پرینستون[4] پذیرفته شد. از آن زمان نام او به جان ون نیومان تغییر یافت و در سال ۱۹۳۳ یکی از اعضای اصلی مؤسسه‌ی مطالعات پیشرفته‌ی پرینستون شد.

ون نیومان در طول حضور گسترده‌ی حرفه‌ای خود در دانشگاه پرینستون سهمی بنیادین در پیشرفت‌های ریاضی، فیزیک و اقتصاد داشت. وی یکی از رهبران تلاش برای ساخت بمب اتمی در جنگ جهانی دوم بود و همچنین یکی از پیشروان اختراع کامپیوتر دیجیتالی به شمار می‌رفت. البته افراد دیگری هم بودند که به اندازه‌ی وی (چه بسا بیشتر) در توسعه‌ی کامپیوترهای دیجیتالی سهم داشتند. برای مثال، آلن تورینگ[5] (همکار ون نیومان در پرینستون) و جان ماوچلی[6] و جان اکرت[7] (رهبران تولید انیاک[8]، اولین کامپیوتر جهان در ۱۹٤۰). اما این ون نیومان بود که برای اولین بار فهمید دستورات برنامه باید به همان طریقی که داده‌ها ذخیره می‌شوند، در کامپیوتر ذخیره شوند. تا به امروز چنین ماشین‌هایی را "ماشین‌های ون نیومان" می‌خوانند.

بعد از جنگ، توجه ون نیومان به هوش مصنوعی جلب شد. در سال ۱۹۵۳ وی نیلس باریسلی[9]، ریاضیدان ایتالیایی، را برای مطالعه در مورد زندگی مصنوعی به پرینستون دعوت کرد. باریسلی از کامپیوترهای دیجیتالی

---

[1] Marconi
[2] Wright
[3] Neumann Janos
[4] Princeton
[5] Alan Turing
[6] John Mauchly
[7] John Eckert
[8] ENIAC
[9] Nils Barricelli

جدید برای شبیه‌سازی فرایندهای تکاملی استفاده کرد. او به تکامل زیستی و مسائل بهینه‌سازی علاقه‌ای نداشت. وی می‌خواست با استفاده از فرایندهایی که در طبیعت وجود دارند (بازتولید و جهش)، یک زندگی مصنوعی داخل یک کامپیوتر ایجاد کند. یکی از نوشته‌های وی در سال ۱۹۵۳ به این قرار است: "سلسله‌ای از آزمایش‌های عددی با هدف بررسی احتمال تکامل در یک دنیای مصنوعی، مانند آنچه در موجودات زنده رخ می‌دهد، در حال انجام است." (دایسون[1]، ۱۹۹۸، صفحه‌ی ۱۱۱). باریسلی اولین فردی بود که نرم‌افزار الگوریتم ژنتیک را نوشت. اولین کار وی بر روی این موضوع در سال ۱۹۵٤ و به زبان ایتالیایی و با عنوان "مدل‌های عددی فرایندهای تکاملی" منتشر شد (باریسلی، ۱۹۵٤).

الکساندر فراسر[2] متولد به سال ۱۹۲۳ در لندن، کمی بعد کار باریسلی را دنبال کرد و از برنامه‌های کامپیوتری برای شبیه‌سازی تکامل استفاده نمود. کار و تحصیلات، وی را به نقاط مختلف دنیا از جمله هونگ کونگ، نیوزلند، اسکاتلند و در نهایت در دهه‌ی ۱۹۵۰ به استرالیا، سازمان تحقیقات صنعتی و مشترک‌المنافع علمی در سیدنی کشاند. فراسر یک مهندس نبود، وی زیست‌شناسی بود که به مبحث تکامل علاقه داشت. وی نمی‌توانست تکامل را در اطرافش مشاهده کند چرا که تکامل فرایندی بسیار کند است و به دوره‌های زمانی از رده‌ی میلیون سال نیاز دارد. بنابراین فراسر تصمیم گرفت با ایجاد دنیای خودش درون یک کامپیوتر دیجیتال به مطالعه‌ی تکامل بپردازد. بدین ترتیب وی می‌توانست فرایندها را تسریع کرده و چگونگی وقوع تکامل را مشاهده کند. در سال ۱۹۵۷ فراسر مقاله‌ای با عنوان "شبیه‌سازی سیستم‌های ژنتیکی با استفاده از کامپیوترهای دیجیتال خودکار" نوشت و با این کار اولین کسی شد که صراحتاً از شبیه‌سازی کامپیوتری برای مطالعه‌ی تکامل زیستی استفاده نمود. وی مقالات زیادی در مورد کارهای خود، به خصوص در ژورنال‌های زیست‌شناسی منتشر کرد. در اواخر دهه‌ی ۱۹۵۰ و دهه‌ی ۱۹٦۰ بسیاری از زیست‌شناسان پا در جا پای وی نهادند و از کامپیوترها برای شبیه‌سازی تکامل زیستی استفاده نمودند.

هانس یوآخیم برمرمن[3]، ریاضیدان و فیزیکدان، نیز شبیه‌سازی‌های کامپیوتری اولیه‌ای از تکامل زیستی اجرا کرد. اولین کار وی بر روی این موضوع در سال ۱۹۵۸ با عنوان "تکامل هوشمند"، هنگامی که استاد دانشگاه واشنگتون[4] بود، منتشر شد (فوگل[5] و اندرسون[6]، ۲۰۰۰). برمرمن بیشتر زندگی حرفه‌ای خود را در برکلی[7] و دانشگاه کالیفرنیا[8] گذراند، جایی که وی در دهه‌ی ۱۹٦۰ از شبیه‌سازی‌های کامپیوتری برای مطالعه‌ی

[1] Dyson
[2] Alexander Fraser
[3] Hans-Joachim Bermermann
[4] Washington
[5] Fogel
[6] Anderson
[7] Berkeley
[8] California

عملکرد سیستم‌های پیچیده[1]، به خصوص تکامل استفاده نمود. اما برنامه‌های کامپیوتری وی تنها تکامل را مدل نمی‌کردند بلکه تراکنش میزبان/انگل، الگوشناسی مغز انسان و واکنش سیستم ایمنی بدن را نیز شبیه‌سازی می‌نمودند.

جورج باکس[2] به سال ۱۹۱۹ در انگلستان متولد شد. وی به تکامل مصنوعی علاقه‌مند بود اما بر خلاف پیشینیان این رشته، وی به زندگی و تکامل مصنوعی محض علاقه‌مند نبود. او می‌خواست مسائل دنیای واقعی را حل کند. باکس از علم آمار برای تحلیل و طراحی نتایج آزمایش‌هایش استفاده کرد و تبدیل به یک مهندس صنایع شد و از آمار برای بهینه ساختن فرایندهای تولیدی استفاده نمود. بهترین چیدمان ماشین‌ها در یک کارخانه چیست اگر بخواهیم میزان تولید را ماکزیمم کنیم؟ بهترین روش برای زمانبندی جریان مواد اولیه در کارخانه چیست؟ در طی دهه‌ی ۱۹۵۰، باکس تکنیکی با نام "عملیات تکاملی"برای بهینه ساختن یک فرایند صنعتی در حین عملیات، به وجود آورد. کار وی به خودی خود یک GA نبود، اما از ایده‌ی "تکامل از طریق انباشتگی تعداد زیادی تغییرات افزایشی" برای بهینه‌سازی یک طراحی مهندسی استفاده نمود. اولین مقاله‌ی وی در سال ۱۹۵۷ با عنوان "عملیات تکاملی: روشی برای افزایش سودمندی صنعتی" به چاپ رسید (باکس، ۱۹۵۷).

جورج فرایدمن[3] نیز مانند جورج باکس یک فرد عملگرا بود. وی در سال ۱۹۵٦، برای پایان‌نامه فوق لیسانس خود در دانشگاه UCLA، روباتی طراحی کرد که می‌توانست بیاموزد چگونه برای کنترل رفتار خود مدار الکتریکی بسازد. عنوان پایان‌نامه وی "کامپیوترهای فیدبکی گزینشی برای ترکیب مهندسی و قیاس سیستم‌های عصبی" بود (فرایدمن، ۱۹۹۸)، (فوگل، ۲۰۰٦). کار وی شبیه GAهای امروزی بود با این تفاوت که وی از عبارت "کامپیوترهای فیدبکی گزینشی" برای توصیف روشش استفاده کرده بود. آخرین پاراگراف از نتیجه‌گیری پایان‌نامه وی اینگونه است: "مفاهیم و تصاویر نموداری در این مقاله اگرچه به‌طور قطعی سودمندی GA را به ما نشان نمی‌دهد اما حداقل به زمینه‌ای احتمالی برای تحقیقات آتی اشاره دارد". به‌طور قطع همینگونه بوده است! امروزه، با گذشت بیش از نیم قرن از پایان‌نامه فرایدمن، هر ساله هزاران مقاله‌ی فنی با موضوع الگوریتم‌های ژنتیک به چاپ می‌رسند.

[1] Complex Systems
[2] Goerge Box
[3] Goerge Friedman

یکی دیگر از پیشروان زمینه‌ی الگوریتم‌های ژنتیک، لورانس[1] فوگل است که در سال ۱۹۶۲ کار بر روی Aها را آغاز کرد. وی در سال ۱۹٦٦ همراه با آلوین اونز[2] و میکائیل (جک) والش[3] اولین کتاب در مورد GAها را با عنوان "هوش مصنوعی به وسیله‌ی تکامل شبیه‌سازی شده" نوشت (فوگل و همکاران، ۱۹۹٦). کارهای اولیه‌ی فوگل در الگوریتم‌های ژنتیک از مسائل مهندسی مانند پیش‌بینی سیگنال‌ها، مدل‌سازی مبارزه و کنترل سیستم‌های مهندسی نشات می‌گرفت. پسر لارنس فوگل، دیوید[4] فوگل، کتابی مهم را تدوین کرد که شامل ۳۱ مقاله‌ی بنیادی در مورد GAها و موضوعات مربوطه بود (فوگل، ۱۹۹۸).

پس از کارهای اصلی و بدوی باریسلی، فراسر، برمرمن، باکس و فرایدمن ر دهه‌ی ۱۹۵۰، سایرین از الگوریتم‌های ژنتیک برای مطالعه‌ی تکامل زیستی و حل مسائل بهنه‌سازی استفاده نمودند. برخی پیشرفت‌های مهم در زمینه‌ی الگوریتم‌های ژنتیک توسط جان هالند، استاد روانشناسی، مهندسی برق و علوم کامپیوتر در دانشگاه میشیگان[5] صورت پذیرفت. در دهه‌ی ۱۹٦۰، هالند به سیستم‌های تطبیقی علاقه‌مند شد. وی به تکامل یا بهینه‌سازی علاقه مند نبود بلکه علاقه‌ی وی به چگونگی تطبیق سیستم‌ها با محیط اطرافشان معطوف بود و در سال ۱۹۷۵ کتاب مشهورش با عنوان "تطبیق در سیستم‌های طبیعی و مصنوعی" را به رشته‌ی تحریر درآورد (هالند، ۱۹۷۵).

کتاب وی به دلیل ارائه‌ی ریاضیات مربوط به بحث تکامل، تبدیل به یک کتاب کلاسیک در این زمینه شد. همچنین در سال ۱۹۷۵ شاگرد هالند، کنث د جونگ، پایان‌نامه‌ی دکترای خود را با عنوان "تحلیل رفتار کلاسی از سیستم‌های تطبیقی ژنتیکی" را به پایان رسانید. پایان‌نامه‌ی د جونگ اولین تحقیق کامل و سیستماتیک از کاربرد GAها برای بهینه‌سازی بود. د جونگ از مجموعه‌ای از مسائل نمونه برای کاوش تأثیر پارامترهای مختلف GAها در کاربرد بهینه‌سازی استفاده نمود. کار وی چنان کامل بود که تا مدت‌ها هر مقاله‌ی بهینه‌سازی که شامل مسائل محک د جونگ نبود نامناسب تلقی می‌شد. در دهه‌های ۱۹۷۰ و ۱۹۸۰ بود که تحقیقات در زمینه‌ی GA با سرعتی نمایی افزایش پیدا کرد. این اتفاق به چند دلیل رخ داد. اول آنکه با تجاری و محبوب شدن ترانزیستورها در دهه‌ی ۱۹۵۰، قدرت محاسباتی کامپیوترها به طرز چشمگیری افزایش یافت. به علاوه، با روشن شدن محدودیت محاسباتی روش‌های سنتی در مقایسه با الگوریتم‌های الهام گرفته از زیست‌شناسی، علاقه‌ی محققان به این دسته از الگوریتم‌ها تفزایش یافت. تحقیقات در زمینه‌ی منطق

[1] Lawrance
[2] Alvin Owens
[3] Michael (Jack) Walsh
[4] David
[5] Michigan

فازی و شبکه‌های عصبی نیز به‌صورت نمایی در این دهه‌ها افزایش پیدا کرد، اگرچه که این دو زمینه به قدرت محاسباتی زیادی نیاز نداشتند.

۳-۴ یک الگوریتم ژنتیک دودویی ساده

فرض کنید مسئله‌ای دارید که می‌خواهید حل کنید. اگر شما بتوانید هر یک از راه‌حل‌های ممکن را به‌صورت یک رشته بیت نمایش دهید آنگاه ممکن است GA بتواند این مسئله را حل کند. هر راه بالقوه یک "راه‌حل نامزد" یا یک "ذره" نامیده می‌شود. یک گروه از ذرات "جمعیت" خوانده می‌شود. این بدان معناست که ما باید هر یک از پارامترهای مسئله را به‌صورت یک رشته بیت کدگذاری کنیم. این بخش GAها را با چند مثال ساده معرفی می‌کند. این مثال‌ها به‌عنوان مثال‌های واقعی ارائه نمی‌شوند بلکه به‌عنوان مثال‌هایی سرراست جهت نشان دادن ویژگی‌های اساسی GAها ارائه می‌شوند.

۳-۴-۱ یک الگوریتم ژنتیک برای طراحی روبات

فرض کنید مسئله‌ی ما شامل طراحی روباتی متحرک و سبک وزن می‌باشد که آنقدر توان در اختیار دارد که بر روی یک زمین ناهموار مسیریابی کرده، بی‌آنکه نیاز باشد به محل آغازین حرکت خود بازگردد. پارامترهایی که در طراحی این روبات باید مشخص کنیم شامل نوع و اندازه‌ی موتور و همچنین نوع و اندازه‌ی منبع توان می‌باشد. نوع و اندازه‌ی موتور را می‌توان به‌صورت زیر کدگذاری نمود:

موتور step ۵-ولتی = ۰۰۰
موتور step ۹-ولتی = ۰۰۱
موتور step ۱۲-ولتی = ۰۱۰
موتور step ۲۴-ولتی = ۰۱۱
موتور servo ۵-ولتی = ۱۰۰
موتور servo ۹-ولتی = ۱۰۱
موتور servo ۱۲-ولتی = ۱۱۰
موتور servo ۲۴-ولتی = ۱۱۱

(۳-۱)

نوع و اندازه‌ی منبع توان را نیز می‌وان به‌صورت زیر کدگذاری نمود:

باتری نیکل-کادمیم ۱۲-ولتی = ۰۰۰
باتری نیکل-کادمیم ۲٤-ولتی = ۰۰۱
باتری لیتیوم-یون ۱۲-ولتی = ۰۱۰
باتری لیتیوم-یون ۲٤-ولتی = ۰۱۱
پنل خورشیدی ۱۲-ولتی = ۱۰۰
پنل خورشیدی ۲٤-ولتی = ۱۰۱
راکتور همجوشی ۱۲-ولتی = ۱۱۰
راکتور همجوشی ۲٤-ولتی = ۱۱۱ (۲-۳)

کد کردن پارامترهای سیستم بخشی بسیار حساس از GA است و تأثیر عمده‌ای بر کار کردن یا نکردن GA دارد.

پس از آنکه در مورد نحوه‌ی کدگذاری پارامترهای سیستم تصمیم‌گیری کردیم، باید در مورد چگونگی ارزیابی برازندگی هر یک از راه‌حل‌های مسئله نیز تصمیم‌گیری نماییم. در مثال روبات، فرمولی وجود دارد که وزن موتور را به نوع/اندازه‌ی موتور و همچنین نوع/اندازه‌ی مولد توان مرتبط می‌سازد. ممکن است فرمول دیگری داشته باشیم که توان روبات را به موتور و منبع توان مرتبط سازد و بدین ترتیب برد روبات را به موتور و منبع توان مرتبط نماید. ما بدون داشتن تعریفی مناسب از برازندگی، قادر به شبیه‌سازی تکامل نخواهیم بود. متناوباً، با یک شبیه‌سازی کامپیوتری می‌توانیم نوع و اندازه‌ی موتور و همچنین نوع و اندازه‌ی منبع توان را به‌عنوان ورودی به کامپیوتر داده و در خروجی نحوه‌ی عملکرد طراحی‌مان را مشاهده کنیم و میزان مطلوب بودن آن را بسنجیم. اینجا جایی است که درک طراح GA از مسئله بسیار مهم می‌شود. در اینجا هیچ قانون سختگیرانه‌ی برای تعیین تابع برازندگی وجود ندارد و این موضوع تنها به درک طراح GA از مسئله بستگی دارد تا بتواند تابع برازندگی منطقی و درستی تعیین کند. در مورد مثال ما تابع برازندگی می‌تواند به شکل زیر باشد:

برازندگی = برد (بر حسب ساعت) + توان (بر حسب وات) − وزن (کیلوگرم) (۳-۳)

هر یک از سه پارامتر برد، توان و وزن خود ممکن است توابعی پیچیده از نوع موتور و نوع منبع توان باشند و یا ممکن است توسط خروجی آزمایش‌های سخت افزاری یا خروجی شبیه‌سازی‌های نرم‌افزاری تعیین شوند[1].

GA را با تولید اتفاقی مجموعه‌ای از ذرات آغاز می‌کنیم. دو ذره از جمعیت GA را در نظر بگیرید. ذره‌ی اول طرحی از روبات است که در آن از موتور step ۱۲ ولتی و پنل خورشیدی ۲٤ ولتی استفاده شده است و ذره‌ی دوم طرحی است که در آن از موتور servo ۹ ولتی و باتری نیکل-کادمیم استفاده شده است. این دو ذره به‌صورت زیر مشخص می‌شوند:

پنل خورشیدی ۲٤ ولتی و موتور step 12 ولتی = ذره‌ی اول
باتری نیکل-کادمیم ۲٤ ولتی و موتور servo 9 ولتی = ذره‌ی دوم (۳-٤)

ذره‌ی اول با رشته بیت ۰۱۰۱۰۱ و ذره‌ی دوم با رشته بیت ۱۰۱۰۰۱ مشخص می‌شود. هر بیت یک دگره (آلل) نام دارد. دنباله‌ای از بیت‌های درون یک ذره که حاوی اطلاعاتی در مورد ویژگی‌های آن ذره است را یک ژن می‌گویند. ژن‌های ویژه را ژنوتیپ[2] نامیده و پارامتر خاصی از مسئله را که یک ژنوتیپ را نمایندگی می‌کند، فنوتیپ[3] می‌نامند. در مثال ما هر ذره دو ژن دارد: یکی برای تعیین نوع و اندازه‌ی موتور و دیگری برای تعیین نوع و اندازه‌ی باتری. ژنوتیپ موتور ذره‌ی اول برابر ۰۱۰ است که این ژنوتیپ، فنوتیپ "موتور step ۱۲ ولتی" را نمایندگی می‌کند. ژنوتیپ باتری این ذره برابر ۱۰۱ بوده و فنوتیپ "باتری خورشیدی ۲٤ ولتی" را نمایندگی می‌کند. مجموعه‌ی تمام ژن‌ها در یک ذره را کروموزوم می‌نامند. ذره‌ی اول کروموزومی برابر ۰۱۰۱۰۱ دارد.

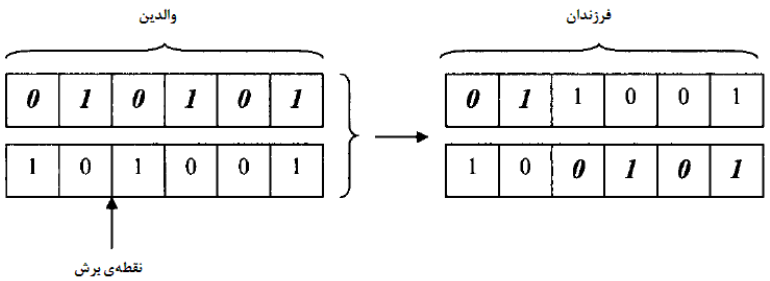
۳-۴-۲ انتخاب و برش

یک GA ممکن است تعداد زیادی ذره داشته باشد. GAهای نوعی ده‌ها و یا صدها ذره دارند. دو ذره‌ی ذکر شده در بالا مانند دو ذره از جمعیت‌های زیستی می‌توانند جفت‌گیری نمایند. برای این کار ذرات عمل *برش* را انجام می‌دهند و این بدان معناست که هر ذره برخی از اطلاعات ژنتیکی خود را با فرزندش به

[1] به دلیل تفاوت واحدها نمی‌توان ساعت و وات و کیلوگرم را با هم جمع کرد. بنابراین به نوعی پارامتر مقیاس‌کننده برای تبدیل این واحدها به واحدهایی که بتوانیم آن‌ها را با هم جمع کنیم نیاز داریم. معهذا، این معادله ایده‌ی کلی چگونگی فرمول‌بندی معادله‌ی برازندگی را نشان می‌دهد.

[2] Genotype
[3] Phenotype

اشتراک می‌گذارد. برای تعیین نقطه‌ی برش عددی اتفاقی میان ۱ و ۵ انتخاب می‌کنیم. فرض کنید عدد ۲ انتخاب شده است. این بدین معنی است که هر دو ذره دگره‌های خود را در کروموزوم‌ها از بیت دوم به بعد با یکدیگر عوض می‌کنند. این موضوع در شکل ۳-۱ نشان داده شده است.



شکل ۳-۱ تصویر برش در یک GA دودویی. نقطه‌ی برش به‌صورت اتفاقی انتخاب شده است. والدین فرزندان را تولید می‌کنند.

دو والد با یکدیگر جفت‌گیری کرده تا دو فرزند را ایجاد کنند. هر فرزند برخی اطلاعات ژنتیکی را از یک والد و برخی دیگر را از والد دیگر دریافت می‌کنند. والدین از بین رفته و فرزندان برای ادامه‌ی فرایند تکاملی به بقای خود ادامه می‌دهند. این فرایند یک نسل از GA نامیده می‌شود.

درست مانند آنچه در زیست‌شناسی نیز رخ می‌دهد، یک فرزند برازندگی بیشتری نسبت به فرزند دیگر دارد. احتمال مرگ ذرات با برازندگی کمتر در نسل‌هایشان زیاد است و این یعنی از شبیه‌سازی حذف می‌شوند. ذرات با برازندگی بالا به بقای خود برای جفت‌گیری با سایر ذرات با برازندگی بالا ادامه داده و بدین وسیله نسلی جدید از ذرات را به وجود می‌آورند. این فرایند آنقدر ادامه می‌یابد تا GA راه‌حلی قابل قبول برای مسئله‌ی بهینه‌سازی پیدا کند.

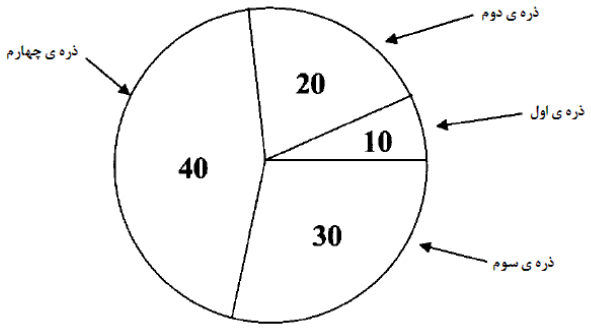
در برخی نقاط از نرم‌افزار GA باید تصمیم بگیریم کدام ذره‌ها برای ایجاد فرزندان با یکدیگر جفت‌گیری کنند. این تصمیم بر مبنای برازندگی ذرات در جمعیت حاضر گرفته می‌شود. ذراتی که بیشترین میزان برازندگی را دارند شانس بیشتری برای جفت‌گیری و تولید فرزندان دارند در حالی که ذراتی که کمترین میزان برازندگی را دارا هستند شانس کمتری برای جفت‌گیری داشته و احتمالاً بدون تولید هیچ فرزندی از بین خواهند رفت.

یک راه معمول برای گزینش والدین جهت انجام عمل جفت‌گیری چرخ رولت است که انتخاب متناسب با برازندگی نیز نامیده می‌شود. فرض کنید یک جمعیت چهار ذره‌ای داریم (یک GA واقعی بسیار بیشتر از چهار ذره خواهد داشت). حال فرض کنید برازندگی ذرات به شرح زیر است:

۱۰ = برازندگی : ذره‌ی اول
۲۰ = برازندگی : ذره‌ی دوم
۳۰ = برازندگی : ذره‌ی سوم
٤۰ = برازندگی : ذره‌ی چهارم (۵-۳)

ذره‌ی چهارم بیشترین برازندگی و ذره‌ی اول کمترین برازندگی را دارند. حال یک چرخ رولت ساخته و در آن ٤ شکاف متناسب با برازندگی هر یک از ذرات به وجود می‌آوریم. چرخ رولت این مثال در شکل ۳-۲ نمایش داده شده است.

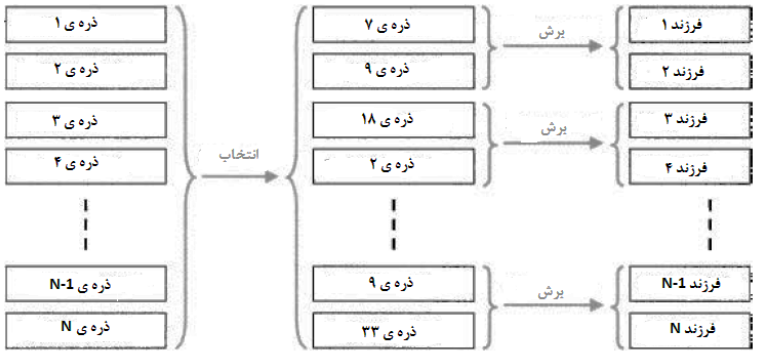
برای ثابت نگه داشتن اندازه‌ی جمعیت در هر نسل، دو دسته‌ی دوتایی برای جفت‌گیری انتخاب می‌کنیم. با این کار در کل ٤ فرزند به وجود خواهد آمد. برای انتخاب اولین زوج با یک چرخاننده‌ی خیالی (که در واقع با کامپیوتر شبیه‌سازی می‌شود) چرخ رولت را می‌چرخانیم. در هر شکافی که بایستد، نمایشگر اولین والد خواهد بود. با نگاه به چرخ رولت در می‌یابیم که ذره‌ی اول تنها ۱۰٪ ، ذره‌ی دوم ۲۰٪، ذره‌ی سوم ۳۰٪ و ذره‌ی چهارم ٤۰٪ احتمال انتخاب شدن دارند.



شکل ۳-۲ نمودار بالا انتخاب چرخ رولت را در یک GA برای جمعیتی چهار ذره‌ای نشان می‌دهد. هر یک از تکه‌ها به یک ذره اختصاص داشته و متناسب با برازندگی‌اش می‌باشد. انتخاب هر یک از ذرات به‌عنوان والد با بزرگی تکه‌ی مربوطه‌اش در چرخ رولت متناسب است.

به عبارت دیگر شانس انتخاب شدن هر ذره با برازندگی‌اش متناسب است. پس برای انتخاب دومین والد چرخ رولت را دوباره می‌چرخانیم. اگر چرخ رولت در همان قسمت ذره‌ی اول متوقف شد، چرخ را دوباره می‌چرخانیم چرا که یک ذره نمی‌تواند با خودش جفت‌گیری کند. پس از آنکه والدین انتخاب شدند، عمل برش جهت تولید فرزندان صورت می‌گیرد. سپس این فرایند را تکرار می‌کنیم تا دو والد جدید به وجود

آمده، جفت‌گیری کرده و دو فرزند دیگر را به وجود آورند. این فرایند آنقدر ادامه می‌یابد تا جمعیت فرزندان با جمعیت والدین برابر شود. این موضوع در شکل ۳-۳ نشان داده شده است.



شکل ۳-۳ نمایش فرایند برش در جمعیت والدین برای تولید جمعیت فرزندان. جمعیت N ذره‌ای اولیه از والدین در سمت چپ نمودار وارد فرایند انتخاب، احتمالا انتخاب چرخ رولت، شده تا مجموعه‌ای N تایی از والدین را به وجود آورد. ممکن است برخی ذرات بیش از یک بار انتخاب شوند درحالی که برخی دیگر اصلا انتخاب نشوند. سپس هر یک از زوج والدین در قسمت وسط نمودار با یکدیگر جفت‌گیری کرده و زوج فرزندان را به وجود می‌آورند. برگرفته شده از (وایتلی[1]، ۲۰۰۱).

با توجه به مقادیر برازندگی که در شکل ۳-۲ نشان داده شده است، می‌توان فرایند چرخ رولت برای انتخاب والدین را مانند شکل ۳-٤ انجام داد. در واقع ما فرایند شکل ۳-٤ را چهار بار تکرار می‌کنیم تا ٤ والد انتخاب شده و این ٤ والد، ٤ فرزند را به وجود آورند.

در مجموع، شکل ۳-۵ شبه‌کدی را برای انتخاب والدین در یک جمعیت N ذره‌ای با استفاده از چرخ رولت ارائه می‌دهد. فرایند شکل ۳-۵ را باید به تعداد دفعات لازم برای انتخاب والدین و ایجاد فرزندان نسل بعد تکرار نمود.

[1] Whitley

عددی اتفاقی r با توزیع یکنواخت میان [0,1] تولید کن

اگر $r < 0.1$ آنگاه

نره ی 1 = والد

اگر $0.1 < r < 0.3$ آنگاه

نره ی 2 = والد

اگر $0.3 < r < 0.6$ آنگاه

نره ی 3 = والد

در غیر این صورت

نره ی 4 = والد

پایان اگر

شکل ۳-٤ شبه‌کد بالا چگونگی انتخاب والدین بر اساس چرخ رولت را نشان می‌دهد.

x_i = نره ی i ام از جمعیت N نره ای

$f_i \leftarrow$ برازندگی x_i برای $i \in [1, N]$

$$f_{sun} = \sum_{i=1}^{N} f_i$$

یک عدد اتفاقی r با توزیع یکنواخت میان $[0, f_{sun}]$ تولید کن

$F \leftarrow f_1$

$K \leftarrow 1$

تا زمانی که $F < r$

$k \leftarrow k + 1$

$F \leftarrow F + f_k$

پایان

$x_k \leftarrow$ والد

شکل ۳-۵ شبه‌کد بالا نحوه‌ی انتخاب یک والد از میان N ذره با استفاده از انتخاب چرخ رولت را نشان می‌دهد. در این شبه‌کد مقادیر برازندگی برای تمام ذرات مثبت فرض شده است.

۳-۴-۳ جهش

گام نهایی در GA جهش نام دارد. جهش در زیست‌شناسی تا آنجا که تأثیر قابل توجهی بر فرزندان بگذارد، نادر است. در بیشتر پیاده‌سازی‌های GA نیز جهش نادر است (از رده‌ی ۲٪). اما نمی‌توان به‌صورت کلی گفت چه تنظیماتی، تنظیمات صحیح برای نرخ جهش GA است. بهترین نرخ جهش به خود مسئله، اندازه‌ی جمعیت، کدگذاری و دیگر فاکتورها بستگی دارد. صرف نظر از تعداد دفعات تکرار آن، جهش بسیار مهم است چرا که راه‌حل‌های بالقوه‌ی جدید برای مسئله می‌یابد. اگر جمعیت فاقد برخی اطلاعات ژنتیکی باشد، جهش فرصت تزریق این اطلاعات به جمعیت را فراهم می‌کند. این موضوع در تکامل‌های زیستی و از آن بیشتر در GAها حایز اهمیت است. این موضوع بدین دلیل است که جمعیت GAها آنقدر کوچک است که در آن‌ها مشکل درون همسری به وجود می‌آید. به دلیلی مشابه بن‌بست‌های تکاملی در GAها معمول‌تر بوده و بیشتر رخ می‌دهند. در تکامل زیستی صحبت از جمعیت‌های میلیونی است در حالی که در GAها از جمیت‌های صدتایی و یا ده‌تایی صحبت می‌شود.

برای پیاده‌سازی جهش یک احتمال جهش، برای مثال ۱٪، انتخاب میکنیم. این بدان معناست که پس از فرایند برش هر بیت در هر فرزند به احتمال ۱٪ ممکن است مقدارش از ۰ به ۱ یا از ۱ به ۰ تغییر کند. جهش فرایندی ساده است، اما انتخاب یک احتمال جهش منطقی بسیار حایز اهمیت است. انتخاب احتمال جهش زیاد باعث می‌شود که GA مانند یک جستجوی اتفاقی رفتار کند و آن را تبدیل به روشی کند که برای حل مسئله مناسب نیست. انتخاب احتمال جهش کم نیز باعث بروز مشکل درون همسری و بن‌بست‌های تکاملی a شده و این مشکلات GA را از یافتن راه‌حلی مناسب برای مسئله باز خواهند داشت.

اگر جمعیتی N ذره‌ای از x_iها داشته باشیم و هر ذره نیز n بیت داشته باشد و نرخ جهش نیز ρ باشد، آنگاه در انتهای هر نسل بیت‌ها را با احتمال ρ عوض خواهیم کرد:

$$r \leftarrow U[0,1]$$

$$x_i(k) \leftarrow \begin{cases} x_i(k) & \text{اگر} \quad r \geq p \\ 0 & \text{اگر} \quad r < p\,,\ x_i(k) = 1 \\ 1 & \text{اگر} \quad r < p\,,\ x_i(k) = 0 \end{cases} \qquad (۶-۳)$$

برای $i \in [1, N]$ و $k \in [1, n]$. U عددی اتفاقی است که به‌صورت یکنواخت بر روی $[0,1]$ توزیع شده است.

۳-۴-۴ خلاصه‌ی GA

این بخش مثالی ساده از GA برای طراحی روبات را ارائه نمود و به بحث در مورد انتخاب، برش و جهش در GAها پرداخت. شکل ۳-٦ با جمع‌آوری موضوعات بحث شده، خلاصه‌ای از طرح کلی یک GA را نشان می‌دهد.

۳-۴-۵ تنظیمات پارامترهای GA و مثال‌ها

شکل ۳-٦ طرح کلی یک GA ساده را نشان می‌دهد، با این حال می‌توان شاهد انعطاف‌پذیری بسیاری در الگوریتم شکل ۳-٦ بود. برای مثال، شرایط توقف برای یک GA می‌تواند شامل گزینه‌های متفاوتی باشد. یکی از گزینه‌ها آن است که GA برای تعداد نسل‌های از پیش تعیین شده‌ای ادامه یابد.

```
جمعیت تولید شده ی اتفاقی ← والدین
تا زمانی که شرایط قطع عملیات فراهم نشده
    برازندگی هر یک از والدین در جمعیت را محاسبه کن
    ∅ ← فرزندان
    تا زمانی که |والدین| > |فرزندان|
        از مقادیر برازندگی برای انتخاب احتمالاتی زوج والدین جهت جفت گیری استفاده کن
        والدین جفت گیری کرده و فرزندان C1 و C2 تولید شوند
        فرزندان U {c1, c2} ← فرزندان
    حلقه
    برخی از فرزندان را به صورت اتفاقی دچار جهش کن
    فرزندان ← والدین
نسل بعد
```

شکل ۳-٦ شبه‌کد بالا یک الگوریتم ساده‌ی ژنتیک را نشان می‌دهد.

گزینه‌ی دیگر آن است که GA تا آنجا ادامه پیدا کند که برازندگی بهترین ذره از یک آستانه‌ی معین بهتر شود. اگر مسئله‌ی ما پیدا کردن راه‌حلی است که به اندازه‌ی کافی خوب باشد آنگاه این کار گزینه‌ای منطقی برای تعیین شرایط توقف GA است. یکی دیگر از گزینه‌ها آن است که GA تا آنجایی ادامه پیدا کند که برازندگی بهترین ذره از نسلی به نسل دیگر بهبود نیابد. این یعنی فرایند تکاملی یکنواخت شده و نمی‌تواند بهتر از آنچه که هست بشود.

یک طراح GA برای آنکه بتواند نتایج خوبی به دست آورد باید چند پارامتر را مشخص کند. انتخاب این پارامترها معمولاً مرز میان موفقیت و شکست را تعیین می‌کند. برخی از این پارامترها به شرح زیرند:

۱. شیوه‌ی کدگذاری که مسئله را به رشته بیت‌ها نگاشت می‌کند. برخی مثال‌های آتی چگونگی کدگذاری دودویی اعداد حقیقی را نشان می‌دهند، بخش ۳–۵ در مورد GAها با پارامترهای دنیای واقعی بحث کرده و بخش ۸–۳ نیز در مورد کدگذاری گری و GAهای دودویی بحث خواهد نمود.
۲. یک تابع برازندگی که راه‌حل‌های مسئله را به مقادیر برازندگی نگاشت می‌کند.
۳. اندازه‌ی جمعیت.
٤. روش انتخاب. در بالا در مورد انتخاب چرخ رولت صحبت کردیم اما روش‌های دیگری نیز برای انتخاب وجود دارد مانند انتخاب مسابقه‌ای، انتخاب رتبه‌ای و بسیاری انواع دیگر. بخش ۸–۷ در مورد برخی از این روش‌ها بحث خواهد کرد.
۵. نرخ جهش. یک GA با نرخ جهش خیلی زیاد تبدیل به یک جستجوی اتفاقی شده و یک GA با نرخ جهش خیلی کم نیز نخواهد توانست فضای جستجو را به میزان کافی بکاود.
٦. مقیاس‌گذاری برازندگی. چگونگی پیاده‌سازی تابع برازندگی در اینجا مشخص می‌شود. برخی اوقات تابع برازندگی به گونه‌ای مناسب تعریف نشده و در نتیجه مقادیر برازندگی تمام ذرات به یکدیگر بسیار نزدیک خواهد بود. در چنین حالتی، فرایند انتخاب به درستی صورت نگرفته و ذرات با برازندگی‌های کم و زیاد به خوبی از هم منفک نمی‌شوند. این موضوع از انتشار ذرات برازنده‌تر به نسل بعد جلوگیری می‌کند. گاهاً عکس این قضیه نیز پیش می‌آید، بدین معنی که گاه مقادیر برازندگی ذرات آن چنان با هم اختلاف دارند که ذرات با برازندگی کمتر هیچ شانسی برای بازتولید نخواهند داشت. بخش ۸–۷ در مورد مقیاس‌گذاری برازندگی بحث خواهد نمود.
۷. نوع برش. پیش از این در مورد ایجاد برش در یک نقطه سخن به میان آوردیم. حال آنکه عمل برش می‌تواند در چندین نقطه صورت بپذیرد. بخش ۸–۸ در مورد انواع مختلف برش‌ها سخن خواهد رفت.
۸. برخی محققان GA فقط به ذراتی که شبیه یکدیگر باشند، متعلق به یک گونه باشند، اجازه‌ی جفت‌گیری می‌دهند. برخی دیگر از محققان تنها ذراتی را مجاز به جفت گیری با یکدیگر می‌دانند که از هم بسیار متفاوت باشند و این یعنی ذراتی که متعلق به خانواده‌های متفاوت‌اند. بخش ۸–٦ در مورد این ایده‌ها بحث خواهد کرد.

این موارد برای الگوریتم‌های تکاملی نیز کاربرد دارند. فصل ۸ در مورد این موارد و پاره‌ای دیگر از موضوعات بحث خواهد نمود.

مثال ۳-۲

مسئله‌ی مینیمم‌سازی مثال ۲-۲ را در نظر بگیرید.

$$min_x \mathrm{f(x)} =? \ , \qquad f(x) = x^4 + 5x^3 + 4x^2 - 4x + 1 \tag{۷-۳}$$

فرض کنید که ما به نحوی می‌دانیم مینیمم $f(x)$ جایی در دامنه‌ی $x \in [-4, -1]$ قرار دارد. حال x را با ٤ بیت به‌صورت زیر کد می‌کنیم:

$$\begin{array}{ll} 0000 = -4.0, & 0001 = -3.8 \\ 0010 = -3.6, & 0011 = -3.4 \\ 0100 = -3.2, & 0101 = -3.0 \\ 0110 = -2.8, & 0111 = -2.6 \\ 1000 = -2.4, & 1001 = -2.2 \\ 1010 = -2.0, & 1011 = -1.8 \\ 1100 = -1.6, & 1101 = -1.4 \\ 1110 = -1.2, & 1111 = -1.0 \end{array} \tag{۸-۳}$$

شیوه‌ی کدگذاری توازنی میان دقت و پیچیدگی است. تعداد بیت بیشتر دقت را افزایش داده اما از سویی نیز پیچیدگی GA را زیاد می‌کند. فرض کنید جمعیت اولیه به‌صورت اتفاقی برابر انتخاب شود:

$$\begin{array}{l} x_1 = 1100 \\ x_2 = 1011 \\ x_3 = 0010 \\ x_4 = 1001 \end{array} \tag{۹-۳}$$

ما می‌خواهیم $f(x)$ را مینیمم کنیم در حالی که GAها برای ماکزیمم‌سازی طراحی شده‌اند.[۱] بنابراین نیاز داریم مسئله را به یک مسئله‌ی ماکزیمم‌سازی تبدیل کنیم تا در چارچوب GA بگنجد. این کار را میتوان با ماکزیمم‌سازی منفی تابع $f(x)$ انجام داد. در این صورت مقادیر برازندگی با استفاده از تبدیل ژنوتیپ/فنوتیپ از معادله‌ی (۳-۸)، و سپس محاسبه‌ی $-f(x)$ به دست می‌آید:

$$\begin{array}{l} \text{برازندگی } (x_1) = -f(-1.6) = -3.71 \\ \text{برازندگی } (x_2) = -f(-1.8) = -2.50 \end{array} \tag{۱۰-۳}$$

[۱] این گفته با فرض آن است که از انتخاب چرخ رولت استفاده نموده‌ایم. برخی روش‌های انتخاب که در بخش ۸-۷ مورد بررسی قرار گرفته‌اند فرضی مبنی بر ماکزیمم‌سازی مسئله ندارند.

$$\text{برازندگی } (x_3) = -f(-3.6) = -1.92$$

$$\text{برازندگی } (x_4) = -f(-2.2) = +0.65$$

حال یک مقدار آفست دلخواه به هر یک از مقادیر برازندگی اضافه می‌کنیم به‌طوری که همه‌ی آن‌ها بزرگتر از ۰ شوند. این موضوع از این لحاظ حایز اهمیت است که بعداً بتوانیم به هر یک از ذرات برازندگی درصدی اختصاص دهیم:

$$\begin{aligned} f_1 &= -3.71 + 10 = 6.29 \\ f_2 &= -2.50 + 10 = 7.50 \\ f_3 &= -1.92 + 10 = 8.08 \\ f_4 &= +0.65 + 10 = 10.65 \end{aligned} \qquad (۱۱-۳)$$

حال برازندگی نسبی هر یک از ذرات را محاسبه می‌کنیم. برازندگی نسبی هر ذره در واقع احتمال انتخاب شدن آن ذره هنگام چرخیدن چرخ رولت است:

$$\begin{aligned} p_1 &= f_1/f_1 + f_2 + f_3 + f_4 = 0.19 \\ p_2 &= f_2/f_1 + f_2 + f_3 + f_4 = 0.23 \\ p_3 &= f_3/f_1 + f_2 + f_3 + f_4 = 0.25 \\ p_4 &= f_4/f_1 + f_2 + f_3 + f_4 = 0.33 \end{aligned} \qquad (۱۲-۳)$$

جمعیت اولیه در جدول ۳-۱ خلاصه شده است. جدول ۳-۱ نشان می‌دهد که هر یک از ذرات x_2 و x_3 در هر بار چرخش، ۲۵٪ شانس اتخاب شدن دارند، در حالی که x_4 شانس انتخاب شدنش تقریبا دو برابر x_1 است. برای آغاز نسل اول GA، اعدادی اتفاقی با توزیع یکنواخت را بر روی [0,1] جهت انتخاب ٤ والد اتخاذ می‌نماییم. فرض کنید نتیجه‌ی این فرایند گزینش x_3، x_4، x_4 و x_1 باشد. به عبارتی x_3 با x_4 و x_4 با x_1 عمل برش را انجام داده و فرزندان را تولید می‌نمایند. به یاد داشته باشید که نقطه‌ی برش برای هر زوج والدین به‌صورت اتفاقی انتخاب می‌شود. این موضوع در جدول ۳-۲ نشان داده شده است.

جدول ۳-۱ مثال ۳-۲: جمعیت اولیه‌ی یک GA ساده

شماره ذره	ژنوتیپ	فنوتیپ	برازندگی	احتمال انتخاب
x_1	1100	-1.4	-4.56	0.19
x_2	1011	-1.8	-2.50	0.23
x_3	0010	-3.6	-1.92	0.25
x_4	1001	-2.2	+0.65	0.33

جدول ۳-۲ مثال ۳-۲: برش یک *GA* ساده. نقاط برش تصادفی با اعداد پررنگ مشخص شده‌اند. نقطه‌ی برش برای زوج اول بین دو بیت اول و برای زوج دوم در میانه‌ی کروموزوم قرار دارد.

	والدین	فرزندان	
ذره	ژنوتیپ	ژنوتیپ	برازندگی
x_1	***0*** 010	0001	-8.11
x_2	1 ***001***	1010	-1.00
x_3	***10*** 01	1000	+2.30
x_4	11 ***00***	1101	-4.56

با توجه به جدول ۳-۲ درمی‌یابیم که بهترین فرزند با برازندگی ۲,۳۰ برازندگی بهتری نسبت به بهترین ذره‌ی جمعیت اولیه (۰,٦۵) دارد. GA قدمی قابل توجه در جهت بهینه‌سازی $f(x)$ برداشته است. هیچ تضمینی وجود ندارد که فرزندان بهتر از والدین باشند اما این مثال ساده نشان می‌دهد که چگونه GA به راه‌حلی جهت بهینه‌سازی یک مسئله می‌رسد.

مثال ۳-۳

مسئله‌ی مینیمم‌سازی مثال ۲-۵ را در نظر بگیرید:

$$min_{x,y} f(x,y) = \min\{e - 20\exp\left(-0.2\sqrt{\frac{x^2+y^2}{2}}\right) - \exp\left(\frac{\cos(2\pi x)+\cos(2\pi y)}{2}\right)\} \quad (۱۳-۳)$$

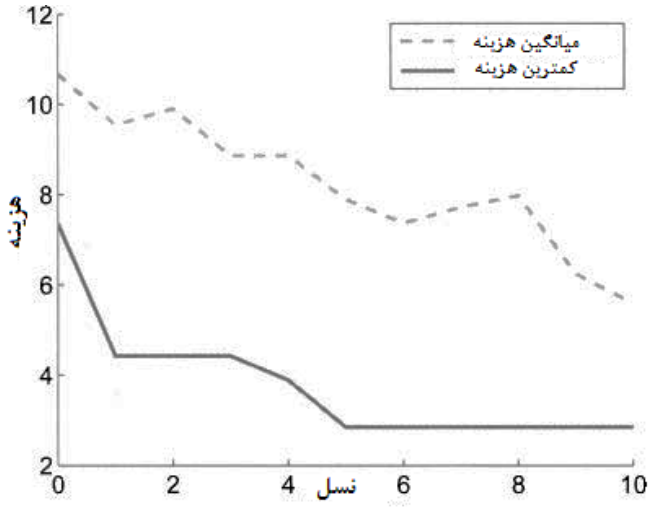
فرض کنید همانند مثال ۲-۵، x و y میان $5+$ و $5-$ متغیر باشند. حال باید در مورد کیفیت و دقت x و y تصمیم بگیریم. اگر برای هر یک از متغیرهای مستقل x و y دقتی برابر ۰,۲۵ مد نظر باشد باید از شش بیت برای نشان دادن x و y استفاده کنیم:

$$\begin{aligned} &x \text{ ژنوتیپ}=x_g \in [0,63] \\ &x \text{ فنوتیپ} = -5 + \frac{10x_g}{63} \in [-5,5] \\ &y \text{ ژنوتیپ} = x_g \in [0,63] \\ &y \text{ فنوتیپ} = -5 + \frac{10y_g}{63} \in [-5,5] \end{aligned} \quad (۱٤-۳)$$

با این کار برای هر بیت از x_g و y_g دقتی برابر $10/63 = 0.159$ به دست خواهد آمد. بیایید برای مینیمم‌سازی $f(x,y)$، GA را برای ۱۰ نسل اجرا کنیم. حال باید در مورد اندازه‌ی جمعیت و نرخ جهش تصمیم‌گیری

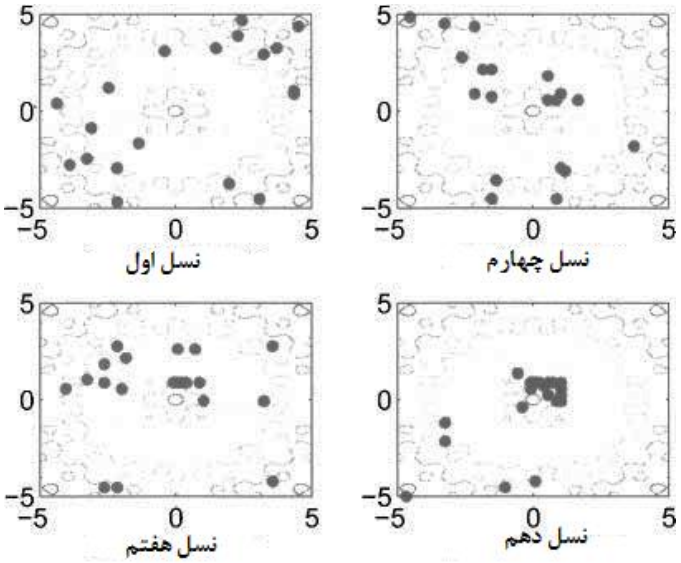
کنیم. اجازه دهید اندازه‌ی جمعیت ۱۰ و نرخ جهش برابر ۲٪ باشد. یک اجرای نوعی از GA نتایجی مانند آنچه در شکل ۳-۷ نشان داده شده است را در بر خواهد داشت. با افزایش شماره‌ی نسل‌ها شاهد کاهش کمترین هزینه (هزینه‌ی بهترین ذره) و همچنین کاهش میانگین هزینه‌ی جمعیت خواهیم بود.

شکل ۳-۸ نمودار کانتور $f(x,y)$ با محل ذرات در نسل‌های اول چهارم، هفتم و دهم را نشان می‌دهد. با توجه به این شکل می‌توان مشاهده نمود که به دلیل مقداردهی اولیه‌ی اتفاقی، جمعیت اولیه در سراسر دامنه پخش شده است. با پیشروی GA جمعیت به خوشه شدن گرایش پیدا کرده و ذرات به سمت مینیمم، که در مرکز نمودار واقع شده است، حرکت می‌کنند.



شکل ۳-۷ مثال ۳-۳: نتایج نوعی شبیه‌سازی یک GA برای مینیمم‌سازی تابع آکلی[1] دو بعدی.

[1] Ackley



شکل ۳–۸ مثال ۳–۳: نتایج نوعی شبیه‌سازی یک GA برای مینیمم‌سازی تابع اَکلی دو بعدی. با پیشروی GA ذرات جمعیت به تدریج شروع به خوشه شدن کرده و به سمت مینیمم، که در مرکز صفحه قرار دارد، حرکت می‌کنند.

نمودار شکل ۳–۸ نشان می‌دهد که مینیمم‌سازی یک تابع چند پیمانه‌ای و با ابعاد زیاد چه قدر می‌تواند سخت باشد. فرض کنید که در چشم‌اندازی با تپه‌ها و دره‌های شکل ۳–۸ ایستاده‌اید و می‌خواهید پست‌ترین نقطه‌ی آن را بیابید. این کار سختی خواهد بود چرا که پستی بلندی‌های بسیاری وجود دارد. با این حال، گروهی از ذرات که در سرتاسر این چشم‌انداز پخش شده‌اند شانس بیشتری برای پیدا کردن پست‌ترین نقطه خواهند داشت. ذرات می‌توانند از یکدیگر یاد بگیرند. یکی از آن‌ها می‌گوید:"این دره بسیار پست به نظر می‌رسد بیایید آن را بگردیم"، دیگری می‌گوید:"نه! این یکی پست‌تر است بیایید اینجا!" ذرات با یکدیگر همکاری کرده تا پست‌ترین نقطه‌ی دره را بیابند. این شبیه طرز کار GA و سایر الگوریتم‌های تکاملی است. ذرات موجود در جمعیت با یکدیگر همکاری کرده تا راه‌حل خوبی برای مسئله بیابند.

۳–۵ یک الگوریتم ژنتیک پیوسته‌ی ساده

شکل ۳–٦ طرح کلی یک GA دودویی ساده را نشان می‌دهد، دقیقاً همان الگوریتمی که ما در مثال‌های ۳–۲ و ۳–۳ از آن استفاده کردیم. با این حال، مسائل موجود در آن مثال‌ها بر روی دامنه‌های پیوسته تعریف شده بودند و به همین دلیل ما باید دامنه را گسسته می‌کردیم تا بتوانیم GA دودویی را بر آن‌ها اعمال نماییم. ساده‌تر و البته طبیعی‌تر است اگر بتوانیم GA را مستقیماً به دامنه‌ی پیوسته‌ی مسائل اعمال کنیم. ما از عبارت

GA پیوسته و یا GA کدشده‌ی حقیقی برای اشاره به GAهایی که مستقیماً بر روی متغیرهای پیوسته عمل می‌کنند، استفاده می‌کنیم.

تعمیم GAهای دودویی به GAهای پیوسته بسیار سرراست است. در حقیقت، می‌توانیم از همان الگوریتم شکل ۳–٦ استفاده کنیم. تنها کاری که باید انجام دهیم اصلاح برخی قدم‌های این الگوریتم است. به عملیات موجود در شکل ۳–٦ نگاه کنید و تصور کنید که این عملیات بر روی یک مسئله‌ی بهینه‌سازی با دامنه‌ی پیوسته چگونه کار می‌کند.

۱. در شکل ۳–٦ ما ابتدا یک جمعیت اولیه اتفاقی تولید می‌کنیم. می‌توان این کار را به راحتی بر روی دامنه‌ی پیوسته نیز انجام داد. فرض کنید می‌خواهیم N ذره در GA تولید کنیم. سپس iامین ذره را با x_i نشان می‌دهیم ($i \in [1, N]$). همچنین فرض کنید می‌خواهیم تابعی n–بعدی را بر روی یک دامنه‌ی پیوسته مینیمم کنیم. آنگاه برای نشان دادن kامین عنصر x_i از عبارت $x_i(k)$ استفاده می‌نماییم:

$$x_i = [x_i(1) \;\; x_i(2) \;\; \dots \;\; x_i(n)] \qquad (۱۵–۳)$$

فرض کنید دامنه‌ی جستجوی k امین بعد، $[x_{min}(k), x_{max}(k)]$ است:

$$x_i(k) \in [x_{min}(k), x_{max}(k)] \qquad (۱٦–۳)$$

در معادله بالا $i \in [1, N]$ و $k \in [1, n]$ می‌باشد. می‌توانیم همانند خط اول شکل ۳–٦ یک جمعیت اولیه‌ی اتفاقی تولید کنیم:

برای $i = 1$ تا N

برای $k = 1$ تا n

$$x_i(k) \leftarrow U[x_{min}(k), x_{max}(k)]$$

k بعدی

i بعدی

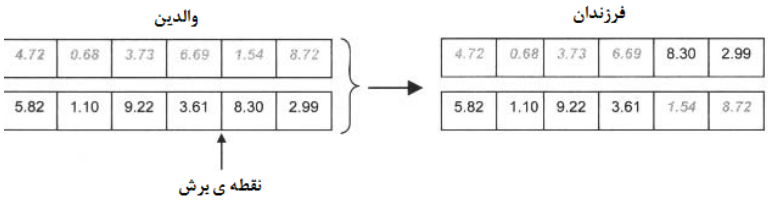
بدین ترتیب و به سادگی هر $x_i(k)$ را برابر تحققی از یک متغیر تصادفی که به‌صورت یکنواخت بین $x_{min}(k)$ و $x_{max}(k)$ توزیع شده، قرار می‌دهیم.

۲. در قدم بعدی حلقهی "تا زمانی که شرایط قطع عملیات فراهم نشده" در شکل ۳–٦ را آغاز می‌کنیم. اولین مرحله در آن حلقه محاسبه‌ی برازندگی تمامی ذرات است. اگر سعی بر ماکزیمم‌سازی $f(x)$

است، برازندگی هر ذره را با محاسبه‌ی $f(x_i)$ به دست می‌آوریم. اگر هدف مینیمم‌سازی $f(x)$ است برازندگی هر ذره را با محاسبه‌ی منفی $f(x_i)$ به دست می‌آوریم.

۳. در مرحله‌ی بعدی حلقه‌ی "تا زمانی که $|\text{والدین}| > |\text{فرزندان}|$" را در شکل ۳–٦ شروع می‌کنیم. قدم اول در این حلقه "استفاده از برازندگی‌ها برای گزینش احتمالاتی والدین جهت جفت‌گیری" است. این مرحله را با استفاده از انتخاب چرخ رولت انجام می‌دهیم. در مورد گزینه‌های دیگر برای انجام این مرحله در بخش ۸–۷ صحبت خواهیم کرد اما فعلاً از انتخاب چرخ رولت استفاده می‌نماییم.

٤. سپس مرحله‌ی "جفت‌گیری والدین" برای تولید فرزندان آغاز می‌شود. برای انجام این مرحله از برش تک نقطه‌ای، که در شکل ۳–۱ نشان داده شده است، استفاده می‌کنیم. تنها تفاوت این است که در اینجا ذره‌های با دامنه‌ی پیوسته را با هم مخلوط می‌نماییم. برش تک نقطه‌ای برای ذره‌های با دامنه‌ی پیوسته در شکل ۳–۹ نشان داده شده است. انواع دیگر برش‌ها برای GAهای پیوسته در بخش ۸–۸ مورد بحث قرار خواهند گرفت.



شکل ۳–۹ نمایش برش در یک GA پیوسته. نقطه‌ی برش به‌صورت اتفاقی انتخاب شده است. والدین فرزندان را به وجود می‌آورند.

۵. در مرحله‌ی بعد، قدم "جهش اتفاقی" در شکل ۳–٦ را پیاده‌سازی می‌کنیم. همان‌طور که معادله‌ی (۳–٦) نشان می‌دهد، در الگوریتم‌های تکاملی دودویی، جهش عملی سرراست است. در GAهای پیوسته، با تخصیص یک عدد اتفاقی با توزیع یکنواخت بر روی دامنه‌ی جستجو به $x_i(k)$ آن را دچار جهش می‌کنیم:

$$r \leftarrow U[0,1]$$
$$x_i(k) \leftarrow \begin{cases} x_i(k) & \text{اگر} \quad r \geq p \\ U[x_{min}(k), x_{max}(k)] & \text{اگر} \quad r < p \end{cases} \qquad (۱۷–۳)$$

P نرخ جهش است. در مورد دیگر گزینه‌های موجود برای جهش در GAها دامنه پیوسته در بخش ۸–۹ بحث خواهیم کرد.

جهش در GAهای پیوسته

توجه داشته باشید که یک نرخ جهش معین تأثیرات متفاوتی بر روی GA دودویی و GA پیوسته دارد. هنگامی که یک مسئله‌ی n بعدی با دامنه پیوسته و با نرخ جهش ρ_c داریم آنگاه هر ویژگی راه‌حل متناظر با هر فرزند با احتمال ρ_c جهش خواهد یافت. برای مثال در شکل ۳–۹، هر یک از شش جزء هر دو فرزند احتمال جهشی برابر ρ_c دارند. علاوه بر این، همان‌طور که در شکل ۳–۱۷ نشان داده شده است، جهش در GA پیوسته باعث می‌شود ویژگی راه‌حل، از یک توزیع یکنواخت میان مقادیر احتمال ماکزیمم و مینیمم آن گرفته شود.[۱]

با این حال در GA دودویی هر بعد از هر ذره را گسسته‌سازی می‌کنیم. اگر یک بعد پیوسته را به وسیله‌ی m بیت به حالت گسسته درآورده و نرخ جهش برابر ρ_b باشد آنگاه، هر بیت به احتمال ρ_b جهش پیدا خواهد کرد. این یعنی احتمال جهش نیافتن هر بیت برابر $1-\rho_b$ خواهد بود. بنابراین، احتمال جهش نیافتن هر بعد برابرست با احتمال آنکه هیچ یک از m بیت آن جهش نیابند که این مقدار برابر $(1-p_b)^m$ خواهد بود. در نتیجه، احتمال جهش m امین بعد برابر $1-(1-p_b)^m$ خواهد بود. به علاوه، اگر جهش رخ بدهد، بعد جهش یافته دارای توزیعی یکنواخت بین مقادیر ماکزیمم و مینیمم خود نخواهد بود و در عوض توزیع آن به شماره‌ی بیت جهش یافته بستگی خواهد داشت.

ما می‌توانیم نرخ جهش ρ_c را برای یک مسئله با دامنه یکنواخت، با تأثیری تقریباً برابر با تأثیر ρ_b برای یک مسئله‌ی گسسته، به دست آوریم. همان‌طور که پیش از این گفته شد، اگر در یک GA دودویی برای یک مسئله‌ی گسسته و با m بیت در هر بعد، نرخ جهش ρ_b باشد، آنگاه احتمال آنکه هیچ بعدی دچار جهش نشود برابر $(1-p_b)^m$ خواهد بود. این مقدار را می‌توان با سری تیلور مرتبه اول تقریب زد:

$$\text{احتمال}\left(\text{عدم جهش } GA \text{ دودویی}\right) = (1-p_b)^m \approx 1-mp_b \qquad (۱۸–۳)$$

این تقریب برای ρ_bهای کوچک معتبر است. اگر یک GA برای مسائل پیوسته دارای نرخ جهش ρ_c باشد آنگاه احتمال آنکه هیچ بعدی دچار جهش نشود برابر $1-p_c$ خواهد بود. اگر این مقدار را با معادله‌ی (۳–۱۸) برابر قرار دهیم خواهیم داشت:

$$\begin{aligned} 1-p_c &= 1-mp_b \\ p_c &= mp_b \end{aligned} \qquad (۱۹–۳)$$

[۱] جهش یکنواخت احتمالاً قدیمی‌ترین نوع جهش در GAهای پیوسته می باشد. با این حال، می توان انواع دیگری از جهش‌ها را به کار برد.

بنابراین فرایند جهش در یک GA دودویی با m بیت در هر بعد و نرخ جهش ρ_b تقریباً با فرایند جهش در یک GA پیوسته با نرخ جهش $m\rho_b$ معادل است. در جمله‌ی قبل بر کلمه‌ی تقریباً تأکید کردیم چرا که معلوم نیست نرخ‌های جهش معادل در GAهای دودویی و پیوسته نتایج مشابهی را به دست دهند. این بدین دلیل است که توزیع اندازه‌ی جهش در GAهای دودویی و پیوسته متفاوت است. یک زمینه‌ی جالب برای مطالعات آینده می‌تواند بررسی کامل هم‌ارزی جهش در GA دودویی و پیوسته باشد.

مثال ۳-٤

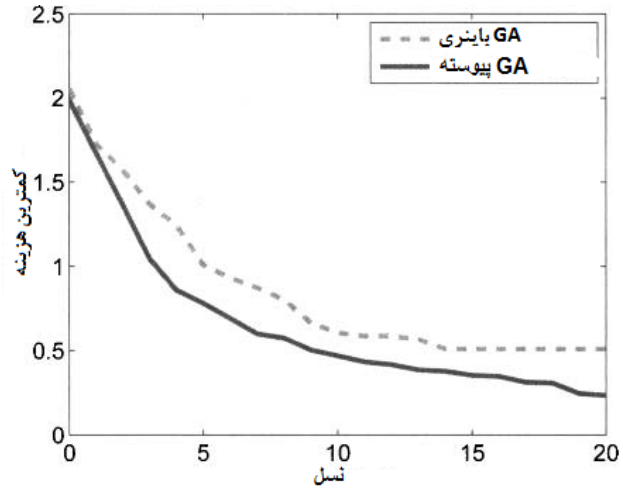
مسئله‌ی مینیمم‌سازی مثال ۳-۳ را در نظر بگیرید:

$$min_{x,y} f(x,y) = \min\{e - 20\exp\left(-0.2\sqrt{\frac{x^2+y^2}{2}}\right) - \exp\left(\frac{\cos(2\pi x)+\cos(2\pi y)}{2}\right)\} \qquad (۲۰-۳)$$

فرض کنید که x و y هر دو بین *1-* و *1+* متغیراند. در مثال ۳-۳ ما فضای جستجو را جهت اعمال GA دودویی گسسته‌سازی کردیم. با این حال، از آنجا که مسئله بر روی دامنه‌ای پیوسته تعریف شده است بهتر است از GA پیوسته استفاده شود. در این مثال هم GA پیوسته و هم GA دودویی را برای ۲۰ نسل و با اندازه جمعیت ۱۰ به اجرا می‌گذاریم. برای GA دودویی همانند مثال ۳-۳، از چهار بیت در هر بعد استفاده کرده و نرخ جهش را برابر ۲٪ می‌گیریم. برای آنکه تأثیر جهش در GA پیوسته تقریبا همانند تأثیر جهش در GA دودویی باشد، نرخ جهش را برای GA پیوسته برابر ۸٪ فرض می‌کنیم. همچنین از یک فاکتور نخبه‌گرایی[1] برابر با ۱ استفاده می‌کنیم. این بدان معناست که بهترین ذره‌ی جمعیت را از نسلی به نسل بعدی نگه می‌داریم (بخش ۸-٤ را ببینید).

شکل ۳-۱۰ بهترین ذره‌ی پیدا شده به‌صورت میانگین را در طول ۵۰ شبیه‌سازی نشان می‌دهد. مشاهده می‌شود که GA پیوسته به طرز قابل توجهی بهتر از GA دودویی است. در مسائل با دامنه‌ی پیوسته که برای حل آن‌ها از GA دودویی استفاده می‌کنیم، معمولاً (نه همیشه) با افزایش تعداد بیت‌ها نتیجه‌ی بهتری به دست آورده و اگر از GA پیوسته استفاده کنیم بهترین عملکرد را خواهیم داشت.

[1] Elitism



شکل ۳-۱۰ مثال ۳-٤: مقایسه‌ی عملکرد GA دودویی و GA پیوسته برای تابع دو بعدی *آکلی*. نمودار هزینه‌ی بهترین ذره در هر نسل به‌صورت میانگین از ۵۰ شبیه‌سازی نشان داده شده است.

جالب است بدانید GAهای پیوسته تاریخچه‌ی بحث برانگیزی دارند. از آنجا که GAها در اصل به‌صورت دودویی ارائه شدند و از آنجا که تمامی نظریات GA در ابتدا معطوف به GA دودویی بود، محققان در مورد ظهور GA پیوسته شک داشتند (گلدبرگ[1]،۱۹۹۱). با این حال، با وجود راحتی استفاده، موفقیت و حمایت نظریه‌ای که اخیراً در مورد آن به وجود آمده است، سخت است که بتوان آن را مورد نقد قرار داد.

۳-۶ نتیجه‌گیری

الگوریتم ژنتیک یکی از اولین الگوریتم‌های تکاملی بود و امروزه محبوب‌ترین آن‌هاست. در سال‌های اخیر بسیاری الگوریتم‌های تکاملی وارد صحنه‌ی رقابت شده‌اند اما GAها به دلیل انسی که با آن‌ها به وجود آمده و به دلیل عملکرد خوبشان در بسیاری از مسائل محبوب مانده‌اند.

در طول سال‌ها کتاب‌ها و مقالات زیادی در مورد GAها نوشته شده است. کتاب گلدبرگ (گلدبرگ، ۱۹۸۹)، یکی از اولین کتاب‌ها در زمینه‌ی GA بود اما مانند بسیاری از کتاب‌های اولیه‌ی دیگر در سایر زمینه‌ها، به دلیل شرح و تفسیر واضحش همچنان محبوب و مشهور است. کتاب‌های خوب دیگری نیز در زمینه‌ی GAها وجود دارند مانند (میچل، ۱۹۹۸)، (میچالویکز[2]، ۱۹۹٦)، هاپت و هاپت[3]، ۲۰۰٤) و (ریوز[4] و روو[5]،

[1] Goldberg
[2] Michalewicz
[3] Haupt
[4] Reevs
[5] Rowe

۲۰۰۳) که به دلیل تأکید محکمش بر نظریات بسیار قابل توجه است. برخی مقالات آموزشی شامل (باک[1] و اشوفل[2]، ۱۹۹۳)، (وایتلی، ۱۹۹٤) و (وایتلی، ۲۰۰۱) می‌باشد.

با توجه به تعداد بسیار زیاد کتاب‌ها و مقالات موجود درباره‌ی GAها، این فصل تنها معرفی مختصری از موضوع ارائه داده است. ما بسیاری از موضوعات مرتبط با GA را نادیده گرفتیم، نه به این خاطر که بی‌اهمیت هستند بلکه به دلیل آنکه دید ما محدود است. برخی از این موارد به این شرح‌اند: GAهای به هم ریخته[3]، که دارای کروموزوم‌هایی با طول متفاوت می‌باشند، (گلدبرگ، *1989b*) و (میچل، ۱۹۹۸)، GAهای جنسیت-محور، که چندین جنسیت مختلف را در جمعیت GA شبیه‌سازی کرده و بیشتر برای بهینه‌سازی‌های چندهدفه استفاده می‌شوند (فصل ۲۰) (لیز[4] و ایبن[5]، ۱۹۹۷)، GAهای جزیره‌ای، که شامل زیرجمعیت‌ها می‌شود (وایتلی و همکاران، ۱۹۹۸)، GAهای سلولی، که رابطه‌ی فضایی خاصی میان ذرات جمعیت ایجاد می‌کند (وایتلی، ۱۹۹٤) و تطبیق ماتریس کوواریانس، که یک استراتژی جستجوی محلی است و می‌توان آن را به هر الگوریتم تکاملی دیگری اضافه نمود (هانسن و همکاران، ۲۰۰۳).

همچنین متغیرهای زیادی در GA اصلی وجود دارد که آن‌ها را در این فصل ارائه کردیم. برخی از آن‌ها بسیار حایز اهمیت‌اند و مرز میان موفقیت و شکست را تعیین می‌کنند. فصل ۸ در مورد بسیاری از متغیرات که در GAها و سایر الگوریتم‌های تکاملی کاربرد دارند بحث خواهد کرد.

مسائل نوشتاری

۳-۱) با توجه به مثال بخش ۳-٤-۱ در مورد طراحی روبات، فرض کنید یک ذره‌ی GA با رشته بیت ۱۱۰۰۱۰ مشخص می‌شود.

a) کروموزوم این ذره‌ی GA چیست؟

b) ژنوتیپ و فنوتیپ این ذره چه هستند؟

۳-۲) می‌خواهیم از یک GA دودویی برای برای مینیمم‌سازی تابع دو بعدی *رستریجین* بر روی بازه‌ی $[-5,5]$ با دقت ۰,۱ استفاده کنیم (قسمت ۱۱-۱.ج را ببینید).

a) برای هر کروموزوم چند ژن نیاز داریم؟

b) در هر ژن به چند بیت نیاز داریم؟

c) با توجه به جواب قسمت قبل، دقت هر x چه قدر است؟

---

[1] Back
[2] Schwefel
[3] Messy GAs
[4] Lis
[5] Eiben

۳-۳) GA با ۱۰ ذره‌ی x_i داریم و برازندگی هر ذره برابر $f(x_i) = i$ می‌باشد. ($i \in [1,10]$). از انتخاب چرخ رولت برای انتخاب ۱۰ والد جهت جفت‌گیری استفاده می‌کنیم. اولین زوج والدین برای تولید دو فرزند جفت‌گیری کرده و به همین ترتیب زوج‌های بعد نیز این عمل را تکرار می‌کنند.

a) احتمال آنکه برازنده‌ترین ذره حداقل یکبار با خودش جفت‌گیری کند چه قدر است؟

b) این احتمال برای ذره با کمترین برازندگی چه قدر است؟

۴-۳) GA با ۱۰ ذره‌ی x_i داریم و برازندگی هر ذره برابر $f(x_i) = i$ می‌باشد. ($i \in [1,10]$). از انتخاب چرخ رولت برای انتخاب ۱۰ والد جهت جفت‌گیری استفاده می‌کنیم.

a) احتمال آنکه ذره‌ی x_{10} بعد از ۱۰ بار چرخاندن چرخ رولت اصلاً انتخاب نشود چه قدر است؟

b) احتمال آنکه ذره‌ی x_{10} در ۱۰ بار چرخاندن چرخ رولت دقیقا یک بار انتخاب شود چه قدر است؟

c) احتمال آنکه ذره‌ی x_{10} بعد از ۱۰ بار چرخاندن چرخ رولت بیش از یک بار انتخاب شود چه قدر است؟

۵-۳) انتخاب چرخ رولت فرض می‌کند که مقادیر برازندگی عبارت $f(x_i) \geq 0$ برای $i \in [1, N]$ را برآورده می‌کنند. فرض کنید جمعیتی با مقادیر برازندگی {3 , 2 , 0 , -5 , -10} در اختیار دارید. اگر بخواهید از انتخاب چرخ رولت استفاده نمایید این اعداد را چگونه اصلاح می‌کنید؟

۶-۳) در انتخاب چرخ رولت فرض شده است که جمعیت با مقادیر برازندگی $\{f(x_i)\}$ شناسایی می‌شود به‌طوری که مقادیر برازندگی بزرگتر از مقادیر برازندگی کمتر، بهتر هستند. حال فرض کنید مسئله‌ای در اختیار داریم که در آن جمعیت با مقادیر هزینه $\{c(x_i)\}$ شناسایی شده به‌طوری که مقادیر هزینه‌ی کمتر از مقادیر هزینه‌ی بیشتر بهتر هستند و برای تمامی i ها $c(x_i) > 0$. اگر بخواهید از انتخاب چرخ رولت استفاده نمایید، مقادیر هزینه را چگونه اصلاح می‌کنید؟

۷-۳) در یک GA دودویی دو والد هر کدام با n بیت وجود دارند. برای $i \in [1, n]$، iامین بیت والد اول با iامین بیت والد دوم متفاوت است. نقطه‌ی برشی را به‌صورت اتفاقی انتخاب می‌کنیم $c \in [1, n]$. با چه احتمالی فرزندان دقیقاً شبیه والدین هستند؟

۸-۳) فرض کنید در یک GA دودویی N ذره داریم که به‌صورت اتفاقی مقداردهی شده‌اند و هر ذره از n بیت تشکیل شده است.

a) احتمال آنکه برای یک i مشخص بیت iام هر ذره یکی باشد چه قدر است؟

b) برای تمامی $i \in [1, n]$ احتمال آنکه iامین بیت هر ذره یکی نباشد چه قدر است؟

c) به یاد آورید که برای مقادیر کوچک am داریم $\exp(-am) \approx 1 - am$ و همین‌طور برای مقادیر کوچک a داریم $(1-a)^m \approx 1 - am$. از این دو جمله استفاده کرده و جواب خود را در قسمت *b* تقریب بزنید.

d) از جواب قسمت *c* برای پیدا کردن اندازه‌ی جمعیت (N) مورد نیاز استفاده کنید، به‌طوری که احتمال آنکه هر دو ژن در جایگاه بیت یک جمعیت اتفاقی اولیه واقع شوند برابر p باشد.

e) فرض کنید می‌خواهیم یک جمعیت از ذرات، هر کدام با ۱۰۰ بیت را به‌صورت اتفاقی مقداردهی کنیم تا با احتمال ۹۹٫۹٪ یا بیشتر هر دو ژن در هر بیت رخ دهند. از جواب قسمت *d* برای پیدا کردن اندازه‌ی جمعیت مورد نیاز استفاده نمایید.

۳–۹) یک GA دودویی با اندازه‌ی جمعیت N و نرخ جهش ρ در اختیار داریم. هر ذره از n بیت تشکیل شده است.

a) احتمال آنکه هیچ بیتی در سراسر جمعیت در یک نسل دچار جهش نشود چه قدر است؟

b) اگر اندازه‌ی جمعیت N باشد و طول بیت هر ذره برابر n فرض شود، کمترین نرخ جهش ρ را به‌گونه‌ای بیابید که احتمال عدم وقوع جهش در هر نسل کمتر از P_{no} باشد. از جواب قسمت *a* استفاده کنید.

c) از جواب قسمت *b* برای پیدا کردن مینیمم نرخ جهش ρ استفاده کنید به‌طوری که احتمال آنکه هیچ بیتی دچار جهش نشود برابر ۰٫۰۱٪ باشد. $N=100$, $n=100$

مسائل کامپیوتری

۳–۱۰) برنامه‌ای کامپیوتری برای مسئله‌ی ۳–۳ بنویسید.

۳–۱۱) برنامه‌ای کامپیوتری برای مسئله‌ی ۳–۸ بنویسید.

۳–۱۲) یک مسئله‌ی one-max مسئله‌ی جستجوی رشته بیتی nتایی با بیشترین تعداد ۱ ممکن است. برازندگی یک رشته بیت تعداد ۱های آن است. صدالبته که می‌توان این مسئله را با نوشتن ۱های متوالی حل نمود، اما در این مسئله می‌خواهیم ببینیم آیا یک *GA* قادر به حل مسئله‌ی one-max می‌باشد یا خیر. یک GA بنویسید که مسئله‌ی one-max را حل کند. $N=20$, $n=30$, تعداد نسل‌ها =۱۰۰، نرخ جهش = ۱٪.

a) برازندگی بهترین ذره و میانگین برازندگی جمعیت را به‌عنوان تابعی از نسل‌ها رسم کنید.

b) ۵۰ شبیه‌سازی مونت کارلو از GA خود اجرا کنید. این کار ۵۰ نمودار برازندگی بهترین ذره بر حسب شماره‌ی نسل‌ها به دست خواهد داد. میانگین آن ۵۰ نمودار را رسم کنید. میانگین بهترین مقادیر برازندگی در نسل ۱۰۰ام را با $\bar{f}(x^*)$ نمایش دهید. $\bar{f}(x^*)$ چیست؟

c) بخش b را با اندازه جمعیت ٤٠ تکرار کنید. $\bar{f}(x^*)$ چگونه تغییر می‌کند؟ چرا؟

d) اندازه‌ی جمعیت را برابر ۲۰ گرفته و این بار نرخ جهش را برابر ۵٪ لحاظ کنید؟ حال $\bar{f}(x^*)$ نسبت به قسمت b چه تغییری می‌کند؟ چرا؟

e) نرخ جهش را برابر ۰٪ قرار دهید. $\bar{f}(x^*)$ نسبت به قسمت b چه تغییری می‌کند؟ چرا؟

f) به جای آنکه برازندگی را برابر تعداد ۱ها تعریف کنید، آن را برابر تعداد ۱ها به علاوه‌ی ۵۰ تعریف کرده و قسمت b را تکرار کنید. $\bar{f}(x^*)$ نسبت به قسمت b چه تغییری می‌کند؟ چرا؟

g) مانند قسمت b دوباره برازندگی را برابر تعداد ۱ها قرار دهید اما این بار برازندگی ذراتی که مقدار برازندگی‌شان کمتر از میانگین است را برابر ۰ قرار دهید. $\bar{f}(x^*)$ نسبت به بخش b چه تغییری می‌کند؟ چرا؟

۳-۱۳) یک GA پیوسته بنویسید که تابع کروی را مینیمم کند (بخش ۱-۱.ج را ببینید). فضای جستجو در هر بعد را در بازه‌ی *[-5,+5]* قرار دهید، بعد مسئله برابر ۲۰ است، تعداد نسل‌ها را ۱۰۰ قرار داده، اندازه‌ی جمعیت را برابر ۲۰ گرفته و نرخ جهش را ۱٪ فرض کنید. برای آنکه بتوانیم از انتخاب چرخ رولت استفاده نماییم باید مقادیر هزینه $c(x_i)$ را به مقادیر برازندگی $f(x_i)$ نگاشت کنیم. برای این کار از تساوی $f(x_i) = 1 \,/\, (c(x_i))$ استفاده نمایید.

a هزینه‌ی بهترین ذره و همچنین میانگین هزینه‌ی جمعیت را به‌عنوان تابعی از شماره‌ی نسل‌ها رسم کنید.

b ۵۰ شبیه‌سازی مونت کارلو از GA خود اجرا کنید. این کار ۵۰ نمودار برازندگی بهترین ذره بر حسب شماره‌ی نسل‌ها به دست خواهد داد. میانگین آن ۵۰ نمودار را رسم کنید. میانگین بهترین مقادیر برازندگی در نسل ۱۰۰ام را با $\bar{c}(x^*)$ نمایش دهید. $\bar{c}(x^*)$ چیست؟

c بخش b را با نرخ جهش ۲٪ تکرار کنید. $\bar{c}(x^*)$ نسبت به بخش b چه تغییری می‌کند؟ با نرخ جهش ۵٪ چطور؟

فصل چهارم

مدل‌های ریاضی برای الگوریتم‌های ژنتیک

مطالعه‌ی الگوریتم‌های تکاملی معمولاً به‌صورت تک‌کاره، شبیه‌سازی محور، ابتکاری و غیرتحلیلی صورت گرفته است. در طول تاریخ مهندسان بیشتر به کار کردن یا نکردن الگوریتم‌های تکاملی علاقه‌مند بوده‌اند تا به چگونگی یا چرایی کار کردن آن‌ها. با این حال، در دهه‌های پایانی قرن بیستم و با به بلوغ رسیدن تحقیقات در زمینه‌ی الگوریتم‌های تکاملی مهندسان تمرکز خود را به سؤالات "چرا" و "چگونه" معطوف ساختند. در این فصل در مورد راه‌های موجود برای پاسخ دادن به این گونه سؤال‌ها در مورد انواع GA بحث خواهیم نمود. این فصل تکنیکی‌ترین و ریاضی‌وارترین فصل در سراسر این کتاب است. دانشجویانی که تنها به دنبال دانش عملی الگوریتم‌های تکاملی هستند می‌توانند از این فصل صرف نظر کنند. با این حال، برای دانشجویانی که به دنبال آنند تا در زمینه‌ی تحقیقاتی الگوریتم‌های تکاملی اطلاعاتی جامع و کامل داشته باشند، فهم ایده‌های این فصل بسیار مهم است. دانشجویی که تلاش و وقت خود را صرف فهم این موضوعات می‌کند، افق‌های تحقیقاتی تازه و غیر منتظره‌ای را خواهد یافت.

مروری بر فصل

یکی از جواب‌های اولیه برای پاسخ‌گویی به سؤالات "چرا" و "چگونه" تئوری شما بود که رشد و زوال ترکیبات مختلف بیت را در انواع GA تحلیل می‌کرد. این موضوع در بخش ٤-١ مورد بررسی قرار گرفته است. برخی تحلیل‌های ریاضی از GAها بر مدل‌های مارکوف و مدل‌های سیستم پویا تکیه دارد که در این فصل مورد بحث قرار خواهند گرفت. این مدل‌ها کمبودهای خود را دارند. اما این کمبودها بیشتر مربوط به حوزه‌ی پیاده‌سازی و منابع محاسباتی می‌باشند تا حوزه‌ی نظری. بخش ٤-٢ مروری بر نظریه‌ی مارکوف خواهد داشت. این نظریه توسط ریاضی‌دان روس اندری[1] مارکوف در سال ١٩٠٦ به وجود آمد (سنتا[2]، ١٩٦٦). نظریه‌ی مارکوف به زمینه‌ای بنیادین در ریاضی تبدیل شده و در فیزیک، شیمی، علوم کامپیوتر، علوم اجتماعی، مهندسی، بیولوژی، موسیقی، ورزش و سایر رشته‌های جالب دیگر کاربرد دارد. در این فصل خواهیم دید که نظریه‌ی مارکوف می‌تواند بینشی در مورد رفتار GA به دست دهد. بخش ٤-٣ برخی نشان‌گذاری‌ها و نتایج مقدماتی را که در بخش‌های بعدی از آن‌ها استفاده خواهیم نمود ارائه می‌کند.

بخش ٤-٤ مدل زنجیرهای مارکوف را تشریح نموده که از انتخاب برازندگی- محور، به همراه جهش و برش تک- نقطه‌ای استفاده می‌نماید. متأسفانه، ابعاد مدل مارکوف با افزایش اندازه‌ی جمعیت و فضای جستجو به‌صورت فاکتوریلی (سریع‌تر از نمایی) رشد می‌کند. این مسئله باعث محدودیت کاربرد آن در مسائل بسیار

---

[1] Andrey
[2] Seneta

کوچک می‌شود. با این حال، مدل‌های مارکوف به دلیل به دست دادن نتایج دقیق بدون نیاز به تکیه بر ماهیت اتفاقی شبیه‌سازی‌های آماری همچنان مفید هستند.

بخش ٤-٥ مدلی سیستمی و پویا برای یک GA ایجاد می‌کند. مدل سیستمی پویا بر اساس مدل مارکوف می‌باشد اما کاربرد آن کاملاً متفاوت است. مدل مارکوف با میل شمار نسل‌ها به بی‌نهایت در هر جمعیت ممکن، مقدار احتمال حالت ماندگار را به دست می‌دهد. مدل سیستم پویا با میل اندازه‌ی جمعیت به سمت بی‌نهایت، نسبت زمانی هر ذره در فضای جستجو را به دست می‌دهد.

۴-۱ نظریه شِما

مسئله‌ی ساده‌ی $max_x f(x), f(x) = x^2$ را در نظر بگیرید. فرض کنید که x را به‌صورت یک عدد صحیح ۵ بیتی کد کرده‌ایم به‌صورتی که رشته بیت ۰۰۰۰۰ عدد ۰ و رشته بیت ۱۱۱۱۱ عدد ۳۱ را نشان می‌دهند. ماکزیمم $f(x)$ هنگامی رخ می‌دهد که $x = 11111$ باشد. به بیان دیگر هر رشته بیتی که با ۱ شروع شود از تمامی رشته بیت‌هایی که با ۰ شروع می‌شوند بهتر خواهد بود. این موضوع مفهوم شِما می‌باشد. یک شِما یک الگوی بیت است که یک مجموعه از بیت‌ها را توصیف می‌کند به‌طوری که از * برای نشان دادن بیت "بی‌اهمیت" استفاده می‌شود. برای مثال، رشته بیت‌های ۱۱۰۰۰ و ۱۰۰۱۱ هر دو به شِمای ****1 تعلق دارند. این شِما یک شِما با برازندگی بسیار بالا برای تابع x^2 به شمار می‌رود. هر رشته بیتی که به این شِما تعلق داشته باشد از همه‌ی رشته بیت‌هایی که به این شِما تعلق ندارند بهتر خواهد بود. GA الگوها را به‌گونه‌ای با هم ترکیب می‌کند که ذرات با برازندگی بسیار بالا حاصل شوند.

رشته بیت‌هایی با طول دو را در نظر بگیرید. شماتا (جمع چند شِما) با طول دو **، *0، *1، 0*، 1*، ۰۰، ۰۱، ۱۰ و ۱۱ می‌باشد. در مجموع ۹ شماتای یکتا با طول دو وجود خواهد داشت. به‌طورکلی برای طول l تعداد 3^l شماتا وجود دارد.

حال تعداد شماره‌ی شِماتایی را در نظر بگیرید که یک رشته بیت به آن تعلق دارد. برای مثال، توجه کنید که ۰۱ به چهار شِماتا تعلق دارد: ۰۱، 1*، *0 و **. به‌طورکلی رشته بیت با طول l بیت به 2^l شِما تعلق خواهد داشت. حال N رشته بیت که طول هر کدام l می‌باشد را در نظر بگیرید. هر رشته بیت از این جمعیت به مجموعه‌ی مشخصی از شِماتا تعلق خواهد داشت. می‌توان گفت که اتحاد این N مجموعه از شِماتا خود مجموعه‌ای تشکیل می‌دهند که کل جمعیت به آن تعلق دارند. اگر تمامی رشته بیت‌ها یکسان باشند آنگاه تمام جمعیت به 2^l شِماتا تعلق خواهد داشت. در حالت دیگر ممکن است تمامی رشته بیت‌ها منحصربه‌فرد بوده و به شِماتاهای یکسان تعلق نداشته باشند (به غیر از شِمای **.....**). در چنین حالتی کل جمعیت به

$N2^l - (N-1)$ شِماتا تعلق خواهد داشت. مشاهده می‌کنید که جمعیت Nتایی از رشته بیت‌ها به چیزی بین 2^l تا $N2^l - (N-1)$ شِماتا تعلق دارند.

تعداد بیت‌های تعریف شده در یک شِما، مرتبه‌ی o شما خوانده می‌شود. برای مثال، شِمای $o(1 *** 0) = 2$ و $o(0 * 11 *) = 3$.

به تعداد بیت‌ها از چپ‌ترین بیت تعریف شده تا راست‌ترین بیت تعریف شده فاصله‌ی تعریفِ δ می‌گویند. برای مثال، $\delta(1 *** 0) = 4$ و $\delta(0 * 11 *) = 3$ و $\delta(****1) = 0$.

یک رشته بیت که به یک شِما تعلق دارد یک نمونه از شِما نامیده می‌شود. برای مثال، شِمای *11*0 چهار نمونه دارد: ۰۰۱۱۰، ۰۰۱۱۱ و ۰۱۱۱۱. در کل، تعداد نمونه‌های یک شِما برابر 2^A می‌باشد که در آن A تعداد ستاره‌های شِما می‌باشد. توجه داشته باشید که $A = l - o$.

از عبارت $m(h,t)$ برای نشان دادن تعداد نمونه‌های شِمای h در نسل tام GA استفاده می‌شود. از $f(x)$ برای نشان دادن برازندگی رشته بیت x استفاده کرده و در نهایت از $f(h,t)$ برای نشان دادن مقدار برازندگی میانگین نمونه‌های شِمای h در نسل tام استفاده می‌شود:

$$f(h,t) = \frac{\sum_{x \in h} f(x)}{m(h,t)} \qquad (۱-٤)$$

ما از $\bar{f}(t)$ برای نشان دادن برازندگی میانگین کل جمعیت در نسل tام استفاده می‌کنیم. اگر از انتخاب چرخ رولت برای انتخاب والدین نسل بعد استفاده شود، خواهیم دید تعداد نمونه‌های hh مورد انتظار بعد از انتخاب برابر مقدار زیر خواهد بود:

$$E[m(h,t+1)] = \frac{\sum_{x \in h} f(x)}{f(t)} = \frac{f(h,t)m(h,t)}{f(t)} \qquad (۲-٤)$$

سپس برش را با احتمال p_c اعمال می‌کنیم. فرض می‌کنیم که نقطه‌ی برش هیچ‌گاه در انتهای رشته بیت واقع نشده و همواره میان بیت‌ها قرار می‌گیرد. از هر زوج از والدین دو فرزند به دست خواهد آمد. احتمال آنکه برش یک شِما را از بین ببرد چقدر خواهد بود؟ بیایید چند مثال را در نظر بگیریم.

- شِمای $h = 1 ****$ را در نظر بگیرید. برش هیچ‌گاه این شِما را از بین نخواهد برد. اگر نمونه‌ای از این شِما با یک رشته بیت برش کنند حداقل یک فرزند نمونه‌ای از این شِما خواهد بود.
- شِمای $h = 11 ***$ را در نظر بگیرید. اگر نمونه‌ای از این شِما با رشته بیت x برش انجام دهد، نقطه‌ی برش می‌تواند یکی از ٤ مکان ممکن باشد. اگر نقطه‌ی برش میان دو بیت با بیشترین ارزش

واقع شود، آنگاه با توجه به مقدار x ممکن است شِما از بین برود. با این حال، اگر نقطه‌ی برش در یکی از سه مکان دیگر قرار بگیرد، شِما هیچ‌گاه از بین نخواهد رفت چرا که حداقل یکی از فرزندان به شِمای h تعلق خواهد داشت. مشاهده می‌کنید که احتمال از بین رفتن شِما، با توجه به محل قرار گرفتن نقطه‌ی برش، کوچکتر یا مساوی $1/4$ است.

- شِمای $h = 1 * 1 **$ را در نظر بگیرید. اگر نمونه‌ای از این شِما با رشته بیت x برش انجام دهد، نقطه‌ی برش یکی از چهار مکان ممکن خواهد بود. اگر نقطه‌ی برش میان دو بیتِ ۱ قرار بگیرد (دو نقطه برش محتمل)، آنگاه با توجه به مقدار x ممکن است شِما از بین برود. از سوی دیگر، اگر نقطه‌ی برش در سمت راستِ راست‌ترین بیتِ ۱ قرار بگیرد (دو نقطه‌ی برش محتمل دیگر)، آنگاه شِما هرگز از بین نخواهد رفت چرا که حداقل یکی از فرزندان نمونه‌ای از h خواهد بود. همان‌طور که می‌بینید، احتمال از بین رفتن شِما بسته به آنکه نقطه‌ی برش در کجا قرار می‌گیرد، کوچکتر یا مساوی $1/2$ می‌باشد.

با تعمیم نتایج فوق می‌توان دید که احتمال آنکه یک شِما توسط برش از بین برود کوچکتر یا مساوی $\delta/(l-1)$ می‌باشد. احتمال وقوع برش خود برابر p_c می‌باشد بنابراین احتمال کلی از بین رفتن یک شِما توسط برش برابر $p_c\,\delta/(l-1)$ خواهد بود. بنابراین احتمال آنکه یک شِما از برش جان سالم به در ببرد برابر است با

$$p_s \geq 1 - p_c\left(\frac{\delta}{l-1}\right) \qquad (۳-٤)$$

سپس عمل جهش را با احتمال p_m در هر بیت، انجام می‌دهیم. تعداد بیت‌های تعریف شده (غیر ستاره) در h، مرتبه‌ی h نام داشته و با علامت $o(h)$ نشان داده می‌شود. احتمال جهش یک بیت تعریف شده برابر p_m و احتمال عدم جهش آن $1 - p_m$ می‌باشد. بنابراین، احتمال آنکه هیچ یک از بیت‌های تعریف شده دچار جهش نشوند نیز برابر $(1-p_m)^{o(h)}$ می‌باشد..

این احتمال به فرم $g(x) = (1-x)^y$ است. بسط تیلور $g(x)$ حول نقطه‌ی x_0 برابر است با

$$g(x) = \sum_{n=0}^{\infty} g^{(n)}(x_0)\frac{(x-x_0)^n}{n!} \qquad (٤-٤)$$

با قرار دادن $x_0 = 0$ خواهیم داشت

$$g(x) = \sum_{n=0}^{\infty} g^{(n)}(0)\frac{x^n}{n!}$$
$$= 1 - xy + \frac{x^2y(y-1)}{2!} - \frac{x^3y(y-1)(y-2)}{3!} + \cdots \qquad (٤-٥)$$
$$\approx 1 - xy$$

برای $xy \gg 1$

اگر $p_m o(h) \ll 1$، آنگاه $(1-p_m)^{o(h)} \approx 1 - {p_m}^{o(h)}$ خواهد بود. با ترکیب این معادله با معادلات (٤-٢) و (٤-٣) خواهیم داشت

$$E[m(h,t+1)] \geq \frac{f(h,t)m(h,t)}{\bar{f}(t)}\left(1 - p_c\left(\frac{\delta}{l-1}\right)\right)\left(1 - p_m o(h)\right)$$
$$\approx \frac{f(h,t)m(h,t)}{\bar{f}(t)}\left(1 - {p_c}\left(\frac{\delta}{l-1}\right) - {p_m} o(h)\right) \qquad (٤-٦)$$

فرض کنید یک شِما کوتاه است. بدین معنی که طول تعریف آن (δ) کوچک است. آنگاه $\delta/(l-1) \ll 1$ خواهد بود. حال فرض کنید که ما از یک نرخ جهش کوچک استفاده کرده و مرتبه‌ی شِما نیز کوچک است، بدین معنی که تعداد بیت‌های تعریف شده در آن اندک‌اند. در این حالت $p_m o(h) \ll 1$ خواهد بود. فرض کنید یک شِما برازندگی بالاتر از میانگین دارد، بدین معنی که $f(t)/\bar{f}(t) = k > 1$، که در آن k یک عدد ثابت است. در آخر، فرض کنید یک جمعیت بزرگ در اختیار داریم به‌طوری که $E[m(h,t+1)] \approx m(h,t+1)$. حال می‌توان به‌طور تقریبی نوشت

$$m(h,t+1) \geq km(h,t) = k^t m(h,0) \qquad (٤-٧)$$

این معادله قضیه‌ی ذیل را نتیجه خواهد داد که قضیه‌ی شِما نام دارد.

قضیه ٤-١ در یک جمعیت GA، تعداد نمایندگان شِماتای کوتاه، کم‌مرتبه و با مقادیر برازندگی بالاتر از میانگین، با سرعتی نمایی افزایش پیدا می‌کنند.

قضیه‌ی شِما اغلب به‌صورت معادله‌ی (٤-٦) یا (٤-٧) نوشته می‌شود.

نظریه‌ی شِما در دهه‌ی ۱۹۷۰ از افکار جان هالند سرچشمه گرفت (هالند، ۱۹۷۵) و به سرعت جای پای خود را در تحقیقات GA تثبیت نمود. نظریه‌ی شِما آن‌چنان در دهه‌ی ۱۹۸۰ غالب بود که اگر پیاده‌سازی‌هایی از GA فرضیات آن را نقض می‌کردند (برای مثال اگر از انتخاب رتبه-محور به‌جای انتخاب برازندگی-محور استفاده می‌کردند)، به آن‌ها شک می‌شد (وایتلی، ۱۹۸۹). توضیح چگونگی کارکرد نظریه‌ی شِما بر روی یک مثال ساده در (گلدبرگ، ۱۹۸۹ش، فصل ۲) فراهم شده است.

از سوی دیگر چند مثال نقض برای نظریه‌ی شِما در (ریوز و روو، ۲۰۰۳، بخش ۳-۲) آورده شده است. نتیجه‌ی آن‌ها آن است که نظریه‌ی شِما همیشه مفید نیست. علت آن به شرح زیر است:

- نظریه‌ی شِما برای زیرمجموعه‌ای دلخواه از فضای جستجو اعمال می‌شود. جدول ۳-۲ در مثال ۳-۲ را در نظر بگیرید. می‌بینیم که x_1 و x_4 هر دو به شِمای $h = 1 * 0 *$ تعلق دارند. اما این دو ذره، به ترتیب بیشترین و کمترین میزان برازندگی در جمعیت را دارند و به همین علت کار چندانی با یکدیگر ندارند، از طرفی هر دو اعضای h می‌باشند. بنابراین قضیه‌ی شِما اطلاعات مفیدی در مورد h به دست نمی‌دهد.
- نظریه‌ی شِما اینکه ممکن است برخی رشته بیت‌های مشابه به یک شِما تعلق نداشته باشند را نمی‌تواند تشخیص دهد. در مثال ۳-۲، می‌بینیم که ۰۱۱۱ و ۱۰۰۰ در فضای جستجو همسایه هستند اما به هیچ شِمای مشترکی تعقل ندارند مگر شِمای جهانی ****. این مشکل را می‌توان با کدگذاری گری تعدیل نمود اما حتی در آن صورت هم، بسته به فضای جستجو ممکن است همسایه‌های فضای جستجو رابطه‌ی نزدیکی در فضای برازندگی نداشته باشند.
- نظریه‌ی شِما تعداد نمونه‌های شِما که از یک نسل به نسل دیگر زنده می‌مانند را به دست می‌دهد اما مهم‌تر از این، آن است که کدام نمونه‌ها جان سالم به در می‌برند. بار دیگر به مثال ۳-۲ نگاه کنید. می‌بینید که x_1 و x_4 هر دو به شِمای $h = 1 * 0 *$ تعلق دارند. نظریه‌ی شِما به ما می‌گوید آیا h از یک نسل به نسل دیگر از بین می‌رود یا خیر، اما ما بیشتر به زنده ماندن یا نماندن x_1 و x_4 علاقه‌مندیم.
- نظریه‌ی شِما تعداد *مورد انتظار* نمونه‌های شِما را به دست می‌دهد. اما طبیعت آماری GA باعث می‌شود هر بار که GA اجرا می‌شود رفتارهای متفاوتی بروز کنند. تعداد نمونه‌های مورد انتظار یک شِما تنها هنگامی با تعداد واقعی نمونه‌های یک شِما برابر است که اندازه‌ی جمعیت به سمت بی‌نهایت میل کند.
- هیچ شِمایی نمی‌تواند به‌صورت هم‌زمان هم برازندگی بالاتر از میانگین داشته باشد و هم به‌صورت نمایی افزایش یابد. اگر یک شِما به‌صورت نمایی رشد کند آنگاه به‌زودی بر جمعیت غالب شده و در این هنگام برازندگی میانگین جمعیت تقریباً با برازندگی شِما برابر خواهد شد. در نتیجه تقریب $f(t)/\bar{f}(t) = k$ که در پاراگراف قبل از قضیه‌ی ٤-١ آورده شده بود، نادرست خواهد بود. مورد دیگری که در ارتباط با این ایده وجود دارد آن است که اکثر GAها با جمعیتی کوچکتر یا مساوی ۱۰۰ عمل می‌کنند. چنین جمعیت‌های کوچکی نمی‌توانند بیشتر از سه یا چهار نسل از رشد نمایی شِما پشتیبانی کنند.

تا دهه‌ی ۱۹۹۰، تأکیدات بیش از اندازه بر کاستی‌های نظریه‌ی شِما باعث شد تا بیانیه‌های تندوتیزی مانند "به نظر من، دیگر جای بحثی نیست.قضیه‌ی شِما هیچ چیزی را در مورد الگوریتم ژنتیک ساده (SGA)[1] توضیح نمی‌دهد." (وُز[2]، ۱۹۹۹، صفحه‌ی xi) صادر شوند. این مغایرت و ناسازگاری در مورد بسیاری از نظریه‌های جدید امری معمول است. نظریه‌ی شِما درست است اما محدودیت‌هایی دارد. یک دیدگاه متعادل از فواید و کاستی‌های نظریه‌ی شِما در (ریوز و روو، ۲۰۰۳، فصل ۳) آورده شده است.

۴-۲ زنجیره‌ی مارکوف

فرض کنید که یک سیستم زمان-گسسته در اختیار داریم و می‌توانیم آن را با مجموعه‌ای از حالت‌های گسسته $s = \{s_1, \ldots., s_2\}$ توصیف کنیم. برای مثال، هوا را می‌توان با مجموعه‌ای از $s = \{$برفی ,خوب ,بارانی$\}$ توصیف نمود. از علامت $S(t)$ برای نشان دادن حالت در زمان t استفاده می‌کنیم. حالت آغازین $S(0)$ است، حالت زمان بعدی $S(1)$ است و به همین ترتیب. حالت سیستم ممکن است از یک زمان به زمان دیگر تغییر کرده و یا بدون تغییر باقی بماند. فرایند انتقال از یک حالت به حالت دیگر یک فرایند کاملاً احتمالی است. در فرایند مارکوف مرتبه‌ی اول، که به آن زنجیره‌ی مارکوف مرتبه اول نیز می‌گویند، احتمال انتقال حالت سیستم به هر حالت ممکن در زمان بعدی، تنها به حالت جاری بستگی دارد یا به عبارت دیگر این احتمال از تمام حالت‌های گذشته مستقل است. احتمال گذر سیستم از حالت i به حالت j از یک زمان به زمان بعدی را با p_{ij} نشان می‌دهند. بنابراین برای تمام iها داریم

$$\sum_{j=1}^{n} p_{ij} = 1 \qquad (۸-٤)$$

با این کار یک ماتریس $n \times n$ تشکیل شده که آن را با P نمایش می‌دهیم. در این ماتریس p_{ij} درایه‌ای است که در ردیف iاُم و ستون jاُم قرار گرفته است. به P ماتریس انتقال یا ماتریس احتمال یا ماتریس آماری فرایند مارکوف می‌گویند.

مثال ٤-۱

سرزمین اُز هیچ‌گاه دو روز خوب متوالی را پشت سر نگذاشته است (کِمنی[3] و همکاران، ۱۹۷٤). اگر امروز روز خوبی است بنابراین فردا به احتمال ۵۰ درصد بارانی و یا به احتمال ۲۵ درصد برفی خواهد بود.

[1] Simple Genetic Algorithm
[2] Vose
[3] Kemeny

اگر امروز بارانی است بنابراین روز بعد به احتمال ۵۰ درصد بارانی خواهد بود و یا به احتمال ۲۵ درصد برفی خواهد بود و یا اینکه به احتمال ۲۵ درصد فردا هوا خوب خواهد بود. می‌بینیم که پیش‌بینی هوا برای یک روز مشخص تنها به هوای روز قبل از آن بستگی دارد. اگر به ترتیب هوای بارانی، خوب و برفی را با حروف R، N و S نمایش دهیم، آنگاه می‌توانیم ماتریس مارکوف را که احتمالات آب‌وهواهای گوناگون را به نمایش می‌گذارد را تشکیل دهیم

$$P = \begin{matrix} R \\ N \\ S \end{matrix} \begin{bmatrix} \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \end{bmatrix} \qquad (۹-٤)$$

فرض کنید یک فرایند مارکوف با حالت i و در زمان ۰ آغاز می‌شود. همان‌طور که گفته شد، احتمال قرار گرفتن فرایند در حالت j در زمان ۱ با $\Pr(S(1) = S_j | S(0) = S_i) = p_{ij}$ به دست خواهد آمد. حال زمان بعدی را در نظر می‌گیریم. می‌توانیم از قضیه‌ی احتمال کل برای محاسبه‌ی احتمال قرار گرفتن فرایند در حالت ۱ در زمان ۲ استفاده کنیم (میتزِنماخر[1] و اوپفال[2]، ۲۰۰۵)

$$\begin{aligned} \Pr(S(2) = S_1 | S(0) = S_i) = {} & \Pr(S(1) = S_1 | S(0) = S_i)\, p_{11} + \\ & \Pr(S(1) = S_2 | S(0) = S_i)\, p_{21} + \cdots + \\ & \Pr(S(1) = S_n | S(0) = S_i)\, p_{n1} \\ = {} & \sum_{k=1}^{n} \Pr(S(1) = S_k | S(0) = S_i)\, p_{k1} \qquad (۱۰-٤) \\ = {} & \sum_{k=1}^{n} p_{ik} p_{k1} \end{aligned}$$

با تعمیم روابط بالا، احتمال آنکه فرایند در زمان ۲ در حالت j قرار گیرد از رابطه‌ی زیر به دست می‌آید

$$\Pr(\mathrm{S}(2) = S_j | S(0) = S_i) = \sum_{k=1}^{n} p_{ik} p_{kj} \qquad (۱۱-٤)$$

[1] Mitzenmacher
[2] Upfal

اما این مقدار برابر است با عنصر سطر iاُم و ستون jاُم ماتریس P که برابر است با

$$\Pr\big(s(2) = S_j \big| S(0) = S_i\big) = [P^2]_{ij} \qquad (۱۲-٤)$$

با ادامه‌ی این منطق به روش استقرایی خواهیم داشت

$$\Pr\big(\mathrm{S(t)} = S_j \big| S(0) = S_i\big) = [P^t]_{ij} \qquad (۱۳-٤)$$

که یعنی احتمال انتقال فرایند مارکوف از حالت i به حالت j بعد از زمان tاُم برابر است با عنصر سطر iاُم و ستون jاُم در ماتریس P^t.

در مثال ٤-۱ می‌توانیم ماتریس P^t را برای مقادیر مختلف t محاسبه کنیم. در این صورت خواهیم داشت

$$\begin{aligned}
P &= \begin{vmatrix} 0.5000 & 0.2500 & 0.2500 \\ 0.5000 & 0.0000 & 0.5000 \\ 0.2500 & 0.2500 & 0.5000 \end{vmatrix} \\
P^2 &= \begin{vmatrix} 0.4375 & 0.1875 & 0.3750 \\ 0.3750 & 0.2500 & 0.3750 \\ 0.3750 & 0.1875 & 0.4375 \end{vmatrix} \\
P^4 &= \begin{vmatrix} 0.4023 & 0.1992 & 0.3984 \\ 0.3984 & 0.2031 & 0.3984 \\ 0.3984 & 0.1992 & 0.4023 \end{vmatrix} \\
P^8 &= \begin{vmatrix} 0.4000 & 0.2000 & 0.4000 \\ 0.4000 & 0.2000 & 0.4000 \\ 0.400 & 0.2000 & 0.4000 \end{vmatrix}
\end{aligned} \qquad (۱٤-٤)$$

به طرز جالبی، هنگامی که $t \to \infty$ ماتریس P^t به سمت ماتریسی با سطرهای یکسان همگرا می‌شود. این موضوع برای تمام ماتریس‌های انتقال اتفاق نمی‌افتد اما، همان‌طور که در قضیه‌ی زیر بیان شده، برای زیرمجموعه‌ی معینی از آن‌ها صادق است.

قضیه ٤-۲ یک ماتریس انتقال معمولی P، که با عنوان ماتریس انتقال اولیه نیز شناخته می‌شود، ماتریسی است که برای آن تمام عنصرهای P^t برای برخی tها غیرصفر هستند. اگر P یک ماتریس انتقال معمولی باشد، خواهیم داشت

۱. $lim_{t\to\infty} P^t = P_\infty$
۲. تمام سطرهای P_∞ یکسان بوده و با p_{ss} نشان داده می‌شوند.
۳. تمامی عناصر p_{ss} مثبت می‌باشند.
٤. احتمال قرار گرفتن فرایند مارکوف در حالت iاُم بعد از بی‌نهایت بار انتقال برابر است با عنصر iاُم در p_{ss}.

۵. p_{ss}^{t} بردار ویژه‌ی P^T متناظر با مقدار ویژه‌ی ۱ می‌باشد. این مقدار نرمالیزه شده است تا جمع عناصر آن برابر ۱ شود.

٦. اگر ماتریس‌های P_i، $i \epsilon [1,n]$، را با قرار دادن سطر iاُم P با صفر تشکیل دهیم، آنگاه iاُمین عنصر p_{ss} توسط رابطه‌ی زیر به دست می‌آید

$$p_{ss,i} = \frac{|P_i - I|}{\sum_{j=1}^{n} |P_j - I|} \qquad (١٥-٤)$$

که در آن I یک ماتریس واحد $n \times n$ است و $|.|$ اپراتور دترمینان می‌باشد.

اثبات: ۵ ویژگی اول شامل قضیه‌ی اساسی حد برای زنجیره‌ی مارکوف معمولی می‌شوند و در (گریناستید[1] و اسنل[2]، ۱۹۹۷، فصل ۱۱) و دیگر کتاب‌هایی که در مورد زنجیره‌ی مارکوف می‌باشند اثبات شده‌اند. برای اطلاعات بیشتر در مورد دترمینان، مقادیر ویژه و بردارهای ویژه می‌توانید به هر کتابی که در زمینه‌ی سیستم‌های خطی نگارش شده‌اند مراجعه کنید (سایمون[3]، ۲۰۰٦، فصل ۱). آخرین ویژگی قضیه‌ی ٤-۲ در (دیویس[4] و پرینسیپ[5]، ۱۹۹۳) اثبات شده است.

مثال ٤-۲

با اعمال قضیه‌ی ٤-۲ به مثال ٤-۱ و با استفاده از معادله‌ی ٤-١٤ می‌بینیم که هر روزی در آینده‌ی دور، به احتمال ٤۰ درصد بارانی، ۲۰ درصد آفتابی و ٤۰ درصد برفی خواهد بود. بنابراین، ٤۰ درصد از روزها در سرزمین اُز بارانی، ۲۰ درصدشان آفتابی و ٤۰ درصد از آن‌ها برفی خواهد بود. علاوه بر این، با محاسبه‌ی مقادیر ویژه‌ی P^T به اعداد ۱، ۰,۲۵، و 0.25- می‌رسیم. بردار ویژه‌ی متناظر با مقدار ویژه‌ی ۱ برابر $[0.4\ 0.2\ 0.4]^T$ می‌باشد.

حال فرض کنید در فرایند مارکوف حالت آغازین نامعلوم است و به‌جای آن احتمال هر حالت مشخص است. احتمال آنکه حالت آغازین $S(0)$ برابر S_k باشد توسط $p_k(0)$، $k \epsilon [1,n]$، نمایش داده می‌شود. حال می‌توانیم از قضیه‌ی احتمال کل (میتزِنماخر، و اوپفال، ۲۰۰۵) استفاده کنیم که در این صورت خواهیم داشت

$$\begin{aligned} \Pr(S(1) = S_i) = {} & \Pr(S(0) = S_1)\, p_{1i} + \Pr(S(0) = S_2)\, p_{2i} + \cdots \\ & + \Pr(S(0) = S_n)\, p_{ni} \end{aligned} \qquad (١٦-٤)$$

[1] Grinstead
[2] Snell
[3] Simon
[4] Davis
[5] Principe

$$= \sum_{k=1}^{n} \Pr(S(0) = S_k)\, p_{ki}$$
$$= \sum_{k=1}^{n} p_{ki} p_k(0)$$

با تعمیم رابطه‌ی فوق خواهیم داشت

$$\begin{vmatrix} \Pr(S(1) = S_1) \\ \dots \\ \Pr(S(1) = S_n) \end{vmatrix}^T = p^T(0)P \qquad (۱۷-٤)$$

که در آن $p(0)$ یک بردار ستونی شامل $p_k(0)$ می‌باشد. با تعمیم این نتیجه برای چند مرحله‌ی زمانی به رابطه‌ی زیر خواهیم رسید

$$p^T(t) = \begin{vmatrix} \Pr(S(t) = S_1) \\ \dots \\ \Pr(S(t) = S_n) \end{vmatrix}^T = p^T(0)P^t \qquad (۱۸-٤)$$

مثال ٤-۳

پیش‌بینی هوای امروز برای سرزمین اُز ۸۰ درصد هوای آفتابی و ۲۰ درصد هوای برفی می‌باشد. پیش‌بینی هوا برای دو روز دیگر چیست؟

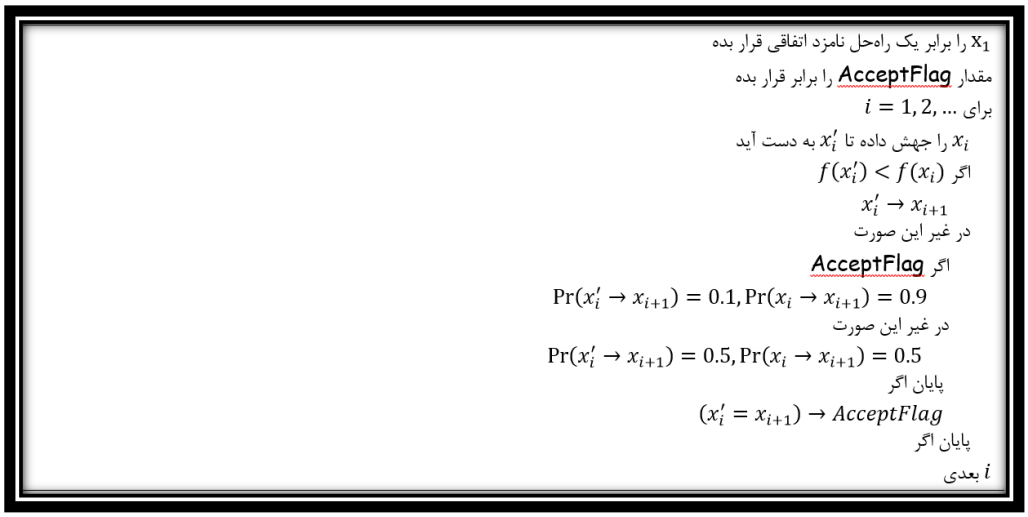
از معادله‌ی ٤-۱۸ داریم: $p^T(2) = p^T(0)P^2$ که در آن P از مثال ٤-۱ گرفته شده است و بدین ترتیب $p(0) = [0.0\ 0.8\ 0.2\ ]^T$ خواهد بود. بدین ترتیب خواهیم داشت: $p(2) = [0.375\ 0.2375\ 0.3875]^T$. یعنی دو روز دیگر به احتمال ۳۷,۵ درصد بارانی و به احتمال ۲۳,۷۵ درصد آفتابی و به احتمال ۳۸,۷۵ درصد برفی خواهد بود.

مثال ٤-٤

یک مثال ساده‌ی تپه‌نوردی که شامل تنها یک ذره است را در نظر بگیرید (ریوز و رو، ۲۰۰۳، صفحه‌ی ۱۱۲). هدف الگوریتم تکاملی مینیمم نمودن $f(x)$ است. ما از علامت x_i برای نشان دادن راه‌حل نامزد در نسل iاُم استفاده می‌کنیم. در هر نسل x_i به‌صورت اتفاقی دچار جهش شده تا x_i' به دست آید. اگر $f(x_i') < f(x_i)$ باشد آنگاه x_{i+1} را برابر x_i' قرار می‌دهیم یعنی: $x_{i+1} = x_i'$.

اگر $f(x_i') > f(x_i)$، آنگاه از منطق زیر برای تعیین x_{i+1} استفاده می‌کنیم. اگر در مرحله‌ی قبل که در آن $f(x_k') > f(x_k)$ بوده، x_{k+1} را برابر x_k' قرار داده باشیم، آنگاه با احتمال ۱۰٪ $x_{i+1} = x_i'$ و با احتمال ۹۰٪ $x_{i+1} = x_i$ خواهد بود. با این حال اگر در دوره‌ی قبلی k، x_{k+1} را برابر x_k قرار داده باشیم آنگاه با احتمال

۵۰٪ $x_{i+1} = x'_i$ و با احتمال ۵۰٪ $x_{i+1} = x_i$ خواهد بود. این الگوریتم تکاملی اصطلاحاً حریص[1] است چرا که همواره جهش‌های سودمند را قبول می‌کند. با این حال، این الگوریتم گاه جهش‌های مخرب را هم پذیرفته و از این رو شامل مقداری اکتشاف نیز می‌باشد. احتمال آنکه یک جهش مخرب مورد قبول واقع شود به این بستگی دارد که آیا جهش مخرب قبلی مورد قبول واقع شده است یا خیر. شبه کد این الگوریتم تکاملی تپه‌نوردی در شکل ٤-١ نشان داده شده است.



شکل ٤-١ شبه کد بالا الگوریتم تکاملی تک‌ذره‌ای تپه نوردی به کار رفته در مثال ٤-٤ را نشان می‌دهد. مقدار *AcceptFlag* اینکه آیا جهش مخرب قبلی جایگزین راه‌حل نامزد شده است یا خیر، را نشان می‌دهد.

می‌توان با فرض برتر بودن x'_i از x_i این الگوریتم تکاملی را تحلیل کرد. از Z_k برای نشان دادن حالت در k اُمین زمان، که در آن $f(x'_i) > f(x_i)$ است، استفاده می‌کنیم. Y_1 را به‌عنوان حالت "پذیرش"، که به معنی $x_{i+1} \leftarrow x'_i$ است، تعریف کرده و Y_2 را حالت "عدم پذیرش"، که در آن $x_{i+1} \leftarrow x_i$، تعریف می‌کنیم. حال با استفاده از الگوریتم شکل ٤-١ می‌توانیم بنویسیم:

$$\begin{aligned}
&\Pr(Z_k = Y_1 | Z_{k-1} = Y_1) = 0.1 \\
&\Pr(Z_k = Y_2 | Z_{k-1} = Y_1) = 0.9 \\
&\Pr(Z_k = Y_1 | Z_{k-1} = Y_2) = 0.5 \\
&\Pr(Z_k = Y_2 | Z_{k-1} = Y_2) = 0.5
\end{aligned} \qquad (١٩-٤)$$

این معادلات نشان‌دهنده‌ی ماتریس انتقال به شکل زیر خواهد بود

[1] Greedy

$$P = \begin{bmatrix} 0.1 & 0.9 \\ 0.5 & 0.5 \end{bmatrix} \qquad (۲۰-٤)$$

توجه داشته باشید که جمع درایه‌های هر سطر p برابر ۱ است. همچنین می‌توان دید که تمام درایه‌های p^t برای برخی از tها (در این مثال تمام tها) غیرصفراند، بنابراین p یک ماتریس انتقال معمولی است. قضیه‌ی ۲-٤ ما را خاطرنشان می‌سازد که: (۱) p^t با میل t به سمت بی‌نهایت همگرا می‌شود، (۲) تمام ردیف‌های P^∞ همسان هستند، (۳) تمام درایه‌های P^∞ مثبت است، (٤) احتمال قرار گرفتن فرایند مارکوف در حالت Y_i بعد از بی‌نهایت بار انتقال با درایه iاُم هر سطر P^∞ برابر است و (۵) هر سطر P^∞ با ترانهاده‌ی بردار ویژه‌ی متناظر با مقدار ویژه‌ی ۱ در P^T برابر است.

برای پیدا کردن P^∞ از محاسبات عددی استفاده کرده و در این صورت خواهیم داشت

$$P^\infty = \frac{1}{14}\begin{bmatrix} 5 & 9 \\ 5 & 9 \end{bmatrix} \qquad (۲۱-٤)$$

بدین ترتیب مقادیر ویژه‌ی P^T برابر *0.4*- و ۱ خواهد بود و بردار ویژه‌ی متناظر با مقدار ویژه‌ی ۱، $\begin{bmatrix} \frac{5}{14} & \frac{9}{14} \end{bmatrix}$ می‌باشد. این نتایج در واقع حاکی از آنند که در طولانی مدت، نسبت "پذیرش" به "عدم پذیرش" جهش‌ها برابر $\frac{5}{9}$ خواهد بود.

۴-۳ نشان‌گذاری مدل مارکوف برای الگوریتم‌های تکاملی

در این بخش، نشان‌گذاری‌هایی را تعریف خواهیم کرد که بعداً برای به دست آوردن مدل مارکوف و مدل سیستم پویا برای الگوریتم‌های تکاملی از آن‌ها استفاده خواهد شد. مدل‌های مارکوف ابزاری ارزشمند برای تحلیل الگوریتم‌های تکاملی هستند چرا که نتایج دقیقی را به دست می‌دهند. می‌توان برای بررسی کارایی الگوریتم‌های تکاملی از شبیه‌سازی استفاده نمود اما این شبیه‌سازی‌ها می‌توانند گمراه‌کننده باشند. برای مثال، ممکن است یک مجموعه از شبیه‌سازی‌های مونت‌کارلو به دلیل یک ترتیب مشخص از اعداد اتفاقیِ تولید شده در طول شبیه‌سازی نتایجی گمراه‌کننده به دست دهند. همچنین، تولیدکننده‌ی اعداد اتفاقی استفاده شده در شبیه‌سازی الگوریتم‌های تکاملی ممکن است ناصحیح بوده و به همین دلیل نتایج گمراه‌کننده‌ای را به دنبال داشته باشند. این موضوع بیش از آنکه فکرش را بکنید اتفاق می‌افتد [ساویکی[1] و روبنیک-سیکونجا[2]، ۲۰۰۸]. در آخر اینکه ممکن است تعداد شبیه‌سازی‌های مونت‌کارلوی مورد نیاز برای تخمین خروجی‌های غیر محتمل آنقدر زیاد باشد که نتوان در یک زمان منطقی به آن‌ها دست پیدا نمود. نتایج مدل مارکوف که به دست

[1] Savicky
[2] Robnik-Sikonja

خواهیم آورد هیچ یک از این نواقص را نداشته و نتایج دقیقی را به دست خواهند داد. تنها ایراد مدل‌های مارکوف نیاز به محاسبات زیاد برای پیاده‌سازی آن‌ها است.

ابتدا توجه خود را به الگوریتم‌های تکاملی با اندازه جمعیت N در یک فضای جستجوی گسسته با کاردینالیتی n معطوف می‌کنیم. فرض می‌کنیم فضای جستجو از تمام رشته بیت‌های $q-$ بیتی تشکیل شده باشد به‌طوری که $n = 2^q$. از علامت x_i برای نشان دادن iاُمین رشته بیت در فضای جستجو استفاده می‌کنیم. از علامت v نیز برای نشان دادن بردار جمعیت استفاده می‌کنیم. در این حالت v_i تعداد ذرات x_i در جمعیت را نشان خواهد داد. واضح است که

$$\sum_{i=1}^{n} v_i = N \qquad (۲۲-٤)$$

این معادله‌ی ساده بیانگر آن است که تعداد کل ذرات موجود در جمعیت برابر N است. از علامت y_k برای نشان دادن kاُمین ذره در جمعیت استفاده می‌کنیم. جمعیت Y از الگوریتم تکاملی را می‌توان به‌صورت زیر نشان داد

$$\begin{aligned} Y &= \{y_1, \dots, y_N\} \\ &= \{\underset{\Downarrow}{x_1}, \underset{\Downarrow}{x_1}, \dots, x_1, \underset{\Downarrow}{x_2}, \underset{\Downarrow}{x_2}, \dots, x_2, \dots, \underset{\Downarrow}{x_n}, \underset{\Downarrow}{x_n}, \dots, x_n\} \\ &\qquad v_1 \text{ بار} \qquad\quad v_2 \text{ بار} \qquad\quad v_n \text{ بار} \end{aligned} \qquad (۲۳-٤)$$

که در آن y_iها هر یک، یک گروه از ذرات همسان را تشکیل داده‌اند. از حرف T برای نشان دادن تعداد جمعیت‌های ممکنِ Y استفاده می‌کنیم. به عبارت دیگر، T تعداد بردارهای صحیح $n \times 1$ است به‌طوری که $\sum_{i=1}^{n} v_i = N$ و $v_i \in [0, N]$.

مثال ٤-٥

فرض کنید $N = 2$ و $n = 4$. یعنی فضای جستجو از رشته بیت‌های $\{00, 01, 10, 11\}$ تشکیل شده و کلاً دو ذره در الگوریتم تکاملی وجود دارند. ذرات فضای جستجو عبارت‌اند از

$$\begin{aligned} x_1 &= 00, \qquad x_2 = 01 \\ x_3 &= 10, \qquad x_4 = 11 \end{aligned} \qquad (۲٤-٤)$$

جمعیت‌های ممکن شامل موارد زیرند:

$$\begin{aligned} &\{00, 00\} \qquad \{00, 01\} \\ &\{00, 10\} \qquad \{00, 11\} \end{aligned} \qquad (۲۵-٤)$$

$\{10, 11\}$ $\{11, 11\}$

مشاهده می‌شود که برای این مثال $T = 10$ می‌باشد.

حال این سؤال پیش می‌آید: برای یک الگوریتم تکاملی با جمعیت N و کاردینالیتی فضای جستجوی n، چند جمعیت ممکن وجود دارد؟ می‌توان نشان داد که [نیکس و وُز، ۱۹۹۲]، T از ضریب دوجمله‌ای زیر، که تابع انتخاب نیز نامیده می‌شود، به دست می‌آید:

$$T = \binom{n + N - 1}{N} \qquad (۲۶-٤)$$

همچنین می‌توان از قضیه‌ی چندجمله‌ای [چوان–چونگ[1] و خی–مِنگ[2]]، [سایمون و همکاران *2011a*] برای پیدا کردن T استفاده نمود. قضیه‌ی چندجمله‌ای را می‌توان به چند طریق بیان نمود که یکی از آن‌ها بدین شرح است: اگر k نوع از اشیا موجود باشد، تعداد راه‌های مختلف برای انتخاب N شی، به‌طوری که از هیچ نوع شی بیش از M بار انتخاب نکنیم و ترتیب آن‌ها نیز بی‌اهمیت باشد، برابر ضریب qN در چندجمله‌ای زیر است:

$$q(x) = (1 + x + x^2 + \cdots + x^M)^K \\ = 1 + q_1 x + q_2 x^2 + \cdots + q_N x^N + \cdots + x^{MK} \qquad (۲۷-٤)$$

بردار جمعیت الگوریتم تکاملی ما یک بردار n عنصری است که هر یک از عناصر آن یک عدد صحیح بین N و ۰ بوده و جمع عناصر آن برابر N است. T تعداد بردارهای جمعیت یکتای v است. بنابراین، T تعداد راه‌های ممکن برای انتخاب N شی از n نوع شی است به‌طوری که ترتیب بی‌اهمیت بوده و از هیچ نوع شی بیش از N بار انتخاب نشود. با اعمال قضیه‌ی چندجمله‌ای ٤–۲۷ به این مسئله خواهیم داشت:

$$T = q_N \\ q(x) = (1 + x + x^2 + \cdots + x^N)^n \\ = 1 + q_1 x + q_2 x^2 + \cdots + q_N x^N + \cdots + x^{Nn} \qquad (۲۸-٤)$$

برای پیدا کردن T می‌توان از شکل‌های دیگر قضیه‌ی چندجمله‌ای نیز استفاده نمود [چوان– چونگ و خی– منگ، ۱۹۹۲]، [سایمون و همکاران، *2011a*]. قضیه‌ی چندجمله‌ای را می‌توان به شکل زیر بیان نمود

$$(x_1 + x_2 + \cdots + x_N)^n = \sum_{S(k)} \frac{n!}{\prod_{j=0}^{N} k_j!} \prod_{j=0}^{N} x_j^{k_j} \\ \sum_{S(k)} \prod_{i=0}^{N} \binom{\sum_{j=0}^{i} k_j}{k_i} \prod_{j=0}^{N} x_j^{k_j} \qquad (۲۹-٤)$$

[1] Chuan-Chong
[2] Khee-Meng

که در آن

$$S(k) = \{k \in \mathbb{R}^N : k_j \in \{0, 1, \dots, n\}, \sum_{j=0}^{N} k_j = n\}$$

حال چندجمله‌ای $(x^0 + x^1 + x^2 + \cdots + x^N)^n$ را در نظر بگیرید. از قضیه‌ی چندجمله‌ای معادله‌ی (٤–٢٩) درمی‌یابیم که ضریب $[(x^0)^{k_0}(x^1)^{k_1} \dots (x^N)^{k_N}]$ توسط معادله‌ی زیر تعیین می‌شود

$$\prod_{i=0}^{N} \binom{\sum_{j=0}^{i} k_j}{k_i} \qquad (٤–٣٠)$$

اگر این عبارات را برای تمامی k_jها اضافه کنیم به‌طوری که

$$\sum_{j=0}^{N} j k_j = N \qquad (٤–٣١)$$

آنگاه ضریب x^N را به دست خواهیم آورد. اما معادله‌ی (٤–٢٨) نشان می‌دهد که T برابر ضریب x^N است. بنابراین داریم:

$$T = \sum_{S'(k)} \prod_{i=0}^{N} \binom{\sum_{j=0}^{i} k_j}{k_i} \qquad (٤–٣٢)$$

که در آن

$$S'(k) = \{k \in \mathbb{R}^{N+1} : k_j \in \{0, 1, \dots, n\}, \sum_{j=0}^{N} k_j = n, \sum_{j=0}^{N} j k_j = n\}$$

معادلات (٤–٢٦) و (٤–٢٨) و (٤–٣٢) عبارات یکسانی را برای T به دست می‌دهند.

مثال ٤–٦

این مثال از [سایمون و همکاران، *2011a*] گرفته شده است. فرض کنید یک فضای جستجوی ۲– بیتی داریم ($q = 2$ و $n = 4$) و اندازه‌ی جمعیت الگوریتم تکاملی برابر ٤ است. بنابر معادله‌ی (٤–٢٤) داریم

$$T = \binom{7}{4} = 35 \qquad (٤–٣٣)$$

از معادله‌ی (٤–٢٨) داریم

$$q(x) = (1 + x + x^2 + x^3 + x^4)^4 \\ = 1 + \cdots + 35x^4 + \cdots + x^{16} \qquad (۳٤-٤)$$

که یعنی $T = 35$. از معادله‌ی (٤-۳۲) خواهیم داشت:

$$T = \sum_{S'(k)} \prod_{i=0}^{4} \binom{\sum_{j=0}^{i} k_j}{k_i}$$

که در آن

$$S'(k) = \left\{k \in \mathbb{R}^5 : \; k_j \in \{0, 1, \dots, 4\}, \sum_{j=0}^{4} k_j = 4 \, , \sum_{j=0}^{4} j k_j = 4\right\} \qquad (۳٥-٤)$$
$$= \{(3\,0\,0\,0\,1), (2\,1\,0\,1\,0), (2\,0\,2\,0\,0), (1\,2\,1\,0\,0), (0\,4\,0\,0\,0)\}$$

این یعنی $T = 4 + 12 + 6 + 12 + 1 = 35$. مشاهده می‌شود که هر سه روش برای محاسبه‌ی T نتایج یکسانی را به دست می‌دهند.

۴-۴ مدل مارکوف الگوریتم‌های ژنتیک

استفاده از مدل‌های مارکوف برای مدل‌سازی *GA*ها برای اولین بار در [نیکس و وُز، ۱۹۹۲] و [دیویس و پرینسیپ، ۱۹۹۱] صورت پذیرفت و توضیحات تکمیلی آن در [ریوز و روو، ۲۰۰۳] و [وُز، ۱۹۹۹] ارایه گشت. همان‌طور که در فصل ۳ مشاهده کردیم، یک *GA* از سه فرایند انتخاب، برش و جهش تشکیل شده است. برای مدل‌سازی مارکوف جای دو فرایند جهش و برش را عوض می‌کنیم، بدین معنا که در این قسمت *GA* را متشکل از سه فرایند انتخاب، جهش و برش در نظر می‌گیریم (به ترتیب فرایندها دقت کنید.)

۴-۴-۱ انتخاب

ابتدا انتخاب متناسب با برازندگی (که همان انتخاب چرخ رولت است) را در نظر می‌گیریم. احتمال انتخاب ذره‌ی x_i با یک بار چرخش چرخ رولت با حاصلضرب برازندگی x_i در تعداد ذرات x_i موجود در جمعیت متناسب است. این احتمال نرمالیزه می‌شود تا جمع تمامی احتمالات برابر ۱ شود. با توجه به تعریفی که در قسمت قبل ارایه گردید، v_i تعداد ذرات x_i موجود در جمعیت است. بنابراین، احتمال انتخاب ذره‌ی x_i با یک بار چرخش چرخ رولت عبارتست از

$$P_s(x_i|v) = \frac{v_i f_i}{\sum_{j=1}^{n} v_j f_j} \qquad (۳٦-٤)$$

که در آن $i \in [0, N]$ و n کاردینالیتی فضای جستجو بوده و f_j میزان برازندگی x_j می‌باشد. از علامت $P_s(x_i|v)$ برای نشان دادن این حقیقت استفاده کردیم که احتمال انتخاب x_i به بردار جمعیت v بستگی دارد. حال فرض کنید اندازه‌ی جمعیت برابر N بوده و ما چرخ رولت را N بار می‌چرخانیم تا N والد حاصل شوند. هر بار چرخش چرخ رولت n خروجی محتمل دارد $\{x_1, \dots, x_n\}$. احتمال ظاهر شدن x_i در خروجی در هر بار چرخش برابر $P_s(x_i|v)$ است. بگذارید $U = [U_1 \dots U_n]$ برداری از متغیرهای اتفاقی باشد به‌طوری که U_i تعداد دفعات ظهور x_i در N بار چرخش چرخ را نشان بدهد. همچنین فرض کنید که $u = [u_1 \dots u_n]$ تحقیقی از بردار U باشد. نظریه‌ی توزیع چندجمله‌ای [ایوانز[1] و همکاران، ۲۰۰۰] بیان می‌دارد که

$$Pr_s(u|v) = N! \prod_{i=1}^{n} \frac{[P_s(x_i|v)]^{u_i}}{u_i!} \qquad (۳۷-٤)$$

این رابطه میزان احتمال رسیدن به بردار جمعیت u بعد از N بار چرخش چرخ رولت را به دست می‌دهد به‌طوری که بردار جمعیت آغازین برابر v بوده باشد. اندیس s در عبارت $P_s(x_i|v)$ نشان‌دهنده‌ی آن است که ما تنها فرایند انتخاب را در نظر گرفته‌ایم.

حال با خاطر آورید که ماتریس انتقال مارکوف تمام احتمالات انتقال از یک حالت به حالت دیگر را در بر می‌گیرد. معادله‌ی (٤,۳۷)، احتمال انتقال از یک بردار جمعیت v به بردار جمعیت دیگری، u، را به دست می‌دهد. همان طور که در قسمت قبل گفته شد، تعداد بردارهای جمعیت ممکن برابر T است. بنابراین اگر معادله‌ی (٤-۳۷) را برای هر u و هر v ممکن حساب کنیم، آنگاه یک ماتریس انتقال $T \times T$ به دست خواهد آمد که مدل احتمالی دقیقی از یک GA، که فقط شامل فرایند انتخاب است، به دست خواهد داد. هر درایه‌ی این ماتریس انتقال شامل احتمال انتقال از یک بردار جمعیت مشخص به یک بردار جمعیت دیگر است.

۴-۴-۲ جهش

حال فرض کنید بعد از انتخاب برخی ذرات دچار جهش می‌شوند. M_{ji} احتمال جهش x_j و تبدیل آن به x_i می‌باشد. آنگاه احتمال حصول ذره‌ی x_i بعد از یک بار چرخش چرخ رولت و بعد از آن یک بار جهش برابر است با

$$P_{sm}(x_i|v) = \sum_{j=1}^{n} M_{ji} P_s(x_j|v) \qquad (۳۸-٤)$$

[1] Evans

که در آن $i \in [0, N]$. این بدان معناست که می‌توانیم برداری n عنصری که عنصر iاُمش برابر $P_{sm}(x_i|v)$ می‌باشد را به‌صورت زیر بنویسیم

$$P_{sm}(x|v) = M^T P_s(x|v) \qquad (۳۹-٤)$$

که در آن M ماتریسی است که M_{ji} درایه‌ی سطر jاُم و ستون iاُمش بوده و $P_s(x|v)$ برداری است n عنصری که عنصر jاُمش برابر $P_s(x_i|v)$ است. حال دوباره از نظریه‌ی توزیع چندجمله‌ای استفاده می‌کنیم

$$Pr_{sm}(u|v) = N! \prod_{i=1}^{n} \frac{[P_{sm}(x_i|v)]^{u_i}}{u_i!} \qquad (٤٠-٤)$$

این عبارت احتمال حصول بردار جمعیت u را در صورتی که بردار جمعیت آغازین برابر v بوده و هر دو فرایند انتخاب و جهش در GA اتفاق بیافتند، نشان می‌دهد. اگر معادله‌ی (٤-٤٠) را برای همه‌ی u و vهای ممکن محاسبه کنیم، آنگاه به یک ماتریس انتقال مارکوف $T \times T$ دست پیدا خواهیم کرد که این ماتریس مدل احتمالی دقیقی از یک GA که شامل هر دو فرایند انتخاب و جهش می‌باشد را به دست خواهد داد.

اگر جهش به‌گونه‌ای تعریف شود که برای تمام i و jها، $M_{ji} > 0$ باشد، آنگاه برای تمام u و v ها، $P_{sm}(x_i|v) > 0$ خواهد بود. این بدان معناست که تمام درایه‌های ماتریس انتقال مارکوف مثبت بوده و بدین ترتیب این ماتریس معمولی خواهد بود. قضیه‌ی ٤-۲ بیان می‌دارد که برای حصول هر توزیع جمعیت ممکن، یک احتمال غیرصفر یکتا وجود دارد. این احتمال را می‌توان با استفاده از قضیه‌ی ٤-۲ و معادله‌ی (٤-٤٠) به دست آورد. GA به یک جمعیت خاص همگرا نمی‌شود بلکه به‌صورت نامتناهی میان فضای جستجو سرگردان بوده و با توجه به مقادیر منتج از قضیه‌ی ٤-۲، برای درصدهایی از زمان به هر یک از جمعیت‌های ممکن می‌رسد.

مثال ٤-۷

فرض کنید یک فضای جستجوی چهار عنصری با ذرات $\{00, 01, 10, 11\}$ در اختیار داریم. فرض کنید هر بیت در هر ذره با احتمال ۱۰٪ شانس جهش دارد. احتمال آنکه 00 بعد از جهش همان 00 باقی بماند برابر است با احتمال آنکه 0 اول بدون تغییر باقی بماند (۹۰٪) ضربدر احتمال آنکه 0 دوم نیز بدون تغییر باقی بماند (۹۰٪) که برابر ۰٫۸۱ خواهد بود. این مقدار برابر M_{11} است ، چراکه M_{11} احتمال آن است که x_{11} بدون تغییر باقی بماند. احتمال آنکه 00 بعد از جهش به 01 تبدیل شود برابر است با احتمال آنکه 0

اول بدون تغییر باقی بماند (۹۰٪) ضربدر احتمال آنکه 0 دوم به 1 تبدیل شود (۱۰٪) و بدین ترتیب خواهیم داشت $M_{12} = 0.09$. با ادامه‌ی این روند خواهیم داشت

$$M = \begin{bmatrix} 0.81 & 0.09 & 0.09 & 0.01 \\ 0.09 & 0.81 & 0.01 & 0.09 \\ 0.09 & 0.01 & 0.81 & 0.09 \\ 0.01 & 0.09 & 0.09 & 0.81 \end{bmatrix} \quad (٤١-٤)$$

توجه داشته باشید که M متقارن است (یعنی M یا ترانهاده‌اش، M^T، برابر است). این خاصیت معمولاً (نه همیشه!) بدین معناست که احتمال جهش x_i به x_j با احتمال جهش x_j به x_i برابر است.

۴-۴-۳ برش

حال فرض کنید بعد از انتخاب و جهش فرایند برش اتفاق بیافتد. اجازه دهید احتمال آنکه x_j با x_k عمل برش را انجام داده تا x_i حاصل شود را با r_{jki} نمایش دهیم. آنگاه احتمال حصول ذره‌ی x_i بعد از دو بار چرخش چرخ رولت و یکبار احتمال جهش برای هر ذره و پس از انجام عمل برش عبارتست از

$$P_{smc}(x_i|v) = \sum_{j=1}^{n}\sum_{k=1}^{n} r_{jki} P_{sm}(x_j|v) P_{sm}(x_k|v) \quad (٤٢-٤)$$

حال با استفاده از نظریه‌ی توزیع چندجمله‌ای داریم

$$Pr_{smc}(u|v) = N! \prod_{i=1}^{n} \frac{[P_{sm}c(x_i|v)]^{u_i}}{u_i!} \quad (٤٣-٤)$$

این عبارت احتمال حصول بردار جمعیت u را در صورتی که بردار جمعیت آغازین v بوده و فرایندهای انتخاب، جهش و برش اتفاق افتاده باشند، به دست می‌دهد.

مثال ٤-٨

فرض کنید فضای جستجویی چهار عنصری با ذرات $x = \{x_1, x_2, x_3, x_4\} = \{00, 01, 10, 11\}$ در اختیار داریم. فرض کنید عمل برش را قرار دادن b برابر ۱ یا ۲ به‌صورت اتفاقی و با احتمال برابر، انجام دهیم و سپس بیت‌های $1 \to b$ والد اول و بیت‌های $2 \to (b+1)$ والد دوم را به یکدیگر الحاق کنیم. برخی احتمالات برش را می‌توان به‌صورت زیر نوشت:

$$\begin{aligned} 00 \times 00 &\to 00 \\ 00 \times 01 &\to 01 \text{ یا } 00 \\ 00 \times 10 &\to 00 \end{aligned} \quad (٤٤-٤)$$

$$00 \times 11 \quad \rightarrow \quad 01 \text{ یا } 00$$

بدین ترتیب مقادیر احتمالات برش برابرند با

$$\begin{array}{llll} r_{111} = 1.0, & r_{112} = 0.0, & r_{113} = 0.0, & r_{114} = 0.0 \\ r_{121} = 0.5, & r_{122} = 0.5, & r_{123} = 0.0, & r_{124} = 0.0 \\ r_{131} = 1.0, & r_{132} = 0.0, & r_{133} = 0.0, & r_{134} = 0.0 \\ r_{141} = 0.5, & r_{142} = 0.5, & r_{143} = 0.0, & r_{144} = 0.0 \end{array} \qquad (٤٥-٤)$$

سایر مقادیر r_{jki} را می‌توان به طریقی مشابه محاسبه نمود.

مثال ٤-٩

در این مثال یک مسئله‌ی سه بیتی و تک- بیشینه را در نظر می‌گیریم. برازندگی هر ذره با تعداد یک‌های موجود در آن ذره متناسب است:

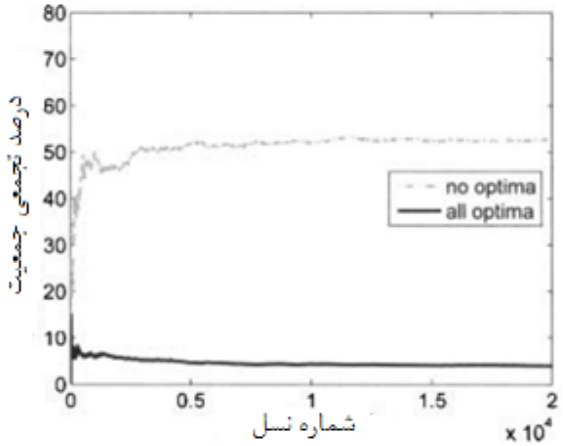
$$\begin{array}{llll} f(000) = 1, & f(001) = 2, & f(010) = 2, & f(011) = 3 \\ f(100) = 2, & f(101) = 3, & f(110) = 3, & f(111) = 4 \end{array} \qquad (٤٦-٤)$$

فرض کنید هر بیت با احتمال ۱۰٪ دچار جهش شود. بدین ترتیب ماتریس جهش همانی خواهد بود که در مثال ٤-۷ به دست آمد. بعد از فرایندهای انتخاب و جهش، فرایند برش را با احتمال ۹۰٪ انجام می‌دهیم. اگر قرعه به نام برش بیافتد، آنگاه این عمل با انتخاب یک موقعیت بیت اتفاقی $b \in [1, q-1]$ که در آن q تعداد بیت‌های هر ذره است، صورت خواهد پذیرفت. سپس بیت‌های $1 \rightarrow b$ از والد اول و بیت‌های $(b+1) \rightarrow 2$ از والد دوم را به یکدیگر پیوند می‌دهیم.

بگذارید از اندازه‌ی جمعیتی برابر $N = 3$ استفاده کنیم. تعداد توزیع جمعیت ممکن برابر است با ترکیب N شی از $n + N - 1$ شی که برابر است با ترکیب ۳ شی از ۱۰ شی که برابر با ۱۲۰ خواهد بود. می‌توانیم از معادله‌ی (٤-٤۳) برای محاسبه‌ی احتمال انتقال بین هر یک از ۱۲۰ توزیع جمعیت ممکن استفاده کنیم که با این کار به یک ماترس P با ابعاد 120×120 دست پیدا خواهیم کرد. پس می‌توان احتمال هر یک از توزیع‌های جمعیت ممکن را به یکی از سه روش زیر محاسبه نمود:

۱. می‌توان از معادله‌ی (٤-۱۵) استفاده نمود.
۲. می‌توان با استفاده از قضیه‌ی ٤-۲، P^{∞} را حساب کرده و سپس از هر یک از سطرهای P^{∞} برای مشاهده‌ی احتمال هر جمعیت ممکن استفاده نمود.
۳. می‌توان مقادیر ویژه‌ی P^T را حساب کرده و سپس بردار ویژه‌ی متناظر با مقدار ویژه‌ی ۱ را به دست آورد.

هر یک از این سه روش نتایج یکسانی برای احتمالِ ۱۲۰ توزیع جمعیت ممکن به دست خواهند داد. می‌توان مشاهده نمود که احتمال جمعیتی که در آن تمام ذرات بهینه هستند، یعنی هر ذره با رشته بیت ۱۱۱ برابر می‌باشد، برابر ٦,۱٪ می‌باشد. احتمال جمعیت که شامل هیچ ذره‌ی بهینه‌ای نباشد برابر ۵۱,۱٪ است. شکل ٤-۲ نتایج شبیه‌سازی برای ۲۰۰۰۰ نسل را نشان می‌دهد و مشخص است که نتایج شبیه‌سازی به نتایج مارکوف بسیار نزدیک‌اند. با این حال نتایج شبیه‌سازی تقریبی بوده و در هر بار اجرا متفاوت خواهند بود. نتاج شبیه‌سازی تنها در حالتی با نتایج مارکوف برابر خواهند شد که تعداد نسل‌ها به سمت بی‌نهایت میل کند.



شکل ٤-۲ مثال ٤-۹: نتایج شبیه‌سازی مسئله‌ی تک- بیشینه‌ی سه بیتی. نظریه‌ی مارکوف پیش‌بینی می‌کند احتمال جمعیت بدون ذره‌ی بهینه ۵۱,۱٪ و احتمال جمعیت تمام بهینه ٦,۱٪ باشد.

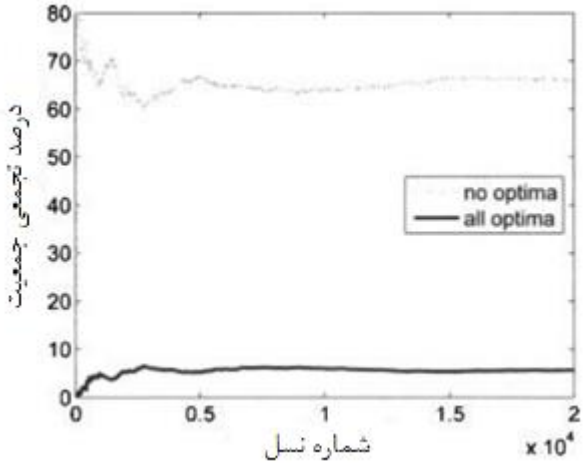
مثال ٤-۱۰

مثال ٤-۹ را با مقادیر برازندگی زیر تکرار می‌کنیم

$$f(000)=5,\quad f(001)=2,\quad f(010)=2,\quad f(011)=3$$
$$f(100)=2,\quad f(101)=3,\quad f(110)=3,\quad f(111)=4 \qquad (٤٧-٤)$$

این مقادیر برازندگی با مقادیر موجود در معادله‌ی (٤-٤٦) برابرند تنها با این تفاوت که در این مثال ذره‌ی ۰۰۰ برازنده‌ترین ذره است. این مسئله یک مسئله‌ی فریبنده است. چرا که در آن معمولاً با اضافه کردن یک بیت ۱ به یکی از ذرات، برازندگی‌اش افزایش می‌یابد اما ذره‌ی ۱۱۱ برازنده‌ترین ذره نیست بلکه برازنده‌ترین آن‌ها ۰۰۰ است. مانند مثال ٤-۹ مجموعه‌ی ۱۲۰ احتمال برای ۱۲۰ توزیع جمعیت ممکن را حساب می‌کنیم. خواهیم دید که احتمال جمعیتی که شامل تمام ذرات بهینه است، یعنی هر ذره برابر رشته بیت ۰۰۰ باشد،

برابر ۵,۹٪ می‌باشد. این مقدار در مثال ٤-۹، ٦,۱٪ بود. احتمال آنکه جمعیت شامل هیچ ذره‌ی بهینه‌ای نباشد برابر ٦٥,۲٪ است که این مقدار برا مثال ٤-۹ برابر ٥۱,۱٪ بود. این مثال نشان‌دهنده‌ی آن است که حل مسائل فریبنده سخت‌تر از مسائل با ساختار معمولی است. شکل ٤-۳ نتایج شبیه‌سازی برای ۲۰۰۰۰ نسل را نشان داده و همچنین نشان می‌دهد نتایج شبیه‌سازی بسیار به نتایج مارکوف نزدیک می‌باشند.



شکل ٤-۳ مثال ٤-۱۰: نتایج شبیه‌سازی مسئله‌ی فریبنده‌ی سه بیتی. نظریه‌ی مارکوف پیش‌بینی می‌کند احتمال جمعیت بی‌بهینه ٦٥,۲٪ بوده و احتمال جمعیت تمام بهینه برابر ۵,۹٪ می‌باشد.

نفرین ابعاد: "نفرین ابعاد" عبارتی بود که در اصل در متن‌های مربوط به برنامه‌ریزیِ پویا استفاده می‌شد [بِلمن[1]، ۱۹٦۱]. با این حال گاه در مورد مدل‌های مارکوف برای GAها مصداق بهتری دارد. اندازه‌ی ماتریس انتقال یک مدل مارکوف برابر با $T \times T$ است که در آن T برابر است با ترکیب N شی از $n + N - 1$ شی. در جدول ٤-۱ برخی ابعاد ماتریس انتقال به ازای برخی ترکیبات مختلف از اندازه‌ی جمعیت N و کاردینالیتی فضای جستجوی n (که برای فضای q بیتی برابر 2^q است) نشان داده شده‌اند. می‌توان دید که ابعاد ماتریس انتقال حتی برای مسایلی با ابعاد نسبتا کوچک به طرز مضحکی افزایش می‌یابد! این موضوع ممکن است به این حقیقت اشاره داشته باشد که مدل‌سازی مارکوف تنها از نقطه نظر نظری جالب توجه است اما هیچ‌گونه کاربرد عملی ندارد. با این حال، چنین پاسخی به چند دلیل نابه‌جا به نظر می‌رسد.

[1] Bellman

جدول ٤-١ ابعاد ماتریس انتقال مارکوف برای مقادیر مختلف کاردینالیتیِ فضای جستجوی n و اندازه‌ی جمعیت N. برگرفته شده از [ریوز و رو، ۲۰۰۳، صفحه‌ی ۱۳۱]

T	N	$n = 2^q$	تعداد بیت (q)
10^{23}	10	2^{10}	10
10^{42}	20	2^{10}	10
10^{102}	20	2^{20}	20
10^{688}	50	2^{50}	50

اولاً با این که نمی‌توان مدل‌های مارکوف را به مسائل با ابعاد حقیقی اعمال نمود، اما مدل‌های مارکوف همچنان احتمالات دقیقی را برای مسائل کوچک به دست می‌دهند. این موضوع این امکان را به ما می‌دهد تا نگاهی به فواید و مضرات الگوریتم‌های تکاملی مختلف برای مسائل کوچک داشته باشیم.

این دقیقاً همان کاری است که هنگام مقایسه‌ی GAها با BBO در [سایمون و همکاران، 2011b] انجام داده‌ایم. امروزه بسیاری از تحقیقات در زمینه‌ی الگوریتم‌های تکاملی بر شبیه‌سازی‌ها متمرکز شده‌اند. ایراد شبیه‌سازی‌ها آن است که خروجی آن‌ها به شدت به جزئیات پیاده‌سازی و تولیدکننده‌ی اعداد اتفاقی مورد استفاده وابسته است. علاوه بر این، اگر احتمال وقوع یک رخداد بسیار کم باشد، آنگاه تعداد شبیه‌سازی‌های بسیار زیادی برای آشکار ساختن این احتمال مورد نیاز است. نتایج شبیه‌سازی مفید و ضروری‌اند اما همیشه باید نیم‌نگاهی شکاکانه به آن‌ها داشت.

ثانیاً، ابعاد ماتریس انتقال مارکوف قابل کاهش‌اند. مدل مارکوف دارای T حالت است اما بسیاری از این حالات مشابه یکدیگرند. برای مثال، یک GA با اندازه‌ی جمعیت ۱۰ و کاردینالیتی ۱۰ را در نظر بگیرید. جدول ٤-١ بیان می‌دارد که مدل مارکوف 10^{23} حالت دارد، اما این حالات شامل موارد زیر می‌باشند:

$$\begin{aligned} v(1) &= \{5,5,0,0,0,0,0,0,0,0\} \\ v(2) &= \{4,6,0,0,0,0,0,0,0,0\} \\ v(3) &= \{6,4,0,0,0,0,0,0,0,0\} \end{aligned} \qquad (٤٨-٤)$$

این حالات به قدری شبیه یکدیگرند که می‌توان هر سه‌ی آن‌ها را در یک گروه قرار داد و یک حالت در نظر گرفت. این کار را می‌توان با بسیاری حالات دیگر نیز انجام داد تا به یک مدل مارکوف با فضای حالت کاهش یافته دست پیدا کرد. آنگاه ماتریس انتقال احتمالات مربوط به انتقال از یک گروه حالات اصلی به گروهی دیگر را مشخص خواهد کرد. این ایده در [اسپیرز[1] و دِجونگ، ۱۹۹۷] ارایه شده و در [ریوز و رو، ۲۰۰۳] بیشتر مورد بحث واقع شده است. شاید تصور چگونگی کاهش ابعاد یک ماتریس $10^{23} \times 10^{23}$ به یک ماتریس قابل اداره سخت باشد، اما حداقل با این ایده می‌توان مسائل بزرگتری را نسبت به قبل حل نمود.

[1] Spears

۴–۵ مدل سیستم پویا برای الگوریتم‌های ژنتیک

در این بخش از مدل‌های مارکوف بخش قبلی برای به دست آوردن یک مدل سیستم پویا برای GAها استفاده خواهیم نمود. مدل مارکوف احتمال وقوع هر یک از توزیع‌های جمعیت را هنگامی که تعداد نسل‌ها به سمت بی‌نهایت میل می‌کند به دست می‌دهد . مدل سیستم پویایی که در اینجا به دست خواهیم آورد کاملاً متفاوت خواهد بود. این مدل درصد هر ذره در جمعیت را به‌عنوان تابعی از و هنگامی که اندازه‌ی جمعیت به سمت بی‌نهایت میل کند، دست می‌دهد. دیدگاه GA به‌عنوان یک سیستم پویا در ابتدا در [نیکس و وُز، ۱۹۹۲]، [وُز، ۱۹۹۶] و [وُز و لیپینز[1]، ۱۹۹۱] منتشر شد و مباحث تکمیلی در مورد آن در [ریوز و رو، ۲۰۰۳] و [وُز، ۱۹۹۹] ارایه گردید.

از معادله‌ی (٤–۲۲) به یاد آورید که $v = [v_1 \dots v_n]^T$ بردار جمعیت بوده، v_i تعداد ذرات x_i در جمعیت بوده و جمع عناصر v برابر با N یا همان اندازه‌ی جمعیت می‌باشد. ماتریس تناسب را به‌صورت زیر تعریف می‌کنیم

$$p = v/N \qquad (٤٩–٤)$$

که بدین معناست که جمع عناصر p برابر ۱ خواهد بود.

۴–۵–۱ انتخاب

برای دست یافتن به مدل سیستم پویای یک GA که تنها شامل فرایند انتخاب است می‌توانیم صورت و مخرج معادله‌ی (٤–۳٦) را بر N تقسیم کرده و بدین ترتیب احتمال انتخاب ذره‌ی x_i از جمعیتِ توصیف شده با بردار جمعیت v را محاسبه کنیم:

$$P_s(x_i|v) = \frac{p_i f_i}{\sum_{j=1}^{n} p_j f_j} = \frac{p_i f_i}{f^T p} \qquad (٥٠–٤)$$

که در آن f برداری ستونی از مقادیر برازندگی است. با نوشتن معادله‌ی (٤–٥٠) برای $i \in [0, N]$ و با ترکیب همه‌ی n معادله خواهیم داشت:

$$P_s(x|v) = \begin{bmatrix} P_s(x_1|v) \\ \dots \\ P_s(x_n|v) \end{bmatrix} = \frac{diag(f)p}{f^T p} \qquad (٥١–٤)$$

[1] Liepins

که در آن $diag(f)$ یک ماتریس قطری $n \times n$ بوده و قطر این ماتریس از عناصر f تشکیل شده است. قانون اعداد بزرگ به ما می‌گوید که میانگین نتایج حاصله از تعداد زیادی از دنباله‌ها باید به امید ریاضی[1] یک دنباله‌ی تنها نزدیک باشد. [گرین استید و اسنل]. این یعنی با بزرگ شدن اندازه‌ی جمعیت، نسبت انتخاب هر ذره‌ی x_i به $P_s(x_i|v)$ نزدیک خواهد بود. اما تعداد انتخاب‌های x_i در نسل بعد برابر v_i خواهد بود. بنابراین، برای اداره جمعیت‌های بزرگ، می‌توان معادله‌ی (٤-٥٠) را به‌صورت زیر نوشت

$$p_i(t) = \frac{p_i(t-1)f_i}{\sum_{j=1}^{n} p_j(t-1)f_j} \qquad (٤-٥٢)$$

که در آن t شماره‌ی نسل است.

حال فرض کنید که

$$p_i(t) = \frac{p_i(0)f_i^t}{\sum_{j=1}^{n} p_j(0)f_j^t} \qquad (٤-٥٣)$$

از معادله‌ی (٤-٥٢) واضح است که معادله‌ی بالا برای $t = 1$ صادق است. با جایگذاری $t-1$ در معادله‌ی (٤-٥٣)، صورت کسرِ معادله‌ی (٤-٥٢) را می‌توان به‌صورت زیر نوشت

$$f_i p_i(t-1) = f_i \frac{p_i(0)f_i^{t-1}}{\sum_{j=1}^{n} p_j(0)f_j^{t-1}} = \frac{p_i(0)f_i^t}{\sum_{j=1}^{n} p_j(0)f_j^{t-1}} \qquad (٤-٥٤)$$

همچنین مخرج کسر معادله‌ی (٤-٥٢) به‌صورت زیر در خواهد آمد

$$\sum_{j=1}^{n} p_j(t-1)f_j = \sum_{j=1}^{n} f_j \frac{p_i(0)f_i^{t-1}}{\sum_{k=1}^{n} p_j(0)f_k^{t-1}} = \sum_{j=1}^{n} \frac{p_j(0)f_j^t}{\sum_{k=1}^{n} p_k(0)f_k^{t-1}} \qquad (٤-٥٥)$$

با جایگذاری معادلات (٤-٥٤) و (٤-٥٥) در معادله‌ی (٤-٥٢) خواهیم داشت

$$p_i(t) = \frac{p_i(0)f_i^t}{\sum_{k=1}^{n} p_k(0)f_k^{t-1}} \qquad (٤-٥٦)$$

[1] Expected Value

این معادله بردار تناسب را به‌صورت تابعی از زمان و مقادیر برازندگی و بردار تناسب آغازین به دست خواهد داد. این نتیجه برای GAهایی که تنها شامل فرایند انتخاب باشد معتبر است.

مثال ٤-١١

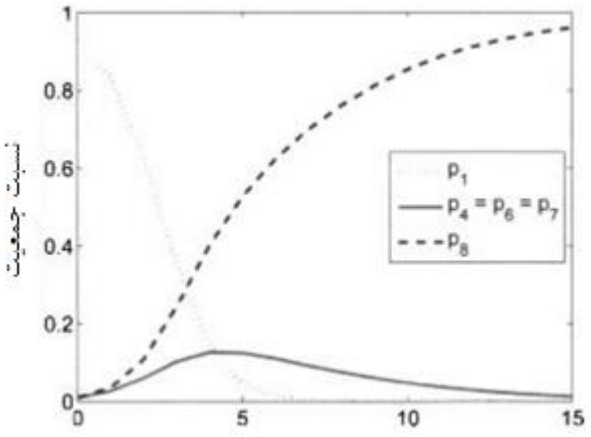
مانند مثال ٤-٩، مسئله‌ی تک‌بیشینه‌ی سه بیتی با مقادیر برازندگی ذیل در نظر می‌گیریم

$$f(000)=1,\quad f(001)=2,\quad f(010)=2,\quad f(011)=3$$
$$f(100)=2,\quad f(101)=3,\quad f(110)=3,\quad f(111)=4 \qquad (٥٧-٤)$$

فرض کنید بردار تناسب آغازین برابر است با

$$p(0)=[0.93\quad 0.01\quad 0.01\quad 0.01\quad 0.01\quad 0.01\quad 0.01\quad 0.01]^T \qquad (٥٨-٤)$$

۹۳٪ از جمعیت اولیه‌ی از ذرات با کمترین میزان برازندگی تشکیل شده و تنها ۱٪ از جمعیت به ذرات با بیشترین برازندگی تعلق دارد. شکل ٤-٤ نموداری از معادله‌ی (٤-٥٦) را نشان می‌دهد. می‌بینیم که با پیشرفت جمعیت GA، x_4، x_6 و x_7 که از لحاظ برازندگی در مقام دوم قرار دارند، بیشتر جمعیت را در اختیار می‌گیرند. ذره با کمترین برازندگی، x_1، به سرعت توسط فرایند انتخاب از جمعیت کنار گذاشته می‌شود. p_2، p_3 و P_5 در این شکل نشان داده نشده‌اند. تعداد نسل‌های زیادی برای رسیدن کل جمعیت به برازنده‌ترین ذره، x_8، نیاز نیست.



شکل ٤-٤ تکامل بردار تناسب جمعیت برای مثال ٤-١١. اگرچه برازنده‌ترین ذره، x_8، در ابتدا تنها ۱٪ از جمعت را تشکیل می‌دهد اما این مقدار به سرعت به سمت ۱۰۰٪ می‌گراید. ذره با کمترین برازندگی، x_1، در ابتدا ۹۳٪ از جمعیت را در اختیار داشته ولی این مقدار به سرعت به سمت ۰٪ می‌گراید.

تا به اینجا در مورد مدل سیستم پویا برای انتخاب متناسب با برازندگی بحث نموده‌ایم اما انتخاب‌هایی چون انتخاب مسابقه‌ای و انتخاب رتبه‌ای را نیز می‌توان با استفاده از مدل سیستم پویا مدل نمود [ریوز و رو، ۲۰۰۳]، [وُز، ۱۹۹۹].

۴-۵-۲ جهش

معادله‌ی (٤-٥١) همراه با قانون اعداد بزرگ بیان می‌دارند که

$$p(t) = \frac{diag(f)p(t-1)}{f^T p(t-1)} \quad \text{(فقط انتخاب)} \tag{٤-٥٩}$$

اگر انتخاب با جهش همراه باشد و M_{ji} احتمال جهش x_j به x_i باشد، آنگاه می‌توان از استنتاجی مانند معادله‌ی (٤-٣٠) استفاده که در این صورت خواهیم داشت

$$p(t) = \frac{M^T diag(f)p(t-1)}{f^T p(t-1)} \quad \text{(انتخاب و جهش)} \tag{٤-٦٠}$$

اگر $p(t)$ به مقدار حالت ماندگار برسد آنگاه می‌توان نوشت $p_{ss} = p(t-1) = p(t)$ که در آن صورت معادله‌ی (٤-٦٠) به‌صورت زیر درخواهد آمد

$$p_{ss} = \frac{M^T diag(f)p_{ss}}{f^T p_{ss}} \tag{٤-٦١}$$
$$M^T diag(f)p_{ss} = (f^T p_{ss})p_{ss}$$

این معادله به فرم $Ap = \lambda p$ است که در آن λ مقدار ویژه‌ی A بوده و p بردار ویژه‌ی A است. می‌بینیم که بردار تناسب حالت ماندگار یک GA با فرایندهای انتخاب و جهش (و نه برش!)، بردار ویژه‌ی $M^T diag(f)$ است.

مثال ٤-١٢

مانند مثال ٤-١٠، یک مسئله‌ی فریبنده‌ی سه بیتی با مقادیر برازندگی زیر در نظر می‌گیریم

$$f(000) = 5,\ \ f(001) = 2,\ \ f(010) = 2,\ \ f(011) = 3$$
$$f(100) = 2,\ \ f(101) = 3,\ \ f(110) = 3,\ \ f(111) = 4 \tag{٤-٦٢}$$

در این مسئله نرخ جهش را ۲٪ در هر بیت فرض می‌کنیم. داریم

$$M^T diag(f) = \tag{٤-٦٣}$$

$$\begin{bmatrix} 4.706 & 0.038 & 0.038 & 0.001 & 0.038 & 0.001 & 0.001 & 0.000 \\ 0.096 & 1.882 & 0.001 & 0.058 & 0.001 & 0.058 & 0.000 & 0.002 \\ 0.096 & 0.001 & 1.882 & 0.058 & 0.001 & 0.000 & 0.058 & 0.002 \\ 0.002 & 0.038 & 0.038 & 2.824 & 0.000 & 0.001 & 0.001 & 0.077 \\ 0.096 & 0.001 & 0.001 & 0.000 & 1.882 & 0.058 & 0.058 & 0.002 \\ 0.002 & 0.038 & 0.000 & 0.001 & 0.038 & 2.824 & 0.001 & 0.077 \\ 0.002 & 0.000 & 0.038 & 0.001 & 0.038 & 0.001 & 2.824 & 0.077 \\ 0.000 & 0.001 & 0.001 & 0.058 & 0.001 & 0.058 & 0.058 & 3.765 \end{bmatrix}$$

بردارهای ویژه‌ی $M^T diag(f)$ را همان‌طور که در معادله‌ی (٤–٦١) نشان داده شده محاسبه کرده و هر یک از آن‌ها را به‌گونه‌ای مقیاس می‌کنیم که جمع عناصرشان برابر ۱ شود. به خاطر داشته باشید که بردارهای ویژهی یک ماتریس به مقدار مقیاس کننده بستگی ندارند، بدین معنی که اگر p یک بردار ویژه است آنگاه cp نیز برای هر مقدار غیر صفر c یک بردار ویژه است. از آن جایی که هر بردار ویژه معرف یک بردار تناسب است بنابراین، همان‌طور که در معادله‌ی (٤–٤٩) نشان داده شده است، جمع عناصر آن باید برابر ۱ باشد. ما هشت بردار ویژه به دست خواهیم آورد اما تنها یکی از آن‌ها تمام عناصرش مثبت خواهد بود. بنابراین، تنها یک بردار تناسب حالت ماندگار وجود خواهد داشت:

$$p_{ss}(1) = [0.90074 \quad 0.03070 \quad 0.03070 \quad 0.00221 \\ 0.03070 \quad 0.00221 \quad 0.00221 \quad 0.0005]^T \qquad (٦٤–٤)$$

این معادله بیانگر آن است که GA به سمت جمعیتی همگرا خواهد شد که ۹۰,۰۷٤٪ از آن را x_1 تشکیل داده و هر یک از ذرات x_2، x_3 و x_5 هر کدام ۳,۰۷٪ جمعیت را در اختیار خواهند داشت. این یعنی بیش از ۹۰٪ از جمعیت GA شامل ذرات بهینه خواهد بود. با این حال، بردار ویژه‌ای از $M^T diag(f)$ وجود دارد که تنها شامل یک عنصر منفی است:

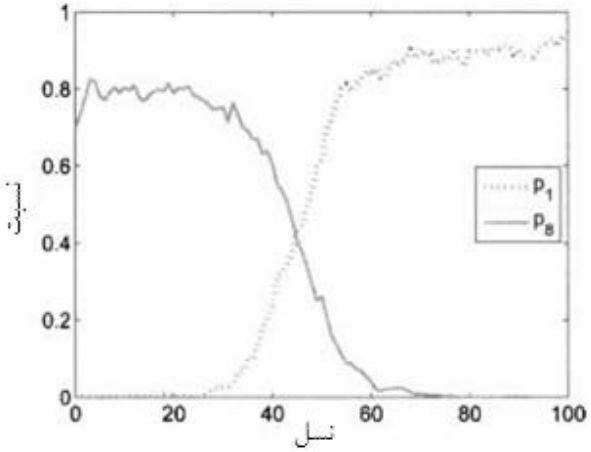
$$p_{ss}(1) = [-0.0008 \quad 0.0045 \quad 0.0045 \quad 0.0644 \\ 0.0045 \quad 0.0644 \quad 0.0644 \quad 0.7941]^T \qquad (٦٥–٤)$$

به این بردار نقطه‌ی فراپایدار می‌گویند [ریوز و رو، ۲۰۰۳] و شامل درصد بسیار زیادی (۷۹,٤۱٪) از ذرات x_8 که از لحاظ برازندگی در مرتبه‌ی دوم قرار دارد، می‌باشد. از آنجا که $p_{ss}(2)$ نقطه‌ای ثابت از معادله‌ی (٤–٦١) است، هر بردار تناسبی نزدیک به آن تمایل خواهد داشت در همانجا بماند. با این حال، $p_{ss}(2)$ یک بردار تناسب معتبر نیست چرا که دارای مقدار منفی در میان عناصرش بوده و همچنین با اینکه جمعیت GA به سمت $p_{ss}(2)$ جذب می‌شود، اما در نهایت از آن دور شده و به سمت $p_{ss}(1)$ خواهد رفت. شکل ٤–۵ نتایج شبیه‌سازی GA با فرایندهای انتخاب و جهش را نشان می‌دهد که در آن اندازه‌ی جمعیت $N = 500$ بوده و از بردار تناسب آغازین ذیل در آن استفاده شده است

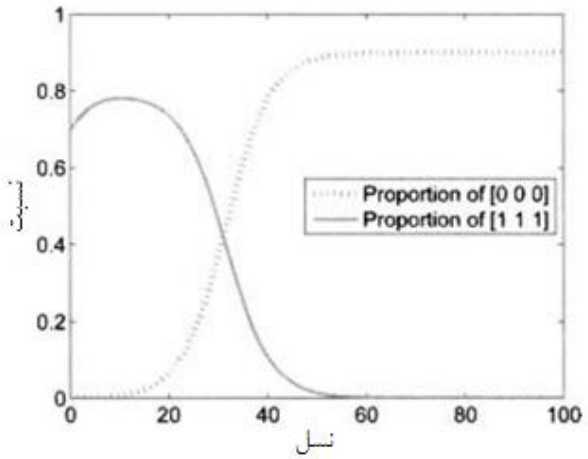
$$p(0) = [0.0 \quad 0.0 \quad 0.0 \quad 0.1 \quad 0.0 \quad 0.1 \quad 0.1 \quad 0.7]^T \qquad (٦٦-٤)$$

که این بردار به نقطه‌ی فراپایدار $p_{ss}(2)$ نزدیک است. از شکل ٤-٥ مشاهده می‌کنیم که تا حدود ۳۰ نسل جمعیت به توزیع ابتدایی که شامل ۸۰٪ از ذرات x_8 بوده و به نقطه‌ی فراپایدار $p_{ss}(2)$ نزدیک است، نزدیک می‌ماند. بعد از حدود ۳۰ نسل جمعیت به سرعت به نقطه‌ی پایدار $p_{ss}(1)$ که شامل حدود ۹۰٪ از ذرات x_1 است همگرا می‌شود. به یاد داشته باشید که اگر شبیه‌سازی دوباره انجام شود، به دلیل وجود تولیدکننده‌ی اتفاقی به کار رفته برای انتخاب و جهش، نتایج متفاوتی به دست خواهد آمد.

شکل ٤-٦، p_1 و p_8 از معادله‌ی (٤-٦٠) را برای ۱۰۰ نسل نشان می‌دهد. این شکل نسبت دقیقی از ذرات x_1 و x_8 را به دست می‌دهد، هنگامی که اندازه‌ی جمعیت به سمت بی‌نهایت میل می‌کند. می‌توان دید که شکل‌های ٤-٥ و ٤-٦ به یکدیگر شباهت دارند اما شکل ٤-٥ نتایج شبیه‌سازی با اندازه‌ی جمعیت محدود است و به دلیل وجود تولیدکننده‌ی اعداد اتفاقی به کار رفته، با هر بار اجرا نتایج متفاوتی به دست خواهد آمد. این در حالی است که نتایج شکل ٤-٦ دقیق هستند.



شکل ٤-٥ شبیه‌سازی برای مثال ٤-١٢. جمعیت ابتدا در حول و حوش نقطه‌ی فراپایدار، که ۷۰٪ از آن از ذرات x_1 تشکیل شده است، قرار دارد و سپس به سمت نقطه‌ی پایدار با ۹۰٪ از ذرات x_8 حرکت می‌کند. به دلیل طبیعت آماری شبیه‌سازی، با هر بار اجرا نتایج متفاوتی به دست خواهد آمد.



شکل ٤-٦ نتایج تحلیل برای مثال ٤-١٢. با شکل ٤-٥ مقایسه کنید. نتایج تحلیلی به تولیدکننده‌ی اعداد اتفاقی بستگی ندارد.

۴-۵-۳ برش

همانند بخش ٤-٤-٣ از علامت r_{jki} برای نشان دادن احتمال برش x_j با x_k و تشکیل x_i استفاده می‌کنیم. اگر جمعیت با استفاده از بردار تناسب p در یک جمعیت نامحدود مشخص شود، آنگاه احتمال ایجاد x_i از یک برش اتفاقی با استفاده از رابطه‌ی زیر به دست خواهد آمد

$$P_c(x_i|p) = \sum_{j=1}^{n}\sum_{k=1}^{n} p_j p_k r_{ijk} = \sum_{k=1}^{n} p_k \sum_{j=}^{n} p_j r_{jik}$$

$$= \sum_{k=1}^{n} p_k [p_1 \dots p_n] \begin{bmatrix} r_{1ki} \\ \dots \\ r_{nki} \end{bmatrix}$$

$$= [p_1 \dots p_n] \sum_{k=1}^{n} p_k \begin{bmatrix} r_{1ki} \\ \dots \\ r_{nki} \end{bmatrix}$$

$$= p^T \begin{bmatrix} \sum_{k=1}^{n} r_{1ki} p_k \\ \dots \\ \sum_{k=1}^{n} r_{nki} p_k \end{bmatrix} \qquad (٤-٦٧)$$

$$= p^T \begin{bmatrix} [r_{11i} & \dots & r_{1ni}]p \\ & \dots & \\ [r_{n1i} & \dots & r_{nni}]p \end{bmatrix}$$

$$= p^T \begin{bmatrix} r_{11i} & \cdots & r_{1ni} \\ \vdots & \ddots & \vdots \\ r_{n1i} & \cdots & r_{nni} \end{bmatrix} p$$

$$= p^T R_i p$$

که در آن درایه‌ی سطر jاُم و ستون kاُم از R_i با r_{jki} برابر است دوباره از قانون اعداد بزرگ [گرین استید و اسنل، ۱۹۹۷] استفاده می‌کنیم. در این حالت اگر اندازه‌ی جمعیت N به سمت بی‌نهایت میل کند، فرایند برش نسبت ذراتِ x_i را به شکل زیر تغییر خواهد داد

$$\hat{p}_i = P_c(x_i|p) = p^T R_i p \qquad (٦٨-٤)$$

اگرچه R_i معمولاً نامتقارن است، $P_c(x_i|p)$ را می‌توان با استفاده از یک ماتریس متقارن به‌صورت زیر نوشت

$$\begin{aligned} P_c(x_i|p) &= p^T R_i p \\ &= \frac{1}{2} p^T R_i p + \frac{1}{2} (p^T R_i p)^T \end{aligned} \qquad (٦٩-٤)$$

توجه داشته باشید که $p^T R_i p$ یک اسکالر است و ترانهاده‌ی یک اسکالر با خودش برابر است. بنابراین با توجه به اینکه $(ABC)^T = C^T B^T A^T$، خواهیم داشت

$$\begin{aligned} P_c(x_i|p) &= \frac{1}{2} p^T R_i p + \frac{1}{2} (p^T R_i p)^T \\ &= \frac{1}{2} p^T \left(R_i + {R_i}^T\right) p \\ &= p^T \hat{R}_i p \end{aligned} \qquad (٧٠-٤)$$

که در آن ماتریس متقارن $\hat{R}_i$ از رابطه‌ی زیر به دست خواهد آمد

$$\hat{R}_i = \frac{1}{2}\left(R_i + {R_i}^T\right) \qquad (٧١-٤)$$

مثال ٤-١٣

مانند مثال ٤-٨ فرض کنید یک فضای جستجوی چهار عنصره با ذرات $x = \{x_1, x_2, x_3, x_4\} =$ $\{00, 01, 10, 11\}$ در اختیار داریم. عمل برش را با قرار دادن $b = 1$ یا $b = 2$ به‌صورت اتفاقی و با احتمال برابر انجام داده و سپس بیت‌های $1 \to b$ والد اول و بیت‌های $2 \to (b+1)$ والد دوم را به یکدیگر الحاق می‌کنیم. احتمالات برش به شرح زیر خواهد بود

$$\begin{aligned} &00 \times 00 \;\to\; 00 \\ &00 \text{یا} \quad 00 \times 01 \;\to\; 01 \\ &00 \times 10 \;\to\; 00 \\ &00 \text{یا} \quad 00 \times 11 \;\to\; 01 \end{aligned} \qquad (٧٢-٤)$$

$$01 \times 00 \quad \rightarrow \quad 00 \quad \text{یا} \quad 01$$
$$01 \times 01 \quad \rightarrow \quad 01$$
$$01 \times 10 \quad \rightarrow \quad 00 \quad \text{یا} \quad 01$$
$$01 \times 11 \quad \rightarrow \quad 01$$
$$10 \times 00 \quad \rightarrow \quad 10$$
$$10 \times 01 \quad \rightarrow \quad 11 \quad \text{یا} \quad 10$$
$$10 \times 10 \quad \rightarrow \quad 10$$
$$10 \times 11 \quad \rightarrow \quad 11 \quad \text{یا} \quad 10$$
$$11 \times 00 \quad \rightarrow \quad 11 \quad \text{یا} \quad 10$$
$$11 \times 01 \quad \rightarrow \quad 11$$
$$11 \times 10 \quad \rightarrow \quad 11 \quad \text{یا} \quad 10$$
$$11 \times 11 \quad \rightarrow \quad 11$$

بدین ترتیب احتمالات r_{jk1}، که در واقع احتمال برش x_j با x_k برای تشکیل x_1 به شرح زیر است

$$\begin{matrix} r_{111} = 1.0, & r_{121} = 0.5, & r_{131} = 0.0, & r_{141} = 0.5 \\ r_{211} = 0.5, & r_{221} = 0.0, & r_{231} = 0.5, & r_{241} = 0.0 \\ r_{311} = 0.0, & r_{321} = 0.0, & r_{331} = 0.0, & r_{341} = 0.0 \\ r_{411} = 0.0, & r_{421} = 0.0, & r_{431} = 0.0, & r_{441} = 0.0 \end{matrix} \qquad (۷۳-٤)$$

که این مقادیر ماتریس برش زیر را به دست خواهند داد

$$R_1 = \begin{bmatrix} 1.0 & 0.5 & 1.0 & 0.5 \\ 0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \qquad (۷٤-٤)$$

کاملاً واضح است که R_1 متقارن نیست اما با این حال می‌توان $P_c(x_i|p)$ را با استفاده از ماتریس متقارن نوشت

$$P_c(x_1|p) = p^T \hat{R}_1 p$$

که در آن

$$\hat{R}_1 = \frac{1}{2}\left(R_1 + {R_1}^T\right) = \begin{bmatrix} 1.0 & 0.5 & 0.5 & 0.25 \\ 0.5 & 0.0 & 0.25 & 0.0 \\ 0.5 & 0.25 & 0.0 & 0.0 \\ 0.25 & 0.0 & 0.0 & 0.0 \end{bmatrix} \qquad (۷۵-٤)$$

سایر ماتریس‌های $\hat{R}_i$ را می‌توان به روشی مشابه پیدا کرد.

حال فرض کنید یک GA داریم که در آن به ترتیب فرایندهای انتخاب، جهش و برش رخ می‌دهند. یک بردار تناسب p در نسل $(t-1)$اُم نیز در اختیار داریم. انتخاب و جهش p را به روش نشان داده شده در معادله‌ی (٤-٦٠) اصلاح می‌کند

$$p(t) = \frac{M^T diag(f) p(t-1)}{f^T p(t-1)} \qquad (٤-٧٦)$$

برش، p_i را به‌گونه‌ای که در معادله‌ی (٤-٦٦) نشان داده شده اصلاح می‌کند. با این حال، p موجود در سمت راست معادله‌ی (٤-٦٨) قبلا توسط فرایندهای انتخاب و جهش اصلاح شده و p نشان داده شده در معادله‌ی (٤-٧٦) حاصل گردیده است. بنابراین، ترتیب فرایندهای انتخاب، جهش و برش، $\widehat{p_i}$ نشان داده شده در معادله‌ی (٤-٦٨) را نتیجه خواهند داد با این تفاوت که p موجود در سمت راست معادله‌ی (٤-٦٨) با p منتج شده از انتخاب و جهش (٤-٧٦) جایگزین شده است:

$$\begin{aligned} p_i(t) &= \left[\frac{M^T diag(f) p(t-1)}{f^T p(t-1)}\right]^T R_i \left[\frac{M^T diag(f) p(t-1)}{f^T p(t-1)}\right] \\ &= \frac{p^T(t-1) diag(f) M R_i M^T diag(f) p(t-1)}{(f^T p(t-1))^2} \end{aligned} \qquad (٤-٧٧)$$

می‌توان در معادله‌ی (٤-٧٧)، R_i را با $\widehat{R_i}$ جایگزین نمود تا عبارتی معادل به دست آید. معادله‌ی (٤-٧٧) عبارتی دقیق و تحلیلی برای دینامیک نسبت ذرات x_i در یک جمعیت نامحدود را به دست می‌دهد.

در هر نسل باید مدل سیستم پویای معادله‌ی (٤-٧٧) را برای هر $i \in [1, n]$ حساب نمود (n اندازه‌ی فضای جستجو است). ماتریس‌های معادله‌ی (٤-٧٧)، $n \times n$ بوده و میزان محاسبات ضرب ماتریسی، در صورت پیاده‌سازی با الگوریتم‌های استاندارد، با n^3 متناسب است. بنابراین، مدل سیستم پویا به محاسباتی از مرتبه‌ی n^4 نیاز دارد. این میزان از محاسبات بسیار کمتر از میزان محاسبات لازم برای فرایند مارکوف است، اما همچنان با افزایش اندازه‌ی فضای جستجو، n بسیار سریع افزایش یافته و به همین دلیل بررسی مسائل با اندازه‌ی نسبتاً کوچک به منابع محاسباتی غیرقابل دستیابی نیاز دارد.

مثال ٤-١٤

بار دیگر یک مسئله‌ی تک- بیشینه‌ی سه بیتی را در نظر بگیرید (مثال ٤-٩ را ببینید). ما از احتمال برشی برابر ۹۰٪، نرخ جهشی برابر ۱٪ در هر بیت، اندازه‌ی جمعیتی برابر ۱۰۰۰ و بردار تناسب آغازین نشان داده شده در زیر استفاده می‌کنیم

$$p(0) = [0.8 \quad 0.1 \quad 0.1 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0] \qquad (٤-٧٨)$$

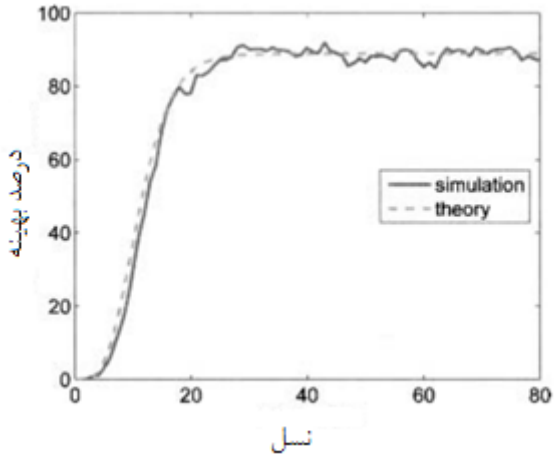
شکل ٤–٧ درصد ذرات بهینه‌ی موجود در جمعیت را از یک بار شبیه‌سازی در مقایسه با نتایج دقیق نظری از معادله‌ی (٤–٧٧) را نشان می‌دهد. نتایج شبیه‌سازی به خوبی با نتایج نظری منطبق است، اما نتایج شبیه‌سازی تقریبی بوده و از یک اجرا به اجرای دیگر متفاوت خواهد بود. این در حالی است که نتایج نظری دقیق هستند.

حال فرض کنید بردار تناسب آغازین را به شکل زیر تغییر دهیم

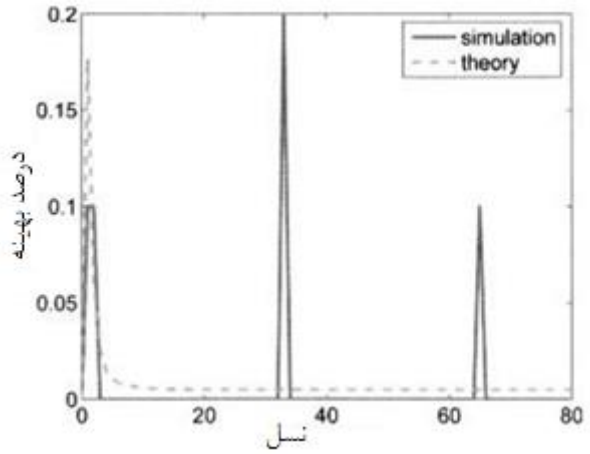
$$p(0) = [0.0 \quad 0.1 \quad 0.1 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.8]^T \qquad (٧٩–٤)$$

شکل ٤–٨ درصد نابرازنده‌ترین ذرات را از یک بار شبیه‌سازی در مقایسه با نتایج دقیق نظری نشان می‌دهد. از آنجا که احتمال حصول یک ذره با کمترین برازندگی بسیار کم است، نتایج شبیه‌سازی چند پرش را در نمودار نشان می‌دهند.

در نگاه اول، این پرش‌ها در مقایسه با اندازه‌ی نمودار بزرگ به نظر می‌ایند اما در واقع این پرش‌ها بسیار کوچک بوده و نقطه‌ی بیشینه‌ی آن‌ها ۰٫۲٪ می‌باشد. با این حال نتایج نظری کاملاً دقیق هستند. این نتایج نشان می‌دهند که نسبت ذرات با کمترین برازندگی در چند نسل ابتدایی به دلیل جهش کمی افزایش یافته و سپس به سرعت به مقدار دقیق حالت ماندگار یعنی ۰٫۰۰۵۰۲٪ می‌رسد. اگر بخواهیم با استفاده از شبیه‌سازی به این نتیجه برسیم، باید تعداد بسیار بسیار زیادی شبیه‌سازی انجام دهیم. حتی بعد از هزاران بار شبیه‌سازی ممکن است نتایج اشتباه حاصل شود و این موضوع به صحت تولیدکننده‌ی اعداد اتفاقی به کار رفته بستگی خواهد داشت.



شکل ٤–٧ نسبت برازنده‌ترین ذرات برای مثال ٤–١٤



شکل ٤-٨ نسبت ذرات با کمترین برازندگی برای مثال ٤-١٤

۴-۶ نتیجه‌گیری

در این فصل مدل‌های مارکوف و مدل‌های سیستم پویا برای GA را تشریح نمودیم. این مدل‌ها، که در دهه‌ی ۱۹۹۰ ابداع شدند، نتایج نظری دقیقی را به دست می‌دهند اما نتایج شبیه‌سازی در هر بار اجرا به دلیل وجود تولیدکننده‌های اعداد اتفاقی به کار رفته برای انتخاب، جهش و برش، متفاوت خواهند بود. اندازه‌ی مدل مارکوف به صرت فاکتوریلی با اندازه‌ی جمعیت و کاردینالیتی فضای جستجو افزایش پیدا می‌کند. مدل سیستم پویا با n^4 رشد پیدا می‌کند که n کاردینالیتی فضای جستجو است. این الزاماتِ محاسباتی کاربرد مدل‌های مارکوف و سیستم پویا را به مسائل کوچک محدود می‌سازند. با این حال، این مدل همچنان برای مقایسه‌ی پیاده‌سازی‌های مختلف GA و همچنین مقایسه‌ی الگوریتم‌های تکاملی مفید هستند [سایمون و همکاران، 2011b].در [ریوز و رو، ۲۰۳۳] و [وُز، ۱۹۹۹] برخی ایده‌ها و دست‌یافته‌های دیگر ارائه شده‌اند.

مدل‌سازی مارکوف و سیستم پویا زمینه‌های بسیار بالغی هستند که نتایج کلی به دست می‌دهند. فضای بسیار زیادی در زمینه‌ی کاربرد آن‌ها در مورد GA و سایر الگوریتم‌های تکاملی وجود دارد.

می‌توان از روش‌های دیگری نیز برای مدل‌سازی و تحلیل رفتار GA استفاده نمود. برای مثال، مکانیک آماری زمینه‌ای است که شامل متوسط‌گیری از بسیاری ذرات مولکول برای مدل‌سازی رفتار یک گروه از مولکول‌ها می‌باشد. می‌توان از این ایده برای مدل‌سازی رفتار GA با جمعیت‌های زیاد استفاده نمود [ریوز و رو، ۲۰۰۳، فصل ۷]. همچنین می‌توان از تبدیلات فوریه و والش برای تحلیل رفتار GA استفاده نمود [وُز و

رایت، 1998a]، [وُز و رایت، 1998b]. در آخر آنکه می‌توان از انتخاب پرایس[1] و قضیه‌ی کوواریانس برای مدل‌سازی GAها استفاده نمود [پولی[2] و همکاران، ۲۰۰۸، فصل ۳].

ایده‌های ارائه شده در این فصل را می‌توان برای بسیاری الگوریتم‌های تکاملی دیگر، به غیر از GA به کار برد. ما این کار را در [سایمون و همکاران، 2011a] و [سایمون، 2011a] برای بهینه‌سازی زیست-جغرافی محور انجام داده‌ایم. بسیاری از محققان دیگر این ایده‌ها را برای سایر الگوریتم‌های تکاملی به کار برده‌اند اما همچنان فضای زیادی برای کاربرد مدل‌های مارکوف و سیستم پویا در زمینه‌ی الگوریتم‌های تکاملی وجود دارد. این کار امکان مقایسه میان الگوریتم‌های تکاملی مختلف در سطح تحلیلی را به دست می‌دهد و دیگر نیازی به اتکا بر شبیه‌سازی نخواهد بود. شبیه‌سازی در مطالعه‌ی الگوریتم‌های تکاملی ضروری است، اما باید از آن برای حمایت از نظریات استفاده نمود.

مسائل نوشتاری

٤-١ چند شماتا با طول ۲ وجود دارد؟ چندتای آن‌ها از مرتبه‌ی ۰، چندتای آن‌ها از مرتبه‌ی ۱ و چندتای آن‌ها از مرتبه‌ی ۲ می‌باشند؟

٤-٢ چند شماتا با طول ۳ وجود دارد؟ چندتای آن‌ها از مرتبه‌ی ۰، چندتای آن‌ها از مرتبه‌ی ۱، چندتای آن‌ها از مرتبه‌ی ۲ و چندتای آن‌ها از مرتبه‌ی ۳ می‌باشند؟

٤-٣ چند شماتا از مرتبه‌ی l وجود دارد؟

الف) چندتای آن‌ها از مرتبه‌ی ۰ اند؟

ب) چندتای آن‌ها از مرتبه‌ی ۱ اند؟

ج) چندتای آن‌ها از مرتبه‌ی ۲ اند؟

د) چندتای آن‌ها از مرتبه‌ی ۳ اند؟

ه) چندتای آن‌ها از مرتبه‌ی p اند؟

٤-٤ فرض کنید نمونه‌های شمای h دارای مقادیر برازندگی هستند که ۲۵٪ از میانگین برازندگی جمعیت GA بزرگترند. فرض کنید احتمال تخریب h توسط جهش و برش قابل صرف نظر کردن است. فرض کنید GA با یک نمونه از h آغاز می‌شود. برای اندازه‌های جمعیت ۲۰،۵۰، ۱۰۰، ۲۰۰، شماره‌ی نسلی را که در آن h تمام جمعیت را در اختیار می‌گیرد را محاسبه کنید [گلدبرگ، 1989a].

[1] Price
[2] Poli

٤–٥ فرض کنید یک GA با ذرات دو بیتی داریم به‌طوری که احتمال جهش هر رشته بیت x_i به هر رشته بیت x_j برابر p_m باشد (برای تمام $j \neq i$ها). ماتریس جهش را بیابید. تحقیق کنید که جمع هر سطر برابر ۱ است.

٤–٦ فرض کنید یک GA با ذرات دو بیتی داریم به‌طوری که احتمال جهش یک بیت ۰ برابر p_0 و احتمال جهش یک بیت ۱ برابر p_1. ماتریس جهش را بیابید. تحقیق کنید که جمع هر سطر برابر ۱ است.

٤–٧ r_{2ij} در مثال ٤–٨ را برای $i \in [1,4]$ و $j \in [1,4]$ بیابید.

٤–٨ R_2 و $\widehat{R_2}$ در مثال ٤–١٣ را بیابید.

٤–٩ فرض کنید در نسل tاُم جمعیت کاملاً از ذرات بهینه تشکیل شده باشد. فرض کنید جهش به‌گونه‌ای پیاده‌سازی شده است که احتمال جهش یک ذره‌ی بهینه به هر ذره‌ی دیگری برابر ۰ است. از معادله‌ی (٤–٧٧) استفاده کرده و نشان دهید که جمعیت در نسل $(t+1)$اُم همچنان از ذرات بهینه تشکیل شده است.

مسائل کامپیوتری

٤–١٠ یک GA را در نظر بگیرید که در آن هر ذره از یک بیت تشکیل شده است. m_1 نشان‌دهنده‌ی تعداد نمونه‌های شِمای $h_1 = 1$ بوده و f_1 برازندگی آن را نشان می‌دهد. به همین ترتیب، m_0 نشان‌دهنده‌ی تعداد نمونه‌های $h_0 = 0$ بوده و f_0 برازندگی آن را نشان می‌دهد. فرض کنید جمعیت GA نامحدود بوده و از فرایند بازتولید و جهش (و نه برش!) استفاده می‌کند. یک رابطه‌ی بازگشتی برای $p(t)$، که نسبت $\frac{m_1}{m_0}$ را در tاُمین نسل مشخص می‌کند، به دست آورید [گلدبرگ، 1989a].

الف) فرض کنید GA با $p(0) = 1$ آغاز شده است. $p(t)$ را برای ۱۰۰ نسل اول رسم کنید. نرخ جهش برای نسبت برازندگی $\frac{f_1}{f_0} = 10, 2, 1.1$ برابر ۱۰٪ است.

ب) قسمت الف را برای نرخ جهش ۱٪ تکرار کنید.

ج) قسمت الف را برای نرخ جهش ۰٫۱٪ تکرار کنید.

د) در مورد نتایجی که به دست آوردید توضیح دهید.

٤–١١ ویژگی ٦ از قضیه‌ی ٤–٢ را برای ماتریس انتقال مثال ٤–١ تحقیق کنید.

٤–١٢ استادی امتحان‌های سختی برگزار می‌کند. ۷۰٪ از دانشجویان که در حال حاضر نمره‌ی A از این درس دارند بعد از هر امتحان به نمره‌ی B یا بدتر نزول می‌کنند. ۲۰٪ از دانشجویان که در حال حاضر نمره‌ی B از این درس دارند بعد از هر امتحان به A صعود می‌کنند. اگر تعداد بی‌نهایت امتحان برگزار شود، چند درصد از دانشجویان نمره‌ی A خواهند گرفت؟

٤–١٣ از معادلات (٤–٢٦) و (٤–٢٨) برای محاسبه‌ی تعداد جمعیت‌های ممکن در یک GA با ذرات ٦ بیتی و اندازه‌ی جمعیت ۱۰ استفاده کنید.

٤–١٤ مثال ٤–١٠ را با مقادیر برازندگی زیر تکرار کنید.

$$f(000) = 7, \quad f(001) = 2, \quad f(010) = 2, \quad f(011) = 4$$
$$f(100) = 2, \quad f(101) = 4, \quad f(110) = 4, \quad f(111) = 6$$

٤–١٥ مثال ٤–١٠ را با نرخ جهش ۱٪ تکرار کنید. احتمال راه‌حل‌های غیربهینه چه قدر است؟ نتایج را با نتایج به دست آمده از مثال ٤–١٠ مقایسه کرده و تفاوتشان را توضیح دهید.

٤–١٦ مثال ٤–٩ را تکرار کنید تنها با این تفاوت که اگر راه‌حل بهینه حاصل شد، آنگاه هیچ جهشی رخ ندهد. این موضوع در ماتریس جهش چه تاثیری خواهد گذاشت؟ احتمال راه‌حل‌های بهینه چه قدر خواهد بود؟ این نتایج را چگونه توضیح می‌دهید؟

فصل پنجم

برنامه‌نویسی تکاملی

پیش‌نیاز رفتار هوشمندانه موفقیت در پیش‌بینی یک محیط است.

لارنس فوگِل [فوگِل، ۱۹۹۹، صفحه‌ی ۳]

برنامه‌نویسی تکاملی (EP[1]) توسط لارنس فوگِل و همکارانش، اونز و جک والش، در دهه‌ی ۱۹۶۰ اختراع شد [فوگِل و همکاران، ۱۹٦٦]، [فوگِل، ۱۹۹۹]. یک EP جمعیتی از ذرات را رشد داده اما شامل بازترکیب نمی‌شود. ذرات جدید تنها توسط عمل جهش به وجود می‌آیند.

در اصل EP برای به تکامل رساندن ماشین‌های حالتِ متناهی (FSM[2]) اختراع شد. یک FSM، یک ماشین مجازی است که به ازای یک دنباله‌ی ورودی، یک دنباله‌ی خروجی ایجاد می‌کند. تولید دنباله‌ی خروجی علاوه بر دنباله‌ی ورودی به مجموعه‌ای از حالت‌ها و قوانین انتقال حالت نیز بستگی دارد. از نظر لارنس فوگِل پیش‌بینی، عنصر کلیدیِ هوش در است. بنابراین وی توسعه‌ی FSMهایی که می‌توانستند خروجی بعدی فرآیند را پیش‌بینی کنند را به‌عنوان قدمی کلیدی در توسعه‌ی هوش محاسباتی در نظر گرفت.

مروری بر فصل

قسمت ۵-۱ مروری از EP برای مسائل پیوسته به دست می‌دهد. اگرچه EP اصالتاً به‌گونه‌ای تعریف شده بود که بر روی دامنه‌ی گسسته عمل می‌کرد، امروزه اغلب بر روی دامنه‌های پیوسته پیاده‌سازی می‌شود. بخش ۵-۲ ماشین‌های حالتِ متناهی را معرفی کرده و نشان خواهد داد چگونه می‌توان با استفاده از EP آن‌ها را بهینه نمود. FSMها بسیار قابل توجه‌اند چرا که می‌توان از آن‌ها برای مدل‌سازی بسیاری از انواع مختلف سیستم‌ها مانند برنامه‌های کامپیوتری، الکترونیک دیجیتال، سیستم‌های کنترل و سیستم‌های طبقه‌بندی استفاده نمود.

بخش ۵-۳ روش اصلی فوگِل از EP برای مسائل گسسته را مورد بحث قرار خواهد داد. بخش ٤-۵ به بحث در مورد معمای زندانی، که یک مسئله‌ی نظریه بازیِ کلاسیک است، خواهد پرداخت. راه‌حل‌های معمای زندانی را می‌توان به‌صورت FSM ارائه نمود؛ بنابراین EPها قادرند راه‌حل‌های بهینه‌ی معمای زندانی را بیابند.

بخش ۵-۵ به بحث در مورد مسئله‌ی مورچه‌ی مصنوعی، که از EP برای به تکامل رساندن یک FSM استفاده نموده تا یک مورچه بتواند مسیری بهینه برای جمع‌آوری غذا بیابد، خواهد پرداخت.

---

[1] Evolutionary Programming
[2] Finite State Machines

۵–۱ برنامه‌نویسی تکاملی پیوسته

فرض کنید می‌خواهیم $f(x)$ را، که در آن x یک بردار n بعدی است، مینیمم کنیم. همچنین فرض کنید برای همه‌ی xها $f(x) \geq 0$ است. یک EP با یک جمعیت تولید شده‌ی اتفاقی از ذرات $\{x_i\}$، $i \in [1, N]$ آغاز می‌شود.

فرزندان را به‌صورت زیر ایجاد می‌کنیم:

$$x_i' = x_i + r_i\sqrt{\beta f(x_i) + \gamma}, \quad i \in [1, N] \qquad (۵–۱)$$

که در آن r_i یک بردار n بعدی اتفاقی است که هر عنصر آن از توزیع گاوسی با میانگین ۰ و واریانس ۱ تبعیت می‌کنند. β و γ پارامترهای تنظیم EP هستند. واریانس جهش x_i برابرست با $(\beta f(x_i) + \gamma)$. اگر $\beta = 0$ باشد، آنگاه دامنه‌ی میانگین جهش همه‌ی ذرات برابر خواهد بود. به‌طور معمول و در یک EP استاندارد، $\beta = 1$ و $\gamma = 0$ می‌باشد. در فصل ٦ خواهیم دید که یک EP با اندازه‌ی جمعیت ۱ با یک ES دو عضوی معادل است.

بررسی معادله‌ی (۵–۱) برخی نکات را در مورد پیاده‌سازی **EP** آشکار می‌سازد [باک، ۱۹۹٦، بخش ۲–۲]:

- اولاً، مقادیر هزینه‌ی $f(x)$ باید به‌گونه‌ای جابه‌جا شوند تا همیشه نامنفی باشند. این کار مشکل نیست اما کاری است که باید انجام شود.
- ثانیاً، β و γ باید تنظیم شوند. مقادیر پیش‌فرض آن‌ها $\beta = 1$ و $\gamma = 0$ می‌باشد اما دلیلی بر مؤثر فرض کردن این مقادیر وجود ندارد. برای مثال، فرض کنید مقادیر x_i دامنه‌ی بزرگی دارند و ما از مقادیر پیش‌فرض $\beta = 1$ و $\gamma = 0$ استفاده می‌کنیم. آنگاه مقدار جهش موجود در معادله‌ی (۵–۱) بسیار کوچک بوه و باعث می‌شود همگرایی یا به کندی صورت گیرد و یا اصلاً صورت نگیرد. از طرف دیگر، اگر مقادیر x_i دامنه‌ای بسیار کوچک داشته باشند، آنگاه مقادیر پیش‌فرض β و γ جهش‌هایی غیرمعقول را نتیجه داده و بدین ترتیب جهش باعث خواهد شد تا برخی مقادیر x_i خارج از دامنه قرار بگیرند.
- ثالثاً، اگر $\beta > 0$ باشد (که معمولاً همینطور است) و تمامی مقادیر هزینه زیاد باشند، آنگاه $(\beta f(x_i) + \gamma)$ برای تمامی x_iها تقریباً یکسان خواهد بود و این موضوع باعث خواهد شد تا تمام ذرات، مستقل از مقادیر هزینه‌شان، میزان جهش تقریباً یکسانی داشته باشند. حتی اگر ذره‌ای با استفاده

از جهش مفیدی هزینه‌اش را بهبود بخشد، این بهبود به احتمال زیاد توسط یک جهش مخرب خنثی خواهد شد.

- برای مثال، فرض کنید دامنه‌ی مقادیر هزینه از $f(x) = 1000$ تا $f(x) = 1100$ متغیر باشد. ذره‌ی x_1 به‌طور نسبی بسیار بهتر از ذره‌ی x_N است اما مقادیر هزینه به‌گونه‌ای مقیاس می‌شوند که هم x_1 و هم x_N با دامنه‌ی تقریباً یکسانی دچار جهش خواهند شد. با این حال، این مسئله مختص EP نیست. مقیاس کردن مقادیر تابع هزینه در مورد سایر الگوریتم‌های تکاملی نیز مصداق دارد که ما بعداً در بخش ۸-۷ به آن خواهیم پرداخت.

بعد از آن که معادله‌ی (۵-۱) N فرزند ایجاد کرد، $2N$ ذره خواهیم داشت: $\{x_i\}$ و $\{x'_i\}$. N ذره از این $2N$ ذره را برای تشکیل جمعیت نسل بعد انتخاب می‌شوند. یک الگوریتم EP پایه در شکل ۵-۱ خلاصه شده است.

پارامترهای غیر منفی *EP* را انتخاب کن (β, γ). β, γ به صورت اسمی برابر **1** و **0** هستند.
$\{x_i\} \leftarrow \{$جمعیت تولید شده به صورت اتفاقی$\}, i \in [1, N]$
تا زمانی که شرایط پایان برآورده نشده است:
هزینه‌ی $f(x_i)$ را برای هر ذره از جمعیت محاسبه کن
برای هر ذره از جمعیت مانند x_i، $i \in [1, N]$:
یک بردار اتفاقی مانند r_i را به گونه‌ای که هر عنصر آن در $N(0,1)$ باشد، تولید کن
$x'_i \leftarrow x_i + r_i\sqrt{\beta f(x_i) + \gamma}$
ذره‌ی بعد
بهترین N ذره از $\{x_i, x'_i\} \leftarrow \{x_i, x'_i\}$
نسل بعد

شکل ۵-۱ شبه کد بالا، طرح کلی یک برنامه‌ی تکاملی (EP) پایه را برای مینیمم‌سازی $f(x)$ نشان می‌دهد.

برای انتخاب ذرات نسل بعد از $\{x_i, x'_i\}$ راه‌های متفاوتی وجود دارد. شکل ۵-۱ نشان می‌دهد که این عمل به‌صورت قاطعانه صورت می‌پذیرد، یعنی بهترین N ذره از $\{x_i, x'_i\}$ انتخاب می‌شوند. با این حال، عمل انتخاب می‌تواند به‌صورت احتمالی نیز صورت بپذیرد. برای مثال، می‌توان از N بار چرخش چرخ رولت برای انتخاب ذرات از $\{x_i, x'_i\}$ استفاده نمود و یا از انتخاب مسابقه‌ای یا بسیاری روش‌های انتخاب دیگر بهره برد (بخش ۸-۷ را ببینید).

یک EP معمولاً به‌گونه‌ای نوشته می‌شود تا نه تنها راه‌حل‌های نامزد تکامل پیدا کنند بلکه واریانس جهش‌هایشان هم تکامل پیدا کند. این EP یک فرا-EP نام داشته و در شکل ۵-۲ خلاصه شده است. در یک

فرا–EP، هر ذره‌ی x_i با یک واریانس جهش، v_i، همبستگی دارد. خودِ واریانس‌های جهش برای پیدا کردن بهترین واریانسِ جهش، دچار جهش می‌شوند. ما در شکل ۵–۲ واریانس جهش را به یک مقدار مینیمِ ε محدود ساخته‌ایم. این ε توسط کاربر تعیین می‌شود. یک فرا–EP با تعدیلِ واریانس‌های جهش به‌صورت خودکار سرعت همگرایی را افزایش می‌دهد، اما از سویی نیز بسته به مسئله، می‌توانند باعث کاهش سرعت همگرایی شوند.

پارامترهای غیر منفی ***EP*** را انتخاب کن (ϵ, c). مقادیر اسمی برابرند با $\epsilon \ll 1$ و $c = 1$.

$\{x_i\} \leftarrow \{$جمعیت تولید شده به صورت اتفاقی$\}, i \in [1, N]$

$\{v_i\} \leftarrow \{$واریانس تولید شده به صورت اتفاقی$\}, i \in [1, N]$

تا زمانی که شرایط پایان برآورده نشده است:

هزینه‌ی $f(x_i)$ را برای هر ذره از جمعیت محاسبه کن

برای هر ذره از جمعیت مانند x_i، $i \in [1, N]$:

بردارهای اتفاقی r_{xi} و r_{vi} را به گونه‌ای که هر عنصر آن در $N(0,1)$ باشد، تولید کن

$$x_i' \leftarrow x_i + r_{xi}\sqrt{v_i}$$
$$v_i' \leftarrow v_i + r_{vi}\sqrt{cv_i}$$
$$v_i' \leftarrow \max(v_i', \epsilon)$$

ذره‌ی بعد

بهترین N ذره از $\{x_i, x_i'\} \leftarrow \{x_i\}$

واریانس‌هایی که متناظر با $\{x_i\}$ هستند $\leftarrow \{v_i\}$

نسل بعد

شکل ۵–۲ شبه کد بالا یک فرا–EP را برای مینیمم ساختن $f(x)$ به تصویر می‌کشد. توجه کنید که برای تمام $i \in [1, N]$، v_i با ذره‌ی x_i همبستگی دارد.

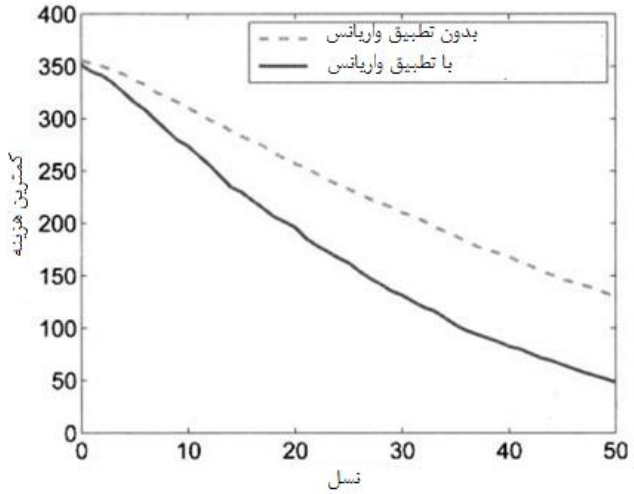
مثال ۵–۱

در این مثال از یک EP برای بهینه‌سازی توابع اَکلی و گرینوانک[1] استفاده می‌کنیم (برای اطلاع از این دو تابع محک به ضمیمه‌ی ج مراجعه کنید). فرض می‌کنیم هر یک از این توابع ۲۰ بعدی هستند. EP استانداردِ شکل ۵–۱ را با $\beta = \frac{x_{max} - x_{min}}{10}$ و $\gamma = 0$ اجرا می‌کنیم. اندازه‌ی جمعیت را برابر ۵۰ در نظر گرفته و مقادیر هزینه‌ی تمام ذرات را به گونه‌ای نرمالیزه می‌کنیم که در هر نسل $f(x_i) \in [1,2]$ باشد.

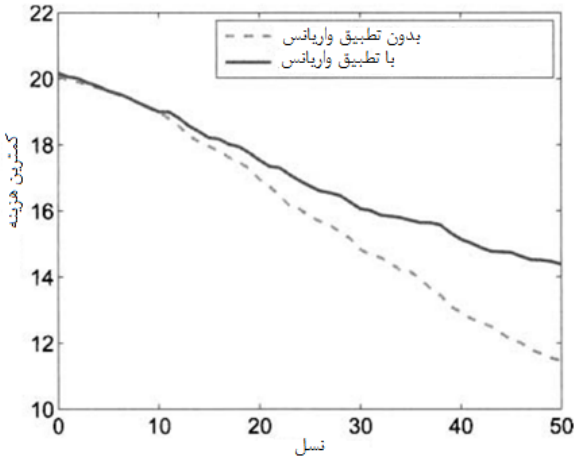
ما همچنین از فرا–EP شکل ۵–۲ با $c = 1$ و $\varepsilon = \frac{\beta}{10}$ استفاده می‌کنیم. شکل‌های ۵–۳ و ۵–٤ میانگین مینیمم هزینه‌ی جمعیت در طول ۲۰ بار شبیه‌سازیِ مونت کارلو را به‌صورت تابعی از شماره‌ی نسل نشان

[1] Grienwank

می‌دهند. می‌توان دید که در مورد تابع گرینوانک، فرا–EP بسیار بهتر از EP استاندارد همگرا شده و در مورد تابع آکلی فرا–EP بسیار بدتر از EP استاندارد همگرا می‌شود. این موضوع بی‌شک به دامنه‌های متفاوتِ دو تابع مربوط است. دامنه‌های متغیر مستقل در گرینوانک ± 600 می‌باشد، در حالی که دامنه‌های هر متغیر مستقل در تابع آکلی تنها ± 36 است. همچنین مقایسه‌ی عددهای درج شده بر روی محور افقی هر دو شکل نشان دهنده‌ی تفاوت بسیار زیاد دو تابع می‌باشد.



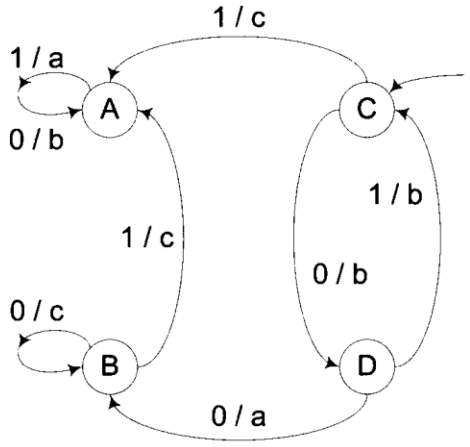
شکل ۵–۳ مثال ۵–۱: همگرایی EP برای تابع گرینوانک ۲۰ بعدی که بر روی ۲۰ بار شبیه‌سازی میانگین‌گیری شده است. فرا EP بسیار سریعتر از EP استاندارد همگرا می‌شود.



شکل ۵–٤ مثال ۵–۱: همگرایی EP برای تابع آکلی ۲۰ بعدی که بر روی ۲۰ بار شبیه‌سازی میانگین‌گیری شده است. EP استاندارد بسیار سریعتر از فرا–EP همگرا می‌شود.

۵–۲ بهینه‌سازی ماشینِ حالت متناهی

EP در اصل برای به تکامل رساندن ماشین‌های حالت متناهی (FSM) به وجود آمد. یک FSM دنباله‌ای از خروجی‌ها را به‌صورت تابعی از یک حالت داخلی و دنباله‌ای از ورودی‌ها تولید می‌کند. شکل ۵–۵ نمونه‌ای از یک FSM را نشان می‌دهد. این ماشین چهار حالت A، B، C و D دارد و ورودی‌های ممکن برای آن ۰ و ۱ هستند. ورودی‌ها در سمتِ چپِ هر خط مورب نشان داده شده‌اند. خروجی‌های محتمل این ماشین a، b و c بوده که در سمت راست خط مورب نمایش داده شده‌اند. پیکانِ موجود در سمت راست و بالای شکل نشان دهنده‌ی آن است که FSM با حالت C آغاز می‌شود. پیکان‌ها نشان دهنده‌ی چگونگی انتقال حالت‌ها بعد از تزریق یک ورودی خاص به ماشین می‌باشند. شکل ۵–۵ را می‌توان به‌صورت جدولی نیز نشان داد که این کار در جدول ۵–۱ انجام شده است.



شکل ۵–۵ ماشینِ حالت متناهیِ جدول ۵–۱ که به‌صورت دیاگرامی نشان داده شده است. این FSM چهار حالت دارد. زوج نشان داده شده در کنار هر پیکان، ورودی و خروجیِ متناظر با آن را نشان می‌دهند اگر FSM در حالتِ نشان داده شده در انتهای پیکان قرار داشته باشد. پیکانِ بالا–راست، نشان‌دهنده‌ی حالت آغازین FSM، C، است.

جدول ۵–۱ ماشین حالت متناهیِ شکل ۵–۵ به فرم جدولی.

A 0	A 1	B 0	B 1	C 0	C 1	D 0	D 1	ورودی حالت فعلی
A b	A a	B c	A c	D b	A c	B a	C b	خروجی حالت بعدی

فرض کنید می‌خواهیم یک FSM ایجاد کنیم که از دنباله‌ی ورودیِ مشخصی، دنباله‌ی خروجی مشخصی را ایجاد می‌کند. برای مثال می‌دانیم که دنباله‌ی ورودی

$$\text{ورودی} = \{1,0,1,0,1,0,0,1,1,0,1,0\} \quad (۲-۵)$$

باید دنباله‌ی خروجی

$$\text{خروجی} = \{0,0,1,1,1,1,0,1,1,0,0,1\} \quad (۳-۵)$$

را تولید کند.

آیا می‌توان ماشینِ حالتی را ایجاد کرد که رفتار مشخصی را از خود نشان دهد؟ این یک مسئله‌ی بهینه‌سازی است: می‌خواهیم یک FSM را به‌گونه‌ای به تکامل برسانیم که اختلافِ میان رفتار FSM و رفتار مطلوب به حداقل برسد. می‌توان یک ماشینِ حالت را به شکل زیر ارائه نمود

$$S = [\left(\text{خروجی}_0 \quad \text{حالت بعد}_0\right) \left(\text{خروجی}_1 \quad \text{حالت بعد}_1\right) \dots]^T \quad (۴-۵)$$

بی‌آنکه از جامعیت مطلب کاسته شود، فرض می‌کنیم FSM با حالت ۱ آغاز شود. عناصرِ S به ترتیب زیر در خواهند آمد

$$\begin{aligned}
&S(1) = \text{خروجی در حالتی که FSM در حالت 1 بوده و ورودی 0 است} \\
&S(2) = \text{حالت بعد در صورتی که FSM در حالت 1 بوده و ورودی 0 است} \\
&S(3) = \text{خروجی در حالتی که FSM در حالت 1 بوده و ورودی 1 است} \\
&S(4) = \text{حالت بعد در صورتی که FSM در حالت 1 بوده و ورودی 1 است} \\
&\dots\dots \\
&S(4n) = \text{حالت بعد در صورتی که FSM در حالت } n \text{ بوده و ورودی 1 است}
\end{aligned} \quad (۵-۵)$$

که در آن فرض کرده‌ایم ورودی دودویی است. می‌توان این ساختار را به راحتی برای ورودی‌های غیر دودویی نیز بسط داد. مشاهده می‌شود که S یک بردار ستونی با $4n$ عنصر است که FSM را توصیف کرده و n در آن تعداد حالت‌ها می‌باشد. می‌توانیم ورودی داده شده در معادله‌ی (۵-۲) را به یک FSM اعمال کنیم و هزینه‌ی خطای FSM را به‌صورت زیر تعریف کنیم

$$\text{هزینه} = \sum_{i=1}^{12} | \left(\text{هزینه مطلوب}\right)_i - (FSM \text{ خروجی})_i | \quad (۶-۵)$$

که در آن i(خروجی مطلوب)، iاُمین خروجی معادله‌ی (۵–۳) بوده و یک دنباله از ۱۲ خروجی موجود است. حال می‌توانیم از الگوریتم EP موجود در شکل ۵–۱ یا ۵–۲ برای تکامل FSM و مینیمم‌سازی معادله‌ی (۵–۶) استفاده کنیم.

یکی از جزئیات پیاده‌سازی که باید آن را در نظر بگیریم آن است که در شکل ۵–۱، x_i یک متغیر پیوسته است، اما در تکامل FSM، عناصرِ هر یک از ذرات اعداد صحیحی هستند که به یک دامنه‌ی خاص محدود می‌باشند. معادله‌ی (۵–۵) این مطلب را برای یک FSM با خروجی‌های دودویی نشان می‌دهد، $S(i) \in [0,1]$ برای مقادیر فرد i و $S(i) \in [1,n]$ برای مقادیر زوج i که در آن n تعدادِ حالت‌ها می‌باشد. این موضوع را می‌توان به سادگی و با انجام جهش‌های نشان داده شده در شکل ۵–۱ و سپس محدود کردن عناصرِ x_i' به یک دامنه‌ی مناسب و در آخر گرد کردن عناصرِ x_i' به سمت نزدیک‌ترین عدد صحیح، مدیریت نمود. در اینجا نیز متذکر می‌شویم که راه‌حل‌های دیگری برای مدیریت این مشکل وجود دارند که تحقیق این راه‌حل‌ها به خلاقیت خواننده واگذار می‌شود.

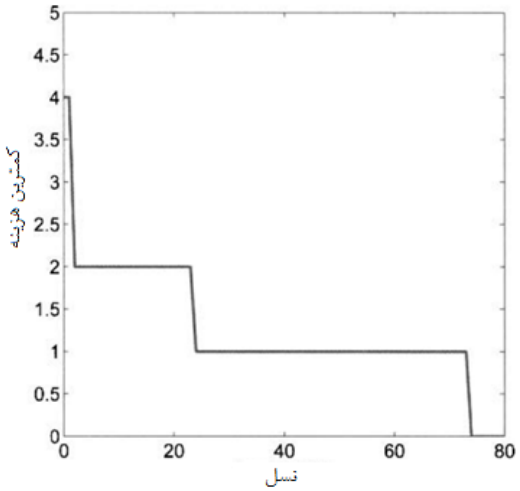
سایر جزئیات پیاده‌سازی شامل تنظیم پارامترهای β و γ و همچنین مقیاس کردن مقادیر هزینه‌ی $f(x_i)$ به شیوه‌ای مناسب، می‌شود.

مثال ۵–۲

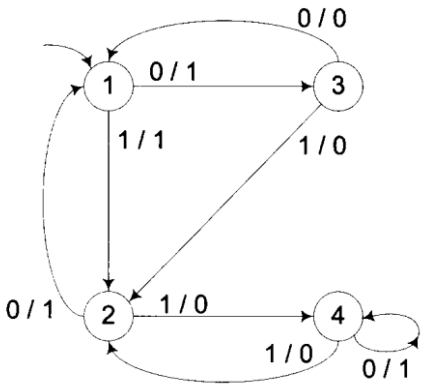
در این مثال ما از EP برای تکامل یک FSM جهت مینیمم ساختن تابع هزینه‌ی معادله‌ی (۵–۶) استفاده می‌کنیم. ورودی و خروجی در معادلات (۵–۲) و (۵–۳) نشان داده شده‌اند. ما در هر ذره‌ی EP از چهار حالت استفاده می‌کنیم. $\beta = 1$ و $\gamma = 0$ بوده و اندازه‌ی جمعیت برابر ۵ می‌باشد. همچنین از الگوریتم EP شکل ۵–۱ برای تکامل جمعیت FSM بهره خواهیم برد. هزینه‌ی هر یک از ذرات را به‌گونه‌ای مقیاس می‌کنیم که در هر نسل هزینه‌ها در بازه‌ی $[1,2]$ واقع شوند. شکل ۵–۶ همگرایی تابع هزینه برای یک بار شبیه‌سازی EP را نشان می‌دهد. یک بردارِ S که هزینه‌ی صفر را به دست خواهد داد به‌صورت زیر می‌باشد

$$S = [1 \quad 3 \quad 1 \quad 2, \quad 1 \quad 1 \quad 0 \quad 4, \quad 0 \quad 1 \quad 0 \quad 2, \quad 1 \quad 4 \quad 0 \quad 2] \qquad (۵–۷)$$

که متناظر FSM نشان داده شده در شکل ۵–۷ می‌باشد.



شکل ۵–٦ مثال ۵–۲: همگرایی ماشینِ حالت متناهی.



شکل ۵–۷ ماشینِ حالت متناهیِ تکامل یافته توسط EP مثال ۵–۲. اگر ورودی این ماشین توسط معادله‌ی (۵–۲) تعیین شود، آنگاه خروجی آن توسط معادله‌ی (۵–۳) به دست خواهد آمد.

۵–۳ برنامه‌نویسی تکاملی گسسته

پیاده‌سازی EP اختراع شده توسط فوگِل برای FSMها از آنچه که ما در قسمت قبل توصیف نمودیم، متفاوت بوده است [فوگِل و همکاران، ۱۹٦٦]، [فوگِل، ۱۹۹۹]. پیاده‌سازی وی مستقیماً قابل اعمال به دامنه‌های گسسته بود. شیوه‌ی وی را علاوه بر بهینه‌سازی FSM می‌توان برای هر مسئله‌ی تعریف شده بر روی یک دامنه‌ی گسسته به کار برد. خلاصه‌ای از این شیوه را در شکل ۵–۸ نشان داده‌ایم.

$\{x_i\} \leftarrow \{$جمعیت تولید شده به صورت اتفاقی$\}, i \in [1, N]$
تا زمانی که شرایط پایان برآورده نشده است:
هزینه‌ی $f(x_i)$ را برای هر ذره از جمعیت محاسبه کن
برای هر ذره از جمعیت مانند x_i، $i \in [1, N]$:
جهش اتفاقی از $x_i \leftarrow x_i'$
ذره‌ی بعد
بهترین N ذره از $\{x_i, x_i'\} \leftarrow \{x_i\}$
نسل بعد

شکل ۵-۸ شبه کد بالا برنامه‌ی تکاملی فوگِل را برای مسائل بهینه‌سازی گسسته نشان می‌دهد. توجه داشته باشید که این الگوریتم تعمیم شکل ۵-۱ است.

"جهش اتفاقی" موجود در شکل ۵-۸ کاملاً به مسئله‌ای بستگی دارد که در صدد حل آن هستیم. به‌عنوان مثال، جهش اتفاقیِ مورد استفاده‌ی فوگِل برای بهینه‌سازی FSM، به‌صورت اتفاقی از یکی از موارد زیر انتخاب می‌شد:

- اضافه کردن یک حالت با زوج‌های خروجی/ورودی و انتقال/ورودی اتفاقی
- حذف یک حالت. هرگونه انتقال به حالت حذف شده به‌صورت اتفاقی به حالت دیگری هدایت می‌شود.
- عوض کردنِ اتفاقی زوجِ خروجی/ورودی برای یک حالت انتخاب شده به‌صورت اتفاقی.
- عوض کردنِ اتفاقی زوجِ انتقال/ورودی برای یک حالت انتخاب شده به‌صورت اتفاقی.
- تعویض اتفاقی حالت آغازین.

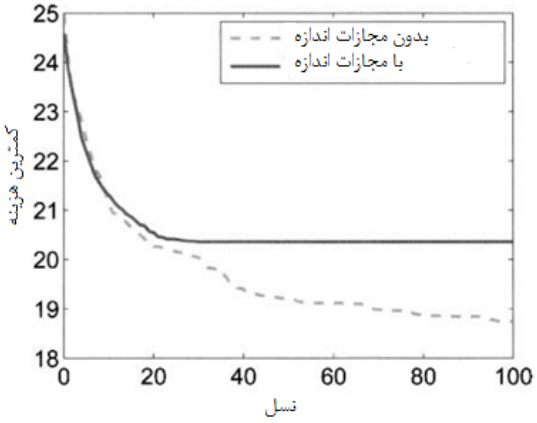
فوگِل همچنین پیشنهاد اضافه کردن یک مجازات متناسب با پیچیدگی ماشین حالت به تابع هزینه را مطرح کرد. این کار انتخاب بهترین N ذره در پایان هر نسل را به سمت ماشین‌های حالتِ ساده‌تر گرایش می‌دهد. این ایده به ما این امکان را می‌دهد تا علاوه بر پیدا کردن FSMهایی که الگوی مطلوب را تولید می‌کنند، FSMهایی با ساختار ساده‌تر به دست آوریم.

مثال ۵-۳

در این مثال تلاش خواهیم کرد ماشین حالتی بیابیم که می‌تواند اعداد اول تولید کند. ما از ۰ برای نشان دادن اعداد غیراول و از ۱ برای نشان دادن اعداد اول استفاده می‌کنیم. ورودی ماشین حالت در هر گامِ زمانی شاخصِ اول گام زمانیِ قبلی است (۰ برای غلط و ۱ برای صحیح). با این کار دنباله‌های ورودی و خروجی به‌صورت زیر به دست می‌آیند

$$\text{ورودی} = \{0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, \dots\}$$
$$\text{خروجی مطلوب} = \{1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, \dots\} \quad (۸-۵)$$

دنباله‌ی ورودی متناظر با این حقیقت است که ۱ عدد اول نیست، ۲ و ۳ اعداد اول‌اند، ٤ عدد اول نیست، ۵ عدد اول است، ٦ عدد اول نیست، ۷ عدد اول است، ۸ و ۹ و ۱۰ اول نیستند، ۱۱ عدد اول است، ۱۲ عدد اول نیست و الی آخر. دنباله‌ی خروجی با یک گام تأخیر زمانی با دنباله‌ی ورودی برابرست. ما از ۱۰۰ عدد صحیح مثبت آغازین برای ارزیابی عملکرد یک FSM استفاده می‌کنیم. در این صورت هر یک از دنباله‌های ورودی و خروجی طولی برابر ۹۹ بیت خواهند داشت. دنباله‌ی ورودی با اعداد صحیح ۹۹ — ۱ متناظر بوده و دنباله‌ی خروجی با اعداد صحیح ۱۰۰ — ۲ متناظر خواهد بود. اندازه‌ی جمعیت را در این مثال برابر ۲۰ در نظر می‌گیریم ($N = 20$). برای هر ذره در هر نسل از EP، یکی از ۵ جهشی که قبلاً در این بخش توصیف شد را به‌صورت اتفاقی در نظر می‌گیریم. ما EP را با مجازاتِ هزینه و همچنین بی‌مجازاتِ هزینه شبیه‌سازی می‌کنیم. شکل ۵–۹ بهترین FSM برای هر نسل، که از آن‌ها بر روی ۱۰۰ شبیه‌سازی مونت کارلو میانگین گرفته شده است، را نشان می‌دهد. اگر مجازاتی در کار نباشد، میانگین تعداد حالت‌ها ٤,۷ بوده و اگر مجازاتِ هزینه‌ای برابر $\frac{n}{2}$ برای تعداد حالت‌ها در نظر گرفته شود، میانگین تعداد حالت‌ها برابر ۲,۸ خواهد بود. n تعداد حالت‌هاست. شکل ۵–۹ نشان می‌دهد که بهترین هزینه‌ی قابل دسترسی چیزی بین ۱۸ و ۱۹ است که این مقدار، با توجه به آنکه میانِ ۱۰۰ عدد صحیح ابتدایی تنها ۲۵ عدد اول وجود دارد، چنگی به دل نمی‌زند. ماشین حالتی که همواره ۰ تولید می‌کند هزینه‌ای برابر ۲۵ خواهد داشت.



شکل ۵–۹ مثال ۵–۳: همگرایی ماشینِ حالت متناهی برای پیش‌بینی اعداد اول که ار آن بر روی ۱۰۰ شبیه‌سازی مونت کارلو میانگین گرفته شده است.

۵-۴ معمای زندانی

معمای زندانی یک نظریه‌ی بازی کلاسیک است. فرض کنید که دو مظنون دستگیر شده‌اند. پلیس از هر یک از دو مظنون به‌طور جداگانه بازجویی می‌کند. پلیس به هر یک از مظنونین پیشنهاد می‌دهد که اگر همدستشان (مظنون دیگر) را لو دهند از محاکمه در امان خواهند بود. اگر هر یک از مظنونین اعتراف کند، پلیس مدرک کافی برای به زندان انداختن همدست وی برای مدتی طولانی در دست خواهد داشت. با این حال، اگر هر دو مظنون سکوت اختیار کنند، پلیس مدرک کافی برای به زندان انداختن آن‌ها برای مدت طولانی نخواهد داشت. در اینجا یک معما برای دو زندانی، که حق ارتباط با یکدیگر را ندارند، پیش خواهد آمد. اگر هر دو زندانی ساکت بمانند (با یکدیگر تعامل کنند) هر دوی آن‌ها آزاد می‌شوند (به آن‌ها حکم تعویق داده می‌شود). اگر هر دوی آن‌ها بر علیه دیگری صحبت کنند (یکدیگر را لو بدهند) هر دوی آن‌ها به زندان می‌افتند (حکم متوسط دریافت خواهند کرد). اگر یکی دیگری را لو داده و دیگری ساکت بماند، آن کس که لو داده است آزاد خواهد شد و آن کس که همکاری کرده و ساکت مانده حکمی طولانی دریافت خواهد کرد. این معمای زندانی در جدول ۵-۲ خلاصه شده است.

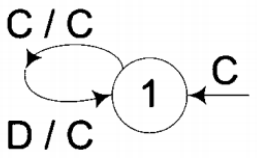
جدول ۵-۲ ماتریس هزینه‌ی معمای زندانی.

	زندانی B همکاری می‌کند	زندانی B همکاری نمی‌کند
زندانی A همکاری می‌کند	زندانی A: ۱ سال زندانی B: ۱ سال	زندانی A: ۱۰ سال زندانی B: آزاد
زندانی A همکاری نمی‌کند	زندانی A: آزاد زندانی B: ۱۰ سال	زندانی A: ۵ سال زندانی B: ۵ سال

فرض کنید شما زندانی A هستید. اگر همدست شما با شما همکاری کرده و ساکت بماند شما می‌توانید با لو دادن وی آزاد شوید. اگر همدستِ شما، شما را لو دهد، آنگاه می‌توانید با لو دادن وی به جای ۱۰ سال، ۵ سال به زندان بروید. بنابراین به نظر می‌رسد بی‌توجه به آنکه همدست شما چه خواهد کرد، شما باید وی را لو بدهید. با این حال، اگر هر دو زندانی از این استراتژی استفاده کنند به مدت ۵ سال محکوم خواهند شد. اگر هر دو زندانی با یکدیگر تعامل کنند آنگاه هر دو تنها ۱ سال محکوم خواهند شد. تصمیمات خودخواهانه می‌توانند وضعیت هر دو زندانی را از حالتی که به نفع یکدیگر کار کنند بسیار بدتر کند. به همین علت است که این مسئله یک معما در نظر گرفته می‌شود.

در معمای زندانیِ دوره‌ای، بازی معمای زندانی چندین بار بازی می‌شود و در آن هدف هر بازیکن ماکزیمم کردن سودِ کلش است (این کار در واقع یعنی مینیمم کردن مدت زمان کلِ زندانی بودن آن بازیکن) [اَکسِلُرد[1]، ۲۰۰۶]. هر بازیکن در هر دوره از تصمیمِ بازیکنِ مقابل در دوره‌های قبلی خبر داشته و با توجه به آن تصمیم به همکاری یا لو دادن در دوره‌های بعدی می‌گیرد. بنابراین، اگر بازیکنِ مقابل مکرراً تصمیم به لو دادن شما داشته باشد، شما می‌توانید با تصمیم به لو دادن وی، سودِ خود را ماکزیمم کنید. اگر همدست شما همکاری کند، شما نیز می‌توانید با تصمیم به همکاریِ متقابل، میزان سودِ متقابل را افزایش دهید. واژه‌ی "دوره‌ای" معمولاً از عبارت "معمای زندانی دوره‌ای" حذف می‌شود. بنابراین عبارت "معمای زندانی" می‌تواند بر یک یا چند نوبت از جدول ۵-۲ اشاره داشته باشد.

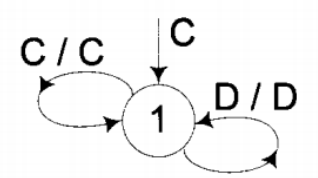
چندین استراتژی برای معمای زندانی ارائه شده است که هر یک از آن‌ها را می‌توان به راحتی به‌صورت یک FSM ارائه کرد [اَشلاک[2]، ۲۰۰۹] و [روبین- اِستین[3]، ۱۹۸۶]. یک استراتژی آن است که همیشه همکاری کنیم. این استراتژی خوشبینانه در شکل ۵-۱۰ با یک FSM تک حالته نشان داده شده است. ما بازی را با همکاری در نوبت اولمان و رفتن به حالت ۱ آغاز می‌کنیم. اگر تصمیمِ قبلی حریفمان C بوده است آنگاه ما C را به خروجی داده و در حالت ۱ باقی می‌مانیم. اگر تصمیم قبلی حریف D بوده، ما باز C را به خروجی داده و در حالت ۱ باقی می‌مانیم.



شکل ۵-۱۰ ماشینِ حالت متناهی برای استراتژی همواره- همکاری در معمای زندانی.

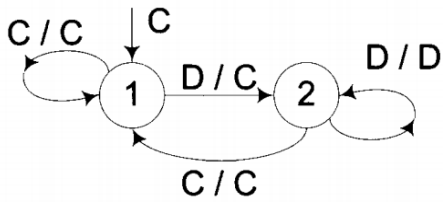
یک استراتژی دیگر استراتژی "این به آن در" است. فلسفه‌ی این استراتژی جمله‌ی "همان کاری را در حق دیگران بکن که دیگران در حق تو می‌کنند" است. ما بازی را با همکاری در نوبت اول شرع می‌کنیم و به حالت ۱ می‌رویم. اگر حریفمان در حرکتِ قبلی خود همکاری کرد ما نیز همکاری می‌کنیم و اگر حریف در حرکت قبلی‌اش ما را لو داد، ما نیز وی را لو می‌دهیم. این استراتژی در شکل ۵-۱۱ نشان داده شده است.

[1] Axelord
[2] Ashlock
[3] Rubin-Stein



شکل ۵-۱۱ ماشینِ حالت متناهی برای استراتژی این به آن در! در معمای زندانی.

یک استراتژی دیگر "این به آن دو در" نام دارد. این استراتژی نسبت به استراتژی "این به آن در" کمی بخشنده‌تر و امیدوارانه‌تر است. در این نوع استراتژی ما همکاری می‌کنیم مگر آنکه حریف برای دو نوبت متوالی ما را لو دهد. این استراتژی در شکل ۵-۱۲ نشان داده شده است.



شکل ۵-۱۲ ماشینِ حالت متناهی برای استراتژی این به آن دو در، در معمای زندانی.

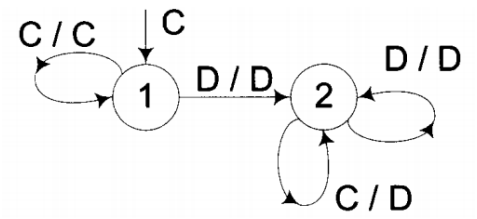
استراتژی شوم بسیار نابخشنده است. در این استراتژی ما خوشبینانه در نوبت اول همکاری می‌کنیم و به حالت ۱ می‌رویم و تا زمانی که حریف همکاری می‌کند ما نیز همکاری می‌کنیم. اما اگر حریف یکبار ما را لو داد، دیگر هیچ وقت همکاری نمی‌کنیم. این استراتژی در شکل ۵-۱۳ نشان داده شده است.

در استراتژی تنبیه، ما از حریف به دلیل لو دادن کمی انتقام می‌گیریم اما در نهایت وی را می‌بخشیم. اگر حریف ما را لو داد ما نیز وی را لو می‌دهیم و این کار را ادامه می‌دهیم تا آنکه حریف سه بار پشت سر هم همکاری کند. بعد از آنکه حریف سه بار متوالی همکاری کرد، ما دوباره شروع به همکاری می‌کنیم. این استراتژی در شکل ۵-۱٤ نشان داده شده است.

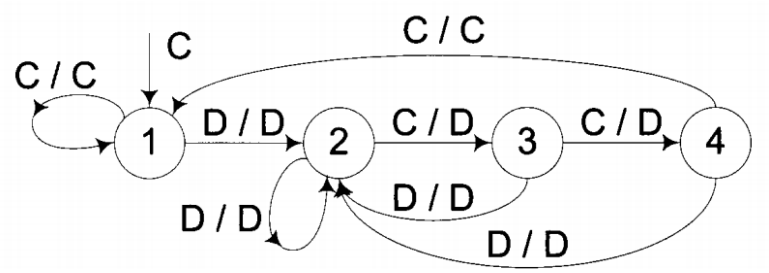
مطالعات زیادی بر روی معمای زندانی انجام شده است چرا که کاربردهای بسیار زیادی دارد. از جمله‌ی این کاربردها می‌توان به اشتراک گذاری فایل به‌صورت نظیر به نظیر[1][الیس[2] و یاو[3]، ۲۰۰۷]، استراتژی تبلیغات

[1] Peer-to-Peer File Sharing
[2] Ellis
[3] Yao

میان شرکت‌های رقیب [کُرفمَن[1] و لِهمَن[2]، ۱۹۹٤]، سیاست [گریکو[3]، ۱۹۸۸]، تقلب در ورزش و سایر زمینه‌ها [اِهرنبورگ[4] و روزِن[5]، ۲۰۰۹] و بسیاری دیگر [پونداِستون[6]، ۱۹۹۳] اشاره کرد.



شکل ۵-۱۳ ماشینِ حالت متناهی برای استراتژی شوم در معمای زندانی.



شکل ۵-۱٤ ماشینِ حالت متناهی برای استراتژی تنبیه در معمای زندانی.

مثال ۵-۱٤

در این مثال ما یک FSM را برای مینیمم کردن هزینه در یک معمای زندانی بهینه می‌کنیم. FSM برای معمای زندانی را می‌توان به‌صورت معادله‌ی (۵-۵) نشان داد تنها با این تفاوت که ما یک عدد صحیح را برای نشان دادن اولین حرکت به ابتدای آن اضافه می‌کنیم. ما از ۰ برای نشان دادن همکاری و از ۱ برای نشان دادن لو دادن استفاده می‌کنیم. چهار حریف اتفاقی ولی ثابت، که هر کدام یک استراتژی FSM ٤-حالته دارند، در نظر می‌گیریم. یک EP با $\beta = 1$ و $\gamma = 0$ اجرا می‌کنیم. اندازه‌ی جمعیت EP را برابر ۵ فرض کرده و جمعیت را به‌صورت اتفاقی ایجاد می‌کنیم. هر ذره شامل چهار حالت است. هر ذره‌ی EP مقابل هر یک از چهار حریف اتفاقی ولی ثابت، ۱۰ بار بازی می‌کند تا بتوان عملکردش را ارزیابی کرد. در این مثال ماشین‌های حالت برای حریف‌ها عبارتند از

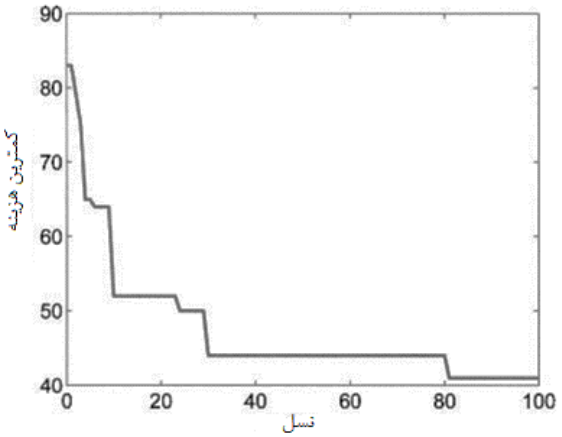
[1] Corfman
[2] Lehmann
[3] Grieco
[4] Ehrnborg
[5] Rosen
[6] Poundstone

$$
\begin{aligned}
S_1 &= [0,\ 0\ 3\ 0\ 3,\ 0\ 3\ 1\ 2,\ 1\ 4\ 0\ 2,\ 1\ 3\ 1\ 3] \\
S_2 &= [0,\ 0\ 1\ 0\ 4,\ 1\ 2\ 0\ 2,\ 0\ 4\ 0\ 3,\ 0\ 2\ 0\ 4] \\
S_3 &= [0,\ 1\ 4\ 0\ 4,\ 0\ 1\ 0\ 1,\ 1\ 4\ 0\ 2,\ 0\ 1\ 1\ 4] \\
S_4 &= [1,\ 0\ 4\ 1\ 4,\ 0\ 4\ 0\ 1,\ 0\ 3\ 0\ 3,\ 0\ 3\ 0\ 2]
\end{aligned}
\qquad (۹-۵)
$$

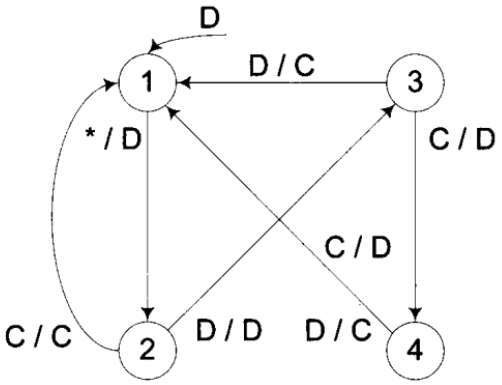
ما از الگوریتم شکل ۵-۱ برای رشد دادن جمعیت FSM استفاده می‌کنیم. شکل ۵-۱۵ همگرایی تابع هزینه‌ی EP را برای یک بار شبیه‌سازی نشان می‌دهد. بردار S، که کمترین هزینه را به دست می‌دهد، به قرارِ زیر به دست می‌آید

$$S = [1,\ 1\ 2\ 1\ 2,\ 0\ 1\ 1\ 3,\ 1\ 4\ 0\ 1,\ 1\ 1\ 0\ 1] \qquad (۱۰-۵)$$

که با FSM نشان داده شده در شکل ۵-۱۶ متناظر است



شکل ۵-۱۵ مثال ۵-٤: همگرایی هزینه‌ی ماشینِ حالت متناهی معمای زندانی.



شکل ۵-۱۶ مثال ۵-٤: بهترین ماشینِ حالت متناهی به تکامل رسیده توسط EP. ستاره‌ی بیرون آمده از حالت ۱ به معنای C یا D می‌باشد. این یعنی اگر FSM در حالت ۱ بوده، آنگاه بدون توجه به آنکه حرکت قبلی حریف چه بوده (C یا D)، خروجی FSM برابر D بوده و حالت بعدی ۲ خواهد بود.

یک آزمایش جالب که می‌توانیم انجام دهیم آن است که جمعیت EP با بازی در مقابل خودش به تکامل برسد. این بدان معنی است که هر ذره‌ی EP در مقابل تمام ذرات دیگرِ موجود در EP بازی کرده و بدین ترتیب هزینه‌ی ذرات ارزیابی می‌شوند.

انواع مختلفی از معمای زندانی وجود دارد. برای مثال، ما فرض کردیم هر ذره در هر نوبت یکی از دو حرکتِ ممکن را انتخاب کند. با این حال، می‌توانیم برای میزان همکاری درجات مختلفی قائل شویم به‌طوری که هر نوبت شامل زنجیره‌ای از حرکات و هزینه‌های مرتبط با آن‌ها باشد. در این حالت ۰ نشانگر همکاری کامل و ۱ نشانگر لو دادن کامل می‌شوند و هر عددی بین این دو می‌تواند نشانگر درجات متغیری از همکاری و لو دادن باشد [هَرالد[1] و فوگِل، ۱۹۹۶]. یک نوع دیگر معمای زندانی آن است که به هر ذره اجازه دهیم در هر زمان که خواست بازی را متوقف کند [دیلاهی[2]و متیو[3]، ۱۹۹۵]. یک نوع دیگر آن است که اجازه دهیم بیش از دو بازیکن به‌صورت همزمان بازی کنند. در چنین حالتی، هزینه‌ی هر بازیکن معمولاً تابعی است از حرکات آن بازیکن و تعدادِ حریفانی که اقدام به همکاری کرده‌اند [بوناکیچ[4] و همکاران، ۱۹۷۶]. نوع دیگر نیز آن است که ماتریس هزینه‌ی ۵,۲ با زمان تغییر کند [وُردِن[5]و لِوین[6]، ۲۰۰۷].

۵–۵ مسئله‌ی مورچه‌ی مصنوعی

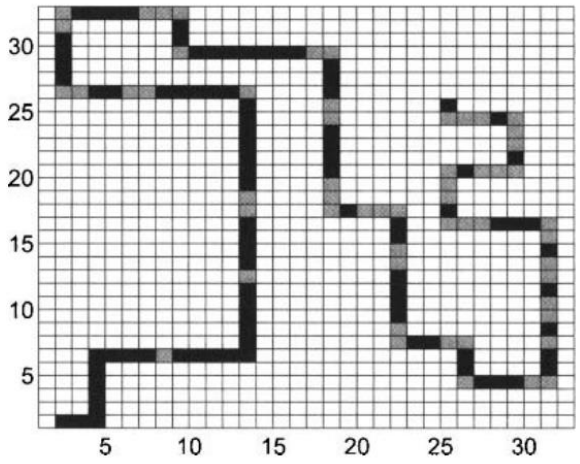
در این بخش به بحث در مورد مسئله‌ی مورچه‌ی مصنوعی (با بهینه‌سازی کلونیِ مورچگان اشتباه نشود!)، که یکی دیگر از مسائل مشهوری است که می‌توان آن را با یک FSM حل نمود، بحث خواهیم کرد. مسئله‌ی مورچه‌ی مصنوعی در سال ۱۹۹۰ معرفی شد [جِفرسون[7] و همکاران، ۲۰۰۳] و در [کوزا[8]، ۱۹۹۲، بخش ۳,۳,۲] به زیبایی توصیف گشت. یک مورچه‌ی مصنوعی بر روی یک شبکه‌ی حلقوی با ابعاد 32 × 32 قرار دارد که ۹۰ مربع از این ۱۰۲٤ مربع حاوی غذا هستند. توانایی‌های حسی مورچه بسیار محدود هستند. این مورچه تنها می‌تواند حضور غذا در خانه‌ی روبه‌رویش را حس کند. این مورچه در هر خانه می‌تواند سه حرکت انجام دهد: می‌تواند در جهت مستقیم (جهتی که صورتش قرار گرفته) یک مربع به جلو رفته و اگر غذایی در آن بود آن را بخورد، یا می‌تواند با ثابت ماندن در مربع خود به سمت راست بچرخد و یا آنکه

[1] Harrald
[2] Delahaye
[3] Mathieu
[4] Bonacich
[5] Worden
[6] Levin
[7] Jefferson
[8] Koza

می‌تواند با ثابت ماندن در مربع خود به سمت چپ بچرخد. رد به جا مانده از مورچه به رد سانتافه[1] شهرت داشته و در شکل ۵–۱۷ نمایش داده شده است.

مورچه حرکت خود را از مربع (۱,۱) که در گوشه‌ی سمت چپ و پایین شبکه قرار دارد آغاز می‌کند (اگر بخواهیم دقیق صحبت کنیم، از آنجایی که شبکه حلقوی است، هیچ گوشه‌ای در آن وجود ندارد). جهتِ آغازین مورچه نیز به سمت راست است. وی غذای روبه‌روی خود در شکل ۵–۱۷ را حس می‌کند پس باید برای خوردن غذا به سمت جلو و به سمت مربع بعدی در مختصات (۲,۱) حرکت کند. هنگامی که در مربع (۲,۱) قرار دارد نیز دوباره غذای روبه‌رویش را حس کرده بنابراین برای خوردن آن نیز باید به مربع (۳,۱) برود. به همین ترتیب به مربع (٤,۱) خواهد رفت. اما حالا در این سفر، که تا به اینجا راضی‌کننده و قابل پیش‌بینی بود، به مانع بر می‌خورد. مادامی که در مربع (٤,۱) قرار دارد می‌تواند نبودِ غذا در مربع روبه‌رویش را حس کند. آیا باید در هر صورت به سمت جلو حرکت کند و به پیدا کردن غذا در مربع بعدی امیدوار باشد؟ یا باید به سمت راست یا چپ چرخیده و امیدوار باشد در مربع مجاورِ مربع کنونی‌اش غذایی یافت شود؟ اگر به سمت چپ بچرخد غذای موجود در (٤,۲) را حس کرده و روی ردِ بهینه باقی خواهد ماند. اما اگر به سمت راست بچرخد غذای موجود در (٤,۳۲) را حس می‌کند (به یاد داشته باشید که شبکه حلقوی است). این کار مقداری لذت و خشنودی کوتاه مدت را برای وی به ارمغان خواهد آورد اما در نهایت باعث گمراهی و گیجی وی خواهد شد. مسئله‌ی مورچه‌ی مصنوعی شامل پیدا کردن یک FSM است تا مورچه را در مسیر سانتافه راهنمایی کرده به‌طوری که مورچه همه‌ی غذاها را با کمترین حرکت ممکن بخورد. یک قدم به جلو، یک چرخش یه راست و یک چرخش به چپ، هر کدام یک حرک محسوب میشوند. مسیر بهینه بر روی شبکه، با حرکت از میان مربع‌های سیاه و خاکستریِ نشان داده شده در شکل ۵–۱۷ حاصل می‌شود. این مسیر شامل ١٦٧ حرکت است.

[1] Santa fe



شکل ۵-۱۷ ردِ سانتافه. یک مورچه بر روی گوشه‌ی سمت چپ پایین و با جهتگیری به سمت راست واقع شده است. مربع‌های سفید خالی بوده و مربع‌های سیاه حاوی غذا هستند. مربع‌های خاکستری نیز خالی هستند اما با رنگ خاکستری نمایش داده شده‌اند تا مسیرِ بهینه‌ی مورچه از میان شبکه را مشخص نمایند.

می‌توان از EP برای به تکامل رساندن یک راه‌حل بهینه برای مسئله‌ی مورچه‌ی مصنوعی استفاده نمود. ابتدا مشخص می‌کنیم که از چه تعداد حالت می‌خواهیم استفاده کنیم. فرض کنید می‌خواهیم پنج حالت داشته باشیم. سپس یک FSM را با استفاده از دنباله‌ی اعداد زیر کدگذاری می‌کنیم:

$$\begin{matrix} 1_{0,m}, 1_{0,s}, 1_{1,m}, 1_{1,s}, \\ 2_{0,m}, 2_{0,s}, 2_{1,m}, 2_{1,s}, \\ .... \\ 5_{0,m}, 5_{0,s}, 5_{1,m}, 5_{1,s} \end{matrix} \qquad (۱۱-۵)$$

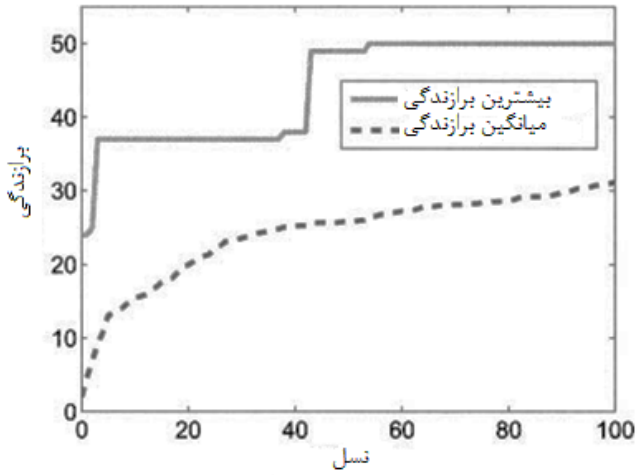
نشان‌گذاری استفاده شده در بالا برای ارائه‌ی FSM به شرح زیر است:

- $n_{0,m}$ حرکتی است که مورچه در صورت حضور در حالت n و حس نکردنِ غذا در خانه‌ی روبه‌رویش انجام می‌دهد. $n_{0,m} = 0, 1$یا2به ترتیب نشانگر حرکت به سمت جلو، چرخش به سمت راست یا چرخش به سمت چپ می‌باشد.
- $n_{0,s}$ حالتی است که اگر مورچه در حالت n بوده و در مقابلش غذا حس نکند به آن منتقل خواهد شد.
- $n_{1,m}$ حرکتی است که مورچه در صورت حضور در حالت n و حس کردنِ غذا در خانه‌ی روبه‌رویش انجام می‌دهد.

- $n_{1,s}$ حالتی است که اگر مورچه در حالت n بوده و در مقابلش حضور غذا را حس کند به آن منتقل خواهد شد.

بدین ترتیب ما یک FSM را با $4N$ عدد صحیح، که N تعداد حالت‌هاست، کدگذاری می‌کنیم. فرض ما بر آن است که مورچه همیشه از حالت ۱ آغاز می‌کند.

ما می‌توانیم هر FSM در یک جمعیتِ EP را ارزیابی کرده و ببینیم در مسیریابی چگونه عمل می‌کند. در ابتدا مورچه را در گوشه‌ی سمتِ چپ و پایینِ شبکه قرار می‌دهیم به‌طوری که روی آن به سمت راست باشد. ما همچنین عدد ۵۰۰ را به‌عنوان محدودیتِ تعداد حرکت‌های مورچه با هر FSM، در نظر می‌گیریم. هزینه‌ی یک FSM با تعداد حرکت‌های مورد نیاز برای آنکه مورچه بتواند تمام غذای موجود در شبکه را بخورد، اندازه‌گیری می‌شود. اگر مورچه پس از انجامِ ۵۰۰ حرکت خود موفق به خوردن تمام غذاها نشده باشد، آنگاه هزینه برابر ۵۰۰ به علاوه‌ی تعداد مربع‌های حاویِ غذایی خواهد بود که مورچه نتوانسته است به آن‌ها برسد. شکل ۵–۱۸ پیشرفتِ یک شبیه‌سازی EP برای FSMهای ۵ حالته و با اندازه جمعیت ۱۰۰ را نشان می‌دهد.



شکل ۵–۱۸ پیشرفتِ یک شبیه‌سازی EP برای تکامل FSM به منظور حل مسئله‌ی مورچه‌ی مصنوعی. هر FSM پنج حالت داشته و تعداد حرکات به ۵۰۰ حرکت محدود شده است. در آغاز، بهترین FSM مورچه را قادر می‌سازد تا ۲۴ غذا از ۹۰ غذا را به دست آورد. بعد از ۱۰۰ نسل، بهترین FSM مورچه را قادر می‌سازد تا ۵۰ غذا را به دست آورد.

میانگینِ تعداد غذاهایی که مورچه‌ها می‌خورند به تعداد حالت‌های به کار رفته در FSM بستگی دارد. اگر از تعداد حالت‌های کم استفاده کنیم آنگاه انعطاف لازم برای پیدا کردن یک راه‌حل خوب را نخواهیم داشت. از سویی اگر از تعداد حالت‌های زیاد استفاده کنیم، عملکرد EP بهتر می‌شود اما ممکن است این بهبود ارزش

آن را نداشته باشد که بخواهیم به خاطر آن مدت عملیات کامپیوتری را زیاد کنیم. میانگینِ غذای خورده شده توسط هر مورچه در جمعیت بعد از ۱۰۰ نسل در یک EP با اندازه‌ی جمعیت ۱۰ به شرح زیر است:

بعد FSM = واحد غذا 50.1 : 4
بعد FSM = واحد غذا 60.5 : 6
بعد FSM = واحد غذا 62.5 : 8 (۱۲-۵)
بعد FSM = واحد غذا 63.1 : 10
بعد FSM = واحد غذا 63.8 : 12

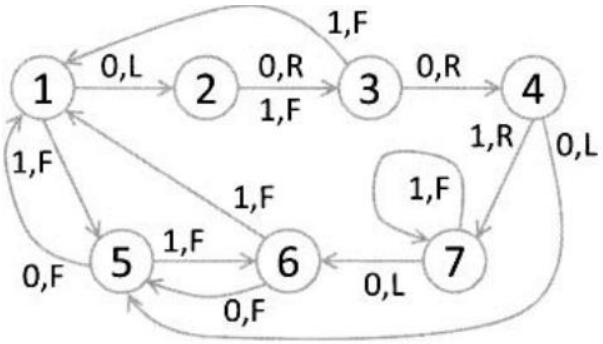
می‌توان دید که با افزایش تعداد حالت‌ها از ٤ به ٦، عملکرد به طرز قابل توجهی بهبود می‌یابد، اما بعد از آن، افزایش تعداد حالت‌ها بهبودِ کمی را به دنبال خواهد داشت.

بعد از چندین بار اجرای مونت کارلو دریافتیم که بهترین FSM که EP به دست داد ۱۲ حالت دارد. با این حال، ۵ حالت از این ۱۲ حالت هیچ گاه مورد استفاده قرار نگرفت و در واقع FSM، ۷ حالتِ عملیاتی داشت. این FSM به فرمِ معادله‌ی (۵-۱۱) در جدول ۵-۳ نشان داده شده است.

شکل ۵-۱۹، FSM را به شکل گرافیکی نشان می‌دهد. مورچه‌ای که از این FSM برای جهت‌یابی استفاده می‌کرد توانست تمام ۹۰ غذا را در ۳٤۹ حرکت، که کمی از دو برابرِ حداقل حرکات لازم برای خوردن غذاها بیشتر است، بخورد. اگر با دقت به شکل ۵-۱۹ نگاه کنیم برخی بی‌کفایتی‌های موجود در FSM آشکار می‌شود. برای مثال، وقتی مورچه در حالت ٤ قرار دارد و در روبه‌رویش حضور غذا را احساس می‌کند، به سمت راست چرخیده و به حالتِ ۷ می‌رود. این در حالی است که انتظار داریم هرگاه مورچه در روبه‌رویش حضور غذا را احساس می‌کند، به سمت جلو حرکت کرده و غذا را بخورد. به نظر می‌رسد که برچسبِ "R و۱" بیرون آمده از حالتِ ٤ بی‌فایده است. ما می‌توانیم این اشکال را با مجبور کردنِ تمام FSMها به حرکت به سوی جلو در هنگام حس کردن حضور غذا، اصلاح کنیم. این کار به نوعی دخیل کردنِ اطلاعات مختصِ مسئله در EP است که می‌تواند عملکردِ EP را بهبود بخشد. به‌طور کلی، همیشه باید سعی کنیم برای بهبود عملکرد، اطلاعاتِ مختص مسئله را درون الگوریتم‌های تکاملی دخالت دهیم.

جدول ۵-۳ بهترین ماشینِ حالت متناهیِ به تکامل رسیده توسط EP برای یک ردِ سانتافه با ابعاد 32 × 32. حرکات به ترتیب این گونه برچسب‌گذاری شده‌اند: حرکت رو به جلو = **0**، چرخش به راست = **1**، چرخش به چپ = **2**. **این FSM در شکل ۵-۱۹ به فرمِ گرافیکی نمایش داده شده است.**

	غذا حس نشده است		غذا حس شده است	
	حرکت	حالت بعد	حرکت	حالت بعد
حالت ۱	۲	۲	۰	۵
حالت ۲	۱	۳	۰	۳
حالت ۳	۱	٤	۰	۱
حالت ٤	۲	۵	۱	۷
حالت ۵	۰	۱	۰	٦
حالت ٦	۰	۵	۰	۱
حالت ۷	۲	٦	۰	۷



شکل ۵-۱۹ بهترین ماشینِ حالت متناهیِ به تکامل رسیده توسط EP برای یک ردِ سانتافه با ابعاد 32 × 32. خروجی‌های هر حالت به فرم (f,s) نشان داده شده‌اند که در آن $f = 0$ بیان‌گرِ آن است که حضور غذا حس نشده است و $f = 1$ بیان‌گرِ آن است که حضور غذا حس شده است. $s = F$ نشان‌دهنده‌ی حرکت به سمت جلو، $s = L$ نشان‌دهنده‌ی چرخش به سمت چپ و $s = R$ نشان‌دهنده‌ی چرخش به سمت راست است. این FSM در جدول ۵-۳ به فرمِ جدولی نشان داده شده است.

نسخه‌های دیگر مسئله‌ی مورچه‌ی مصنوعی شاملِ ردِ تپه‌های لوس آتلوس[1] [کوزا، ۱۹۹۲، بخش ۷,۲]، ردِ سان ماتئو[2] [کوزا، ۱۹۹٤، فصل ۱۲] و ردِ جان موییر[3] [جِفِرسون و همکاران، ۲۰۰۳] می‌باشد.

۵-۶ نتیجه‌گیری

از لحاظ تاریخی،EP برای پیدا کردن FSMهای بهینه مورد استفاده قرار گرفته است. با این حال، ما به دو نکته‌ی بسیار مهم برای جمع‌بندی این فصل اشاره می‌کنیم. نخست آنکه می‌توانیم از EP به‌عنوان یک الگوریتم با مقاصدِ کلی برای حل هرگونه مسئله‌ی بهینه‌سازی استفاده کنیم. در حقیقت EP یک الگوریتم مشهور برای بهینه‌سازی‌های با مقاصدِ کلی است. دوم آنکه ما می‌توانیم مسائل FSM را نه تنها با EP بلکه با هر الگوریتم تکاملیِ بحث شده در این کتاب حل کنیم. معمای زندانی و انواع آن، مسائل بهینه‌سازی کلی هستند که می‌توان از بسیاری از انواع الگوریتم‌های بهینه‌سازی برای حل آن‌ها استفاده نمود. دلیل آنکه بخش زیادی از این فصل را به معمای زندانی اختصاص دادیم آن است که EP در اصل برای حل FSMها به وجود آمدند. در انتها متذکر می‌شویم که کتاب‌ها و مقالات بسیار زیادی هستند که EP را از زوایای دیگر، و در برخی موارد با جزئیات بیشتری مورد بحث قرار داده‌اند. از جمله‌ی آن‌ها می‌توان به [باک و اشوِفِل، ۱۹۹۳]، [باک، ۱۹۹٦] و [یائو و همکاران، ۱۹۹۹] اشاره نمود.

مسائل نوشتاری

۵-۱ واریانسِ جهش

$$\sigma_i^2 = \beta f(x_i) + \gamma$$

معمولاً به‌صورت یک رابطه‌ی خطی میانِ $f(x_i)$ و σ_i^2 در نظر گرفته می‌شود اما در واقع این رابطه نه خطی بلکه تکراری است. معادله‌ی بالا را به چه صورت باید بازنویسی کرد تا رابطه‌ی خطی میان $f(x_i)$ و σ_i^2 حاصل شود؟

۵-۲ یک آسانسور می‌تواند در یکی از دو حالت ممکن قرار گیرد: در طبقه‌ی اول یا در طبقه‌ی دوم. ورودیِ آن یکی از دو ورودیِ ممکن است: کاربر می‌تواند دکمه‌ی طبقه‌ی ۱ یا دکمه‌ی طبقه‌ی ۲ را فشار دهد. یک FSM برای این سیستم، هم به فرمِ گرافیکی و هم به فرم جدولی بنویسید.

۵-۳ مسئله‌ی ۵-۲ را بسط دهید به‌طوری که آسانسور ٤ حالت داشته باشد (در طبقه‌ی اول، در طبقه‌ی دوم، در حال حرکت از طبقه‌ی اول به طبقه‌ی دوم، در حال حرکت از طبقه‌ی دوم به طبقه‌ی اول) که بدین

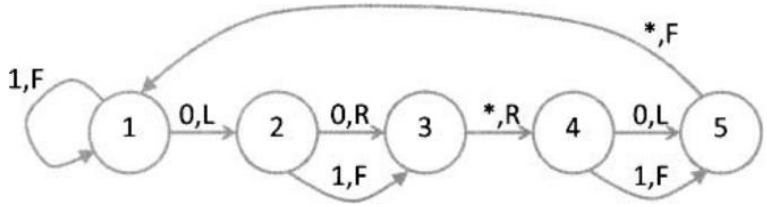
[1] Los Atlos Hills Trail
[2] San Mateo Trail
[3] John Muir Trail

ترتیب ۳ ورودی ممکن وجود خواهد داشت: کاربر می‌تواند دکمه‌ی طبقه‌ی اول را فشار دهد، کاربر می‌تواند دکمه‌ی طبقه‌ی دوم را فشار دهد، کاربر هیچ کاری انجام ندهد. یک FSM برای این سیستم هم به فرم گرافیکی و هم به فرم جدولی بنویسید.

۵-٤ استراتژی همواره- همکاری برای معمای زندانی را به فرمِ برداریِ معادله‌ی (۵-۹) بنویسید. این کار را برای استراتژی‌های این به آن در، این به آن دو در و استراتژی شوم تکرار کنید. با توجه به فرم‌های برداریِ شما، کدام دو استراتژی بیشتر به یکدیگر شباهت دارند؟ استراتژی‌های همواره- همکاری و این به آن در یا این به آن دو در و استراتژی شوم؟

۵-۵ فرض کنید در یک مسابقه‌ی معمای زندانی، استراتژی‌های همواره- همکاری، این به آن در، این به آن دو در و شوم با یکدیگر به رقابت می‌پردازند. کدام یک برنده‌ی بازی خواهد بود؟

۵-٦ FSM شکلِ ۵-۲۰ برای یک مورچه‌ی مصنوعی در مسیر سانتافه پیشنهاد شده است و در آن ورودیِ ۰ به معنای عدم حس کردنِ غذا و ورودیِ ۱ به معنای حس کردن غذا بوده و خروجی‌های L، R و F به ترتیب به معنای حرکت به سمتِ چپ، راست و مستقیم می‌باشند [میلیو و همکاران، ۱۹۹۹]. این FSM را به فرم معادله‌ی (۵-۱۱) بازنویسی کنید.



شکل ۵-۲۰ مسئله‌ی ۵-٦: FSM برای مسئله‌ی مورچه‌ی مصنوعی.

۵-۷ تعداد حداقل حرکات لازم برای آنکه یک مورچه‌ی مصنوعی بتواند تمام مربع‌های موجود در مسیرِ سانتافه را ملاقات کند چه قدر است؟

۵-۸ فرض کنید یک مورچه‌ی مصنوعی β مربعِ اتفاقی و یکتا از مسیرِ سانتافه‌ی 32×32 را ملاقات کند. احتمال آنکه مورچه تمامی غذاها را پیدا کند چه قدر خواهد بود؟

مسائل کامپیوتری

۵–۹ EP شکل ۵–۱ را برای مینیمم کردن تابع کروی ۱۰ بعدی شبیه‌سازی کنید. دامنه‌ی جستجو را $[-5.12, +5.12]$، β را برابر $\frac{x_{max}-x_{min}}{10} = 1.024$، $\gamma = 0$، اندازه‌ی جمعیت را برابر ۵۰ و محدودیت نسل را نیز برابر ۵۰ در نظر بگیرید. هنگام محاسبه‌ی واریانس جهش، مقادیر تابع هزینه را به گونه‌ای نرمالیزه کنید که برای تمام $f(x_i) \in [1,2]$ها i باشد.

الف) بهترین راه‌حل به دست آمده از EP، با میانگین‌گیری روی ۲۰ شبیه‌سازی مونت کارلو، چیست؟

ب) واریانس را با تابعِ $\beta f^2(x_i)$ جایگذاری کنید و شبیه‌سازی را تکرار کنید.

ج) واریانس را با تابعِ $\beta\sqrt{f(x_i)}$ جایگذاری کنید و شبیه‌سازی را تکرار کنید.

د) از β به‌عنوان واریانس برای تمامی iها استفاده کرده و شبیه‌سازی را تکرار کنید.

۵–۱۰ مثال ۵–۳ را با مجازاتِ اندازه‌ی FSM برابر n، که در آن n تعداد حالت‌هاست، دوباره اجرا کنید. تعداد حالت‌ها و هزینه‌ی بهترین FSM، با میانگین‌گیری بر روی ۱۰۰ بار شبیه‌سازی مونت کارلو چه قدر خواهند بود؟ نتایج خود را با نتایج به دست آمده در مثال ۵–۳ مقایسه کنید.

۵–۱۱ اگر در معمای زندانی بدانیم که حریف در هر نوبت ما را لو می‌دهد، بهترین استراتژی آن خواهد بود که ما نیز در هر نوبت وی را لو دهیم. از EP برای به تکامل رساندن یک FSM استفاده کنید به‌طوری که FSM مذکور به خوبیِ استراتژی همواره– لو دادن عمل کند.

۵–۱۲ تعداد حرکات لازم برای مورچه‌ای که از FSM مسئله‌ی ۵–۶ برای خوردن تمام غذاهای ردِ سانتافه استفاده می‌کند چه قدر است؟

۵–۱۳ از جوابی که به مسئله‌ی ۵–۸ داده اید برای پیدا کردن احتمال آنکه مورچه بعد از ملاقاتِ β مربعِ یکتا، اتفاقی تمام غذاهای رد سانتافه را خورده باشد، استفاده کنید. فرض کنید $\beta \in [1,1024]$ است. از مقیاس لگاریتمی برای محور احتمال استفاده کنید. اگر بخواهیم شانسِ مورچه برای پیدا کردن تمامی غذاها حداقل ۵۰٪ باشد، چند مربع را باید ملاقات کند؟

فصل ششم

استراتژی‌های تکامل

اولین نسخه‌ی ES[1] با تنها یک فرزند در هر نسل فعالیت می‌کرد چرا که جمعیتی از اهداف برای فعالیت وجود نداشت.

هانس-پاول[2] اشوفل [اشوفل و مندس[3]، ۲۰۱۰]

اولین حمله‌ی اروپائیان به الگوریتم‌های تکاملی توسط سه دانشجو در دانشگاه فنی و مهندسی برلین در دهه‌ی ۱۹۶۰ صورت گرفت. این سه دانشجو به دنبال دستیابی به یک طراحی بهینه‌ی بدنه در یک تونل باد به منظور مینیمم ساختن مقاومت هوا بودند. اینگو روچنبرگ[4]، هانس-پاول اشوفل و پیتر بینرت[5] برای حل تحلیلی این مسئله با مشکل مواجه بودند. بنابراین تصمیم گرفتند تغییراتی اتفاقی در بدنه ایجاد کرده، آن‌هایی را که عملکرد بهتری داشته انتخاب کرده و این فرآیند را آنقدر ادامه دهند تا راه‌حل‌هایی مناسب برای مسئله‌شان پیدا کنند.

اولین نوشته‌ی روچنبرگ در زمینه‌ی استراتژی‌های تکامل (ES)، که گاهاً استراتژی‌های تکاملی نیز خوانده می‌شوند، در سال ۱۹٦٤ به چاپ رسید [روچنبرگ، ۱۹۹۸]. نکته‌ی قابل توجه آن است که اولین پیاده‌سازی‌ها از ES به صورت تجربی صورت گرفته است. منابع محاسباتی برای شبیه‌سازی با دقت بالا کافی نبوده است و به همین علت، توابع برازندگی به‌صورت آزمایشی به دست می‌آمده است و جهش‌ها به‌صورت سخت‌افزاریِ فیزیکی پیاده‌سازی می‌شده‌اند. روچنبرگ مدرک دکترای خود را در سال ۱۹۷۰ در همین زمینه دریافت نمود و چندی بعد این اثر را در قالب کتاب منتشر نمود [روچنبرگ، ۱۹۷۰]. اگرچه که این کتاب به زبان آلمانی نوشته شده است، اما به دلیل در بر داشتن اشکال گرافیکی از فرآیندهای تکاملی، برای خوانندگان غیرآلمانی زبان نیز بسیار جالب توجه است. این کتاب، تکامل اشکال بال برای مینیمم ساختن کشش در یک میدان جریان باد، تکامل شکل کلاهک موشک‌ها برای مینیمم ساختن کشش سوخت در هنگام عبور از کلاهک و همچنین تکامل شکل لوله برای مینیمم ساختن کشش سیال در هنگام عبور از لوله را نشان می‌دهد. این الگوریتم‌های اولیه، *مسیرهای راه‌حل سایبرنتیک*[6] نام داشتند.

اشوفل مدرک دکترای خود را در سال ۱۹۷۵ دریافت نمود و در سال‌های بعد چندین کتاب در مورد ES منتشر نمود [اشوفل، ۱۹۷۷]، [اشوفل، ۱۹۸۱]، [اشوفل، ۱۹۹۵]. وی از سال ۱۹۸۵ تاکنون در دانشگاه دورتموند مشغول به کار است. بینرت نیز مدرک دکترای خود را در سال ۱۹٦۷ دریافت نمود. تاریخچه‌ای جالب از سال‌های ابتدایی ES را می‌توان در مصاحبه‌ای با اشوفل در [اشوفل و مندس، ۲۰۱۰] یافت.

---

[1] Evolutionary Strategiy
[2] Hans-Paul
[3] Mendes
[4] Ingo Rochenberg
[5] Peter Bienert
[6] Cybernetic Solution Paths

مروری بر فصل

بخش ٦-١ به بحث در مورد (1+1)-ES خواهد پرداخت. (1+1)-ES ساده‌ترین و همچنین اولین نوع ES است که مورد استفاده قرار گرفت. این ES تنها شامل جهش بوده و بازترکیب را شامل نمی‌شود. بخش ٦-٢ قانون $1/5$ را برای ES استنتاج می‌کند، قانونی که چگونگی تنظیم نرخ جهش برای دستیابی به بهترین عملکرد را توضیح می‌دهد. خوانندگانی که علاقه‌ای به اثبات‌های ریاضی ندارند می‌توانند از خواندن این بخش صرف نظر نمایند. بخش ٦-٣، (1+1)-ES را به الگوریتمی بسط می‌دهد که در آن در هر نسل تعداد μ والد وجود دارد. μ توسط کاربر تعیین می‌شود. والدین ترکیب شده تا فرزندان را به وجود آورند. اگر فرزندان به اندازه‌ی کافی برازنده باشند در نسل بعد حضور پیدا خواهند کرد. چندین گزینه برای بازترکیب وجود دارد. بخش ٦-٤، (1+1)-ES را باز هم تعمیم داده به‌گونه‌ای که در هر نسل تعدادِ λ فرزند وجود داشته باشد. بخش ٦-٥ به ما نشان می‌دهد که چگونه با تطبیق نرخ جهش، عملکرد ES را به‌صورت قابل توجهی بهبود بخشیم. این تطبیق‌ها شامل الگوریتم‌های بی‌نظیرِ CMA-ES و CMSA-ES می‌شود.

٦-١ استراتژی تکامل

فرض کنید $f(x)$ یک تابع از بردار حقیقی x بوده و ما می‌خواهیم $f(x)$ را ماکزیمم کنیم. عملکرد الگوریتم ES بدین صورت است که ابتدا یک راه‌حل نامزد را اعمال کرده و سپس برازندگی آن را اندازه‌گیری می‌کند. سپس این راه‌حل نامزد دچار جهش شده و برازندگی ذره‌ی دچار جهش شده نیز اندازه‌گیری می‌شود. از بین این دو راه‌حل نامزد (فرزند و والد) بهترینشان نقطه‌ی آغازین برای نسل بعد را تشکیل می‌دهد. الگوریتم ES ابتدایی، ذاتاً برای حل مسائل گسسته طراحی شده بود، بدین ترتیب که از جهش‌های کوچک در یک فضای جستجوی گسسته استفاده می‌کرد. به همین علت، مستعد گیر افتادن در بهینه‌های محلی بود. بدین ترتیب ES ابتدایی اصلاح شد تا از جهش‌های پیوسته در فضای جستجوی پیوسته استفاده نماید [بِیر و اشوفل، ۲۰۰۲]. این الگوریتم در شکل ٦-١ خلاصه‌سازی شده است.

شکل ٦-١ را (1+1)-ES می‌نامند چرا که در آن هر نسل شامل یک فرزند و یک والد است که بهترینِ آن‌ها به‌عنوان ذره‌ی نسل بعد انتخاب می‌شود. (1+1)-ES، که ES دو عضوه نیز نامیده می‌شود، بسیار شبیه استراتژی‌های تپه‌نوردی در بخش ٦-٢ است. همچنین بسیار شبیه یک EP با اندازه جمعیت ۱ است (بخش ٥-١ را ببینید). قضیه‌ی زیر تضمین می‌کند که (1+1)-ES در نهایت ماکزیمم سراسری $f(x)$ را پیدا می‌کند.

قضیه‌ی ٦-١) اگر $f(x)$ یک تابع پیوسته با مقدار بهینه‌ی سراسری $f^*(x)$ باشد که بر روی یک بازه‌ی بسته تعریف شده است، آنگاه

$$\lim_{t\to\infty} f(x) = f^*(x) \qquad (١-٦)$$

که در آن t شماره‌ی نسل است.

```
واریانس جهش σ² را با یک مقدار نامنفی آغاز کن
یک ذره‌ی تولید شده‌ی اتفاقی ← x0
تا زمانی که شرایط توقف برآورده نشده است
  یک بردار اتفاقی r را با ri~N(0,σ²) برای تمامی i ∈ [1,n] ایجاد کن
  x1 ← x0 + r
  اگر x1 بهتر از x0 بود آنگاه
    x0 ← x1
  پایان اگر
نسل بعد
```

شکل ٦-١ شبه کد بالا طرح کلی استراتژی تکامل (1+1) را نشان می‌دهد. در این شبه کد n ابعاد مسئله بوده و x_1 با x_0 برابرست تنها با این تفاوت که هر عنصر x_0 دچار جهش شده است.

قضیه‌ی ٦-١ در [دِوروی[1]، ۱۹۷۸]، [رودالف[2]، ۱۹۹۲]، [باک، ۱۹۹٦، قضیه‌ی ۷] و [مایکِلویکز، ۱۹۹٦] اثبات شده است. این قضیه همچنین با شهود و درک ما همخوانی دارد. از آنجا که ما از جهش‌های تصادفی برای کاوش فضای جستجو استفاده می‌کنیم، اگر مدت زمان کافی در اختیار داشته باشیم، سرانجام تمام فضای جستجو را کاوش کرده و ماکزیمم سراسری را خواهیم یافت.

واریانس σ^2 در الگوریتمِ شکل ٦-١ یک پارامتر میزان‌سازی است.

- σ باید به اندازه‌ی کافی بزرگ باشد تا جهش‌ها تمامی نواحی مورد جستجو را در یک بازه‌ی زمانی معقول پوشش دهند.
- σ باید به اندازه‌ی کافی کوچک باشد تا جستجوی راه‌حل بهینه با دقت مورد نظرِ کاربر صورت بپذیرد. شاید مناسب باشد که مقدار σ را با پیشرفت ES کاهش داد. مقادیر بزرگ σ در ابتدای ES باعث می‌شود جستجوی اصطلاحاً دانه درشت صورت گرفته و به جواب بهینه نزدیک شویم. با

[1] Devroye
[2] Rudolph

نزدیک شدن ES به خط پایان، مقادیر کوچکتر σ به ES این اجازه را می‌دهند تا راه‌حل‌های نامزد بهتر تنظیم شده و همگرایی به راه‌حل بهینه با دقت بیشتری صورت بپذیرد.

جهش در شکل ٦-١ همه سو یکسان[1] خوانده می‌شود چرا که جهشِ تمامی عناصر x_0 دارای واریانس یکسانی هستند. در عمل شاید بخواهیم جهش‌هایی غیر همه سو یکسان را به شکل زیر پیاده‌سازی کنیم:

$$x_1 \leftarrow x_0 + N(0, \Sigma) \qquad (٢-٦)$$

که در آن Σ یک ماتریس قطری $n \times n$ با عناصر قطری σ_i برای $i \in [1, n]$ می‌باشد. این بدان معناست که هر عنصر از x_0 با واریانس متفاوتی دچار جهش می‌شود. ما مقدار هر σ_i را مستقلاً و با توجه به دامنه‌ی اُمین i عنصر x_0 و همچنین شکل تابع هدف، اختصاص می‌دهیم.

روچنبرگ (1+1)-ES را برای چند مسئله بهینه‌سازی ساده تحلیل کرد و به این نتیجه رسید که ٢٠٪ از جهش‌ها باید باعث بهبود تابع برازندگی $f(x)$ شوند [روچنبرگ، ١٩٧٣]، [باک، ١٩٩٦، بخش ٢-١-٧]. در بخش ٦-٢ برخی از این تحلیل‌ها را ارایه خواهیم داد. اگر نرخ موفقیت جهش بزرگتر از ٢٠٪ باشد، آنگاه جهش‌ها بسیار کوچک بوده و بهبودهای کوچکی را ایجاد می‌کنند که این موضوع باعث طولانی شدن مدت زمان همگرایی می‌شود. اگر نرخ موفقیت جهش کمتر از ٢٠٪ باشد، آنگاه جهش‌ها بسیار بزرگ بوده که باعث ایجاد پیشرفت‌هایی بزرگ اما ناکافی می‌شوند که این موضوع نیز باعث طولانی شدن زمان همگرایی می‌شود. کار روچنبرگ به قانون $1/5$ انجامید:

در (1+1)-ES، اگرنسبت جهش‌های موفق به کل جهش‌ها کمتر از $1/5$ باشد، آنگاه میزان انحراف استاندارد σ باید کاهش پیدا کند. از سوی دیگر اگر این نسبت بزرگتر از $1/5$ باشد، میزان انحراف استاندارد باید افزایش پیدا کند.

در بخش ٦-٢ خواهیم دید که این قانون تنها برای تعداد اندکی از توابع هدف مصداق دارد، اما در کل خط مشی مفیدی برای پیاده‌سازی‌های ES به دست می‌دهد. اما قانون $1/5$ این سؤال را ایجاد می‌کند که انحراف استاندارد تا چه حد باید کاهش یا افزایش یابد؟ اشوفل فاکتوری را به‌صورت نظری به دست آورد که میزان افزایش یا کاهش σ را مشخص می‌کند:

$$\begin{aligned} &\sigma \leftarrow c\sigma \quad &&\text{: کاهش انحراف معیار} \\ &\sigma \leftarrow \sigma / c \quad &&\text{: کاهش انحراف معیار} \\ &c = 0.817 \quad &&\text{که در آن} \end{aligned} \qquad (٣-٦)$$

[1] Isotropic

این نتایج به (1+1)-ES انطباقی، که در شکل ٦-٢ نشان داده شده است، منجر می‌شود. برای (1+1)-ES انطباقی نیاز به تعریف یک پنجره به طول G داریم. ما می‌خواهیم G به اندازه‌ی کافی بزرگ باشد تا ایده‌ی خوبی از نرخ موفقیت جهش‌های ES به دست آید، اما نه آنقدر بزرگ که تطبیق σ به کندی صورت بپذیرد. مقدار توصیه شده در [بِیر و اشوفل، ۲۰۰۲] برابرست با

$$G = \min(n, 30) \qquad (٦-٤)$$

که در آن n ابعاد مسئله است.

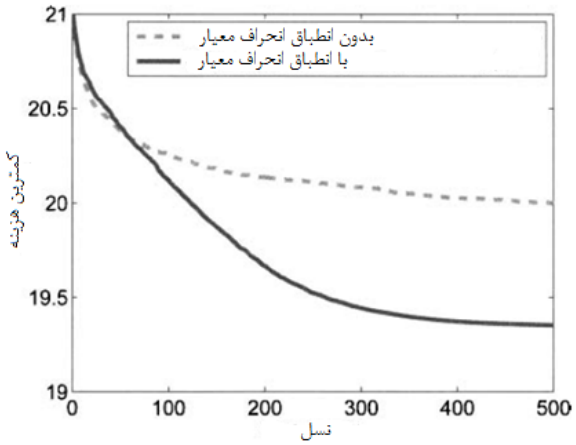
واریانس جهش σ^2 را با یک مقدار نامنفی آغاز کن
یک ذره‌ی تولید شده‌ی اتفاقی $\leftarrow x_0$
تا زمانی که شرایط توقف برآورده نشده است
  یک بردار اتفاقی r را با $r_i \sim N(0, \sigma^2)$ برای تمامی $i \in [1, n]$ ایجاد کن
  $x_1 \leftarrow x_0 + r$
  اگر x_1 بهتر از x_0 بود آنگاه
    $x_0 \leftarrow x_1$
  پایان اگر
  نسبت جهش‌های موفق در طول G نسل قبل $\leftarrow \phi$
  اگر $\phi < 1/5$ آنگاه
    $\sigma \leftarrow c^2\sigma$
  اگر $\phi > 1/5$ آنگاه
    $\sigma \leftarrow \sigma/c^2$
  پایان اگر
نسل بعد

شکل ٦-٢ شبه کد بالا استراتژی تکامل *(1+1)* انطباقی را نشان می‌دهد که در آن n ابعاد مسئله است. x_1 با x_0 برابرست تنها با این تفاوت که هر ویژگی آن دچار جهش شده است. ϕ نسبتی از جهش‌های G نسل قبل است که به بهتر شدن x_1 از x_0 منجر شده‌اند. واریانس جهش به‌صورت خودکار به‌منظور افزایش نرخ همگرایی تنظیم می‌شود. مقدار نامی c برابر ۰,۸۱۷ است.

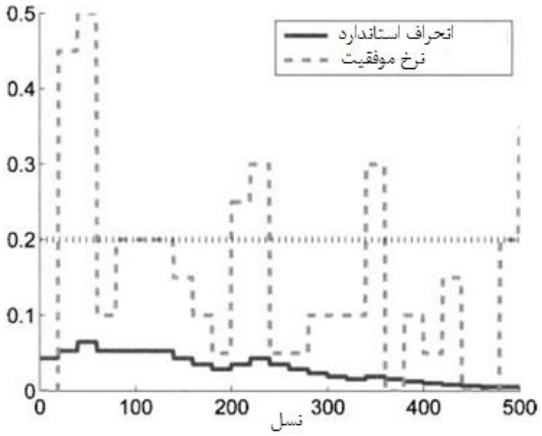
مثال٦-١

در این مثال ما از (1+1)-ES برای بهینه ساختن تابع محک اَکلیِ ۲۰ بعدی استفاده کرده‌ایم (ضمیمه ج.٢-١ را ببینید). در این مثال (1+1)-ES استاندارد که در شکل ٦-١ نشان داده شده است را با (1+1)-ES انطباقیِ نشان داده شده در شکل ٦-٢، مقایسه خواهیم نمود. در مورد ES انطباقی، آمار جهش‌های موفق و کل تعداد جهش‌ها را ثبت خواهیم نمود. تعداد کل جهش‌ها با تعداد کل نسل‌ها برابر است. هر ۲۰ نسل یکبار

میزان نرخ موفقیت جهش را محاسبه کرده و مقدار انحراف استاندارد را مطابق با معادله‌ی (٦-٣) تنظیم خواهیم نمود. شکل ٦-٣ میانگین نرخ همگراییِ (1+1)-ES استاندارد را با (1+1)-ES انطباقی مقایسه نموده است. شکل ٦-٤ منحنی رایجی از نرخ موفقیتِ جهش و انحراف استاندارد جهش را نشان می‌دهد. می‌توان دید که هر گاه نرخ موفقیت در طول ۲۰ نسل گذشته بزرگتر از ۲۰٪ بوده است، انحراف استاندارد جهش به‌صورت خودکار افزایش یافته و هرگاه نرخ موفقیت کمتر از ۲۰٪ بوده است، انحراف استاندارد جهش به‌صورت خودکار کاهش یافته است.



شکل ٦-٣ مثال ٦-١: همگراییِ الگوریتم (1+1)-ES که در طول ۱۰۰ شبیه‌سازی میانگین‌گیری شده است. ES انطباقی، که انحراف استاندارد جهش را به‌صورت خودکار تنظیم می‌نماید، بسیار سریعتر از ES استاندارد همگرا شده است.



شکل ٦-٤ مثال ٦-١: نرخ موفقیت جهش و انحراف استاندارد جهشِ (1+1)-ES انطباقی. ES انطباقی هرگاه نرخ موفقیت بزرگتر از ۲۰٪ باشد، به‌صورت خودکار انحراف استاندارد جهش را افزایش داده و هرگاه نرخ موفقیت کمتر از ۲۰٪ باشد، انحراف استاندارد جهش را کاهش خواهد داد.

۶-۲ قانون $1/5$: یک استنتاج

این بخش به استنتاج قانون $1/5$ می‌پردازد. این قانون بیان می‌دارد که حدود ۲۰٪ از کل جهش‌ها به ایجاد بهبود در ذرات جهش یافته‌ی ES منجر می‌شود. محرک اصلی این بخش از کتاب، [روچنبرگ، ۱۹۷۳، فصل ۱۴-۱۵] می‌باشد. خوانندگانی که علاقه‌ای به اثبات‌های ریاضی ندارند می‌توانند از خواندن این بخش صرف نظر نمایند.

فرض کنید که یک مسئله‌ی مینیمم‌سازیِ n بعدی با تابع هزینه‌ی $f(x)$ که در آن $x = [x_1 x_2 \dots x_n]$ است، در اختیار داریم. در این قسمت بر روی مسئله‌ی دهلیز[1] که دارای دامنه‌ی زیر است تمرکز می‌نماییم.

$$\begin{aligned} &x_1 \in [0, \infty) \\ &x_j \in (-\infty, \infty), \quad j \in [2, n] \end{aligned} \tag{۶-۵}$$

مسئله‌ی دهلیز دارای تابع هزینه‌ی زیر است:

$$f(x) = \begin{cases} c_0 + c_1 x_1, & \text{اگر } x_j \in [-b, b] \quad \text{برای تمامی } j \in [2, n] \\ \infty & \text{در غیر این صورت} \end{cases} \tag{۶-۶}$$

که در آن c_0، c_1 و b ثابت‌هایی مثبت هستند. در این مسئله هزینه با کاهش x_1 بهبود می‌یابد اما تنها در صورتی که x در بازه‌ی (یا دهلیز!) $x_j \in [-b, b]$ برای تمامی jها قرار داشته باشد.

از شکل ۶-۱ و ۶-۲ به یاد آورید که هر ذره‌ی x_0 از ES بر اساس معادله‌ی $x_1 \leftarrow x_0 + r$، که در آن r یک بردار n عنصری تصادفی است، دچار جهش می‌شود. ما از نماد x_{0j} برای اشاره به jاُمین عنصر بردار x_0 و از نماد x_{1j} برای اشاره به jاُمین عنصر بردار x_1 استفاده می‌کنیم.

جهش هر یک از عناصر x_0 از یک توزیع گاوسی با میانگین صفر و واریانس σ^2 انتخاب می‌شود. بدین ترتیب PDF مربوط به x_{1j} را می‌توان به‌صورت زیر نوشت

$$PDF(x_{1j}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x_{1j} - x_{0j})^2}{2\sigma^2}\right], \quad j \in [1, n] \tag{۶-۷}$$

جهشی که سبب بهبود x می‌شود به وقوع n اتفاق مستقل نیاز دارد:

$$\begin{aligned} &r_1 < 0 \\ &x_{1j} \in [-b, b], \quad j \in [2, n] \end{aligned} \tag{۶-۸}$$

[1] Corridor Problem

که در آن r_1 جهش اولین عنصر x_0 است (یعنی: $x_{11} \leftarrow x_{01} + r_1$). بنابراین φ' یا دامنه‌ی مورد انتظارِ یک جهش مفید برابرست با

$$\phi' = |E(r_1|r_1 < 0)| \prod_{j=2} Prob(x_{1j} \in [-b, b])$$

$$= |\int_{-\infty}^{0} \frac{r_1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{r_1^2}{2\sigma^2}\right) dr_1 | \prod_{j=2}^{n} \int_{-b}^{b} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{\left(x_{1j} - x_{0j}\right)^2}{2\sigma^2}\right) dx_{1j} \qquad (۹-٦)$$

$$= \frac{\sigma}{\sqrt{2\pi} \prod_{j=2}^{n} \frac{1}{2} [\mathrm{erf}\left(\frac{b - x_{0j}}{\sigma\sqrt{2}}\right) + \mathrm{erf}(\frac{b + x_{0j}}{\sigma\sqrt{2}})]}$$

(مسئله‌ی ٦-۲ را ببینید) که در آن $erf(.)$ تابع خطا می‌باشد:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2)\, dt, \quad x \geq 0 \qquad (۱۰-٦)$$

اگر x_0 قبل از جهش در بازه‌ی $[-b, b]$ قرار داشته باشد، دامنه‌ی مورد انتظارِ جهش مفید را می‌توان به‌صورت زیر نوشت

$$\phi = E(\phi' | x_{0j} \in [-b, b] \text{ برای تمامی } j \in [2, n])$$

$$= \int_{-b}^{b} \dots \int_{-b}^{b} \phi' PDF(x_{02}) \dots PDF(x_{0n}) dx_{02} \dots dx_{0n} \qquad (۱۱-٦)$$

اگر x_{0j} در بازه‌ی $[-b, b]$ واقع شده باشد و فرض کنیم x_{0j} به‌صورت یکنواخت بر روی این بازه توزیع شده باشد، خواهیم داشت

$$\phi = \frac{\sigma}{\sqrt{2\pi}} \prod_{j=2}^{n} \int_{-b}^{b} \frac{1}{2} \left[\mathrm{erf}\left(\frac{b - x_{0j}}{\sigma\sqrt{2}}\right) + \mathrm{erf}\left(\frac{b + x_{0j}}{\sigma\sqrt{2}}\right)\right] (\frac{1}{2b}) dx_{0j} \qquad (۱۲-٦)$$

حال به یاد آورید که

$$\int \mathrm{erf}(z_{dz} = zerf(z) + \frac{\exp(-z^2)}{\sqrt{\pi}} \qquad (۱۳-٦)$$

با به کار بردن معادله‌ی بالا در معادله‌ی (٦-۱۲) خواهیم داشت

$$\phi = \frac{\sigma}{\sqrt{2\pi}(4b)^{n-1}} \prod_{j=2}^{n} \left[4\mathrm{b}\ \mathrm{erf}\left(\frac{2b}{\sigma\sqrt{2}}\right) + \mathrm{erf}\left(\frac{2\sigma\sqrt{2}(\exp\left(-\frac{2b^2}{\sigma^2}\right) - 1}{\sqrt{\pi}}\right)\right] \qquad (۱٤-٦)$$

$$\frac{\sigma}{\sqrt{2\pi}}[\mathrm{erf}\left(\frac{2b}{\sigma\sqrt{2}}\right)-\frac{\sigma}{b\sqrt{2\pi}}(1-\exp\left(-\frac{2b^2}{\sigma^2}\right))]^{n-1}$$

به یاد آورید که $\lim_{x\to\infty} erf(x)=1$ و $\lim_{x\to\infty} exp(x)=0$. بنابراین،

$$\phi \approx \frac{\sigma}{\sqrt{2\pi}}\left(1-\frac{\sigma}{b\sqrt{2\pi}}\right)^{n-1} \quad \text{برای } \frac{b}{\sigma} \text{ بزرگ} \qquad (۱۵-٦)$$

ϕ مقدار امید ریاضی یک جهش مفید است و مطلوب است که بزرگ باشد. برای به دست آوردن مقادیری از σ که مقادیر بزرگ ϕ را به دست می‌دهند، کافیست از ϕ نسبت به σ مشتق گرفت و آن را برابر ۰ قرار داد. بدین ترتیب خواهیم داشت

$$\frac{d\phi}{d\sigma}=\frac{1}{\sqrt{2\pi}}\left(1-\frac{\sigma}{b\sqrt{2\pi}}\right)^{n-1}-\frac{\sigma(n-1)}{b2\pi}\left(1-\frac{\sigma}{b\sqrt{2\pi}}\right)^{n-2} \qquad (۱٦-٦)$$

اگر این معادله را برابر ۰ قرار داده و آن را برای σ حل کنیم خواهیم داشت

$$\sigma^* = b\sqrt{2\pi}/n \qquad (۱۷-٦)$$

که بیشترین مقدار ϕ و به تبع آن بیشترین دامنه‌ی مورد انتظارِ یک جهش مفید را به دست خواهد داد.

حال احتمال w'، که احتمال مفید بودن یک جهش است را در نظر بگیرید. یک جهش مفید است اگر n واقعه‌ی مستقلِ نشان داده شده در معادله‌ی (٦-۸) رخ دهند. احتمال این اتفاق را می‌توان به‌صورت زیر نوشت

$$w' = Prob(r_1 < 0)\prod_{j=2}^{n} Prob(x_{1j} \in [-b,b]) \qquad (۱۸-٦)$$

از آنجایی که r_1 دارای میانگین صفر است، پس احتمال آنکه $r_1 < 0$ باشد برابر $\frac{1}{2}$ است. بنابراین معادله‌ی بالا را می‌توان به‌صورت زیر بازنویسی کرد

$$w' = \frac{1}{2}\prod_{j=2}^{n} Prob(x_{1j} \in [-b,b]) \qquad (۱۹-٦)$$

با مقایسه‌ی معادله‌ی بالا با معادله‌ی (٦-۹) خواهیم دید که

$$w' = \frac{\sqrt{2\pi}}{2\sigma}\phi' \qquad (۲۰-٦)$$

حال با فرض آنکه x_0 قبل از وقوع جهش در بازی $[-b,b]$ واقع شده باشد، مقدار امید ریاضی w مربوط به احتمال آنکه یک جهش مفید باشد را در نظر بگیرید. w را می‌توان به‌صورت زیر نوشت:

$$w = E(w'|x_{0j} \in [-b,b] \quad \text{برای تمامی} \quad j \in [2,n]) \\ = \int_{-b}^{b} \dots \int_{-b}^{b} w' PDF(x_{02}) \dots PDF(x_{0n}) dx_{02} \dots dx_{0n} \tag{۲۱-٦}$$

با مقایسه‌ی معادله‌ی بالا با معادلات (٦-۱۱) و (٦-۲۰) خواهیم دید که

$$w = \frac{\sqrt{2\pi}}{2\sigma}\phi \tag{۲۲-٦}$$

حال ϕ را از معادله‌ی (٦-۱۵) جایگزین می‌کنیم:

$$w = (\frac{\sqrt{2\pi}}{2\sigma})(\frac{\sigma}{\sqrt{2\pi}})(1 - \frac{\sigma}{b\sqrt{2\pi}})^{n-1} \tag{۲۳-٦}$$

سپس برای به دست آوردن مقدار بهینه‌ی w، مقدار بهینه‌ی σ را از معادله‌ی (٦-۱۷) در معادله‌ی بالا قرار می‌دهیم:

$$w^* = \frac{1}{2}(1 - \frac{1}{n})^{n-1} \tag{۲٤-٦}$$

به یاد آورید که برای مقادیر کوچکِ x داریم: $\exp(-x) \approx 1 - x$. بنابراین، برای nهای بزرگ داریم: $(1 - \frac{1}{n}) \approx \exp\left(-\frac{1}{n}\right)$. بدین ترتیب خواهیم داشت

$$w^* = \frac{1}{2}(\exp\left(-\frac{1}{n}\right))^{n-1} \approx 1/2e \approx 0.18 \tag{۲۵-٦}$$

بنابراین، انحراف استاندارد بهینه‌ی σ^* دامنه‌ی جهشی را به دست خواهد داد که در ۱۸٪ موارد باعث ایجاد بهبود خواهد شد.

روچنبرگ همچنین یک تابع کروی، که در آن هدف مینیمم‌سازی تابع زیر بود را تحلیل نمود.

$$f(x) = \sum_{j=1}^{n} x_j^2 \tag{۲٦-٦}$$

وی دریافت که نرخ موفقیت جهش بهینه برای تابع کروی برابر ۲۷٪ است.

این نتایج تنها قابل اعمال به توابع خاصی هستند و تحت تقریب‌های ساده‌سازی به دست آمده‌اند، اما به قانون $1/5$، که مفید بودن آن در بسیاری از مسائل به اثبات رسیده است، منتهی شده‌اند. برای ماکزیمم ساختن

نرخ همگرایی، میزان انحراف استاندارد در یک ES باید به‌گونه‌ای تنظیم شود که نسبت جهش‌های موفق به کل جهش‌ها برابر $1/5$ باشد.

۶–۳ استراتژی تکامل $(\mu + 1)$

اولین تعمیم از ES-(1+1)، ES-$(\mu + 1)$ می‌باشد. در ES-$(\mu + 1)$، در هر نسل از μ والد استفاده می‌شود و μ پارامتری است که توسط کاربر تعیین می‌شود. همچنین هر والد دارای یک بردار σ است که دامنه‌ی جهش آن را کنترل می‌نماید. والدین با یکدیگر ترکیب شده تا فرزندان را به وجود آورند و سپس فرزندان دچار جهش می‌شوند. از میان μ والد و فرزندان، تعداد μ ذره که بهترین بوده‌اند انتخاب شده تا μ والد نسل بعد را تشکیل دهند. این الگوریتم در شکل ٦–٥ خلاصه‌سازی شده است. از آنجایی که ES-$(\mu + 1)$ بهترین ذرات هر نسل را نگه می‌دارد، بنابراین این الگوریتم یک الگوریتم نخبه‌گراست، بدین معنی که بهترین ذره‌ی آن هیچگاه از یک نسل به نسل دیگر بدتر نمی‌شود. ES-$(\mu + 1)$ گاهاً ES حالت ماندگار نیز خوانده می‌شود. از آنجایی که در انتهای هر نسل تنها یک ذره از جمعیت کل حذف می‌شود، این استراتژی را می‌توان *انقراض بدترین*، که در واقع نقطه‌ی مقابل *بقای برازنده‌ترین* است، نیز نامید.

والدین موجود در شکل ٦–٥ هم ویژگی‌های راه‌حل و هم واریانس جهششان را با هم ترکیب می‌کنند. با این حال، شکل ٦–٥ نوعی از انطباق واریانس جهش را که پیش از این در شکل ٦–٢ دیدیم، شامل نمی‌شود. در حقیقت، بعد از تعداد نسل‌های مشخصی که با μ متناسب است، مقدار σ در شکل ٦–٥ دچار فروپاشی شده و به یک مقدار می‌رسد که این مقدار شاید قدرت جهش مناسب را بازتاب داده و یا ندهد [بِیر، ۱۹۹۸]. شاید بهتر باشد شکل ٦–٥ آن گونه که نشان داده شده است پیاده‌سازی نشود، اما به هر حال نقطه‌ی آغازی است به سوی ES مؤثرتر و خود–انطباق در بخش ٦–٥.

ذرات تولید شده‌ی اتفاقی، $\{(x_k, \sigma_k)\} \leftarrow k \in [1, \mu]$

هر ذره‌ی x_k یک راه‌حل نامزد بوده و هر σ_k یک بردار انحراف استاندارد است

توجه داشته باشید که $x_k \in R^n$ و $\sigma_k \in R^n$ با عناصر مثبت می‌باشند

تا زمانی که شرایط توقف برآورده نشده است

دو والد را به صورت اتفاقی از جمعیت انتخاب کن $\{(x_k, \sigma_k)\}$

از یک روش بازترکیب برای ترکیب نمودن والدین و به دست آوردن فرزند استفاده کن

که با $(x_{\mu+1}, \sigma_{\mu+1})$ نشان داده می‌شود

$\Sigma_{\mu+1} \leftarrow diag(\sigma^2_{\mu+1,1}, \dots, \sigma^2_{\mu+1,n}) \in R^{n \times n}$

یک بردار اتفاقی r از $N(0, \Sigma_{\mu+1})$ تولید کن

$x_{\mu+!} \leftarrow x_{\mu+!} + r$

بدترین ذره را از جمعیت جدا کن، یعنی:

بهترین μ ذره از $\{(x_1, \sigma_1), \dots, (x_{\mu+1}, \sigma_{\mu+1})\} \leftarrow \{(x_k, \sigma_k)\}$

نسل بعد

شکل ٦-٥ شبه کد بالا طرح کلی استراتژی تکامل $\mu + 1$ را نشان می‌دهد. n ابعاد مسئله است.

مرحله‌ی بازترکیب در شکل ٦-٥ بیان می‌دارد، "برای ترکیب والدین از روش‌های بازترکیب استفاده کنید...". راه‌های زیادی برای انجام بازترکیب وجود دارد[1]. برش گسسته‌ی جنسی[2] با انتخاب هر فرزند از x_p یا x_q به‌صورت اتفاقی و همچنین انتخاب انحراف استاندارد هر فرزند از σ_p یا σ_q به‌صورت اتفاقی، به طوری که هر انتخاب از انتخاب دیگر مستقل باشد، صورت می‌پذیرد. این نوع برش گسسته است چرا که هر ویژگی فرزند تنها از یکی از والدین گرفته می‌شود و همچنین این نوع برش را جنسی می‌خوانند چرا که هر ویژگی فرزند از یکی از دو والدین گرفته می‌شود. برش گسسته‌ی جنسی در شکل ٦-٦ نمایش داده شده است.

گزینه‌ی دیگر برای انجام عمل بازترکیب، برش جنسی میانی[3] است. در این روش، ویژگی‌های فرزند در حد میانی ویژگی‌های والدین قرار می‌گیرد. برش جنسی میانی در شکل ٦-٧ نمایش داده شده است.

دیگر گزینه برای عمل بازترکیب، برش سراسری[4] یا برش پانمیکتیک[5] است. یک جمعیت پانمیکتیک جمعیتی است که در آن هر ذره یک زوج بالقوه برای سایر ذرات می‌باشد. در برش گسسته‌ی سراسری، هر ویژگی فرزند از یک والد که به‌صورت تصادفی از سراسر جمعیت انتخاب شده است، گرفته می‌شود. این

---

[1] در جامعه‌ی ES ترجیح بر آن است که به جای واژه‌ی برش، از واژه‌ی بازترکیب یا آمیختن استفاده شود.

[2] Discrete Sexual Crossover

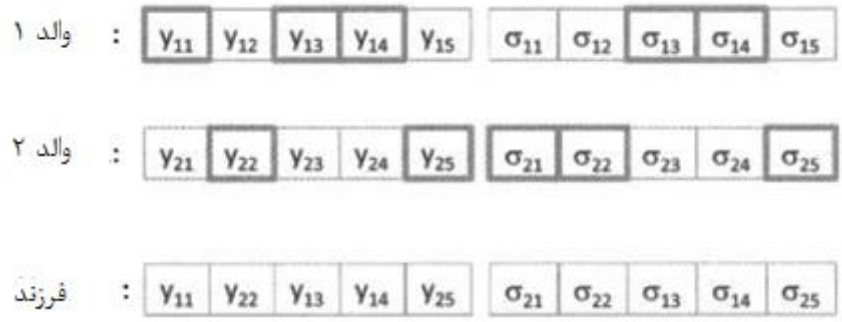
[3] Intermediate Sexual Crossover

[4] Global Crossover

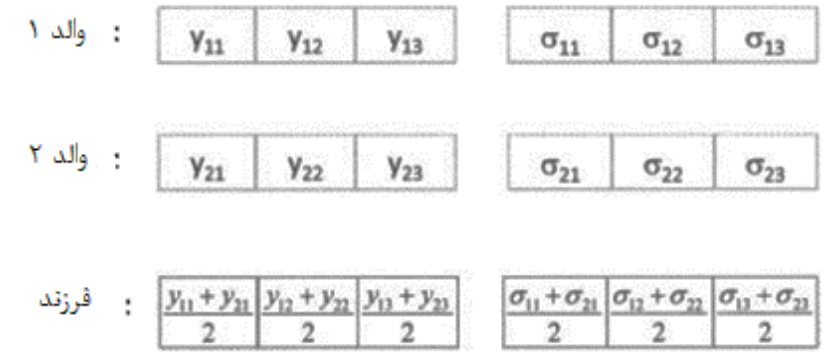
[5] Panmictic Crossover

موضوع در شکل ٦–٨ نشان داده شده است. برش‌های گسسته‌ی نشان داده شده در شکل‌های ٦–٦ و ٦–٨، برش غالب[1] نیز نامیده می‌شوند.

می‌توان برش سراسری و برش میانی را با یکدیگر ترکیب نمود و به برش سراسری میانی[2] دست یافت. در این روش، هر ویژگی فرزند ترکیبی است خطی از زوج والدینی که به‌صورت تصادفی انتخاب می‌شوند. این روش در شکل ٦–٩ نشان داده شده است. برش سراسری میانی روشی است که در عمل بیشتر استفاده شده و توصیه می‌شود [بِیر و اشوفل، ۲۰۰۲].



شکل ٦–٦ برش گسسته‌ی جنسی در یک ES. ابعاد مسئله برابر ۵ است ($n = 5$). هر ویژگی راه‌حل و انحراف استاندارد در فرزند به‌صورت تصادفی از یکی از دو والد انتخاب می‌شود.

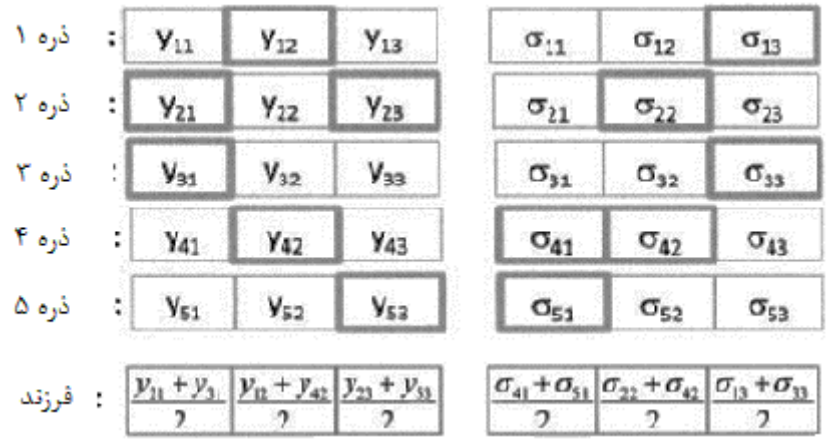


شکل ٦–٧ برش جنسی میانی در یک ES. ابعاد مسئله برابر ۵ است ($n = 5$). هر ویژگی راه‌حل و انحراف اسانداد در هر فرزند در میانه‌ی دو والد قرار دارد.

[1] Dominant Crossover
[2] Intermediate Global Crossover

شکل ٦–٨ برش گسسته‌ی سراسری در یک ES پج عضوی ($\mu = 5$). ابعاد مسئله برابر ۵ است ($n = 5$). هر ویژگی راه‌حل و انحراف استاندارد در فرزند، به‌صورت اتفاقی از کل جمعیت انتخاب می‌شود.

از دیگر روش‌های برش نیز می‌توان استفاده نمود. برای مثال می‌توان والد $x_p(0)$ را برای فرزند انتخاب نمود و n والد دیگر را $\{x_p(k)\}$، برای $k \in [1,n]$، هر کدام برای یکی از ویژگی‌های راه‌حل، انتخاب نمود. سپس ویژگی kاُم فرزند $(x_{\mu+1,k})$ را می‌توان با برش دادنِ $x_p(0)$ و $x_p(k)$ برای $k \in [1,n]$، تولید نمود. به همین ترتیب، انحراف استاندارد kاُمِ فرزند $(\sigma_{\mu+1,k})$، با برش میان $\sigma_p(0)$ و $\sigma_p(k)$ تولید خواهد شد. برای اطلاع از سایر اپراتورهای برش به بخش ۸–۸ مراجعه کنید.



شکل ٦–٩ برش سراسری میانی در یک ES پنج عضوه ($\mu = 5$) که در آن ابعاد مسئله برابر ۳ است ($n = 3$). هر ویژگی راه‌حل و هر انحراف استاندارد در فرزند، در میانه‌ی دو والد، که به‌صورت اتفاقی برگزیده شده‌اند، قرار دارد.

۶-۴ استراتژی تکامل $(\mu + \lambda)$ و (μ, λ)

دیگر تعمیمِ استراتژی تکامل، $(\mu + \lambda)$-ES می‌باشد. در $(\mu + \lambda)$-ES، ما یک جمعیت با اندازه‌ی μ در اختیار داشته و در هر نسل λ فرزند تولید می‌نماییم. پس از تولید فرزندان جمعیتی با $\mu + \lambda$ ذره در اختیار خواهیم داشت که هم فرزندان و هم والدین را شامل می‌شود. از این بین، μ ذره را که بهترین هستند به‌عنوان والدین نسل بعد نتخاب می‌کنیم.

یکی دیگر از استراتژی‌های تکامل که از آن زیاد استفاده می‌شود، (μ, λ)-ES است. در این الگوریتم، والدین نسل بعد، بهترین μ ذره از میان λ فرزند می‌باشند. به عبارت دیگر، هیچ یک از μ والد به نسل بعد راه پیدا نمی‌کنند؛ در عوض، زیرمجموعه‌ای μ عضوه از λ فرزند به‌عنوان والدین نسل بعد انتخاب می‌شوند. در (μ, λ)-ES حتما باید $\lambda \geq \mu$ باشد. والدین نسل قبل هرگز به نسل بعدی نخواهند رسید و زندگی هر ذره تنها به یک نسل محدود می‌شود.

در $(\mu + \lambda)$-ES و (μ, λ)-ES اگر $\mu > 1$ باشد، ES، چند عضوه خوانده می‌شود. علیرغم موفقیت‌های این تعمیم‌های ES، در ابتدا مخالفت‌های شدیدی با قرار دادن μ و λ بزرگتر از ۱ وجود داشت. مخالفتی که در برابر قرار دادن $\lambda > 1$ وجود داشت بر پایه‌ی این استدلال بود که اگر λ بزرگتر از ۱ باشد، استخراج اطلاعات با تاخیر مواجه می‌شود. از سوی دیگر، مخالفتی که در برابر قرار دادن $\mu > 1$ وجود داشت نیز بر پای‌ی این استدلال بود که در این صورت، بقای ذرات نامرغوب باعث کند شدن روند پیشرفت ES می‌شود [دِجونگ و همکاران، ۱۹۹۷].

در مواردی که تابع هدف نویزی و یا متغیر با زمان است، معمولاً (μ, λ)-ES بهتر از $(\mu + \lambda)$-ES عمل می‌کند (فصل ۲۱). در $(\mu + \lambda)$-ES، یک ذره‌ی (x, σ) ممکن است برازندگی خوبی داشته باشد اما به دلیل σ نامناسب، بهبود نیابد. بنابراین ممکن است ذره‌ی (x, σ) برای نسل‌های متمادی در جمعیت باقی بماند، بدون آنکه بهبودی داشته باشد که این موضوع باعث هدر رفتن یک مکان در جمعیت خواهد شد. (μ, λ)-ES این مشکل را با بیرون راندن تمامی ذرات در انتهای هر نسل و انتخاب بهترینِ فرزندان، مرتفع می‌سازد. این کار باعث می‌شود تا بقا برای نسل بعد به فرزندان با σ مناسب محدود شود. σ خوب، σای است که بردار جهشی به دست دهد که باعث بهبود x شود. در [بِیر و اشوفل، ۲۰۰۲]، (μ, λ)-ES برای مسائل با فضای جستجوی نامحدود و $(\mu + \lambda)$-ES برای مسائل با فضای جستجوی گسسته توصیه شده است.

شکل ٦-۱۰، (μ, λ)-ES و $(\mu + \lambda)$-ES را خلاصه‌سازی کرده است. توجه داشته باشید که اگر σ_k برای تمامی kها ثابت باشد، آنگاه σ_k از یک نسل به نسل بعد تغییری نخواهد کرد. شکل ٦-۱۰، همچون شکل

۶–۵، تطبیق واریانس جهش را شامل نمی‌شود. بنابراین، شکل ۶–۱۰ نباید به همان ترتیبی که نشان داده شده است پیاده‌سازی شود، اما نقطه‌ی شروعی برای ES خود–انطباق است. در بخش بعد، شکل ۶–۱۰ را برای رسیدن به (μ, λ)-ES و $(\mu + \lambda)$-ES خود–انطباق، بسط خواهیم داد.

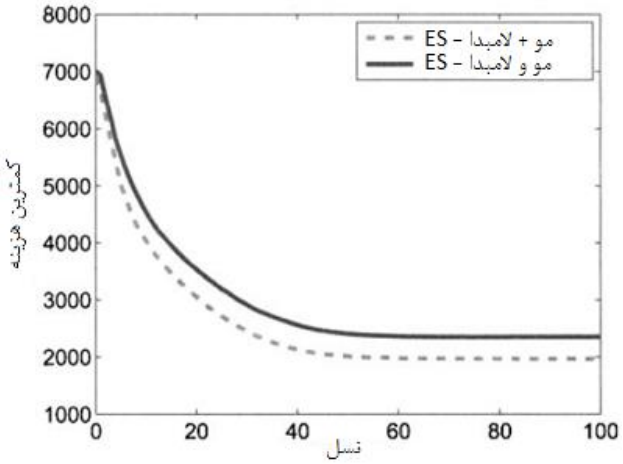
ذرات تولید شده‌ی اتفاقی، $\{(x_k, \sigma_k)\} \leftarrow k \in [1, \mu]$
هر ذره‌ی x_k یک راه‌حل نامزد بوده و هر σ_k یک بردار انحراف استاندارد است
توجه داشته باشید که $x_k \in R^n$ و $\sigma_k \in R^n$ با عناصر مثبت می‌باشند
تا زمانی که شرایط توقف برآورده نشده است
  برای $k = 1, \dots, \lambda$
    دو والد را به صورت اتفاقی از جمعیت انتخاب کن $\{(x_k, \sigma_k)\}$
    از یک روش بازترکیب برای ترکیب نمودن والدین و به دست آوردن فرزند استفاده کن
    که با (x'_k, σ'_k) نشان داده می‌شود
    $\Sigma'_k \leftarrow diag((\sigma'_{k1})^2, \dots, (\sigma'_{kn})^2) \in R^{n \times n}$
    یک بردار اتفاقی r از $N(0, \Sigma'_k)$ تولید کن
    $x_k \leftarrow x_k + r$
  k بعدی
  اگر این یک ES $(\mu + \lambda)$ است آنگاه
    بهترین μ ذره از $\{(x_k, \sigma_k)\} \cup \{(x'_k, \sigma'_k)\} \leftarrow \{(x_k, \sigma_k)\}$
  اگر این یک ES (μ, λ) است آنگاه
    بهترین μ ذره از $\{(x'_k, \sigma'_k)\} \leftarrow \{(x_k, \sigma_k)\}$
  پایان اگر
نسل بعد

شکل ۶–۱۰ شبه کد بالا طرح کلی استراتژی تکامل $(\mu + \lambda)$ و (μ, λ) را با فرض آنکه ابعاد مسئله برابر n است، به تصویر می‌کشد.

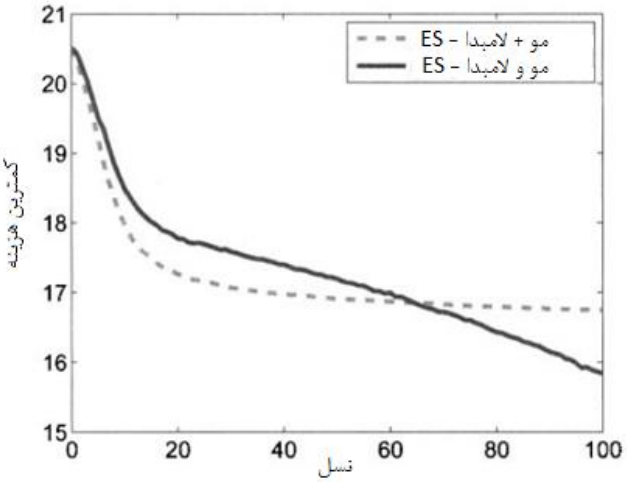
مثال ۶–۲

در این مثال دو استراتژی تکامل $(\mu + \lambda)$ و (μ, λ) را مقایسه کرده‌ایم. ما هر دو استراتژی را با $\mu = 10$ و $\lambda = 20$، برش سراسری جنسی و ابعاد مسئله‌ی ۲۰ اجرا نمودیم. شکل ۶–۱۱، میانگین عملکرد هر دو الگوریتم بر روی تابع محک اشوفل ۲,۲۶ را نشان می‌دهد. می‌توان دید که $(\mu + \lambda)$-ES، (μ, λ)-ES را از میدان به در می‌کند. این موضوع به دلیل محدودیت تک نسلی برای هر ذره و در نتیجه احتمال از بین رفتن ذرات خوب در (μ, λ)-ES، قابل انتظار است. با این حال، مقایسه‌ی عملکرد دو ES به نوع مسئله بستگی دارد. شکل ۶–۱۲ میانگین عملکرد دو الگوریتم را بر روی تابع محک آکلی نشان می‌دهد. در ابتدا

$(\mu + \lambda)$-ES، (μ, λ)-ES را از میدان به در می‌کند، اما در نهایت (μ, λ)-ES به $(\mu + \lambda)$-ES رسیده و حتی عملکرد بهتری از خود نشان می‌دهد. در برخی از موارد استفاده از (μ, λ)-ES مفید است. این موضوع هم به دلیل قابلیت انطباق زیاد آن با توابع نویزی و متغیر با زمان و هم به دلیل تأکید آن بر عمل کاوش برای برخی توابع است، به همین دلیل در برخی موارد عملکرد بهتری نسبت به $(\mu + \lambda)$-ES خواهد داشت.



شکل ٦–١١ مثال ٦–٢: نرخ همگرایی $(\mu + \lambda)$-ES و (μ, λ)-ES بر روی تابع محک اشوفل ۲,۲٦ که بر روی ۱۰۰ شبیه‌سازی میانگین گرفته شده است. $(\mu + \lambda)$-ES به وضوح بهتر از (μ, λ)-ES عمل می‌کند.



شکل ٦–١٢ مثال ٦–٢: نرخ همگرایی $(\mu + \lambda)$-ES و (μ, λ)-ES بر روی تابع محک اشوفل ۲,۲٦ که بر روی ۱۰۰ شبیه‌سازی میانگین گرفته شده است. $(\mu + \lambda)$-ES در ابتدا بهتر از (μ, λ)-ES عمل می‌کند اما در نهایت این (μ, λ)-ES است که عملکرد بسیار بهتری نسبت به $(\mu + \lambda)$-ES خواهد داشت.

۶–۵ استراتژی تکامل (μ, k, λ, ρ)

از شکل ٦–١٠ به یاد آورید که هر فرزند دارای دو والد است. اما دلیلی برای محدود کردن تعداد والدین به عدد ۲ وجود ندارد. به جای آن می‌توانیم بیش از ۲ والد را با یکدیگر ترکیب نماییم و از ρ برای نشان دادن تعداد والدین مشارکت‌کننده در ایجاد هر فرزند استفاده کنیم. در انتهای بخش ٦–٣، در مورد برخی اپراتورهای چند–والده بحث کردیم و در بخش ٨–٨ در مورد برخی دیگر از اپراتورهای ممکن بحث خواهیم کرد. همچنین می‌توان برای هر ذره از جمعیت یک بیشینه‌ی عمر (k) تعریف کرد. اگر بیشینه‌ی عمر یک نسل باشد، آنگاه $k = 1$ بوده و یک (μ, λ)-ES خواهیم داشت چرا که والدین مجاز به زنده ماندن برای نسل بعد نخواهند بود. اگر بیشینه‌ی عمر نامحدود باشد، آنگاه $k = \infty$ بوده و ما یک $(\mu + \lambda)$-ES خواهیم داشت چرا که هیچ محدودیتی در زنده ماندن برای نسل بعد برای والدین وجود نداشته و تا زمانی که والد یکی از μ ذره‌ی برتر در جمعیت باشد، می‌تواند به نسل بعد راه پیدا کند، بی‌آنکه مدت حضورش در جمعیت مهم باشد. در کل ممکن است برای جلوگیری از رکود، به خصوص در مسائل متغیر با زمان، مجبور به محدود کردن عمر ذرات ES باشیم (بخش ۲۱–۲ را ببینید).

ترکیب این دو تعمیم، (μ, k, λ, ρ)-ES را نتیجه خواهد داد [اشوفل، ۱۹۹۵]. جمعیت (μ, k, λ, ρ)-ES، μ والد دارد، هر ذره‌ی آن دارای بیشینه عمری برابر k بوده و در هر نسل از آن، λ فرزند که هر کدامشان دارای ρ والد هستند، تولید می‌شود.

۶–۶ استراتژی‌های تکامل خود–انطباق

الگوریتم‌های ES که تا به اینجا مورد مطالعه قرار دادیم، آزادی چندانی برای تنظیم انحراف‌های استانداردِ جهش‌ها در اختیار ما قرار نمی‌دهند. تنها گزینه‌ی ما تا به اینجا، (1+1)-ES انطباقی از شکل ٦–٢ است که انحراف‌های استاندارد را بر اساس نرخ موفقیت جهش تنظیم می‌نماید. این الگوریتم را می‌توان با امتحان کردن همه‌ی λ جهش‌ها در هر نسل و پیگیری نمودن آن‌هایی که به بهبود منجر می‌شوند، به $(1 + \lambda)$-ES خود–انطباق بسط داد. با این حال، هیچ راه روشنی برای بسط دادن این ایده به $(\mu + \lambda)$-ES و یا (μ, λ)-ES با $\mu > 1$ وجود ندارد. در این حالت، فرزندان نه تنها از جهش، بلکه از ترکیب والدینشان شکل گرفته‌اند. بنابراین، تعیین نرخ جهش مناسب با استفاده از مقایسه‌ی برازندگی فرزند با برازندگی والد چندان معنی نخواهد داشت. با این حال، همان‌طور که برای پیدا کردن x بهینه، ویژگی‌های راه‌حل $\{x_i\}$ را برای $i \in [1, n]$ دچار جهش می‌کنیم، برای پیدا کردن σ بهینه نیز می‌توانیم عناصر $\{\sigma_i\}$ از بردار انحراف استاندارد را

دچار جهش نماییم. پس از آنکه فرزند (x', σ') تولید شد، فرزند را به قرار زیر دچار جهش می‌نماییم [اشوفل، ۱۹۷۷]، [باک، ۱۹۹۶، بخش ۲-۱-۲]:

$$\sigma_i' \leftarrow \sigma_i' exp(\tau' \rho_0 + \tau \rho_i)$$
$$x_i' \leftarrow x_i' + \sigma_i' r_i \qquad (۶-۲۷)$$

برای $i \in [1, n]$ که در آن ρ_0، ρ_i و r_i متغیرهای اسکالر تصادفی از $N(0,1)$ هستند. τ و τ' نیز پارامترهای میزان‌سازی می‌باشند. فاکتور $\tau' \rho_0$ امکان ایجاد تغییر کلی در نرخ جهش x' را فراهم کرده و فاکتور $\tau \rho_i$ امکان ایجاد تغییرات در نرخ جهش عناصر خاص x' را ایجاد می‌کند. فرم جهشِ σ' به‌گونه‌ای است که مثبت باقی ماندن σ' را تضمین می‌کند.

توجه داشته باشید که ρ_0 و ρ_i به همان اندازه که ممکن است مثبت باشند، ممکن است منفی نیز باشند. این بدان معنی است که توانِ نمایی در معادله‌ی (۶-۲۷) به همان اندازه که ممکن است بزرگتر از ۱ باشد، امکان کوچکتر بودنش از ۱ نیز وجود دارد. بدین ترتیب σ' همانقدر که ممکن است افزایش یابد، احتمال کاهش نیز دارد. اشوفل نتیجه می‌گیرد که این روش جهش نسبت به تغییرات τ و τ' مقاوم است، اما رابطه‌ی آن‌ها را به‌صورت زیر پیشنهاد می‌کند [اشوفل، ۱۹۷۷]، [باک، ۱۹۹۶، بخش ۲-۱-۲]:

$$\tau = P_1(\sqrt{2\sqrt{n}})^{-1}$$
$$\tau' = P_2(\sqrt{2n})^{-1} \qquad (۶-۲۸)$$

که در آن n ابعاد مسئله بوده و P_1 و P_2 ثابت‌های تناسبی هستند که معمولاً برابر با ۱ در نظر گرفته می‌شوند.

پیاده‌سازی جهش به ترتیب نشان داده شده در معادله‌ی (۶-۲۷)، یعنی وقوع جهش در σ' پیش از وقوع جهش در x'، حائز اهمیت است. این بدان خاطر است که از σ' برای ایجاد جهش در x' استفاده می‌شود به‌طوری که برازندگی x' به دقیق‌ترین نحو ممکن، میزان تناسب σ' را نشان می‌دهد. این ایده‌ها، استراتژی تکامل $(\mu + \lambda)$ و (μ, λ) خود-انطباقِ نشان داده شده در شکل ۶-۱۳ را نتیجه می‌دهند. توجه داشته باشید که در شکل ۶-۱۳، ES خود-انطباق نام داشته و شکل ۶-۲ تنها نمایانگر یک ES انطباقی می‌باشد. ES خود-انطباق معرفی شده در [روچنبرگ، ۱۹۷۳] به احتمال زیاد مهمترین ES در تحقیقات الگوریتم‌های تکاملی است. امروزه همه‌ی الگوریتم‌های تکاملی به‌صورت مجازی، از نوعی روش خود-انطباقی برای تنظیم

نمودن پارامترهای میزان‌سازیِ الگوریتم استفاده می‌کنند. به علاوه، [بِیر و دِب[1]، ۲۰۰۱] نشان داده حتی الگوریتم‌های تکاملی بدونِ خاصیت خود–انطباقی صریح، رفتاری خود–انطباق از خود به نمایش می‌گذارند. تعبیر الگوریتم‌های تکاملی به‌عنوان الگوریتم‌های خود–انطباق و اثر رفتار خود–انطباقی بر عملکرد الگوریتم‌های تکاملی، به‌عنوان تکلیفی مهم برای تحقیقات آینده باقی می‌ماند.

در الگوریتم شکل ٦–۱۳، ماتریس کوواریانس جهش (Σ'_k) قطری است. در کل، می‌توان از یک کوواریانس غیرقطری برای ایجاد جهشِ r استفاده نمود. در این حالت باید به جای عناصر قطری، تمامی عناصر ماتریس کوواریانس را بهینه نمود [باک،۱۹۹٦، بخش ۲,۱].

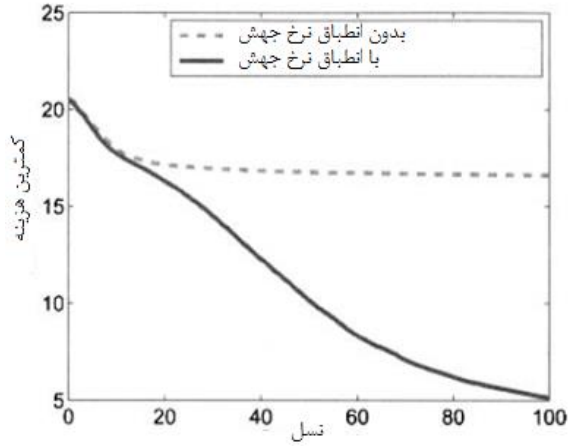
مثال ٦–۳

در این مثال از $(\mu + \lambda)$-ES برای بهینه‌سازی تابع محک اَکلی استفاده می‌کنیم و به مقایسه‌ی $(\mu + \lambda)$-ES استاندارد که در شکل ٦–۱۰ نشان داده شده است و $(\mu + \lambda)$-ES خود–انطباقِ شکل ٦–۱۳ می‌پردازیم. ما در هر دو الگوریتم از $\mu = 10$، $\lambda = 20$، برش گسسته‌ی جنسی و ابعاد مسئله‌ای برابر با ۲۰ استفاده می‌کنیم. ما همچنین از مقادیر استاندارد نشان داده شده در معادله‌ی (٦–۲۸) برای τ و τ' و همچنین از $P_1 = P_2 = 1$ استفاده می‌نماییم. شکل ٦–۱٤، نرخ همگرایی $(\mu + \lambda)$-ES استاندارد و $(\mu + \lambda)$-ES خود–انطباق را مقایسه می‌نماید. می‌توان دید که ES خود–انطباق بسیار سریعتر از ES استاندارد همگرا می‌شود. شکل ٦–۱۵ مقادیر انحراف استانداردِ σ_{ki} برای $i \in [1, n]$، برای بهترین ذره‌ی (x_k, σ_k) از جمعیت در آخرین نسل را نشان می‌دهد. شکل ٦–۱۵ بیانگر آن است که انحراف‌های استاندارد برای ویژگی‌های مختلف به طُرُق متفاوتی نمو کرده‌اند. در واقع آن‌ها به‌گونه‌ای نمو نموده‌اند تا اثربخشی جهش‌ها بهینه شود.

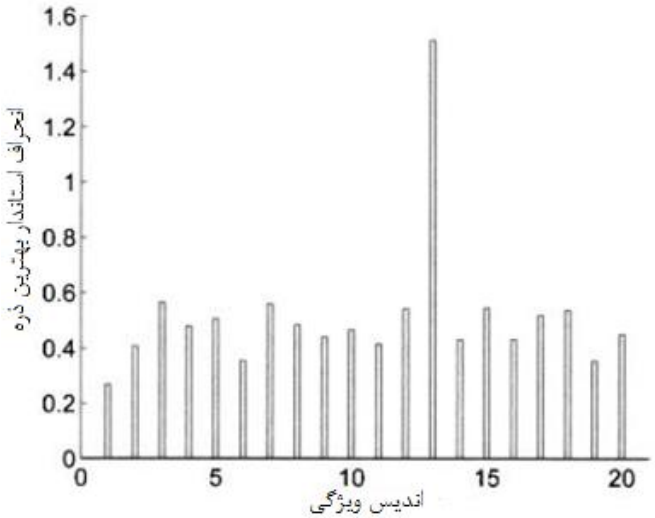
[1] Deb

مقادیر ثابت τ و τ' از معادله‌ی (6.28) را مقداردهی اولیه کن
ذرات تولید شده‌ی اتفاقی، $k \in [1, \mu] \leftarrow \{(x_k, \sigma_k)\}$
هر ذره‌ی x_k یک راه‌حل نامزد بوده و هر σ_k یک بردار انحراف استاندارد است
توجه داشته باشید که $x_k \in R^n$ و $\sigma_k \in R^n$ با عناصر مثبت می‌باشند
تا زمانی که شرایط توقف برآورده نشده است
  برای $k = 1, \ldots, \lambda$
    دو والد را به صورت اتفاقی از جمعیت انتخاب کن $\{(x_k, \sigma_k)\}$
    از یک روش بازترکیب برای ترکیب نمودن والدین و به دست آوردن فرزند استفاده کن
      که با (x'_k, σ'_k) نشان داده می‌شود
    یک اسکالر اتفاقی مانند ρ_0 را از $N(0,1)$ تولید کن
    بردار $[\rho_1 \ldots \rho_n]$ را به صورت اتفاقی از $N(0, I)$ تولید کن
    $\sigma'_{ki} \leftarrow \sigma'_{ki} \exp(\tau' \rho_0 + \tau \rho_i)$
    $\Sigma'_k \leftarrow diag((\sigma'_{k1})^2, \ldots, (\sigma'_{kn})^2) \in R^{n \times n}$
    یک بردار اتفاقی r از $N(0, \Sigma'_k)$ تولید کن
    $x'_k \leftarrow x'_k + r$
  k بعدی
  اگر این یک ES- $(\mu + \lambda)$ است آنگاه
    بهترین μ ذره از $\{(x_k, \sigma_k)\} \cup \{(x'_k, \sigma'_k)\} \leftarrow \{(x_k, \sigma_k)\}$
  اگر این یک ES- (μ, λ) است آنگاه
    بهترین μ ذره از $\{(x'_k, \sigma'_k)\} \leftarrow \{(x_k, \sigma_k)\}$
  پایان اگر
نسل بعد

شکل ٦-١٣ شبه کد بالا طرح کلی استراتژی‌های تکامل $(\mu + \lambda)$ و (μ, λ) خود-انطباق را نشان می‌دهد. n ابعاد مسئله است.



شکل ٦-١٤ مثال ٦-٣: همگرایی الگوریتم‌های استاندارد و خود-انطباق ES-($\mu + \lambda$) بر روی تابع محک آکلی ۲۰ بعدی که بر روی ۱۰۰ شبیه‌سازی میانگین گرفته شده است. ES خود-انطباق ، که انحراف‌های استانداردجهش را به‌صورت خودکار تنظیم می‌نماید، بسیار سریعتر از ES استاندارد همگرا می‌شود.



شکل ٦-١٥ مثال ٦-٣: مقادیر انحراف استانداردِ بهترین ذره در آخرین نسل از ES خود-انطباق که به تابع آکلیِ ۲۰ بعدی اعمال شده است. ۲۰ انحراف استاندارد، متناظر با ۲۰ بعدِ مسئله‌ی بهینه‌سازی وجود دارد. ES خود-انطباق انحراف استاندارد جهش را به‌گونه‌ای تنظیم می‌نماید تا اثربخشی جهش‌ها افزایش یابد.

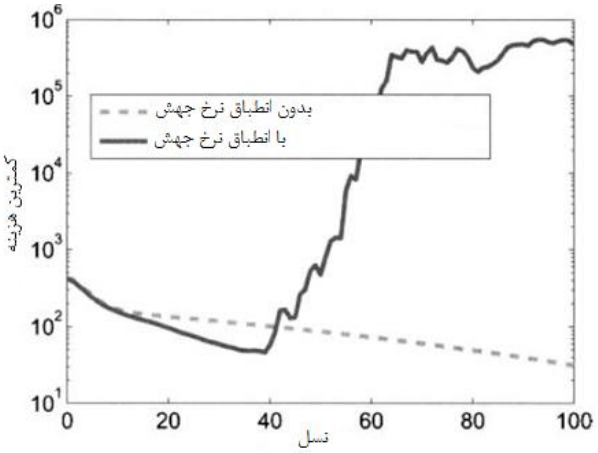
مثال ٦-٤

این مثال همانند مثال ٦-٣ است تنها با این تفاوت که در این مثال از ES-(μ, λ) و تابع محک گرینوانک استفاده نموده‌ایم. شکل ٦-١٦ میانگین نرخ همگراییِ ES-(μ, λ) استاندارد و ES-(μ, λ) خود-انطباق را

مقایسه می‌کند. می‌توان دید که ES خود–انطباق بسیار ضعیف عمل می‌کند. دلیل این اتفاق آن است که اگرچه جهشِ σ' در معادله‌ی (٦–۲۷) مقدار میانه‌ای برابر ۱ دارد (بدین معنی که همان قدر که احتمال کاهش σ' وجود دارد احتمال افزایش آن نیز هست)، اما جهش σ' دارای میانگینی بزرگتر از ۱ است. آرگومان تابع نمایی در معادله‌ی (٦–۲۷) برابر است با جمع دو متغیرِ تصادفی گاوسی با میانگین صفر. برای سادگی فرض کنید آرگومانِ x از تابع نمایی $\exp(x)$ یک متغیر تصادفی گاوسی با میانگین صفر و واریانس ۱ باشد. در این حالت تابع نمایی دارای مقدارِ میانه‌ای برابر ۱ است، اما مقدار میانگین آن برابر خواهد بود با

$$E[\exp(x)] = \int_{-\infty}^{\infty} PDF(x)\exp(x)\,dx$$
$$= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}}\exp(-x^2/2)\exp(x)\,dx \qquad (٦–۲۹)$$
$$= \exp(\frac{1}{2}) \approx 1.65$$

می‌توان دید که جهشِ σ' بیشتر متمایل به افزایش σ' است تا کاهش آن. این موضوع می‌تواند به ایجادِ مزیت مقادیر σ' بزرگ در فرزندان منجر شود. اگر در انتهای هر نسل تمامی والدین از جمعیت حذف شوند، همانند (μ, λ)-ES، مقادیری از σ' که به طرز غیر قابل قبولی بزرگ هستند ممکن است در جمعیت ادامه یافته و باعث عملکرد ضعیف الگوریتم شوند.



شکل ٦–١٦ مثال ٦–٤: نرخ همگراییِ روش‌های استاندارد و خود–انطباقِ (μ, λ)-ES بر روی تابع ۲۰ بعدی گرینوانک که بر روی ۱۰۰ شبیه‌سازی میانگین گرفته شده است. ES خود–انطباق که انحراف‌های استاندارد جهش را به‌صورت خودکار تنظیم می‌نماید، عملکرد بسیار ضعیفی دارد.

۶-۷ انطباق ماتریس کوواریانس

یکی از انواع موفق ES، CMA-ES می‌باشد. CMA[1] مخفف *انطباق ماتریس کوواریانس* است [هانسن، ۲۰۱۰]. هدف CMA-ES، که موفقیت‌های زیادی بر روی توابع محک از خود نشان داده، آن است که تا حد ممکن جهش‌های ES را در حد فاصل تابعِ هدف جای دهد. این کار تنها در مورد توابع هدفِ درجه دوم به طور کامل موفقیت‌آمیز است، اما بسیاری از توابع هدف را می‌توان در نزدیکی مقدار بهینه‌شان با یک تابع درجه دوم تقریب زد. معایب CMA-ES عبارتند از استراتژی انطباق پیچیده و همچنین تنظیمات پیچیده‌ی مربوط به پارامترهای میزان‌سازی. در اینجا ما یک نسخه‌ی ساده از CMA-ES که *ES خود-انطباقی ماتریس کوواریانس (CMSA-ES*[2]*)* نام دارد [بِیِر و سِندهاف[3]، ۲۰۰۸] را معرفی می‌کنیم. ایده‌ی به کار رفته در CMSA-ES، یادگیریِ شکل فضای جستجو در طول تکامل و تطبیق واریانسِ جهش می‌باشد. شکل ٦-۱۷، طرح کلی CMSA-ES را نشان می‌دهد.

در شکل ٦-۱۷، τ یک پارامتر یادگیری است که میزان سرعت انطباق مقادیر σ_k را تعیین می‌کند. مقادیر σ_k میزان قدرت جهش را کنترل می‌کنند. توجه داشته باشید که در اینجا، بر خلاف شکل ٦-۱۳، σ_k یک اسکالر است. ثابت زمانیِ τ_c میزان سرعت انطباق ماتریس کوواریانسِ C، که دامنه‌های نسبی و همبستگی جهش‌ها در طولِ هر بعد را کنترل می‌کنند، تعیین می‌نماید. مقادیر توصیه شده برای τ و τ_c به شرح زیرند [بِیِر و سِندهاف، ۲۰۰۸]:

$$\tau \leftarrow 1/\sqrt{2n}$$
$$\tau_c \leftarrow 1 + n(n+1)/2\mu \qquad (۳۰-٦)$$

هر چند با قراردادن مقادیرِ متغیر با زمان برایر τ و τ_c می‌توان عملکرد مطلوبتری را به دست آورد.

مقدار $\sqrt{c}$ در شکل ٦-۱۷ را می‌توان از چند راه مختلف محاسبه نمود. CMA-ES از تجزیه‌ی طیفی یا تجزیه‌ی مقدار ویژه استفاده می‌کند [هانسن و اُستر مِیِر[4]، ۲۰۰۱] و CMSA-ES از تجزیه‌ی چولسکی[5] استفاده می‌نماید [بِیِر و سِندهاف، ۲۰۰۸].

شکل ٦-۱۷ میانگین‌های $\bar{\sigma}$، $\bar{x}$ و $\hat{s}$ را به‌صورت میانگین‌های ساده نشان می‌دهد. با این حال، می‌توان آن‌ها را به‌صورت میانگین‌های وزنی نیز محاسبه نمود، بدین معنی که به ذرات برازنده‌تر وزن بیشتری

[1] Covariance Matrix Adaptation
[2] Covariance Matrix Self-Adaptive ES
[3] Sendhoff
[4] Ostermeier
[5] Cholesky

اختصاص یابد. با توجه به وجود تعادلی زیبا میان اثربخشی و سادگی بر روی توابع محک، به نظر می‌آید که CMSA-ES حرف‌های زیادی در تحقیقات آینده برای گفتن داشته باشد که از جمله‌ی آن می‌توان به پیوند آن با سایر الگوریتم‌های تکاملی اشاره نمود.

مقادیر ثابت τ و τ_c را مقداردهی اولیه کن
ماتریس واحد $C \leftarrow I = n \times n$
ذرات تولید شده‌ی اتفاقی، $\{(x_k, \sigma_k)\} \leftarrow k \in [1, \mu]$
هر ذره‌ی x_k یک راه‌حل نامزد بوده و هر σ_k یک بردار انحراف استاندارد است
توجه داشته باشید که $x_k \in R^n$ و $\sigma_k \in R$ با عناصر مثبت می‌باشند
تا زمانی که شرایط توقف برقرار نشده است
$\bar{\sigma} \leftarrow \sum_{k=1}^{\mu} \sigma_k / \mu$
$\bar{x} \leftarrow \sum_{k=1}^{\mu} x_k / \mu$
برای $k = 1, \ldots, \lambda$
اسکالر اتفاقی گاوسی $= N(0,1) \leftarrow \tau$

شکل ٦–۱۷ شبه کد بالا استراتژی تکامل خود–انطباقِ ماتریس کوواریانس (CMSA-ES) را به تصویر می‌کشد به‌طوری که در آن n ابعاد مسئله است. برای آگاهی از جزئیات به متن مراجعه کنید.

۶–۸ نتیجه‌گیری

ما در مورد (1+1)-ES، تعمیم آن یعنی $(\mu + 1)$-ES و حتی تعمیم بیشتر آن یعنی $(\mu + \lambda)$-ES بحث نمودیم. همچنین در مورد (μ, λ)-ES نیز سخن راندیم. می‌توان مشاهده نمود که یک ES مشابه یک GA است تنها با این تفاوت که GAها راه‌حل‌ها را به‌صورت رشته‌های بیت کدگذاری می‌کنند اما ESها همواره بر روی پارامترهای پیوسته عمل می‌کنند. اگرچه GAها معمولاً به‌گونه‌ای بسط داده می‌شوند که بر روی پارامترهای پیوسته عمل کنند، اما این موضوع به‌عنوان یک تفاوت فلسفی میان دو الگوریتم باقی می‌ماند: ES تمایل دارد بر تماثیلی عمل کند که به شرح مسئله نزدیک‌ترند، در حالی که GAها مایل‌اند بر تماثیلی عمل نمایند که پیشتر از شرح اصلی مسئله حذف شده‌اند. یک تفاوت دیگر میان دو الگوریتم آن است که GAها بر بازترکیب تأکید داشته و ES بر جهش تأکید دارد. این موضوع می‌تواند در انتخاب الگوریتم مناسب به هنگام مواجه با یک مسئله‌ی بهینه‌سازی به ما کمک کند. اگر در یک مسئله‌ی خاص جستجو مهم‌تر از استخراج باشد، بهتر است از یک ES استفاده شود. از سوی دیگر، اگر استخراج مهم‌تر از جستجو باشد، آنگاه بهتر است از یک GA استفاده نماییم[1].

[1] برای آگاهی از استخراج (انتفاع) و جستجو (کاوش) به بخش ۲–۷–۵ مراجعه نمائید.

در تمامی انواع ES که تاکنون در موردشان بحث نمودیم، مکانیزم انتخاب قاطع است بدین معنا که همواره بهترین μ ذره برای نسل بعد انتخاب می‌شوند. از این لحاظ می‌توان انتخاب ذرات برای نسل بعد را به‌صورت احتمالی انجام داد. برای مثال، در $(\mu + \lambda)$-ES، می‌توانستیم از انتخاب چرخ رولت برای انتخاب والدین نسل بعد به‌صورت احتمالی بهره ببریم. در این حالت هر قسمت از چرخ رولت با برازندگی ذره‌ی متناظر متناسب خواهد بود.

موارد اضافی در مورد ES در [باک و اشوفل، ۱۹۹۳] و [بِیِر، ۲۰۱۰] ارایه شده‌اند. یک مدل مارکوف از ES در [فرانکیس[1]، ۱۹۹۸] ارایه شده است. استراتژی‌های تکامل برای مسائل چندهدفه (فصل ۲۰ را ببینید) نیز در [رودالف و اشوفل، ۲۰۰۸] مورد بحث قرار گرفته‌اند.

مسائل

تمارین نوشتاری

٦–١ نتیجه‌گیریِ ارائه شده در انتهای این فصل بیان می‌دارد که هرگاه در مسئله‌ای کاوش و جستجو مورد نیاز باشد، ES مناسب‌تر بوده و هرگاه انتفاع و استخراج مورد نیاز باشد، GA مناسب‌تر خواهد بود. در چه نوع مسائلی کاوش مطلوب‌تر است و در چه نوع مسائلی استخراج؟

٦–٢ نشان دهید خط دوم و سوم معادله‌ی (٦–٩) با هم برابرند.

٦–٣ با استفاده از معادله‌ی (٦–١٦)، معادله‌ی (٦–١٧) را ثابت کنید.

٦–٤ فرض کنید می‌خواهیم از (1+1)-ES برای بهینه‌سازی تابع کروی تک بعدی استفاده کنیم. احتمال آنکه یک جهش با انحراف استاندارد σ باعث بروز بهبودی در راه‌حل نامزد x شود، چه قدر است؟

٦–٥ معادله‌ی (٦–٢٧) نشان می‌دهد که انحراف استانداردِ جهشِ ES خود–انطباق با یک فاکتور نمایی تغییر می‌کند. از سویی معادله‌ی (٦–٢٩) بیان می‌دارد که میانگین این فاکتور بزرگتر از ١ است.، که این موضوع ممکن است باعث عملکرد ضعیف ES شود. معادله‌ی (٦–٢٧) را چگونه تغییر دهیم تا میانگینِ این فاکتور برابر ١ شود؟

تمارین کامپیوتری

٦–٦ الگوریتم (1+1)-ES، ارایه شده در شکل ٦–٢ را برای مینیمم ساختن تابع کروی ١٠ بعدی (ضمیمه‌ی ج.١–١ را ببینید)، بر روی دامنه‌ی $[-5.12, +5.12]$ شبیه‌سازی کنید. مقدار اولیه‌ی انحراف استانداردِ جهشِ

[1] Francqis

هر بعد را برابر $\frac{0.1}{2\sqrt{3}}$ قرار دهید. شبیه‌سازی را برای ۵۰۰ نسل اجرا کرده و مقدار هزینه را در هر نسل محاسبه نمایید. شبیه‌سازی را ۵۰ بار تکرار کرده و میانگینِ ۵۰ مقدارِ هزینه برای هر نسل را محاسبه نمایید. مقدار میانگینِ هزینه برای هر نسل را به‌عنوان تابعی از شماره‌ی نسل رسم نمایید. این کار را برای $C =$ 1.0 و 0.6, 0.8 انجام دهید. کدام مقدارِ C بهترین نتیجه را به دست می‌دهد؟

۶–۷ مسئله‌ی ۶–۶ را تکرار کنید. این بار به جای شبیه‌سازی برای سه مقدار مختلف از C، شبیه‌سازی را برای مقدار پیش‌فرض C و سه مقدار آستانه‌ی نرخِ موفقیتِ جهشِ مختلف (ϕ_{thresh}) انجام دهید. یعنی به جای استفاده از مقدار پیش‌فرض $\phi_{thresh} = 1/5$، از 0.4 و $\phi_{thresh} = 0.1, 0.2$ استفاده نمائید. کدام مقدار ϕ_{thresh} بهترین عملکرد را به دست می‌دهد؟

۶–۸ تابع دهلیزِ دو بعدی را بر روی دامنه‌ی $[-50, +50]$ با ثابت‌های $c_0 = 0$، $c_1 = 1$ و $b = 10$ رسم کنید.

۶–۹ از الگوریتم $(\mu + \lambda)$-ES برای مینیمم ساختن تابع کروی ۱۰ بعدی بر روی دامنه‌ی $[-5.12, +5.12]$ با $\mu = 10$ و $\lambda = 20$ استفاده کنید. انحراف استانداردِ جهشی هر بعد را برابر $\frac{0.1}{2\sqrt{3}}$ قرار دهید. شبیه‌سازی را برای ۱۰۰ نسل انجام داده و مقدار هزینه در هر نسل را ثبت کنید. شبیه‌سازی را ۵۰ بار تکرار کرده و میانگینِ ۵۰ مقدارِ هزینه برای هر نسل را محاسبه نمائید. مقدار هزینه‌ی مینیمم را بر حسب شماره‌ی نسل رسم کنید. این کار را برای برش گسسته‌ی جنسی، گسسته‌ی سراسری، جنسی میانی و سراسری میانی انجام دهید. کدام یک از انواع برش بهترین عملکرد را به دست می‌دهد؟

۶–۱۰ از معادله‌ی (۶–۲۹) به یاد آورید که اگر $x \sim N(0,1)$ باشد، آنگاه $\exp(x)$ دارای میانه‌ای برابر ۱ و میانگینِ ۱٫۶۵ خواهد بود. معادله‌ی (۶–۲۷) نسان می‌دهد که انحراف استانداردِ جهشِ ES خود–انطباق با یک فاکتور نمایی تغییر می‌کند. میانه و میانگین این فاکتور نمایی را با فرض $x = 10$به‌صورت عددی تقریب بزنید. افزایش مقدار n چه تأثیری در میانه و میانگین فاکتور نمایی خواهد داشت؟

۶–۱۱ مقادیر پیش‌فرض P_1 و P_2 برابر ۱ می‌باشند اما ممکن است مقادیرِ دیگر عملکردِ بهتری را به دست دهند. از $(\mu + \lambda)$-ES خود–انطبا با $\mu = 10$ و $\lambda = 20$ برای مینیمم‌سازی تابع کروی ۱۰ بعدی بر روی دامنه‌ی $[-5.12, +5.12]$ استفاده کنید. مقدار اولی‌ی انحراف استانداردِ جهشِ هر بعد را برابر $\frac{0.1}{2\sqrt{3}}$ قرار دهید. شبیه‌سازی را برای ۱۰۰ نسل انجام داده و مقدار هزینه را در هر نسل ثبت نمایید. شبیه‌سازی را ۶۰ بار تکرار کرده و از ۵۰ مقدار هزینه در هر نسل میانگین بگیرید. میانگینِ مقدار هزینه‌ی مینیمم را به‌عنوان تابعی از شماری نسل رسم نمایید. این کار را برای $P_1 = P_2 = 0.1$، $P_1 = P_2 = 1$ و $P_1 = P_2 = 10$ انجام دهید. کدام مقادیر P_1 و P_2 بهترین عملکرد را به دست می‌دهند؟

فصل هفتم

برنامه‌نویسی ژنتیک

اگر ماشین‌ها بتوانند وظایفی را انجام دهند که در موردشان روش دقیقی وجود ندارد، بسیار مفیدتر واقع خواهند شد.

ریچارد فرایدبرگ[1] [فرایدبرگ، ۱۹۵۸]

الگوریتم‌های ژنتیک و سایر الگوریتم‌های تکاملی مشابه، تکنیک‌های بهینه‌سازی نیرومندی می‌باشند، اما دارای یک محدودیت ذاتی می‌باشند: آن‌ها ساختار مسئله‌ی مفروض را در نمایش راه‌حل‌های نامزدشان جای می‌دهند. برای مثال، اگر بخواهیم از یک GA برای حل یک مسئله‌ی بهینه‌سازی پیوسته با ۱۰ متغیر استفاده کنیم، آنگاه کروموزوم‌های GA معمولاً به صورت $(x_1, x_2, \dots, x_{10})$ نمایش داده می‌شوند. این نکته هم یک مزیت و هم یک مضرت دارد. مزیت این نکته در آن است که به مهندس اجازه می‌دهد تا اطلاعات مختص مسئله را به فرمِ نمایشی از راه‌حل کدگذاری کند. اگر بدانیم که می‌توان راه‌حل مورد نظر را بدون هیچ اشکالی با ۱۰ پارامتر حقیقی نشان داد، آنگاه تعریف کروموزوم به صورت $(x_1, x_2, \dots, x_{10})$ بسیار منطقی خواهد بود. با این حال، شاید در یک مسئله معلوم نباشد کدام پارامتر نیازمند بهینه‌سازی است. همچنین، شاید از ساختار پارامترهایی که باید بهینه شوند اطلاعاتی در دست نداشته باشیم. آیا این پارامترها اعداد حقیقی‌اند؟ یا ماشین‌های فضای حالت؟ یا برنامه‌های کامپیوتری؟ یا ارائه‌های مختلط؟ یا زمانبندی و یا یک چیز دیگر؟

برنامه‌نویسی ژنتیک (GP[2]) تلاشی است برای تعمیم الگوریتم‌های تکاملی به الگوریتمی که بتواند علاوه بر ساختن بهترین راه‌حل برای یک مسئله با ساختار خاص، ساختار بهینه را نیز بیابد. GP برنامه‌های کامپیوتری را برای حل مسائل بهینه‌سازی نمو می‌دهد. این خصیصه‌ی GP آن را از سایر الگوریتم‌های تکاملی ممتاز می‌سازد؛ سایر الگوریتم‌های تکاملی راه‌حل‌ها را رشد می‌دهند در حالی که GP برنامه‌هایی را رشد می‌دهد که می‌توانند راه‌حل‌ها را محاسبه نمایند. در واقع، این یکی از اهداف ابتدایی جامعه‌ی هوش مصنوعی بود. آرتور ساموئل[3]، یکی از پیشگامان آمریکاییِ ابتدایی در زمینه‌ی هوش مصنوعی، در سال ۱۹۵۹ چنین نوشت:"نیاز به چنین برنامه‌نویسی‌های مفصلی سرانجام با برنامه‌نویسی کامپیوترها به‌گونه‌ای که بتوانند از تجارب بیاموزند، از بین خواهد رفت."

ویژگی‌های بنیادین GP را می‌توان در سه قاعده‌ی کلی خلاصه نمود [کوزا، ۱۹۹۲، فصل ۲]. اول آنکه GP، که برنامه‌های کامپیوتری را رشد می‌دهد، به ما این امکان را می‌دهد تا شیوه‌های راه‌حلی به دست آوریم که قابل اعمال به گستره‌ی وسیعی از مسائل خواهند بود. بسیاری از مسائل مهندسی را می‌توان با ساختارهایی که به‌صورت برنامه‌های کامپیوتری، درخت انتخاب یا معماری شبکه می‌باشند، حل نمود. دوم آنکه، GP به

---

[1] Friedberg
[2] Genetic programming
[3] Arthur Samuel

اندازه‌ی سایر الگوریتم‌های تکاملی راه‌حل‌هایش را محدود نمی‌کند. برنامه‌های رشد یافته اختیار آن را دارند تا بهترین و مناسب‌ترین اندازه، شکل و ساختار را برای مسئله در نظر بگیرند. سوم آنکه GP برنامه‌های کامپیوتری را با استفاده از استنتاج و قیاس رشد می‌دهد. این موضوع هم نقطه‌ی قوت و هم نقطه‌ی ضعف است. GP برنامه‌های کامپیوتری را، مانند انسان‌ها، با ساختن آن‌ها به روشی منطقی و استراتژیک رشد نمی‌دهد. با این حال، برخی مسائل در برابر استقرا رام‌نشدنی هستند. اگر بخواهیم بر اساس یک مجموعه از نمونه‌های یادگیری یک برنامه‌ی کامپیوتری بنویسیم و از تکنیک‌های استاندارد برنامه‌نویسی استفاده نماییم، کار سختی در پیش رو خواهیم داشت. اما GP برنامه‌های کامپیوتریِ بهینه را به روش قیاسی و استنتاجی خواهد ساخت.

نتایج اولیه در برنامه‌نویسی ژنتیک

آلن تورینگ، یکی از پدران علم کامپیوتر و هوش مصنوعی، هنگام نوشتن جمله‌ی زیر در یک مقاله‌ی معروف در سال ۱۹۵۰، چیزی مانند GP را در خیال خود پرورانده بود:

"ما نمی‌توانیم انتظار داشته باشیم در اولین تلاش خود یک بچه ماشین خوب پیدا کنیم. ابتدا باید یک ماشین را تعلیم داد و دید که ماشین چه قدر خوب می‌آموزد. سپس می‌توان یک ماشین دیگر را امتحان نمود و دید که آیا بهتر از ماشین اول است یا نه" [تورینگ، ۱۹۵۰، صفحه‌ی ٤٥٦].

ریچارد فرایدبرگ که دانش آموخته‌ی هوش کامپیوتری در دهه‌ی ۱۹۵۰ بود و سپس به شغلی در زمینه‌ی دارو روی آورد، یکی از اولین کسانی بود که بر روی مسائلی کار کرد که می‌توان آن‌ها را در دسته‌ی برنامه‌نویسی ژنتیک قرار داد. فرایدبرگ برنامه‌های کامپیوتری نوشت که می‌توانستند سایر برنامه‌های کامپیوتری را رشد دهند و آن برنامه‌ها نیز خود می‌توانستند به حل مسائل بپردازند. کار وی در اواخر دهه‌ی ۱۹۵۰ با عنوان "یک ماشین یادگیری" منتشر شد [فرایدبرگ، ۱۹۵۸]، [فرایدبرگ و همکاران، ۱۹۵۸]. وی در آن زمان به دلیل وجود محدودیت‌های توان محاسباتی، مجبور به اتخاذ برخی میانبرها بود. برای مثال، وی برنامه‌های مشابه را گروه‌بندی کرد و فرض نمود برازندگی آن‌ها با هم همبسته‌اند و بدین ترتیب توانست تعداد محاسبات برازندگی را کاهش دهد. این روش طلایه‌دار سایر روش‌های کاهش محاسبات برازندگی است که امروزه در الگوریتم‌های تکاملی می‌بینیم (فصل ۲۱ را ببینید). توان محاسباتی (سرعت و حافظه) از سال ۱۹٦۰ تا ۲۰۱۰ حدود یک میلیون برابر شده است، با این حال امروزه ما همان قدر نگران توان پردازشی هستیم که فرایدبرگ در دهه‌ی ۱۹۵۰ بود.

یکی از پیشروان GP مدرن، GA متغیر است که توسط استفان اسمیت[1] در سال ۱۹۸۰ و در رساله‌ی دکتری وی ارائه شد [اسمیت، ۱۹۸۰]. در این نوع GA هر ذره در جمعیت یک مجموعه از قوانین تصمیم‌گیری را ارائه می‌دهد. یکی دیگر از کارهایی که منادی GP امروزی بود، مقاله‌ی ریچارد فورسیث[2] در سال ۱۹۸۱ می‌باشد که قوانین طبقه‌بندی الگوها را نمو می‌دهد [فورسیث، ۱۹۸۱]. شاید بتوان گفت نایکل کرامر[3] در سال ۱۹۸۵ اولین مقاله‌ی خاصه‌ی GP را نگارش کرد [کرامر، ۱۹۸۵]. این مقاله تقریباً بر پایه‌ی رساله‌ی اسمیت می‌باشد. در سال ۱۹۹۰، هوگو دِ گاریس[4] برای اشاره به بهینه‌سازی شبکه‌های عصبی با استفاده از الگوریتم‌های ژنتیک از عبارت "برنامه‌نویسی ژنتیک" استفاده نمود [دِ گاریس، ۱۹۹۰]، اما مفهوم این عبارت به مرور زمان دستخوش تغییر شد و امروزه به معنای تکامل برنامه‌های کامپیوتری است. کتابِ سالِ ۱۹۹۲ جان کوزا، که امروزه نیز بهترین کتاب برای آشنایی با موضوع حاضر است، ابزاری بود برای محبوب‌سازی تحقیقات در زمینه‌ی GP [کوزا، ۱۹۹۲]. از آن زمان تا کنون کوزا سه کتاب دیگر در زمینه‌ی GP نوشته است [کوزا، ۱۹۹٤]، [کوزا و همکاران، ۱۹۹۹]، [کوزا و همکارا، ۲۰۰۵]. این کتاب‌ها به بحث در مورد کاربردهای عملی و جنبه‌های پیشرفته‌تر GP می‌پردازند. اطلاعات بیشتر در مورد تاریخ اولیه‌ی GP را می‌توان در [کوزا، ۲۰۱۰] یافت.

مروری بر فصل

ما این فصل را با بحث در مورد زبان برنامه‌نویسی Lisp در بخش ۷-۱ آغاز می‌کنیم. غالباً از Lisp برای GP استفاده می‌شود چرا که ساختار آن قابلیت تبعیت از قوانین برش و جهش را دارد. بخش ۷-۲ یک دید کلی و بنیادین از GP که شامل برخی تصمیمات مربوط به طراحی نیز می‌شود را به دست می‌دهد. بخش ۷-۳ به بحث در مورد یک مثال از GP برای کمترین زمان کنترل می‌پردازد. بخش ۷-٤ در مورد GP باد کرده بحث خواهد کرد. GP باد کرده در واقع بیانگر تمایل GP برای افزایش کنترل نشده‌ی اندازه است. بخش ۷-۵ استفاده از GP برای رشد دادن راه‌حل‌ها به جای برنامه‌های کامپیوتری، در مدارهای الکتریکی و سایر طراحی‌های مهندسی را مورد بحث قرار خواهد داد. بخش ۷-٦ به ارائه‌ی روش‌هایی برای مدل‌سازی ریاضی عملکرد GP، خصوصاً با استفاده از نظریه شِما می‌پردازد. خوانندگانی که تنها به دنبال دانشی ابزاری برای استفاده از GP هستند می‌توانند از خواندن این بخش صرف نظر نمایند. در انتها بخش ۷-۷ خلاصه‌ای از فصل را ارائه خواهد داد و پیشنهاداتی را برای آینده‌ی GP مطرح خواهد کرد.

---

[1] Stephan Smith
[2] Richard Forsyth
[3] Nicheal Cramer
[4] Hugo De Garis

۷–۱ LISP: زبان برنامه‌نویسی ژنتیک

برنامه‌نویسی ژنتیک غالباً با استفاده از زبان برنامه‌نویسی LISP پیاده‌سازی می‌شود چرا که ساختار LISP به زیبایی با مفهوم برش و جهش پیوند می‌خورد. این بخش دیدی کلی از LISP به دست داده و یک توصیفِ ادراکی از چگونگی ترکیب برنامه‌های LISP برای ایجاد برنامه‌های جدید به دست خواهد داد.

به‌طور کلی تکامل برنامه‌های کامپیوتری چالش‌آمیز است چرا که معمولاً برنامه‌ها به‌گونه‌ای ارائه می‌شوند که در آن‌ها جهش و برش عملی نیستند. این اولین مانعی است که برای نمو دادن برنامه‌های کامپیوتری باید بر آن غلبه کنیم. برای مثال، دو برنامه‌ی استاندارد MATLAB را در نظر بگیرید:

برنامه اول	برنامه دوم
$if\ x < 1$ $\ z = [1,2,3,4,5];$ $else$ $\ for\ i = 1:5$ $\ \ z(i) = i/x;$ $\ end$ end	$For\ i = 1:10$ $\ If\ x > 5$ $\ \ z(i) = xi;$ $\ else$ $\ \ z(i) = x/I;$ $\ end$ end

چگونه می‌توان عمل برش را در این نوع برنامه‌ها انجام داد؟ یک عمل برش اگر دقت کافی به دستورات زبانِ برنامه‌نویسی نداشته باشد، ممکن است منجر به ایجاد برنامه‌ای شود که از لحاظ نحوی غیرمجاز هستند (برنامه‌ای غیرمجاز است که اجرا نشده و یا حتی کامپایل نشود). برای مثال در جدول بالا اگر دو خط اول از برنامه‌ی اول را با دو خط اول از برنامه‌ی دوم جابه‌جا کنیم، به برنامه‌ی زیر خواهیم رسید:

برنامه فرزند

```
For i = 1:10
  If x < 5
Else
  For i = 1:5
    z(i) = i/x;
  end
end
```

واضح است که برنامه‌ی فرزند یک برنامه‌ی مجاز نیست. بسیاری از جهش‌ها و برش‌هایی که بر روی برنامه‌های MATLAB صورت می‌پذیرند، منجر به ایجاد برنامه‌های غیرمجاز خواهند شد. این موضوع در مورد سایر زبان‌های برنامه‌نویسی نظیر C، JAVA، Fortran، Basic، Perl و Python نیز صدق می‌کند. همه‌ی

این زبان‌ها ساختار یکسانی دارند، پس عمل جهش و برش میان برنامه‌هایی که به این زبان‌ها نوشته شده‌اند به سادگی صورت نمی‌پذیرد.

با این حال یک زبان وجود دارد که برای جهش و برش مناسب است. این زبان LISP نام دارد [وینستون و هورن، ۱۹۸۹]. LISP که در سال ۱۹۵۸ اختراع شد، به احتمال زیاد دومین زبان برنامه‌نویسی دنیاست چرا که تنها یک سال پس از Fortran ارائه شد. LISP خلاصه‌ی عبارت "پردازش لیستی[1]" است. لیست‌های پیوندی، بزرگترین و عمده‌ترین ساختار در LISP هستند و همین ویژگی آن را به محتمل‌ترین انتخاب برای کاربردهای هوش مصنوعی در سال‌های اولیه‌ی آن تبدیل نمود. LISP به دلیل تفاوتی که با سایر زبان‌ها دارد، دیگر از محبوبیت چندانی برخوردار نیست. با این حال، این زبان به دلیل مناسب بودن برای جهش و برش، در میان محققان GP از محبوبیت خاصی برخوردار است.

کدِ برنامه‌های LISP با پرانتز نوشته می‌شود و در آن آرگومان تابع به دنبال نام تابع می‌آید. برای مثال، کد زیر x و ۳ را با یکدیگر جمع می‌کند

$$(+\ x\ 3)$$

این یک مثال از نشان‌گذاری پیشوندی[2] است چرا که اپراتورهای ریاضی بر ورودیشان مقدم‌اند. در LISP یک عبارت پرانتزی گاها "عبارت s"[3] که خلاصه‌ی عبارت سمبلیک است، نیز خوانده می‌شود. می‌توان به عبارت‌های s به عنوان توابعی نگاه کرد که مقادیر مورد محاسبه‌شان را باز می‌گردانند. آن دسته از عبارت‌های s که چندین مقدار را محاسبه می‌نمایند، آخرین مقدار محاسبه شده را باز می‌گردانند.$(+\ x\ 3)$ نه تنها x را با ۳ جمع می‌نماید، بلکه $x+3$ را به سطح بالاتر اجرائیات تابع بازمی‌گرداند.

بگذارید چند مثال دیگر ارائه بدهیم. کد زیر مقدار $\cos(x+3)$ را محاسبه می‌کند:

$$\cos(x+3)$$

کد زیر نیز مینیمم مقدارِ میان $\cos(x+3)$ و $z/14$ را محاسبه می‌نماید:

$$(\min\ (\cos\ (+\ x\ 3)\ )\ (/z14)$$

کد بعدی نیز اگر $z>4$باشد، مقدار x را در y کپی می‌نماید:

$$(if\ (>\ z\ 4)\ (setf\ x\ y)\ )$$

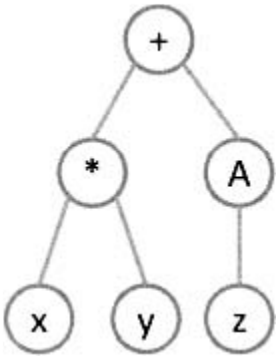
[1] List Processing
[2] Prefix Notation
[3] S-expression

توجه داشته باشید که عبارت‌های s مانند مجموعه‌ها هستند. بدین معنا که عبارت‌های s خود می‌توانند شامل دیگر عبارت‌های s باشند. در عبارتِ s بالا، $(> z\ 4)$ و $(setf\ x\ y)$ عبارت‌های s هستند که خود جزیی از یک عبارت s سطحِ بالاتر یعنی $(if\ (> z\ 4)\ (setf\ x\ y))$ هستند.

دلیل آنکه LISP قابل رشد و نمو می‌باشد آن است که عبارت‌های s مستقیماً با ساختار درخت یا همان درخت نحو[1] در ارتباط هستند. برای مثال، یک عبارت s از LISP برای محاسبه‌ی $xy + |z|$ را می‌توان به‌صورت زیر نوشت:

$$(+\ (*\ x\ y)\ (abs\ z))$$

این عبارت s را می‌توان با درخت نحوی مانند شکل ۷-۱ نشان داد. برای تفسیر این درخت نحو باید از پایینِ آن شروع کرد و به بالای آن رسید. شکل ۷-۱ نشان می‌دهد که x و y در پایین درخت واقع شده و با یک عملگر ضرب به هم متصل شده‌اند. این موضوع xy یا همان $(*\ xy)$ را نتیجه می‌دهد. می‌توان دید که این زیرعبارت s با یک زیردرخت در شکل ۷-۱ مرتبط است. نمادهایی که در پایین‌ترین سطح از یک درخت ظاهر می‌شوند (مانند x، y و z در شکل ۷-۱)، برگ خوانده می‌شوند.



شکل ۷-۱ درخت نحو برای تابع $xy + |z|$ که با عبارت s به‌صورت $(+\ (*\ x\ y)(abs\ z))$ نمایش داده می‌شود. گره‌ی "A" نشانگر عملگر قدر مطلق است.

همچنین شکل ۷-۱ نشان می‌دهد z در انتهای پایینی درخت واقع شده و تنها عملگر قدر مطلق بر روی آن عمل می‌کند. این موضوع عبارت $|z|$ یا $(abs\ z)$ را نشان می‌دهد. بار دیگر می‌بینیم که یک زیرعبارت s با یک زیردرخت در شکل ۷-۱ متناظر است.

[1] Syntax Tree

در نهایت، شکل ۷-۱ نشان می‌دهد که دو عبارت xy و $|z|$ در گره‌ی جمع و در انتهای بالایی درخت به یکدیگر متصل می‌شوند و بدین ترتیب $xy + |z|$ یا $(+ (* x\ y)\ (abs\ z))$ به وجود می‌آید. می‌بینیم که عبارت s سطح بالا با کل ساختار درخت در شکل ۷-۱ متناظر است.

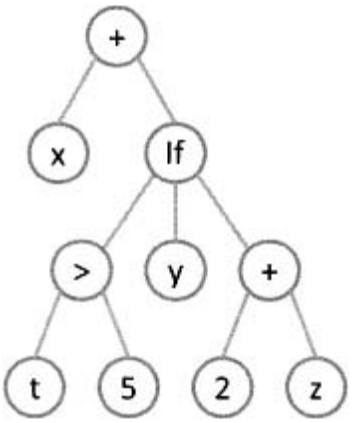
به‌عنوان مثالی دیگر، تابعی را در نظر بگیرید که در صورتی که $t > 5$ باشد مقدار $(x + y)$ و در غیر این صورت مقدار $(x + 2 + z)$ را برگرداند:

$If\ t > 5$
$Return\ (x + y)$
$Else$
$Return\ (x + 2 + z)$

این تابع را با استفاده از نشان‌گذاری‌های Lisp می‌توان به‌صورت زیر نوشت:

(if (>t5)(+xy)(+x2z))

شکل ۷-۲ درخت نحو این مثال را به تصویر می‌کشد. توجه داشته باشید که بسیاری از توابع Lisp، مانند تابع جمع در مثال بالا، می‌توانند هر تعداد آرگومان را بپذیرند.



شکل ۷-۲ درخت نحو برای تابعِ "اگر $t > 5$ آنگاه $(x + y)$ را باز گردان، در غیر این صورت $(x + 2 + z)$ را بازگردان".

برش با برنامه‌های Lisp

تناظر میان عبارت‌های s و زیردرخت‌ها باعث می‌شود تا بتوان عملیاتی مانند برش را بر روی برنامه‌های Lisp پیاده نمود. برای مثال توابع زیر را در نظر بگیرید:

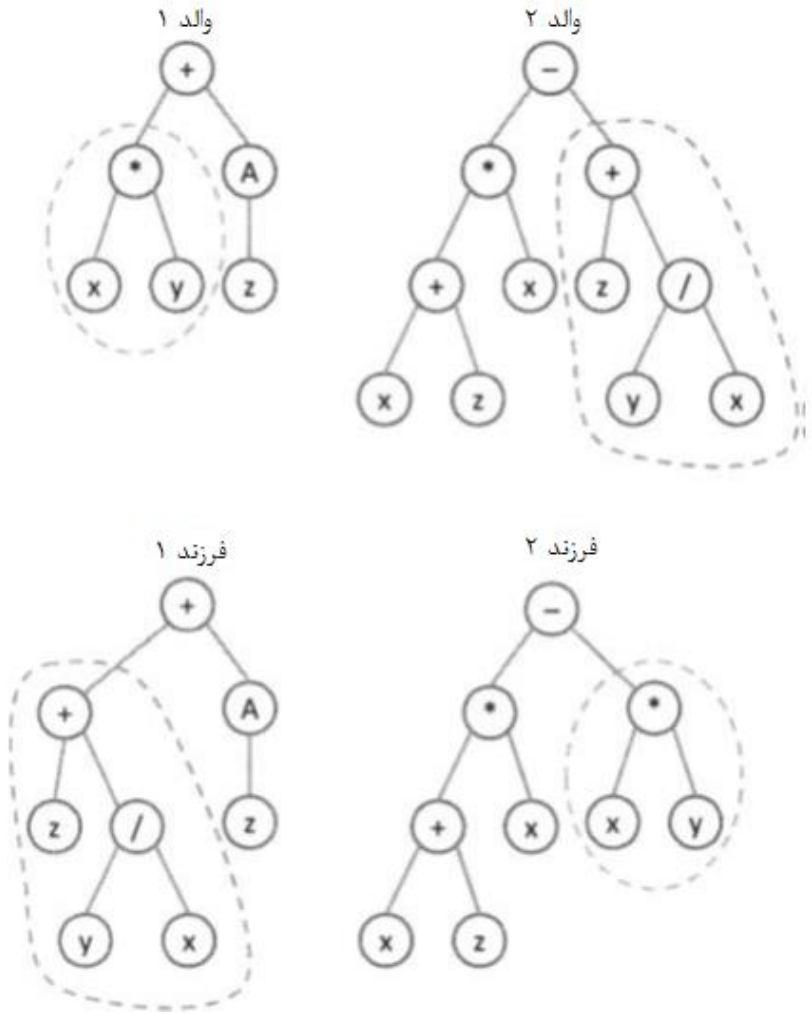
$$\text{والد 1:} \quad xy + |z| \Rightarrow \big(+(* \; x \; y)\big) \; abs \; z) \qquad (۱-۷)$$
$$\text{والد 2:} \quad (x+z)x - \left(z - \frac{y}{x}\right) \Rightarrow (-(* \; (+ \; x \;\; z) \; x) \; (+z \; (/ \; yx)))$$

این دو تابعی والد در شکل ۷-۳ نشان داده شده‌اند. ما می‌توانیم با انتخاب یک نقطه‌ی برش تصادفی در دو والد و جابه‌جا نمودن زیردرخت‌های موجود در زیر آن نقاط، دو تابع فرزند ایجاد کنیم. برای مثال فرض کنید که در والد اول گره‌ی ضرب و در والد دوم، دومین گره‌ی جمع به عنوان نقاط برش انتخاب شوند. شکل ۷-۳ چگونگی جابه‌جایی زیردرخت‌های واقع شده در زیر این نقاط و در نتیجه تولید فرزندان را نشان می‌دهد. توابع فرزندی که به این گونه ایجاد می‌شوند همواره از لحاظ نحوی معتبر خواهند بود.

حال برش میان عبارت‌های s از معادله‌ی ۷-۱ را در نظر بگیرید. معادله‌ی زیر بر زوج‌های پرانتزی، که با زیردرخت‌های شکل ۷-۳ متناظر هستند، تاکید کرده و نشان می‌دهند چگونه می‌توان با جابه‌جایی زوج‌های پرانتزیِ عبارت‌های s اصلی (والدین)، عبارت‌های s جدید (فرزندان) را تولید نمود:

$$\left\{\begin{matrix}(+ \; [* \; xy]) \; abs \; z) \\ (-(* \; (+ \; x \; z)x)\{+ \; z \; (/y \; x)\})\end{matrix}\right\} \Rightarrow \left\{\begin{matrix}(+ \; (+ \; z(/ \; y \; x)\}) \; abs \; z) \\ (-(* \; (+ \; x \; z)x)[* \; x \; y])\end{matrix}\right. \qquad (۲-۷)$$

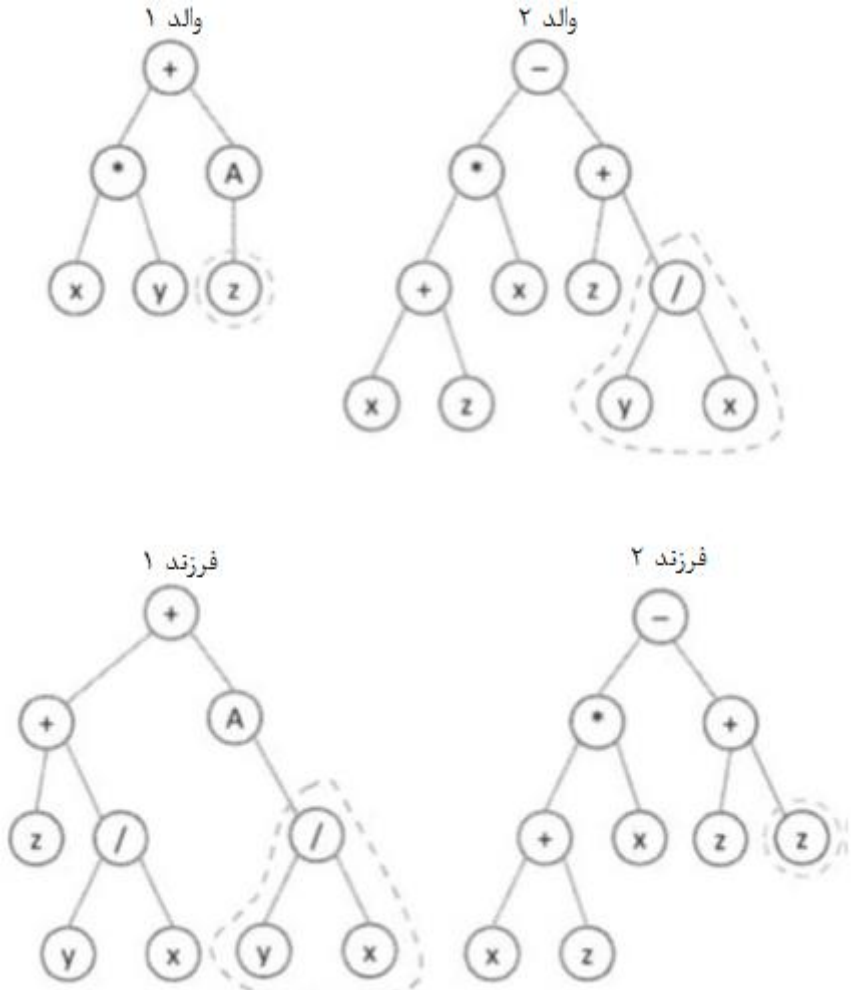
در معادله‌ی بالا، عبارت‌های s برش خورده با فونت پررنگ نمایش داده شده‌اند. این نوع برش، برشِ درخت- محور نام دارد. هر عبارتِ s در درخت نحو را می‌توان با هر عبارتِ s دیگر جایگزین نمود و در عین حال درخت نحو معتبر خواهد ماند. این همان موضوعی است که Lisp را به زبانی تمام عیار برای GP مبدل می‌سازد. اگر بخواهیم میان دو برنامه‌ی Lisp عمل برش انجام دهیم، کافیست یک پرانتز چپ تصادفی در والد اول پیدا کرده و سپس پرانتزی راست متناظر با آن را پیدا کنیم. عبارت میان این دو پرانتز یک عبارتِ s معتبر خواهد بود. این کار را در والد دوم نیز انجام می‌دهیم. پس از آنکه دو عبارت s را در دو والد جابه‌جا نمودیم، دو فرزند خواهیم داشت. عمل جهش نیز به روشی مشابه‌صورت می‌پذیرد، بدین ترتیب که یک عبارت s را که به‌صورت اتفاقی انتخاب شده است را با یک عبارت s که به‌صورت اتفاقی تولید شده است، جایگزین می‌نماییم. این نوع جهش، جهشِ درخت- محور نام دارد.



شکل ۷-۳ برشِ دو درخت نحو (والدین) برای ایجاد دو درخت نحو جدید (فرزندان). برش‌هایی که بدین گونه انجام می‌شوند همواره درختانِ نحوِ فرزند معتبر ایجاد خواهند کرد.

شکل ۷-٤ یک مثال دیگر از برش را نشان می‌دهد. در اینجا نیز والدین همان والدین شکل ۷-۳ می‌باشند، اما این بار گره‌ی z در والد اول و گره‌ی تقسیم در والد دوم به عنوان نقاط برش انتخاب شده‌اند. عملیات برش در شکل ۷-٤ را می‌توان با استفاده از نشان‌گذاری عبارتِ s به‌صورت زیر نشان داد (در اینجا نیز عبارات s بریده شده با فونت پررنگ نمایش داده شده‌اند)

$$\left\{\begin{matrix}(+\ (*\ x\ y))\ abs\ z) \\ (-\ (*\ (+x\ z)x\ )\{+z[/y\ x]\})\end{matrix}\right\} \Rightarrow \left\{\begin{matrix}(+(*\ x\ y))abs\ [/yx]) \\ (-(*\ (+x\ z)\ x)(+z\ z))\end{matrix}\right. \qquad (۳-۷)$$



شکل ۷-٤ برشِ دو درخت نحو (والدین) برای ایجاد دو درخت نحو جدید (فرزندان). در اینجا نیز والدین همان والدین شکل ۷-۳ می‌باشند اما نقاط برش فرق کرده‌اند.

۷-۲ مبانی برنامه‌نویسی ژنتیک

حال که با چگونگی ترکیب برنامه‌های Lisp آشنا شدیم، ابزار لازم برای تعمیم الگوریتم‌های تکاملی به تکاملِ برنامه‌های کامپیوتری را در اختیار داریم. شکل ۷-۵ یک طرح کلیِ ساده از GP را نشان می‌دهد. می‌توان دید که GP مشابه یک الگوریتم ژنتیک است، با این تفاوت که GA راه‌حل‌ها را برای یک مسئله‌ی بهینه‌سازی نمو می‌دهد در حالی که GP برنامه‌هایی کامپیوتری را نمو می‌دهد که خود می‌توانند یک مسئله‌ی بهینه‌سازی را حل نمایند.

قبل از آنکه بتوانیم یک GP را پیاده‌سازی نماییم باید چند تصمیم اساسی بگیریم:

۱. اندازه‌گیری برازندگی در شکل ۷–۵ چگونه است؟
۲. معیارهای پایان‌دهی در شکل ۷–۵ چه هستند؟
۳. مجموعه‌ی ترمینال برای برنامه‌های کامپیوتری چیست؟ به عبارت دیگر، چه نمادهایی می‌توانند در برگ‌های درخت نحو ظاهر شوند؟
٤. مجموعه‌ی توابع برای برنامه‌های کامپیوتری چیست؟ به عبارت دیگر، چه نمادهایی می‌توانند در گره‌های غیرترمینالِ درخت ظاهر شوند؟
۵. جمعیت آغازین برنامه‌های کامپیوتری را چگونه باید ایجاد کنیم؟
٦. چه پارامترهای دیگری را باید جهت کنترل GP تعریف کنیم؟

برخی از این تصمیمات، تصمیماتی هستند که در مورد سایر الگوریتم‌های نیز مورد نیاز می‌باشند اما برخی دیگر از آن‌ها مختص GP هستند. زیربخش‌های آتی به نوبت در مورد هر یک از این موارد بحث خواهند کرد.

```
{برنامه‌های کامپیوتری تولید شده اتفاقی} والدین
تا زمانی که شرایط توقف برقرار نشده است
    برازندگی هر والد موجود در جمعیت را محاسبه کن
    ∅ ← فرزندان
    تا زمانی که |والدین| > |فرزندان|
        از مقادیر برازندگی برای انتخاب احتمالاتی والدین p1 و p2 استفاده کن
        p1 و p2 را با یکدیگر ترکیب کرده تا فرزندان c1 و c2 به دست آیند
        فرزندان ∪ {c1, c2} ← فرزندان
    حلقه
    برخی از فرزندان را به صورت اتفاقی دچار جهش کن
    فرزندان ← والدین
نسل بعد
```

شکل ۷–۵ مروری مفهومی بر یک برنامه‌ی ژنتیک ساده.

۷–۲–۱ اندازه‌گیری برازندگی

اندازه‌گیری برازندگی در شکل ۷–۵ چگونه است؟ این تصمیم تصمیمی است که در مورد همه‌ی الگوریتم‌های تکاملی باید گرفته شود، اما در مورد GP پیچیدگی بیشتری دارد. یک برنامه‌ی کامپیوتری باید

بتواند برای گستره‌ی وسیعی از ورودی‌ها، شرایط اولیه و محیط‌ها به خوبی عمل نماید. برای مثال، برنامه‌ای که برای پیدا کردن مسیر ماهواره از یک مدار به مدار دیگر، به‌گونه‌ای که مصرف سوخت بهینه باشد نوشته شده است، باید برای پارامترهای مختلف ماهواره و همچنین برای مدارهای مختلف به خوبی عمل نماید. بنابراین، هنگام تعیین برازندگی یک برنامه‌ی کامپیوتری، شرایط و وضعیت‌های مختلف بسیاری باید مورد توجه قرار بگیرند. برای یک برنامه‌ی کامپیوتری خاص، هر مجموعه از شرایط ورودی و شرایط عامل، "زیر برازندگی" خاص خود را نتیجه خواهد داد. چگونه باید این زیربرازندگی‌ها را برای دستیابی به یک اندازه‌گیریِ برازندگیِ واحد ترکیب نمود؟ آیا باید از میانگین‌گیری استفاده نماییم؟ یا باید برای ماکزیمم ساختن عملکرد بدترین حالت تلاش نماییم؟ این سؤال‌ها به‌صورت طبیعی ما را به سمت بهینه‌سازی چندهدفه هدایت خواهد کرد (فصل ۲۰). با این حال استفاده از بهینه‌سازی چندهدفه در GP ضرورتی ندارد.

۷–۲–۲ معیارهای پایان‌دهی

معیارهای پایان‌دهی در شکل ۷–۵ چه هستند؟ این سؤال باید در مورد همه‌ی الگوریتم‌های تکاملی پاسخ داده شود (بخش ۸–۲ را ببینید). اما این سؤال در مورد GP از اهمیت ویژه‌ای برخوردار است. این به آن دلیل است که در GP، اندازه‌گیری برازندگی نسبت به سایر الگوریتم‌های تکاملی نیاز به تلاش محاسباتی بیشتری دارد. انتخاب معیارهای پایان‌دهی تأثیر مستقیمی در موفقیت و یا عدم موفقیت GP دارد. همانند دیگر الگوریتم‌های تکاملی، معیارهای پایان‌دهی GP می‌تواند شامل فاکتورهایی همچون تعداد دوره‌ها، تعداد ارزیابی‌ها برازندگی، مدت زمان اجرا، بهترین مقدار برازندگی، تغییراتِ بهترین مقدار برازندگی در طول چندین نسل و یا مقدار انحراف استانداردِ برازندگی در میان کل جمعیت باشد.

۷–۲–۳ مجموعه‌ی ترمینال

مجموعه‌ی ترمینال برای برنامه‌های کامپیوتری در حال رشد چیست؟ این مجموعه، نمادهایی را که می‌توانند در برگ‌های درخت نحو ظاهر شوند را توصیف می‌کند. مجموعه‌ی ترمینال، مجموعه‌ی تمامی ورودی‌های ممکنِ یک برنامه‌ی کامپیوتری در حال رشد است. این مجموعه شامل متغیرهای ورودی یک برنامه‌ی کامپیوتری می‌باشد. همچنین، این مجموعه شامل ثابت‌هایی است که از نظر ما دارای اهمیت هستند. این ثابت‌ها خود می‌توانند شامل اعداد صحیح مانند ۰ و ۱ و همچنین شامل اعدادی باشند که برای یک مسئله‌ی بهینه‌سازی خاصِ بهینه‌سازی حائز اهمیت هستند. درخت‌های نحو نشان داده شده در شکل ۷–۳ دارای سه ترمینال هستند: x، y و z. برخی ثابت‌ها را می‌توان به‌صورت غیرمستقیم به دست آورد. برای مثال،

$x - x = 0$ و $x/x = 1$ بنابراین نیازی به ثابت‌های ۰ و ۱ نداریم. با این حال، بیشترِ پیاده‌سازی‌های GP باید دارای ثابت‌هایی در مجموعه‌ی ترمینال‌شان باشند.

علاوه بر این ما می‌توانیم از اعداد تصادفی در مجموعه‌ی ترمینال استفاده نماییم هرچند که نمی‌خواهیم بعد از ایجاد شدن عدد تصادفی در آن تغییری حاصل شود. این نوع از اعداد تصادفی، ثابت‌های تصادفی بی‌دوام خوانده می‌شوند [کوزا، ۱۹۹۲، فصل ۲۶]. ثابت‌های تصادفی بی‌دوام با اختصاص دادن یک مقدار R در مجموعه‌ی ترمینال ایجاد می‌شود. اگر در هنگام مقداردهی اولیه‌ی جمعیت، R به عنوان یک ترمینال انتخاب شود، ما یک عدد تصادفی r_1 را بین دو مقدار حدی ایجاد کرده و آن را به ذرات GP الحاق می‌کنیم. از اینجا به بعد دیگر مقدار r_1 تغییر نخواهد کرد. با این حال، اگر R برای مقداردهی اولیه‌ی یک ذره‌ی دیگر و یا برای جهش انتخاب شود، ما باید یک ثابت اتفاقی دیگر مانند r_2 تولید نماییم. تعیین حدهای بالا و پایین ثابت‌های تصادفی بی‌دوام، خود یکی دیگر از جنبه‌های طراحی GP است.

تعیین مجموعه‌ی ترمینال برای کاربردهای GP، یک عمل نیازمند تعادل است. اگر از یک مجموعه‌ی ترمینالِ کوچک استفاده نماییم، آنگاه GP قادر به حل مؤثر مسئله‌ی ما نخواهد بود. از سوی دیگر، اگر از یک مجموعه‌ی ترمینال خیلی بزرگ استفاده نماییم، آنگاه GP نخواهد توانست در زمانی معقول به جوابی مناسب دست پیدا کند. کوزا این موضوع را مورد مطالعه قرار داده است [کوزا، ۱۹۹۲، فصل ۲۴]. وی برای این کار از تابع ساده‌ی $x^3 + x^2 + x$ بر پایه‌ی ۲۰ مورد امتحانی استفاده کرده است. برای این مسئله، تنها ترمینال مورد نیاز GP، x است. هنگامی که از مجموعه‌ی ترمینال با کمترین اندازه یعنی $\{x\}$ استفاده می‌شود، GP در ۹۹٫۸٪ مواقع جواب صحیح را در کمتر از ۵۰ نسل پیدا می‌کند. جدول ۷-۱ چگونگی کاهش احتمال موفقیت با اضافه کردن اعضای اضافه (اعداد اعشاری اتفاقی) به مجموعه‌ی ترمینال را نشان می‌دهد. در مورد این مسئله‌ی ساده، احتمال موفقیت با افزایش تعداد متغیرهای اضافه در مجموعه‌ی ترمینال، به‌صورت خطی کاهش می‌یابد. البته خوشبختانه حتی اگر ۳۲ متغیر از ۳۳ متغیر در مجموعه‌ی ترمینال اضافه باشند، باز هم GP در ۳۵٪ موارد قادر به حل نمودن مسئله است.

۷-۲-۴ مجموعه‌ی تابع

مجموعه‌ی تابع برای برنامه‌های کامپیوتری در حال رشد چیست؟ این مجموعه توابعی را توصیف می‌کند که می‌توانند در گره‌های غیرترمینال از درخت نحو ظاهر شوند:

- اپراتورهای استاندارد ریاضی را می‌توان در این مجموعه قرار داد (برای مثال جمع، تفریق، ضرب تقسیم و قدرمطلق).

- توابع مختص مسئله که از نظر ما برای مسئله‌ی بهینه‌سازی مهم هستند را می‌توان در این مجموعه قرار داد (مانند توابع نمایی، توابع لگاریتمی، توابع مثلثاتی، مشتق‌گیرها و انتگرال‌گیرها).
- توابع مقایسه‌ای را می‌توان در این مجموعه قرار داد (مانند بزرگتر، کوچکتر، برابر).
- توابع منطقی نیز اگر برای حل مسئله‌ی بهینه‌سازی مورد نظر قابل استفاده باشند می‌توانند در این مجموعه قرار بگیرند (برای مثال، and، nand، or، nor، not).
- توابع تخصیص متغیر را نیز می‌توان در این مجموعه قرار داد.
- عبارت‌های حلقه می‌توانند در این مجموعه قرار بگیرند (مانند حلقه‌های $while$، for).
- اگر مجموعه‌ای از توابع از پیش تعریف شده برای مسئله‌مان در نظر گرفته باشیم، آنگاه فراخوانی‌های زیرروال[1] را می‌توان در مجموعه‌ی تابع قرار داد.

جدول ۷-۱ احتمال موفقیت *GP* برای پیدا کردن برنامه‌ی $x^3 + x^2 + x$ بعد از ۵۰ نسل. اندازه‌ی جمعیت برابر ۱۰۰۰ است. این اطلاعات از [کوزا، ۱۹۹۲، فصل ٢٤] برگرفته شده است.

تعداد متغیرهای اضافی	احتمال موفقیت (والد)
۰	۹۹٫۸
۱	۹٦٫٦
٤	۸٤٫۰
۸	٦۷٫۰
۱٦	۵۷٫۰
۳۲	۳۵٫۰

درخت نحو در شکل ۷-۳ دارای ۵ تابع است: جمع، تفریق، ضرب، تقسیم و قدر مطلق. ما باید به تعادلی درست در تعاریفمان از مجموعه‌ی تابع و مجموعه‌ی ترمینال برسیم. این مجموعه‌ها باید به قدری بزرگ باشند که *GP* بتواند راه‌حلی مناسب برای مسئله‌مان پیدا کند. اما اگر این مجموعه‌ها زیادی بزرگ باشند آنگاه فضای جستجو بسیار بزرگ بوده و *GP* برای پیدا کردن یک راه‌حل خوب به مشقت خواهد افتاد.

برخی توابع برای استفاده در *GP* باید مورد اصلاح قرار گیرند چرا که رشد درخت نحو ممکن است به توابعی منجر شوند که آرگومان آن‌ها غیرقابل قبول باشد. برای مثال، *GP* نمی‌تواند عبارتِ $(/x0)$ را رشد دهد چرا که تقسیم بر صفر در *Lisp* باعث بروز خطا شده و عمل *GP* قطع خواهد شد. بنابراین برای جلوگیری

[1] Subroutine Calls

از وقوع تقسیم بر صفر و همچنین سرریزی ناشی از تقسیم بر یک عدد بسیار کوچک، به جای استفاده از اپراتور استاندارد تقسیم از اپراتور DIV استفاده می‌نماییم:

$$\begin{array}{l}(defun\ DIV\ (x\ y)): \text{ یک تابع تقسیم حفاظت شده تعریف کن} \\ (if(< abs\ y)\ \epsilon)(return-from\ DIV\ 1): \text{ اگر مقدار مقسوم علیه خیلی کوچک بود ۱ را برگردان} \\ return-from\ DIV\ (/x\ y): \text{ در غیر این صورت } x/y \text{ را برگردان}\end{array} \quad (۷-۴)$$

که در آن ε یک ثابت مثبت بسیار کوچک مانند 10^{-20} است. اگر مقسوم‌علیه مقداری بسیار کوچک باشد، اپراتور DIV عدد ۱ را باز خواهد گرداند. بدین ترتیب برای سایر توابع نیز باید تعریفی درست ارائه کرد تا مطمین شویم که این توابع می‌توانند تمامی مقادیر ممکن ورودی را بپذیرند. از جمله‌ی این توابع می‌توان به توابع لگاریتمی، توابع معکوس مثلثاتی و غیره اشاره نمود.

۷-۲-۵ مقداردهی اولیه

جمعیت آغازین یک برنامه‌ی کامپیوتری چگونه باید ایجاد شود؟ برای این کار دو گزینه‌ی اساسی به نام‌های روش کامل[1] و روش رشد[2] در اختیار داریم. همچنین می‌توان با ترکیب این دو روش به گزینه‌ی سومی نیز دست یافت که به روش نگاهداشته‌ی نصف- نصف مشهور است [کوزا، ۱۹۹۲].

روش کامل، برنامه‌های کامپیوتری را به نحوی ایجاد می‌کند که تعداد گره‌ها از گره‌ی ترمینال تا بالاترین گره برابر D_c، که یک ثابت تعریف شده توسط کاربر است، باشد. D_c را عمق درخت نحو می‌نامند. برای مثال، در شکل ۷-۳ والد ۱ عمقی برابر ۳ و والد ۲ عمقی برابر ٤ دارد. والد ۱ در شکل ۷-۳ یک درخت نحو کامل است چرا که در آن بین هر گره‌ی ترمینال و بالاترین گره‌ی درخت، سه گره وجود دارد. حال آنکه والد ۲ یک درخت نحو کامل نیست چرا که در آن بعضی از شاخه‌های برنامه دارای عمقی برابر چهار و برخی دیگر دارای عمقی برابر سه هستند.

ما می‌توانیم از عمل بازگشت برای تولید درخت‌های نحو اتفاقی استفاده نماییم. برای مثال، اگر بخواهیم یک درخت نحو با ساختاری مانند ساختار والد ۲ در شکل ۷-۳ ایجاد کنیم، ابتدا گره‌ی تفریق در قسمت بالایی درخت را، که نیازمند دو آرگومان است، ایجاد می‌کنیم. برای آرگومان اول، یک گره‌ی ضرب، که آن نیز نیازمند دو آرگومان است ایجاد می‌کنیم. این فرایند برای هر گره و هر آرگومان آنقدر ادامه می‌یابد تا به

[1] Full Method
[2] Grow Method

سطح و عمق مورد نظر برسیم. هنگامی که عمق مطلوب حاصل شد، یک گره‌ی ترمینال اتفاقی ایجاد می‌کنیم تا آن شاخه از درخت نحو کامل گردد. شکل ۷–٦ ایده‌ی الگوریتم بازگشتی برای تولید برنامه‌های کامپیوتری اتفاقی را به تصویر می‌کشد. ما می‌توانیم یک درخت نحو اتفاقی را با بازخوانی روتین $GrowProgramFull(Dc, 1)$، که در آن D_c بیانگر عمق مطلوب درخت نحو است، تولید کنیم. $GrowProgramFull$ هرگاه نیاز به اضافه کردن یک لایه‌ی جدید به درخت نحو در حال رشد باشد، خود را فرا می‌خواند.

روش رشد به‌گونه‌ای برنامه‌های کامپیوتری را ایجاد می‌کند که در آن‌ها تعداد گره‌ها از هر گره‌ی ترمینال تا بالاترین گره برابر یا کوچکتر از D_c باشد. اگر در شکل ۷–۳ والدین با استفاده از مقداردهی اولیه‌ی اتفاقی ایجاد شده باشند، آنگاه می‌توان گفت که والد اول یا با روش کامل و یا با روش رشد ایجاد شده است، اما والد دوم قطعا با روش رشد ایجاد شده چرا که یک درخت نحو کامل نیست. روش رشد را می‌توان به همان ترتیبِ روش کامل پیاده‌سازی نمود تنها باید توجه داشته باشیم که هنگامی که یک گره‌ی اتفاقی در عمقِ کمتر از D_c ایجاد می‌کنیم، هم ممکن است یک گره‌ی ترمینال ایجاد شود و هم امکان دارد یک تابع ایجاد شود. اگر یک گره‌ی تابع ایجاد شود، درخت نحو می‌تواند به رشد خود ادامه دهد. اما اگر گره‌ی ترمینال ایجاد شود، رشد درخت در آن شاخه متوقف خواهد شد. این در حالی است که در روش کامل، گره‌ی ترمینال هنگامی ایجاد می‌شود که عمق مطلوب D_c حاصل شده باشد. شکل ۷–۷ مفهوم الگوریتم بازگشتی برای تولید برنامه‌های کامپیوتری اتفاقی به روش رشد را نشان می‌دهد.

در روش نگاهداشته‌ی نصف– نصف، نیمی از جمعیت آغازین با استفاده از روش کامل و نیمی دیگر با استفاده از روش رشد ایجاد می‌شود. همچنین، این روش برای هر مقداری از عمق که بین ۲ و D_c قرار داشته باشد، تعداد مساوی درخت نحو ایجاد می‌کند. این مقدار از عمق، بیشترین مقدار مجازی است که توسط کاربر تعیین می‌شود. شکل ۷–۸، مفهوم مقداردهی اولیه با روش نگاهداشته‌ی نصف– نصف را نشان می‌دهد.

$function(SyntaxTree) = GrowProgramFull(Depth, NumArgs)$
$SyntaxTree \leftarrow \emptyset$
برای $i = 1$ تا $NumArgs$
اگر $Depth = 1$
ترمینال اتفاقی $\leftarrow SyntaxTree$
در غیر این صورت
یک تابع انتخاب شده اتفاقی $\leftarrow NewFunction$
تعداد آرگومان‌های مورد نیاز $NewFunction \leftarrow NewNumArgs$
$SyntaxTree \leftarrow (NewFunction + GrowProgramFull(Depth - 1, NewNumArg)$
پایان
i بعدی

شکل ۷-۶ نمایش مفهومی از الگوریتم بازگشتی برای رشد درختان نحو اتفاقی به فرم عبارت s در روش کامل. این روند ابتدا توسط دستور $GrowProgramFull(Dc, 1)$، که در آن D_c عمق مطلوب درخت نحو اتفاقی است، فراخوانی می‌شود. اپراتور جمع نشان‌دهنده‌ی الحاق رشته است. توجه داشته باشید که این الگوریتم مفهومی است و شامل تمام جزئیات لازم برای برای تولیدِ معتبر درخت نحو نمی‌باشد.

$function(SyntaxTree) = GrowProgramFull(Depth, NumArgs)$
$SyntaxTree \leftarrow \emptyset$
برای $i = 1$ تا $NumArgs$
اگر $Depth = 1$
ترمینال اتفاقی $\leftarrow SyntaxTree$
در غیر این صورت
یک ترمینال یا تابع انتخاب شده اتفاقی $\leftarrow NewNode$
اگر $NewNode$ یک ترمینال است
$SyntaxTree \leftarrow (SyntaxTree + NewNode)$
در غیر این صورت
تعداد آرگومان‌های مورد نیاز $NewNode \leftarrow NewNumArgs$
$SyntaxTree \leftarrow (NewNode + GrowProgramFull(Depth - 1, NewNumArg)$
پایان
پایان
i بعدی

شکل ۷-۷ نمایش مفهومی از الگوریتم بازگشتی برای رشد درختان نحو اتفاقی به فرم عبارت s در روش رشد. این روند ابتدا توسط دستور $GrowProgramFull(Dc, 1)$، که در آن D_c عمق مطلوب درخت نحو اتفاقی است، فراخوانی می‌شود. اپراتور جمع نشان‌دهنده‌ی الحاق رشته است، اما این الگوریتم تمام جزئیات لازم برای پیاده‌سازی را شامل نمی‌شود.

بیشترین عمق درخت نحو $= D_c$
اندازه جمعیت $= N$
برای $i = 1$ تا N
$Depth \leftarrow U[2, D_c]$
$r \leftarrow U[0,1]$
اگر $r < 0.5$
$SyntaxTree(i) \leftarrow GrowProgramGrow(Depth, 1)$
در غیر این صورت
$SyntaxTree \leftarrow GrowProgramFull(Depth, 1)$
پایان
i بعدی

شکل ۷–۸ الگوریتمی برای ایجاد جمعیت اولیه‌ی GP به روش نگاهداشته‌ی نصف– نصف. $U[2, D_c]$ یک عدد صحیح اتفاقی با توزیع یکنواخت بر روی $[2, D_c]$ بوده و $U[0,1]$ یک عدد حقیقی اتفاقی با توزیع یکنواخت بر روی $[0,1]$ می‌باشد. این الگوریتم روتین‌های شکل ۷–٦ و ۷–۷ را فرامی‌خواند.

کوزا سه روش بالا را بر روی چند مسئله‌ی ساده‌ی GP آزمایش کرد [کوزا، ۱۹۹۲، فصل ۲۵]. وی تفاوت‌هایی در نرخ موفقیت GP، بسته به روش مقداردهی اولیه، پیدا نمود. این نتایج در جدول ۷–۲ نشان داده شده‌اند. این جدول نشان می‌دهد که در کل روش نگاهداشته‌ی نصف– نصف بهتر از دو روش دیگر می‌باشد.

جدول ۷–۲ نرخ موفقیت GP برای مسائل و روش‌های مقداردهی اولیه‌ی گوناگون. این اطلاعات از [کوزا، ۱۹۹۲، فصل ۲۵] استخراج شده است.

مسئله	Full	Grow	Ramped Half-and-Half
رگرسیون نمادی	٪۳	٪۱۷	٪۲۳
منطق بولی	٪٤۲	٪۵۳	٪٦٦
مورچه مصنوعی	٪۱٤	٪۵۰	٪٤٦
معادله خطی	٪٦	٪۳۷	٪۵۳

برای پایان دادن به بحثمان در مورد مقداردهی اولیه، به این توجه می‌کنیم که گاهاً بذرافشانی کردن جمعیت اولیه‌ی یک الگوریتم تکاملی با برخی ذرات خوب شناخته شده مفید است. این ذرات خوب ممکن است توسط خود کاربر ایجاد شده باشند و یا ممکن است از یک الگوریتم بهینه‌سازی دیگر و یا هر منبع

دیگری گرفته شده باشند. با این حال این نوع بذرافشانی ضرورتاً عملکرد الگوریتم تکاملی را بهبود نمی‌بخشد. اگر تنها چند ذره‌ی خوب در جمعیت آغازین وجود داشته باشند و سایر ذرات، ذرات ضعیفِ اتفاقی باشند، همان چند ذره‌ی خوب به زودی فرایند انتخاب را به تسخیر خود درآورده و ذرات ضعیف‌تر را از جمعیت بیرون می‌رانند. این موضوع ممکن است باعث بروز یک بن بست تکاملی و همگرایی نارس شود. این پدیده به نام "بقای متوسط"[1] نیز شناخته می‌شود [کوزا، ۱۹۹۲، صفحه‌ی ۱۰٤]. با این حال، شانس بروز این پدیده‌ی بد به نوع انتخاب مورد استفاده‌ی ما بستگی دارد (بخش ۸-۷ را ببینید). اگر از انتخاب چرخ رولت استفاده نماییم، فشار انتخاب بالا بوده و تعداد کمی از ذرات برازنده سریعاً بر جمع غالب می‌شوند. اگر از انتخاب مسابقه‌ای استفاده نمائیم، فشار انتخاب بسیار کمتر بوده و به همین علت احتمال غلبه‌ی اقلیتِ برازنده بر جمعیت نیز کاهش می‌یابد.

۷-۲-۶ پارامترهای برنامه‌نویسی ژنتیک

چه پارامترهایی اجرای GP را کنترل می‌کنند؟ این پارامترها علاوه بر آن دسته از پارامترهای کنترلی که برای کنترل سایر الگوریتم‌های تکاملی به کار می‌روند، برخی پارامترهای مختص GP را نیز شامل می‌شوند:

۱. ما باید روش انتخاب مورد استفاده برای انتخاب والدین جهت انجام عمل برش را تعیین کنیم. می‌توان از انتخاب متناسب با برازندگی، انتخاب مسابقه‌ای یا یک روش دیگر استفاده نمود. در حقیقت می‌توان از هر یک از روش‌های ارائه شده در بخش ۸-۷ استفاده نمود.

در این برهه خوب است به این نکته اشاره کنیم که ما می‌توانستیم با هوشمندی بیشتر، به جای آنکه خیلی ساده نقاط برش اتفاقی را انتخاب کنیم، برشِ درخت- محور را پیاده‌سازی نماییم. ممکن است زیردرخت‌هایی وجود داشته باشند که بهتر از سایر زیردرخت‌ها بوده و به همین علت نخواهیم این زیردرخت‌ها را بشکنیم. می‌توان برازندگی زیردرخت‌ها را با استفاده از همبستگی میان نقاط برش و برازندگی برنامه‌های فرزند و استفاده از این همبستگی‌ها برای گرایش‌دهی به انتخاب نقاط برش در آینده، محاسبه نمود [ایبا[2] و دِ گریس، ۱۹۹٦].

۲. ما باید اندازه‌ی جمعیت را تعیین کنیم. معمولاً از آنجایی که درجه‌ی اختیارات زیادی در برنامه‌های کامپیوتری وجود دارد، GP اندازه‌ی جمعیت بزرگتری نسبت به سایر الگوریتم‌های تکاملی دارد. به‌طور معمول حداقل اندازه‌ی جمعیت GP برابر ۵۰۰ بوده و گاه به مرتبه‌ی هزار نیز می‌رسد.

[1] Survival of the Mediocre
[2] Iba

۳. ما باید روش جهش را نیز مشخص کنیم. در طول سال‌ها از روش‌های جهش مختلفی برای GP استفاده شده است که برخی از آن‌ها در زیر ارائه شده‌اند:

a. می‌توان یک گره را به‌صورت اتفاقی انتخاب نمود، سپس هر آنچه در زیر آن گره قرار دارد را با یک زیردرخت نحو که به‌صورت اتفاقی ایجاد شده است جایگزین نمود. این روش جهش زیردرخت نام دارد [کوزا، ۱۹۹۲، صفحه‌ی ۱۰۶]. این کار مانند آن است که یک برنامه را با یک برنامه‌ی اتفاقی دیگر برش دهیم و به همین علت برشِ مرغِ بدون سر نیز خوانده می‌شود [آنجلین[1]، ۱۹۹۷].

b. جهش انبساطی، یک ترمینال را با یک زیردرخت جایگزین می‌کند. این روش معادل و روش جهش زیردرخت معادل هستند، اگر در روش زیردرخت گره‌ی مورد نظر یک ترمینال باشد.

c. می‌توان یک گره یا ترمینال را با یک گره یا ترمینال جدید که به‌صورت اتفاقی ایجاد شده است جایگزین نمود. این روش جهش نقطه‌ای یا جهشِ جایگزینی گره نام دارد و نیازمند آن است که آریتی[2] گره‌ی جایگزین شده با آریتی گره‌ی جایگزین شونده برابر باشد[3]. برای مثال، می‌توان یک اپراتور جمع را با یک اپراتور ضرب و یا یک اپراتور قدرمطلق را با یک اپراتور سینوس جایگزین نمود.

d. جهش جرثقیلی یک برنامه‌ی جدید را با انتخاب اتفاقی زیردرخت‌های برنامه‌ی والد ایجاد می‌کند [کینیر[4]، ۱۹۹٤].

e. جهش انقباضی، یک زیردرخت نحو را که به‌صورت اتفاقی انتخاب شده است را با یک ترمینال اتفاقی جایگزین می‌نماید [آنجلین، ۱۹۹٦a]؛ این روش را جهش فروپاشی زیردرخت نیز می‌نامند. جهش جرثقیلی و جهش انقباضی در اصل برای کاهش تعداد خط‌های کد (تورم کد) ارائه شدند (بخش ۷,٤ را ببینید).

f. جهش جایگشتی، آرگومان‌های یک تابعِ اتفاقی انتخاب شده را به‌صورت اتفاقی جایگردانی می‌کند [کوزا، ۱۹۹۲]. برای مثال، می‌توان آرگومان‌های x و y از یک تابع تقسیم را جابه‌جا نمود. روشن است که این نوع جهش تأثیری بر توابع جابه‌جایی‌پذیر ندارد.

---

[1] Angeline

[2] Arity

[3] آریتی یک تابع برابر تعداد آرگومان‌های آن می‌باشد. برای مثال، یک ثابت، دارای آریتی برابر ۰ است، تابع قدرمطلق آریتی برابر ۱ دارد و یک تابع جمع، می‌تواند آریتی برابر ۲ و یا بیشتر داشته باشد.

[4] Kinnear

g. در نهایت می‌توان ثابت‌های یک برنامه را به‌صورت اتفاقی دچار جهش نمود [شونوئر[1] و همکاران، ۱۹۹۶].

جهش‌ها اغلب به‌گونه‌ای پیاده‌سازی می‌شوند که برنامه‌های جهش‌یافته برازنده‌تر از برنامه‌های اصلی باشند. این ایده را می‌توان در مورد سایر الگوریتم‌های تکاملی نیز به کار برد.

٤. ما باید احتمال جهش یا p_m را تعیین کنیم. این کار شبیه همانی است که برای سایر الگوریتم‌های تکاملی انجام می‌دهیم. جهش در یک GP با تعداد N ذره اغلب با روشی مشابه روش زیر انجام می‌پذیرد:

برای هر برنامه‌ی کامپیوتری نامزد x_i که در آن $i \in [1, N]$
عددی اتفاقی مانند r با توزیع یکنواخت بر روی $[0,1]$ تولید کن
اگر $r < p_m$
گره‌ی k را در برنامه‌ی کامپیوتری x_i به‌صورت اتفاقی برگزین
زیردرخت شروع‌شونده از گره‌ی k را با یک زیردرخت تولید شده به‌صورت اتفاقی جایگزین کن
پایان
برنامه‌ی کامپیوتری بعدی

اندازه‌ی جمعیت بزرگ به کار رفته در GP همراه با تعداد زیاد گره‌های محتمل برای وقوع برش، معمولاً بدین معنی هستند که نتایج خوب در GP بستگی چندانی به جهش ندارند [کوزا، ۱۹۹۲، فصل ۲۵]. معمولاً می‌توان با $p_m = 0$ نتایج خوبی به دست آورد. با این حال، امکان دارد در مواردی که یک تابع یا ترمینال مهم از جمعیت از دست رفته است، وقوع جهش مطلوب باشد. اگر چنین اتفاقی بیافتد، جهش تنها راه برگشت آن تابع یا ترمینال به جمعیت خواهد بود.

۵. احتمال برش p_c نیز از جمله‌ی مواردی است که باید تعیین شود. این کار شبیه آن چیزی است که برای GAها انجام می‌دهیم. پس از انتخاب دو والد در شکل ۷–۵، یا می‌توان از فرایند برش برای ترکیب آن‌ها استفاده نمود و یا می‌توان آن‌ها را برای نسل بعد نشر (کلون) داد. در این صورت، عبارتِ:

[1] Schoenauer

p_1 و p_2 را برای تولید فرزندان c_1 و c_2 مزدوج کن

در شکل ۷–۵ با عبارتی مانند عبارت زیر جایگزین خواهد شد:

عددی اتفاقی مانند r با توزیع یکنواخت بر روی $[0,1]$ تولید کن
اگر $r < p_c$
p_1 و p_2 را برای تولید فرزندان c_1 و c_2 مزدوج کن
در غیر این صورت
$c_1 \leftarrow p_1$
$c_2 \leftarrow p_2$
پایان

تجارب به دست آمده حاکی از آنند که برش جز مهمی از GP بوده و باید با احتمالی بیش از ۰٫۹ به کار رود [کوزا، ۱۹۹۲، فصل ۲۵].

۶. ما باید تصمیم بگیریم که از نخبه‌گرایی استفاده کنیم یا نه. مانند سایر الگوریتم‌های تکاملی، می‌توانیم m برنامه‌ی کامپیوتری برتر را از یک نسل به نسل دیگر حفظ کنیم تا مطمئن شویم که در نسل بعد از دست نخواهند رفت (بخش ۸–٤ را ببینید). پارامتر m پارامتر نخبه‌گرایی نام دارد. نخبه‌گرایی را به چند روش مختلف می‌توان پیاده‌سازی نمود. برای مثال، می‌توان بهترین m ذره را در آخر هر نسل آرشیو کرد، سپس فرزندان را مانند همیشه ایجاد کرد و در انتها بدترین m فرزند را با نخبه‌های نسل قبل جایگزین نمود. یا می‌توان m نخبه‌ی هر نسل را به عنوان m فرزند اول نسل بعد در نظر گرفت و سپس اقدام به تولید تنها $(N - m)$ فرزند نمود (که در آن N اندازه‌ی جمعیت است).

۷. ما باید D_i یا بیشینه‌ی اندازه‌ی برنامه در جمعیت آغازین را مشخص نماییم. اندازه‌ی یک برنامه را می‌توان با عمقِ آن، که برابر تعداد گره‌های میان بالاترین گره و پایین‌ترین گره است، نشان داد. برای مثال، والد ۱ در شکل ۷–۳ عمقی برابر سه دارد در حالی که والد ۲ دارای عمقی برابر چهار می‌باشد.

۸. ما همچنین باید D_c، یا بیشینه‌ی عمقِ برنامه‌ی فرزند را نیز مشخص کنیم. در طول عملیات GP و با گذشت نسل‌ها، برنامه‌های فرزند بزرگ و بزرگتر می‌شوند. اگر یک عمق بیشینه برای آن‌ها در نظر گرفته نشود، برنامه‌های فرزند به طور غیرمعقولی طولانی شده و این موضوع باعث به هدر رفتن فضا و زمان اجرا می‌شود. این پدیده تورمِ GP نام دارد (بخش ۷–٤ را ببینید). بیشینه عمق D_c را به چندین روش می‌توان تعیین نمود. یک روش آن است که اگر عمقِ فرزندی از D_c تجاوز کرد، آن را با والدش

جایگزین نماییم. یک راه دیگر نیز آن است که درخت نحو والد را بیازماییم و نقطه‌ی برش را به‌گونه‌ای انتخاب نماییم که درخت نحو فرزند عمقی برابر یا کمتر از D_c پیدا کند.

۹. ما باید تصمیم بگیریم که آیا می‌خواهیم اجازه دهیم که در طول فرایند برش یک ترمینال در درخت نحو با یک زیردرخت جایگزین شود یا خیر. شکل ۷-٤ نشان می‌دهد که ترمینال z در والد ۱ به‌عنوان نقطه‌ی برش انتخاب شده و با یک زیردرخت جایگزین شده است. ما از نماد p_i برای نشان دادن احتمال برش در یک گره‌ی داخلی استفاده می‌کنیم. هنگام انتخاب نقطه‌ی برش، یک عدد اتفاقی مانند r با توزیع یکنواخت بر روی $[0,1]$ تولید می‌نماییم. اگر r کوچکتر از p_i باشد، آنگاه یک ترمینال را به‌عنوان نقطه‌ی برش برمی‌گزینیم. اما اگر r بزرگتر از p_i باشد، آنگاه یک عبارت s را به‌عنوان نقطه‌ی برش انتخاب خواهیم کرد، بدین معنی که زیردرختی را انتخاب خواهیم نمود که با پرانتزهای چپ و راست متناظر احاطه شده باشد.

۱۰. ما باید تصمیم بگیریم که آیا باید نگران ذرات تکراری در جمعیت باشیم یا خیر. ذرات تکراری باعث به هدر رفتن منابع کامپیوتر خواهند شد. در الگوریتم‌های تکاملی که دارای فضای جستجوی کوچک بوده و از اندازه‌ی جمعیت کمی برخوردارند، اغلب ذرات تکراری به وجود می‌آیند و به همین علت سروکله زدن با این ذرات را می‌توان جنبه‌ی مهمی از الگوریتم‌های تکاملی دانست (بخش ۸-٦-۱ را ببینید). با این حال، در GP فضای جستجو آنقدر بزرگ است که ذرات تکراری به ندرت به وجود می‌آیند. به همین دلیل معمولاً در GP نیازی به نگرانی در مورد این مسئله نیست.

۷-۳ برنامه‌نویسی ژنتیک برای کمترین زمانِ کنترل

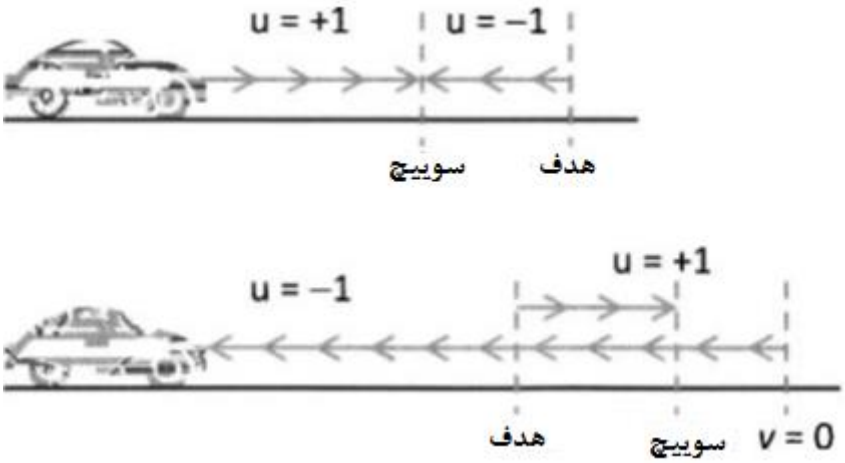
در این بخش، که از [کوزا، ۱۹۹۲، بخش ۷-۱] الهام گرفته است، کاربرد GP برای کمترین زمان کنترل در یک سیستم نیوتونی مرتبه دو را نشان می‌دهیم. یک سیستم نیوتونی مرتبه‌ی دوم، یک سیستم ساده‌ی مکان- سرعت- شتاب است که معادلات زیر را برآورده می‌سازد

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= u \end{aligned} \qquad (۵-۷)$$

که در آن x مکان، v سرعت و u شتاب است. با توجه به معادلات بالا، مشتق مکان، سرعت و مشتق سرعت، شتاب است. ما حرکت را فقط در یک بعد در نظر می‌گیریم. در اینجا ما می‌خواهیم $u(t)$ را به‌گونه‌ای پیدا کنیم که سیستم را از یک نقطه‌ی آغازین در $x(0)$ و با سرعت اولیه‌ی $v(0)$ به مکانِ $x(t_f) = 0$ با سرعت $v(t_f) = 0$ برساند به طوری که t_f مینیمم مقدار ممکن باشد. با استفاده از شهود می‌توان به طور تقریبی دریافت که چگونه باید این کار را انجام داد: ابتدا باید با بیشترین سرعت ممکن در جهت مورد نظر شتاب

گرفت تا به یک نقطه‌ی مشخص رسید. سپس و از آن نقطه به بعد باید شتاب را به سریعترین حالت ممکن و در خلاف جهت اولیه افزایش داد تا در نهایت به $x(0) = v(0) = 0$ برسیم.[۱]

برای سادگی، بدون آنکه کلیت مطلب از بین برود، فرض می‌کنیم بیشترین اندازه‌ی شتاب ممکن برابر ۱ است و می‌توان در هر یک از دو جهت شتاب گرفت. مسئله‌ی کمترین زمان کنترل در بالای شکل ۷-۹ نشان داده شده است. در این شکل ابتدا در جهت مثبت (جهت راست) شتاب می‌گیریم و سپس در یک نقطه‌ی استراتژیک که با کلمه‌ی "سوئیچ" نشان داده شده است، جهت شتاب خود را عوض کرده و در جهت منفی (جهت چپ) شتاب می‌گیریم تا به نقطه‌ی نهایی برسیم. توجه داشته باشید که جهتِ سرعت ماشین در تمام طول حرکت به سمت راست است. اگر زمان‌بندی‌مان درست باشد، با سرعت صفر به مقصد خواهیم رسید.



شکل ۷-۹ نمایش کمترین زمان کنترل. در شکل بالایی، ما به سمت راست شتاب گرفته تا به نقطه‌ی سوئیچ برسیم و سپس به سمت چپ شتاب می‌گیریم تا با سرعت صفر به مقصد برسیم. در شکل پایینی، سرعت اولیه آنقدر زیاد است که به ناچار از مقصد رد خواهیم شد. در چنین حالتی، ما به سمت چپ شتاب می‌گیریم، از مقصد رد می‌شویم، جهت شتاب را در نقطه‌ی سوئیچ به سمت راست عوض کرده و با سرعت صفر به مقصد می‌رسیم.

ممکن است سرعت اولیه چنان زیاد باشد که چاره‌ای به جز رد کردن مقصد نداشته باشیم. در چنین حالتی باید به ناچار مقصد را رد کرده و سپس به سمت مقصد بازگردیم تا با سرعت صفر به آن برسیم. این وضعیت

[۱] در حقیقت این همان کاری است که این روزها راننده‌های جوان بر سر چراغ قرمزها انجام می‌دهند؛ هنگامی که چراغ سبز است تا جایی که می‌توانند شتاب می‌گیرند. وقتی چراغ را رد کردند، در یک نقطه‌ی مشخص قبل از چراغ بعدی پای خود را بر روی ترمز گذاشته و سرعت خود را کم می‌کنند. اگر زمان‌بندی راننده دقیق باشد، ماشین درست قبل از چراغ بعدی می‌ایستد و بدین ترتیب زمان جابه‌جایی میان چراغ قرمزها مینیمم می‌شود.

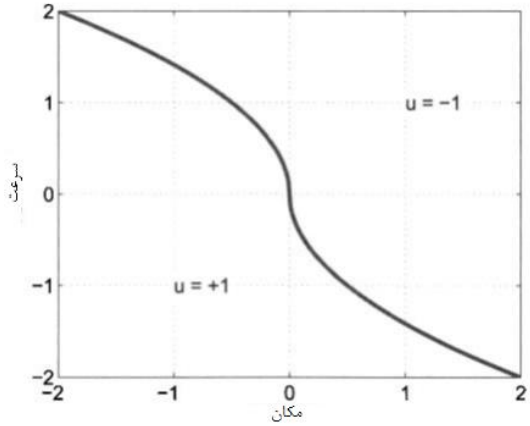
در قسمت پایینی شکل ۷-۹ نشان داده شده است. راه‌حل کمترین زمان برای این حالت عبارتست از شتاب گرفتن تا بیشترین حد ممکن در جهت منفی (جهت چپ) و رد کردن مقصد. در نهایت ماشین در نقطه‌ای ورای مقصد خواهد ایستاد و سپس شروع به حرکت به سمت چپ خواهد نمود. شتاب گرفتن در جهت منفی ادامه خواهد یافت تا وقتی که ماشین به نقطه‌ی استراتژیک که با برچسب سوئیچ نمایش داده شده است برسد. از آنجا به بعد ماشین در جهت مثبت شتاب خواهد گرفت تا در نهایت در نقطه‌ی مقصد با سرعت صفر بایستد.

مسئله‌ی کمترین زمان کنترل یک مسئله‌ی بهینه‌سازی کنترلی کلاسیک است و کاربردهای زیادی در زمینه‌ی هوا فضا داشته و در بسیاری از کتاب‌های بهینه‌سازی کنترل مورد مطالعه قرار گرفته است [کِرک[1]، ۲۰۰٤]. راه‌حل این مسئله به راه‌حل بنگ- بنگ مشهور است چرا که این راه‌حل برای هر $x(0)$ و $v(0)$ شامل دوره‌ای از شتاب‌گیری با بیشترین شتاب ممکن در جهت مثبت و در ادامه، دوره‌ای از شتاب‌گیری با بیشترین شتاب ممکن در جهت منفی می‌باشد. همان‌طور که در شکل ۷-۱۰ نمایش داده شده است، مسئله‌ی کمترین زمان کنترل را می‌توان با نمودار صفحه‌ی فازی به‌صورت گرافیکی نمایش داد. برای سادگی فرض نموده‌ایم که جرم ماشین برابر ۲ است. در چنین حالتی، نمودار رسم شده در شکل ۷-۱۰، که نمودار سوئیچینگ نام دارد، توسط معادله‌ی زیر به دست می‌آید

$$x = v|v|/2 \qquad (٦-٧)$$

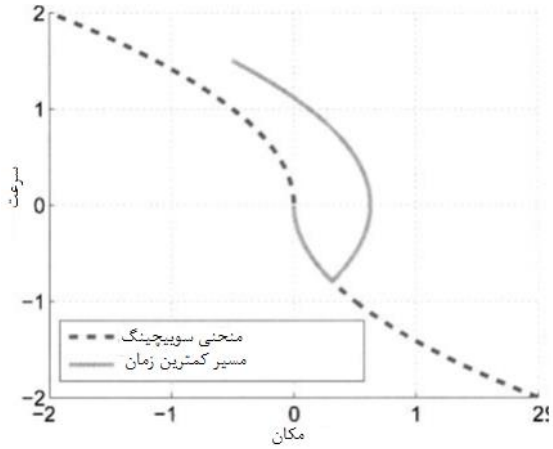
هدف این است که در کمترین زمان ممکن از هر نقطه بر روی صفحه‌ی فاز به مبدأ $x = 0$ و $v = 0$ رسید. اگر مکان و سرعت در بالای منحنی سوئیچینگ قرار دارند باید بیشترین شتاب را در جهت منفی اعمال کنیم. اگر مکان و سرعت در پایین منحنی سوئیچینگ قرار دارند باید بیشترین شتاب را در جهت مثبت اعمال نماییم. این کار ما را در مسیری قرار خواهد داد که در نهایت به منحنی سوئیچینگ خواهد رسید، نقطه‌ای که با رسیدن به آن باید جهت شتاب را معکوس نماییم. سپس منحنی سوئیچینگ را دنبال خواهیم کرد تا به مبدأ صفحه‌ی فاز برسیم.

[1] Kirk



شکل ۷-۱۰ منحنی سوئیچینگ برای کمترین زمان کنترل. اگر مکان و سرعت در بالای منحنی قرار داشته باشند، آنگاه شتاب ماکزیمم باید در جهت منفی اعمال گردد. اگر مکان و سرعت در پایین منحنی واقع شده باشند، آنگاه شتاب ماکزیمم باید در جهت مثبت اعمال شود.

شکل ۷-۱۱ بهترین مسیر را برای شرایط اولیه‌ی $x(0) = -0.5$ و $v(0) = 1.5$ نشان می‌دهد. این مسئله با قسمت پایینی شکل ۷-۹ متناظر است. سرعت ماشین به حدی زیاد است که نمی‌تواند در نقطه‌ی $x = 0$ بایستد. بنابراین ما بیشترین شتاب را آنقدر در جهت منفی اعمال می‌کنیم تا به منحنی سوئیچینگ برسیم؛ توجه داشته باشید که $v = 0$ در مدت زمانی که شتابِ ماشین در جهت منفی است اتفاق می‌افتد. هنگامی که ماشین به منحنی سوئیچینگ می‌رسد، بیشترین شتاب را در جهت مثبت اعمال می‌کنیم. این مسیر در کمترین زمان ممکن به مبدأ صفحه‌ی فاز ($x = 0$ و $v = 0$) خواهد رسید.



شکل ۷-۱۱ کمترین مسیر زمانی برای شرایط اولیه‌ی $x(0) = -0.5$ و $v(0) = 1.5$. در بالای منحنی سوئیچینگ شتاب -1 بوده و پس از رسیدنِ مسیر به منحنی سوئیچینگ، +1 می‌شود.

حال از GP برای رشد یک برنامه‌ی کمترین زمان کنترل برای این مسئله استفاده می‌نماییم. ما دو تابع Lisp خاص را برای این برنامه تعریف می‌کنیم. اولین آن‌ها تابع تقسیمِ حفاظت شده است که در معادله‌ی ۷–٤ نمایش داده شده است. دومی نیز اپراتور "بزرگتر است از" می‌باشد:

$$(defun\ GT\ (x\ y)\ (if\ (>\ x\ y)\ (return-from\ GT\ 1)\ (return-from\ GT\ -1))) \quad (۷–۷)$$

تابع GT اگر $x > y$ باشد ۱ را برگردانده و در غیر این صورت 1- را برمی‌گرداند.

برای ارزیابی هزینه‌ی برنامه از ۲۰ نقطه‌ی ابتداییِ اتفاقی با $|x| < 0.75$ و $|v| < 0.75$ در صفحه‌ی فاز استفاده می‌کنیم تا ببینیم آیا برنامه می‌تواند هر یک از این نقاط را در کمتر از ۱۰ ثانیه به مبدأء مختصات برساند یا خیر. اگر برنامه برای یک شرط آغازین موفق عمل نماید، آنگاه سهمِ آن شبیه‌سازی در هزینه‌ی برنامه برابرست با زمان مورد نیاز برای آوردن (x, v) به مبدأ. اگر برنامه در طول ۱۰ ثانیه موفق به انجام این کار نشود، آنگاه سهمِ هزینه‌ی آن شبیه‌سازی برابر ۱۰ خواهد بود. هزینه‌ی کلی یک برنامه‌ی کامپیوتری برابرست با میانگینِ هزینه‌ی همه‌ی ۲۰ شبیه‌سازی. جدول ۷–۳ پارامترهای GP برای این مسئله را خلاصه کرده است. این پارامترها عمدتا از [کوزا، ۱۹۹۲، بخش ۷–۱] گرفته شده است.

جدول ۷–۳ پارامترهای GP برای مسئله‌ی کمترین زمانِ کنترل ماشین.

گزینه GP	تنظیمات
هدف	پیدا نمودن برنامه کمترین زمان کنترل وسیله نقلیه
مجموعه ترمینال	x (مکان)، v (سرعت)، -1
مجموعه تابع	$+$، $-$، $*$، DIV، GT، ABS
هزینه	زمان آوردن وسیله نقلیه به مبدأ صفحه فاز که بر روی ۲۰ وضعیت ابتدایی اتفاقی میانگین‌گیری می‌شود
حد جمعیت	۵۰
اندازه جمعیت	۵۰۰
بیشترین عمق درخت ابتدایی	٦
روش مقداردهی اولیه	Ramped Half-and-Half
بیشترین عمق درخت	۱۷
احتمال برش	۰,۹
احتمال جهش	۰
تعداد نخبه‌ها	۲
روش انتخاب	مسابقه (بخش ۸–۷–٦ را ببینید)

شکل ۷–۱۲ هزینه‌ی بهترین راه‌حل GP را به عنوان تابعی از شماره‌ی نسل نشان می‌دهد. در مورد این اجرای به خصوص، بهترین برنامه‌ی کامپیوتری بعد از کمتر از ۱۰ نسل پیدا می‌شود، اما میانگین عملکرد کل جمعیت در طول ۵۰ نسل به طور پیوسته کاهش می‌یابد. در اکثر مسائل GP، پیدا شدن بهترین راه‌حل بیشتر از ۱۰ نسل به طول می‌انجامد. دلیل این که این اجرا سریعتر از سایر اجرائیات GP بود احتمالاً به خاطر آن است که مسئله بسیار آسان بوده و یا شاید صرفا یک اتفاق آماری بوده باشد. بهترین راه‌حل پیدا شده توسط GP به شرح زیر است

$$u = (* (GT\ (-(DIV\ x\ v)(-\ -1\ v)\ \big(GT\ (+\ v\ x)\ (DIV\ x\ v)\big)) \\ (DIV\ \big(GT\ (+\ x\ v)(+\ v\ x)\big)\ (GT\ (+v\ x)\ x)))) \qquad (۸–۷)$$

نمودار سوئیچینگ برای این کنترل به همراه بهترین نمودار سوئیچینگ به لحاظ نظری در شکل ۷–۳ نمایش داده شده‌اند. برای $v < 0$ دو نمودار بسیار شبیه هم هستند. برای $v > 0$ اختلاف بیشتری میان دو شکل وجود دارد اما همچنان شکل کلی آن‌ها شبیه هم است.

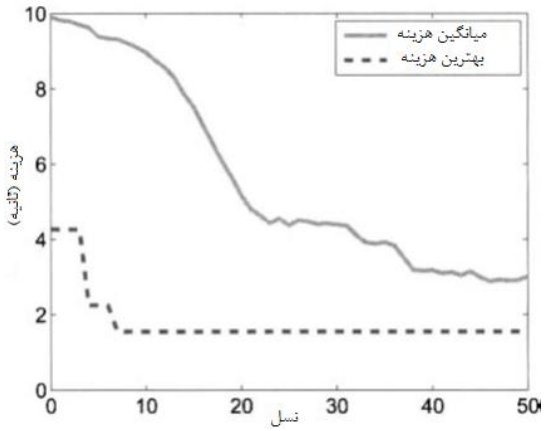
در نمودار سوئیچینگ بهینه‌ی نظری، میانگین زمان مورد نیاز ماشین برای رسیدن به مبدأء صفحه‌ی فاز برای ۱۰۰۰۰ شرط اولیه‌ی مختلف در بازه‌ی فضایی $x \in [-0.75, +0.75]$ و $v \in [-0.75, +0.75]$ برابر ۱٫۵۳ می‌باشد. این درحالی است که این مقدار برای نمودار سوئیچینگ به دست آمده از GP برابر ۱٫۵۰ است. به طرز بسیار جالبی، نمودار سوئیچینگ به دست آمده از GP کمی بهتر از نمودار سوئیچینگ بهینه‌ی نظری عمل می‌نماید. این موضوع از لحاظ نظری ممکن نیست اما عمل و نظریه همیشه با هم همخوانی ندارند. در عمل، مواردی از پیاده‌سازی وجود دارد که امکان عملکرد بهتر از استراتژی بهینه‌ی نظری را ایجاد می‌کنند. برای مثال، ما هنگامی که به مقادیر $|x| < 0.01$ و $|v| < 0.01$ رسیدیم، عملیات شبیه‌سازی را متوقف کردیم و آن را موفقیت به حساب آوردیم. به‌صورت نظری، می‌توان با خطایِ دقیقا صفر به مقصد رسید ولی در عمل نمی‌توان. همچنین، در عمل ما از گام‌های زمانی $\tau = 0.02$ برای شبیه‌سازی سیستم متحرک استفاده نمودیم. به عبارت دیگر به جای استفاده از معادلات بسیار دقیق و نظری زیر

$$\dot{x} = v \\ \dot{v} = u \qquad (۹–۷)$$

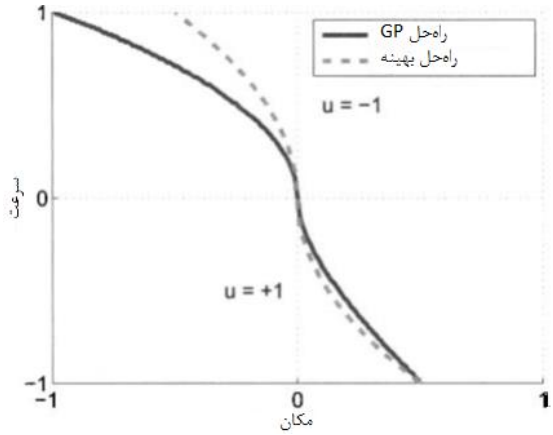
از معادلات تقریبی زیر استفاده نمودیم

$$v_{k+!} = v_k + \tau u_k \\ x_{k+!} = x_k + \tau(v_k + v_{k+1})/2 \qquad (۱۰–۷)$$

که در آن‌ها k اندیس زمان بوده و در بازه‌ی میان ۰ و ۵۰۰ قرار دارد (که برابر ۰ تا ۱۰ ثانیه است). این بدان معناست که ما از انتگرال‌گیری مستطیلی برای به دست آوردن سرعت و از انتگرال‌گیری ذوزنقه‌ای برای به دست آوردن مکان استفاده نموده‌ایم [سایمون، ۲۰۰۶، فصل ۱]. این تفاوت‌ها میان عمل و نظریه ممکن است باعث بروز ناهمخوانی‌های بیشتری میان نمودار سوئیچینگ GP و نمودار سوئیچینگ بهینه‌ی نظری شود و به همین علت ممکن است ناهمخوانیِ میان نمودار سوئیچینگ GP و نموار سوئیچینگ بهینه‌ی نظری از ناهمخوانی‌های میان نمودارهای سوئیچینگ مختلف به دست آمده از GP هم بیشتر باشد.



شکل ۷–۱۲ عملکرد GP برای مسئله‌ی کمترین زمان کنترل. بهترین راه‌حل بعد از کمتر از ۱۰ نسل پیدا شده است.



شکل ۷–۱۳ بهترین نمودار سوئیچینگ به دست آمده از GP همراه با نمودار سوئیچینگ بهینه‌ی نظری.

نظریه در مقابل عمل

برتری نمودار سوئیچینگ GP بر نمودار بهینه‌ی نظری باعث قوت گرفتن نکته‌ای مهم در مورد تفاوت‌های میان نظریه و عمل می‌شود. راه‌حل‌های مهندسی اغلب بر پایه‌ی نظریات قرار دارند، اما همان‌طور که هر مهندس عملگرایی می‌داند، اگر بخواهیم نتایج نظری را در جهان واقعی به کار ببریم، باید آن‌ها را اصلاح کنیم. این مثال نشان می‌دهد که GP می‌توان شرایط دنیای واقعی را در نظر بگیرد و راه‌حلی پیدا کند که از راه‌حلِ بهینه‌ی نظری برای یک مسئله بهتر باشد.

شاید یادگیری نظریه کنترل بهینه و حل مسئله‌ی کمترین زمان کنترل به روش‌های سنتی‌تر ساده‌تر از یادگیری کار با GP باشد. اما شاید هم نه! این مسئله به ما نشان می‌دهد که می‌توان از GP برای حل مسائلی استفاده نمود که خودمان مهارت لازم برای حلشان را نداریم. در ادامه نیز نشان داده خواهد شد که با در نظرگیری موارد عملی می‌توان به راه‌حلی بهتر از راه‌حل بهینه نیز دست یافت.

۷–۴ نفخ (تورم) برنامه‌نویسی ژنتیک

برنامه‌نویسی ژنتیک ممکن است منجر به ایجاد برنامه‌هایی شود که به‌طور غیرمعقولی بزرگ بوده و به سطح محاسباتی بالایی نیاز دارند. نام‌های مختلفی برای کد اضافی که در طول GP رشد می‌کند وجود دارد، از جمله آن‌ها می‌توان به اینترون[1]، کد بنجل، کَرک، کد بی‌تأثیر و کد نامرئی اشاره نمود [لانگدون[2] و پولی، ۲۰۰۲، فصل ۱۱]. برخی مثال‌های اینترون در زیر آمده‌اند:

$$(not\ (not\ x)) \\ (+\ x\ 0) \\ (if\ (>\ 1\ 2)\ x\ y) \qquad (۱۱–۷)$$

هر پیاده‌سازی جدی از GP باید در برابر تورم کد محافظت شود تا از رشد کنترل‌نشده‌ی طول کد جلوگیری به عمل آید. چندین راه برای جلوگیری از این تورم وجود دارد.

اولین راه برای مبارزه با تورم کد استفاده از یک پارامتر بیشترین عمق (D_c) است که قبلا در زیربخش ۷–۲–٦ در مورد آن بحث نمودیم. با این حال، این کار نیاز به تعادل دارد. اگر D_c خیلی کوچک باشد، آنگاه فضای جستجوی GP را محدود کرده‌ایم و بدین ترتیب ممکن است برازندگی بهترین برنامه‌ای که می‌تواند پیدا شود را کاهش دهیم.

[1] Intron
[2] Langdon

راه دوم برای مبارزه با تورم کد، تنظیم پیاده‌سازی‌های برش و جهش است. برای مثال، اندازه‌ی عادلانه‌ی برش نقاط برش را به‌گونه‌ای انتخاب می‌کند که اندازه‌ی تکه‌های کد والدین متعادل باشد و بدین ترتیب باعث می‌شود کدهای فرزند طولانی‌تر از والدین نباشند [لانگدن، ۲۰۰۰]. اگر از جهش زیردرخت استفاده شود، می‌توان آن را به‌گونه‌ای تنظیم نمود که محدود شدن برنامه‌ی جهش یافته تضمین شود [کینیر، ۱۹۹۳]. جهش جرثقیلی کدها را برای کاهش طول برنامه می‌زداید [کینیر، ۱۹۹٤]. برشِ یک= نقطه‌ای روشی است که ما در این فصل بررسی نکرده‌ایم، اما این روش به‌صورت خودکار عمق فرزندان را به عمق بزرگترین والد محدود می‌سازد [پولی و همکاران، ۲۰۰۸، فصل ۲۵].

سومین راه برای مبارزه با تورم کد مجازات کردن برنامه‌های طولانی در عملیات انتخاب، بازتولید و برش می‌باشد. این ایده را می‌توان به چند طریق می‌توان پیاده‌سازی نمود. برای مثال، می‌توان یک مجازاتِ هزینه به برنامه‌های بزرگ اضافه نمود:

$$\text{هزینه تنبیه} \leftarrow \text{هزینه} + \text{اندازه برنامه} \qquad (۷-۱۲)$$

در کل، اگر برنامه‌های بزرگ مجازات شوند پیدا کردن یک راه‌حل خوب به ارزیابی‌های برازندگی بیشتری نیاز دارد [کوزا، ۱۹۹۲، فصل ۲۵]. از سوی دیگر، اگر برنامه‌ها کوچک‌تر باشند، ارزیابی‌های برازندگی سریعتر خواهند بود. این رویکرد الزاماً فرایند انتخاب را به سمت برنامه‌های کوچکتر گرایش می‌دهد. این ایده با نام‌های فشار صرفه‌جویی، تیغ اَکام[1] و کمترین طول توضیحات شناخته می‌شود [لانگدن و پولی، ۲۰۰۲، فصل ۱۱]. یک رویکرد دیگر برای مجازات کردن برنامه‌های بلند روش تارپیَن[2] می‌باشد [پولی، ۲۰۰۳]. در این روش احتمال انتخاب برنامه‌های با طول بیشتر از میانگین برابر صفر قرار داده می‌شود. با تغییر دادن تعداد دفعات تکرار این کار می‌توان قابلیت ضدتورم این روش را تنظیم نمود. همچنین فایده‌ی دیگر این کار این است که با این روش زمان اجرا کاهش می‌یابد چرا که برنامه‌هایی که دارای احتمال انتخاب صفر هستند نیار به ارزیابی ندارند.

همچنین راه‌های دیگری برای مبارزه با تورم کد وجود دارد، برای مثال استفاده از بهینه‌سازی چندهدفه با دو هدفِ برازندگی برنامه و طول برنامه (فصل ۲۰، زدودن خودکار کد اضافی و استفاده از توابع تعریف شده به‌صورت خودکار(ADF[3]) را ببینید). ناگفته نماند که این روش‌ها پیچیده‌تر بوده و صرفا از تورم کد جلوگیری نمی‌کنند [لانگدن و پولی، ۲۰۰۲، فصل ۱۱].

[1] Occam's Razor
[2] Tarpeian Method
[3] Automatically Defined Functions

در آخر نیز متذکر می‌شویم که تورم کد در برخی شرایط می‌تواند سودمند باشد. تورم دارای همانندی زیستی است و می‌تواند به برنامه‌های کامپیوتری کمک کند فرزندانشان را در برابر برش‌های مضر مراقبت کنند [آنجلین، ۱۹۹۶b]، [نوردین[1] و همکاران، ۱۹۹۶]. این موضوع باعث به وجود آمدن عبارت *برازندگی مؤثر* می‌شود. این عبارت هم به میزان برازندگی یک برنامه و هم به میزان برازندگی مورد انتظار برای فرزندان آن برنامه، اشاره دارد [بنژاف[2] و همکاران، ۱۹۹۸، فصل ۷]. امکان ایجاد فرزند برازنده توسط یک والد برازنده‌ی شکننده از امکان ایجاد فرزند برازنده توسط یک والد برازنده با تورم کد زیاد کمتر است. به عبارت دیگر هرچه والد برازنده دارای نورم کد بیشتری باشد، احتمال ایجاد فرزنده برازنده توسط وی بیشتر خواهد شود. عملیات برش میان دو برنامه را در نظر بگیرید. امکان ایجاد فرزند برازنده توسط یک برنامه‌ی خوب با تورم کد زیاد بیشتر است، چرا که احتمال قرار گرفتن نقطه‌ی برش در قسمت بلااستفاده‌ی والد بیشتر است.

ما همچنین می‌توانیم در مورد پیچیدگی مؤثر صحبت کنیم. پیچیدگی مطلق یک برنامه‌ی کامپیوتری تابعی است از طول و ساختار آن. این در حالی است که پیچیدگی مؤثر تابعی است از طول، ساختار و قسمت غیرمتورم از برنامه‌ی کامپیوتری.

۷–۵ رشد نهادهایی غیر از برنامه‌های کامپیوتری

موفقیت‌های GP بسیار تأثیرگذارند. [کوزا، ۱۹۹۲] مثال‌هایی از GP برای آشکار ساختن نهادهای مثلثاتی، آشکار ساختن قوانین علمی، حل معادلات ریاضی به فرم سمبولیک، استنتاج کردن فرم سمبولیک برای یک دنباله و پیدا کردن برنامه‌هایی برای فشرده‌سازی تصویر ارائه می‌دهد. وی همچنین مثال‌هایی از مسائل کنترلی مانند متعادل نمودن یک آونگ معکوس بر روی یک چرخ متحرک به دست می‌دهد.

اما دامنه‌ی استفاده‌ی GP همچنان به گستردگی سایر الگوریتم‌های تکاملی نیست. دلایل زیادی برای این عقب‌ماندگی وجود دارد [کوزا، ۱۹۹۲، فصل ۱].

۱. مهندسان آموزش دیده‌اند برای مسائل راه‌حل‌های درست پیدا کنند. در تکالیف مدارس، معمولاً تنها یک راه‌حل درست برای یک مسئله وجود دارد. اما GP راه‌حل‌هایی را می‌یابد که تقریباً درست هستند. به هر حال، در دنیای مهندسی واقعی تمام همه‌ی راه‌حل‌های ما تقریبی هستند. دلایل زیادی برای این تقریب‌ها وجود دارند که مهمترین آن‌ها فرض‌هایی است که به هنگام به دست آوردن راه‌حل به صراحت و یا به‌طور ضمنی انجام می‌دهیم.

[1] Nordin
[2] Benzhaf

۲. مهندسین آموزش دیده‌اند راه‌حل‌ها را با استفاده از بهبودهای تدریجی بیابند. GP نیز این رویه را دنبال می‌کند و علاوه بر آن به جستجو در نقاط کور نیز می‌پردازد. در دنیای واقعی شکست مایه‌ی ننگ است، اما بهترین و موفق‌ترین مهندسان می‌دانند که شکست پیش‌نیاز پیروزی است [پِتروسکی[1]، ۱۹۹۲]. این موضوع نه تنها برای انسان‌ها، بلکه برای GP نیز صادق است.

۳. مهندسین آموزش دیده‌اند مسائل را به‌صورت قیاسی حل نمایند. ما ابتدا با مسئله‌ی مورد نظر آشنا می‌شویم، سپس به‌صورت گام به گام به تولید راه‌حل می‌پردازیم. اگر بخواهیم آزادانه صحبت کنیم، نوعی قیاس در GP نیز وجود دارد چرا که در آن برنامه‌ها با بالاترین میزان برازندگی با هم ترکیب شده تا برنامه‌هایی با برازندگی بالاتر ایجاد شوند. اما برنامه‌های کامپیوتری ایجاد شده توسط GP به‌صورت منطقی و با اضافه کردنِ گام به گام قابلیت‌های مختلف ایجاد نمی‌شوند.

٤. مهندسین آموزش دیده‌اند مسائل را به‌صورت قاطع حل نمایند. هر چه بیشتر بتوانیم اتفاقی بودن را از محیطمان بزداییم، کنترل بیشتری به دست می‌آوریم و هر چه کنترل بیشتری به دست آوریم، با روش حلمان بهتر پیش خواهیم رفت. اما GP مانند سایر الگوریتم‌های تکاملی برای پیدا کردن راه‌حل‌های خوب به اتفاقی بودن وابسته است.

۵. مهندسین آموزش دیده‌اند مسائل را به طور اقتصادی حل نمایند. یک راه‌حل کوتاه و ساده بهتر از یک راه‌حل پیچیده است. اما GP برنامه‌هایی را رشد می‌دهد که دارای شاخه‌هایی هستند که هیچگاه اجرا نمی‌شوند، و یا دارای ترمینال‌هایی هستند که در نتیجه‌ی نهایی هیچ دخالتی ندارند و دارای ساختاری غیرکارآمد هستند. این مانند فرایندهای حل مسائلی است که در طبیعت می‌بینیم. بسیاری از حیوانات چندصد فرزند به دنیا می‌آورند تا یکی از آن‌ها بتواند به بقای خود ادامه دهد. تکامل فرایندی است که به طرز بدنامی افراط‌کارانه و غیرکارآمد است.

٦. مهندسین آموزش دیده‌اند مسائل را با معیارهای موفقیت خاصی حل نمایند. ما وظیفه، زیروظیفه، مراحل برجسته و زمانبندی داریم. فرایند اعتبارسنجی موفقیت و یا شکست ما را تعیین می‌کند. با این حال، GP نقطه‌ی پایان مشخصی ندارد. این موضوع در زیربخش ۷-۲-۲ مورد بحث قرار گرفته است.

اگرچه بسیاری از این فاکتورهای به سایر الگوریتم‌های تکاملی نیز قابل اعمال هستند اما به نظر می‌رسد که خاصه‌ی GP باشند. تفاوت در این است که الگوریتم‌های تکاملی راه‌حل‌ها را می‌یابند ولی GP روش حل را می‌یابد. به نظر می‌رسد که بی‌ملاحظگی در راه‌حل‌ها را بهتر بتوان تحمل کرد تا در روش حل. هنگامی که

[1] Petroski

برای یک مسئله یک راه‌حل خوب پیدا می‌کنیم، تا زمانی که راه‌حل به خوبی عمل می‌نماید، نگران این نیستیم که راه‌حل از کجا آمده است. اما وقتی یک روش حل می‌یابیم، اگر طرز کارش را نفهمیم، نسبت به آن بی‌اعتماد خواهیم بود حتی اگر درست کار کند.

در این فصل دیده‌ایم که GP برنامه‌های کامپیوتری را رشد می‌دهد. با این حال، عرف آن است که برنامه‌های کامپیوتری توسط انسان‌ها نوشته شوند تا توسط GP. این به آن دلیل است که برنامه‌های کامپیوتری را به‌گونه‌ای می‌توان طرح‌ریزی، ساختاربندی و مدیریت نمود که جوابگوی انسان‌های متخصص باشند. برنامه‌های کامپیوتری را می‌توان بخش‌بندی نمود و بدین ترتیب یک وظیفه را به چندین زیروظیفه تقسیم کرده و هر زیروظیفه را به یک برنامه‌نویس کامپیوتری واگذار نمود. پروژه‌های نرم‌افزاری بزرگ آنقدر درجه‌ی اختیار دارند که نمی‌توان از GP، که یک فرایند متکی به اتفاقی بودن است، انتظار موفقیت در آن‌ها را داشت. حتی اگر GP موفق هم بشود، برنامه‌ی نهایی به احتمال زیاد غیرکارآمد بوده و نگه‌داری از آن دشوار. به‌طور خلاصه، کارهای برنامه‌نویسی کامپیوتری ساده برای GP بسیار ساده‌اند، چرا که انسان‌ها می‌توانند بدون تلاش چندانی آن‌ها را به انجام برسانند. اما وظایف برنامه‌نویسی کامپیوتری دشوار برای GP زیادی سخت هستند و به همین علت به نبوغ انسان نیاز دارند. این موضوع باعث ایجاد سؤالاتی مهم در مورد GP می‌شود. چه نوع مسائلی برای GP مناسب هستند که به واقع برای انسان جزء مسائل دشوار به شمار می‌آیند؟ آیا GP هیچ کاربرد عملی دارد؟

برای بحث در مورد این سؤالات، نگاهی به نواحی کاربردی الگوریتم‌های تکاملی می‌اندازیم. الگوریتم‌های تکاملی در پیدا کردن راه‌حل برای مسائل بهینه‌سازی سخت، چندبعدی و چندپیمانه‌ای خوب هستند. برنامه‌نویسی کامپیوتری می‌تواند سخت، چندبعدی و چندپیمانه‌ای باشد. با این حال، برنامه‌نویسی کامپیوتری کاری است که بسیاری از افراد در آن استعداد دارند. بهینه‌سازی پارامتر کاری است که انسان‌ها در آن استعداد ندارند. تقریبا تمامی مسائل کوچک اما مهمِ بهینه‌سازی پارامتر توسط برنامه‌های کامپیوتری حل می‌گردنند. امروزه می‌بینیم که استفاده از الگوریتم‌های تکاملی بسیار گسترش یافته چرا که آن‌ها در حل مسائلی که برای انسان‌ها دشوارند، بسیار خوب عمل می‌کنند.

از آنجایی که بسیاری از انسان‌ها برنامه‌نویسان توانمند و ماهری هستند، GP کاربرد چندانی در مسائل برنامه‌نویسی دنیای واقعی ندارد. با این حال، GP را می‌توان در مورد مسائلی مشابه برنامه‌نویسی کامپیوتر که انسان‌ها در آن از توانمندی لازم برخوردار نیستند به کار برد. بسیاری از مسائل مهندسی وجود دارند که انسان‌ها استعدادی در حلشان نداشته و می‌توان راه‌حل‌های نامزدشان را با ساختارهایی درخت مانند نشان

داد. این مسائل شامل طراحی سیستم لنزها [کوزا و همکاران، ۲۰۰۸]، ساختارهای کریستال فوتونیک [پریبِل[1] و همکاران، ۲۰۰۵]، الگوریتم‌های دسته‌بندی پروتئین‌ها [کوزا، ۱۹۹۷]، ماشین‌های خودکار سلولی [آندره[2] و همکاران، ۱۹۹٦]، الگوریتم‌های پیدا نمودن راه‌حل عددی برای معادلات دشوار [بالاسوبرامانیم[3] و کومار، ۲۰۰۹]، الگوریتم‌های حل پازل‌ها و پیدا کردن استراتژی بازی [هاپمن[4] و سیپر[5]، ۲۰۰۷]، [هاپمن و همکاران، ۲۰۰۹]، مدارهای الکتریکی [مک کوناهی[6] و همکاران، ۲۰۰۸]، ارائه‌های گیت‌های میدانی قابل برنامه‌ریزی [کوزا و همکاران، ۱۹۹۹] و آنتن‌ها [لون[7] و همکاران، ۲۰۰٤] می‌باشند.

ویژگی کلیدی این مسائل آن است که نمی‌توان به سادگی آن‌ها را به مسائل بهینه‌سازی پارامتر کاهش داد و بنابراین نمی‌توان آن‌ها را با GA ها، ES، EP و سایر روش‌های مشابه حل نمود. آن‌ها با یک GP که یک الگوریتم و یا یک برنامه‌ی طراحی را رشد می‌دهد، قابل حل هستند. نتایج حاصل از GP معمولاً با استفاده از مقدار کمی از اطلاعات خاصه‌ی مسئله، اختراعات موجود را بهبود می‌بخشند [کوزا، ۲۰۱۰]. امروزه آنقدر از عبارت "هوش کامپیوتری" استفاده می‌شود که دیگر تقریباً معنایی درست ندارد و تبدیل به یک کلمه‌ی بی‌محتوا شده است. اما وقتی یک برنامه‌ی کامپیوتری یک اختراع با قابلیت ثبت ایجاد می‌کند، احتمالاً در نظر بسیاری از مردم از درجه‌ی هوشی بالایی برخوردار است.

۷–۶ تحلیل ریاضی برنامه‌نویسی ژنتیک

GPها را نیز می‌توان مانند سایر الگوریتم‌های تکاملی تحلیل ریاضی نمود (فصل ٤ را ببینید). این بخش نظریه شِمای GA را به GP بسط می‌دهد [لانگدن و پولی، ۲۰۰۲،، فصل ٤]. رویکرد کلی ما در این بخش آن است که با استفاده از مثال‌های ساده، ایده‌ی کلی چگونگی کارکرد نظریه شِمای GP را دریابیم و سپس از این مثال‌ها برای ارائه‌ی یک فرمول کلی برای شِمای GP استفاده نماییم.

۷–۶–۱ تعریف و نمادگذاری

در اولین مثال فرض بر آن است که مجموعه‌ی ترمینال $\{x, y\}$ است. ما از علامت # برای نشان دادن یک مجموعه‌ی ترمینال "بی‌اهمیت" استفاده می‌کنیم. شِمای زیر را در نظر بگیرید

[1] Prible
[2] Andre
[3] Balasubramaniam
[4] Hauptman
[5] Sipper
[6] McConaghy
[7] Lohn

$$H = ((+ (- \# y) \#) \quad (۱۳-۷)$$

اگر x و y تنها ترمینال‌های در دسترس باشند آنگاه این شِما دارای چهار نمونه است:

$$\begin{matrix} (+ (- x\, y) x), & (+ (- x\, y) y), \\ (+ (- x\, y) x), & (+ (- y\, y) y) \end{matrix} \quad (۱٤-۷)$$

گوییم یک شِما با یک عبارت s همخوانی دارد اگر عبارتِ s یکی از نمونه‌های شِما باشد. برای مثال، شِمای $(+\#y)$ با عبارتِ $(+xy)$ و $(+2y)$ همخوانی دارد اما با عبارت $(-xy)$ همخوانی ندارد.

حال به تعریف سه عبارت مهم می‌پردازیم که با شماتای GP مرتبط هستند.

۱. مرتبه‌ی H یا o(H)، برابرست با تعداد نمادهای تعریف شده در H که شامل توابع و ترمینال‌های می‌شود. در معادله‌ی (۷-۱۳)، $o(H) = 3$.

۲. طول H یا n(H)، برابرست با تعداد کل نمادهای موجود در H که هم شامل توابع و ترمینال‌های تعریف‌شده و هم شامل توابع و ترمینال‌های بی‌اهمیت می‌شود. در معادله‌ی (۷-۱۳)، $n(H) = 5$.

۳. طول تعریف H یا L(H)، برابرست با کمترین تعداد لینک در زیردرخت‌های نحو که تمامی نمادهای تعریف شده را شامل می‌شوند. تعیین کردن طول تعریف از روی عبارت s کمی دشوار است اما می‌توان آن را به راحتی با نگاه کردن به درخت نحو مربوطه تعیین نمود.

شکل ۷-۱٤ چند نمونه از درخت‌های محو همراه با مرتبه، طول و طول تعریفشان نشان می‌دهد.

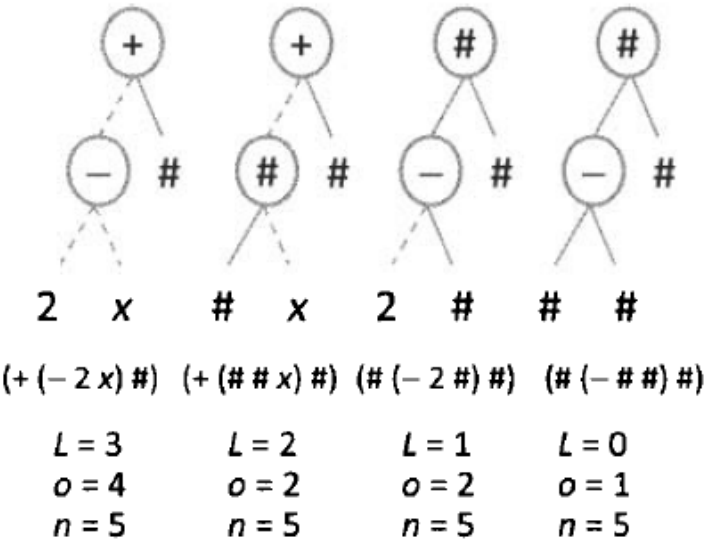
حال تعداد شماتاهایی را که با یک عبارت s با طول n همخوانی دارند را در نظر بگیرید. به‌عنوان یک مثال عبارت s زیر را در نظر بگیرید

$$((+ (- 2\, x) (* 3\, y)) \quad (۱۵-۷)$$

یک شِما با این عبارت s همخوانی دارد اگر در هر گره نمادِ # یا نمادی برابر با نمادِ عبارت s داشته باشد. بنابراین می‌توان دید که تعداد شماتاهایی که با یک عبارت s به طول n همخوانی دارند برابر با 2^n می‌باشد. برای مثال تعداد شماتاهایی که با عبارت s معادله‌ی (۷-۱۵) همخوانی دارند برابر $2^7 = 128$ می‌باشد.

حال به تعریف ساختار یک شِما می‌پردازیم. ما از حرف G برای نشان دادن ساختار یک شِما استفاده می‌کنیم. برای به دست آوردن G باید تمام نمادهای موجود در H را با نماد # جایگزین نماییم. برای مثال، شِمای معادله‌ی (۷-۱۳) دارای ساختار زیر است

$$G = ((\# (\# \# \#) \#) \quad (۱٦-۷)$$



شکل ۷-۱٤ چهار شماتای GP به فرم درخت نحو و به فرم عبارت s. طول تعریف L، مرتبه o و طول n در زیر هر شِما نشان داده شده است. لینک‌هایی که از آن‌ها برای تعیین طول تعریف استفاده شده است آن‌هایی هستند که در زیردرختی قرار دارند که همه‌ی نمادهای غیر # را شامل می‌شود. این لینک‌ها با خط چین نمایش داده شده‌اند.

۷-۶-۲ انتخاب و برش

یک GP را در نظر بگیرید که از انتخاب چرخ رولت و برش استفاده می‌نماید. ما از نماد $m(H,t)$ برای نشان دادن تعداد نمونه‌های شمای H در جمعیت GP در tاُمین نسل استفاده می‌کنیم. همچنین از نماد $m(H,t+1/2)$ برای نشان دادن تعداد نمونه‌های شِما در جمعیت بعد از عمل انتخاب استفاده می‌نماییم. بدین ترتیب، $m(H,t+1)$ برابر با تعداد نمونه‌های شِما پس از انتخاب، برش و جهش خواهد بود. اگر از انتخاب چرخ رولت استفاده نماییم آنگاه به‌طور میانگین خواهیم داشت

$$m\left(H,t+\frac{1}{2}\right)=m(H,t)f(H,t)/f_{ave}(t) \qquad (۱۷-۷)$$

که در آن $f(H,t)$ برابرست با میانگین برازندگی تمام نمونه‌های H در نسل tاُم و $f_{ave}(t)$ برابرست با میانگین برازندگی تمام ذرات در نسل tاُم.

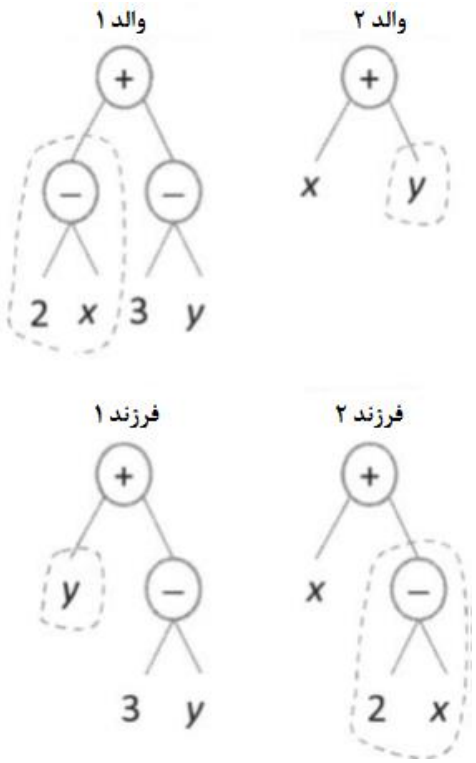
حال تأثیر برش بر روی جمعیت را در نظر بگیرید. اگر یکی از والدین موجود در جمعیت نمونه‌ای از H باشد، آنگاه برش ممکن است این والد را از بین برده و فرزندانی ایجاد کند که هیچ کدام نمونه‌ای از H نباشند. بدین ترتیب تعداد نمونه‌های H نسل بعد کاهش می‌یابند. برش به دو طریق ممکن است یک نمونه از H را از بین ببرد. والد p_1 را در نظر بگیرید به طوری که $p_1 \in H$ و $p_1 \in G$ باشد، G ساختار H است. اول

آنکه یک نمونه از H می‌تواند با برش داده شدن p_1 با p_2 که $p_2 \notin G$ از بین برود. ما این حالت را اتفاق D_1 می‌نامیم. در حالت دوم، یک نمونه از H می‌تواند با برش داده شدن p_1 با p_2 که $p_2 \in G$ اما $p_2 \notin H$ از بین برود. ما این اتفاق را D_2 می‌نامیم. از آنجا که D_1 و D_2 اتفاق‌هایی دوبه‌دو ناسازگار هستند، بنابراین احتمال آنکه برش یک نمونه از H را از بین ببرد برابرست با

$$\Pr(D) = \Pr(D_1) + \Pr(D_2) \qquad (۱۸-۷)$$

مثال ۷-۱

شکل ۷-۱۵ مثالی از اتفاق D_1 را نشان می‌دهد. والدین $((-2x)(-3y))+$ و $(+xy)$ برای برش انتخاب شده‌اند. نقاط برش عبارتند از چپ‌ترین تابع تفریق در والد اول و ترمینال y در والد دوم. این نقاط به‌صورت اتفاقی انتخاب شده‌اند. عمل برش فرزندانِ $((-3y)+y)$ و $((-2x)+x)$ را ایجاد می‌کند. می‌توان دید که هیچ یک از فرزندان ساختاری مشابه هیچ یک از والدین ندارد. برش تمام شماتای H_1 و همچنین شماتای H_2 را از بین برده است.



شکل ۷-۱۵ برش میان این دو والد موجب ایجاد فرزندانی می‌شود که ساختار هیچ یک از دو والد را ندارند. این مثالی از اتفاق D_1 است که در آن برش میان ذرات با ساختارهای متفاوت باعث از بین رفتن شماتا می‌شود.

مثال ۷-۲

ما احتمال وقوع D_2 را با نگاه به یک مثال در نظر می‌گیریم. شمای $H = (\#xy)$ را در نظر بگیرید. فرض کنید دو والد به‌صورت زیر باشند

$$p_1 = (+\ x\ y) \in H \\ p_2 = (-\ y\ x) \notin H \qquad (۱۹-۷)$$

هر دو والد دارای ساختار مشابه G هستند با این تفاوت که p_1 به شمای H تعلق دارد ولی p_2 به این شِما تعلق ندارد. اگر در هر دو والد نقاط برش در بالاترین لینک انتخاب شوند آنگاه فرزندان ایجاد شده به‌صورت زیر خواهند بود

$$c_1 = (+\ y\ x) \notin H \\ c_2 = (-\ x\ y) \in H \qquad (۲۰-۷)$$

می‌توانیم ببینیم که نمونه‌ی H برای نسل بعد حفظ شده است.

حال شِمای $H = (+x\#)$ را همراه با همان دو والد نشان داده شده با معادله‌ی (۷-۱۹) در نظر بگیرید. مانند قبل $p_1 \in H$ و $p_2 \notin H$ می‌باشد. با این حال، در این مورد نه c_1 و نه c_2 از معادله‌ی (۷-۲۰) متعلق به H نمی‌باشند و بدین ترتیب نمونه‌ی H از بین رفته است.

حال تأثیر D_1 بر روی تعداد نمونه‌های شما در جمعیت را در نظر می‌گیریم. به یاد آورید D_1 اتفاقی است که در آن نمونه‌ای از H که دارای ساختار G است به دلیل وقوع برش میان والد p_1 که $p_1 \in H$ و $p_1 \in G$ و والد p_2 که $p_2 \notin G$، از بین می‌رود. یعنی

$$D_1 = D \cap (p_2 \notin G) \qquad (۲۱-۷)$$

که در آن D واقعه‌ای است که در آن نمونه‌ای از H از بین می‌رود. قضیه‌ی بیز به ما می‌گوید که

$$\Pr(D_1) = \Pr(D \cap p_2 \notin G))\Pr(p_2 \notin G) \qquad (۲۲-۷)$$

اما احتمال آنکه $p_2 \notin G$ با تعداد ذراتی از جمعیت که بعد از انتخاب به G تعلق ندارند، متناسب است. یعنی

$$\Pr(p_2 \notin G) = (\mathrm{N} - \mathrm{m}\left(\mathrm{G}, \mathrm{t} + \frac{1}{2}\right))/\mathrm{N} \qquad (۲۳-۷)$$

که در آن N اندازه‌ی جمعیت است. با ترکیب این معادله با معادله‌ی (۷-۲۲) خواهیم داشت

$$\Pr(D_1) = \Pr(D \cap p_2 \notin G))\frac{N - m(G, t + \frac{1}{2})}{N} \qquad (۲۴-۷)$$

حال احتمال واقعه‌ی D_2 را در نظر بگیرید. به یاد آورید که D_2 واقعه‌ای است که در آن یک نمونه از شمای H، که دارای ساختار G است، به دلیل برش میان والدین p_1 و p_2 به طوری که $p_1 \in H$ و $p_1 \in G$ و $p_2 \in G$ از بین می‌رود. یعنی

$$D_2 = D \cap (p_2 \in G) \\ = D \cap (p_2 \in G) \cap (p_2 \notin H) \quad (۲۵-۷)$$

که در آن تساوی دوم از این حقیقت ناشی می‌شود که تخریب شِما اتفاق نمی‌افتد مگر آنکه $p_2 \notin H$. قضیه‌ی بیز بیان می‌دارد

$$\Pr(D_2) = \Pr(D \cap p_2 \in G, p_2 \notin H)\Pr(p_2 \in G, p_2 \notin H) \quad (۲۶-۷)$$

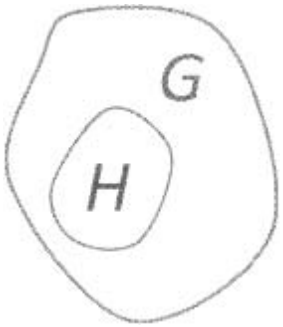
عبارت دوم در سمت راست معادله‌ی (۷-۲۶) برابرست با احتمال آنکه $p_2 \in G$ و $p_2 \notin H$. با این حال $p_2 \in H$ خود زیرمجموعه‌ای است از $p_2 \in G$. بنابراین داریم

$$\Pr(p_2 \in G, p_2 \notin H) = \Pr(p_2 \in G) - \Pr(p_2 \in H) \quad (۲۷-۷)$$

که این موضوع در شکل ۷-۱۶ نشان داده شده است.

احتمالات موجود در سمت راست معادله‌ی (۷-۲۷) به ترتیب با تعداد نمونه‌های G و H پس از فرایند انتخاب متناسب‌اند. بنابراین داریم

$$\Pr(p_2 \in G, p_2 \notin H) = \frac{m\left(G, t+\frac{1}{2}\right) - m\left(H, t+\frac{1}{2}\right)}{N} \quad (۲۸-۷)$$



شکل ۷-۱۶ مجموعه‌ی برنامه‌هایی که به شِمای H تعلق دارند زیرمجموعه‌ای است از برنامه‌های که به ساختار G تعلق دارند. بنابراین، $\Pr(p \in G, p \notin H) = \Pr(p \in G) - \Pr(p \in H)$.

حال عبارت سمت راست معادله‌ی (۷-۲۶) را در نظر بگیرید. این عبارت در واقع احتمال D را در شرایطی که $p_2 \in G$ و $p_2 \notin H$ باشد را به دست می‌دهد. یک نمونه از شِما تنها در صورتی از بین می‌رود که نقطه‌ی برش در یکی از لینک‌های تعریف شده‌ی شِما واقع شود. برای مثال در شکل ۷-۱٤، شِمای سمت چپ تنها در صورتی نابود می‌شود که برش در یکی از لینک‌های خط‌چین واقع شود. این موضوع برای تمامی شِماتای نشان داده شده در شکل صادق است. حتی اگر هم برش در یکی از آن لینک‌ها اتفاق بیافتد، باز هم بسته به محتویات زیردرختی که به نمونه‌ی شِما متصل است، امکان عدم نابودی شِما وجود دارد. از آنجا که تعداد لینک‌هایی که برش باید در آن‌ها اتفاق بیافتد تا امکان نابودی شِما وجود داشته باشد برابر $L(H)$ است، و همچنین از آنجا که که کلاً تعداد $n(H)-1$ لینک وجود دارد، احتمال وقوع برش در یکی از لینک‌های تعریف شده برابر $L(H)/(n(H)-1)$ است. از آنجا که وقوع برش در یکی از این لینک‌ها شرط لازم اما غیرکافی برای نابودی نمونه‌ی شِما است، احتمال وقوع D با فرض $p_2 \in G$ به‌صورت زیر خواهد بود

$$\Pr(D|p_2 \in G, p_2 \notin H) \leq \frac{L(H)}{n(H)-1} \qquad (۷-۲۹)$$

سمت راست معادله‌ی بالا، شکنندگی ترکیب گره‌ی شِمای H نام دارد [لنگدن و پولی، ۲۰۰۲، بخش ٤,٤]. ترکیب معادلات (۷-۱۸)، (۷-۲٤) و (۷-۲۸) حد بالای احتمال نابودیِ شِمای H را به دست می‌دهد:

$$\Pr(D) \leq \Pr(D|p_2 \in G)\frac{N - m\left(G, t+\frac{1}{2}\right)}{N} + \frac{L(H)}{n(H)-1}\frac{m\left(G, t+\frac{1}{2}\right) - m(H, t+\frac{1}{2})}{N} \qquad (۷-۳۰)$$

جمله‌ی بالا عبارتی کلی برای احتمال نابودی یک نمونه از شِمای H به دلیل برش را به دست می‌دهد.

با در نظر گرفتن این حقیقت که برش با احتمال p_c رخ می‌دهد، می‌توان جمعیت GP را آنقدر بزرگ فرض کرد که بتوان قانون اعداد بزرگ را در مورد $m(H, t+1)$، که برابر با تعداد نمونه‌های H در نسل $(t+1)$اُم است، به کار برد[گرین‌استید و اِسنِل، ۱۹۹۷]. این مقدار از جمع دو عبارت به دست می‌آید: (۱) تعداد نمونه‌های H پس از فرایند انتخاب، ضرب شده در احتمال عدم وقوع برش، (۲) تعداد نمونه‌های H پس از فرایند انتخاب، ضرب شده در احتمال وقوع برش، ضرب شده در احتمال عدم نابودی یک نمونه از H توسط فرایند برش. بدین ترتیب داریم

$$\begin{aligned} m(H, t+1) &= m\left(H, t+\frac{1}{2}\right)(1-p_c) + m\left(H, t+\frac{1}{2}\right)p_c(1-\Pr(D)) \\ &= m\left(H, t+\frac{1}{2}\right)(1-p_c\Pr(D)) \end{aligned} \qquad (۷-۳۱)$$

که در آن $m(H,t+\frac{1}{2})$ در معادله‌ی (۷–۱۷) نشان داده شده است. معادله‌ی (۷–۳۱) مقداری تقریبی از تعداد نمونه‌های شِما پس از برش به دست می‌دهد.

۷–۶–۳ جهش و نتایج نهایی

حال احتمال نابودی شِما توسط جهش را در نظر می‌گیریم. فرض کنید احتمال وقوع جهش در هر گره برابر p_m است. آنگاه احتمال عدم وقوع جهش در هر گره برابر $1-p_m$ خواهد بود. در این صورت احتمال عدم وقوع جهش در هیچ یک از گره‌های تعریف شده برابرست با $(1-p_m)^{o(H)}$، که در آن $o(H)$ مرتبه‌ی شِما (تعداد گره‌های تعریف شده) است. بنابراین احتمال وقوع جهش در یکی از گره‌های تعریف شده برابرست با

$$\Pr(D_m) = 1-(1-p_m)^{o(H)} \approx p_m o(H) \qquad (۷–۳۲)$$

که در آن تقریب استفاده شده از بسط تیلور $\Pr(D_m)$ حول نقطه‌ی $p_m = 0$ حاصل شده است. توجه داشته باشید که وقوع جهش در یک گره‌ی تعریف شده شرط لازم ولی غیرکافی نابودی شِما است. با ترکیب معادله‌ی (۷–۳۱) و (۷–۳۲) خواهیم داشت

$$m(H,t+1) \geq m\left(H,t+\frac{1}{2}\right)(1-p_c\Pr(D))(1-p_m o(H)) \qquad (۷–۳۳)$$

حال معادله‌ی بالا را با معادلات (۷–۱۷) و (۷–۳۰) ترکیب می‌نماییم

$$\begin{aligned} m(H,t+1) \geq{}& \frac{m(H,t)f(H,t)}{f_{ave}(t)}[1-p_m o(H))] \times \\ &\{1-p_c[\Pr(D|p_2 \notin G)\left(1-\frac{m(G,t)f(G,t)}{Nf_{ave}(t)}\right)+ \\ &\frac{L(H)}{n(H)-1}\frac{m(G,t)f(G,t)-m(H,t)f(H,t)}{Nf_{ave}(t)}]\} \end{aligned} \qquad (۷–۳۴)$$

معادله‌ی بالا کران بالای تعداد نمونه‌های شِمای H در نسل $t+1$اُم را به دست می‌دهد، نسلی که در آن برش و جهش در نظر گرفته شده‌اند. همان‌طور که انتظار می‌رفت، این کمی پیچیده‌تر از نظریه شِمای GA است که در بخش ٤–۱ استنتاج کردیم. GP به دلیل اندازه‌ی متغیرها و همچنین شکل ذراتش پیچیده‌تر از GA هاست. با این حال، می‌توان در معادله‌ی (۷–۳٤) ساده‌سازی‌هایی انجام داد. برای مثال، در اجرای GP که پیش از این انجام شد گوناگونی بسیاری در جمعیت دیده می‌شد، بنابراین بسیار غیرمحتمل بود دو ذره با شکل‌های متفاوت در نقطه‌ای با هم برش انجام دهند که باعث حفظ شِما شود. بنابراین، می‌توان گفت:

$\Pr(D|g \notin G) \approx 1$. همچنین گوناگونی زیاد باعث می‌شود برازندگی کل ذرات متعلق به یک ساختار G بسیار کوچک باشد چرا که تعداد کمی از برنامه‌های این چنین وجود خواهد داشت و این یعنی: $m(G,t)f(G,t)/(Nf_{ave}) \ll 1$. بنابراین معادله‌ی (۷-۳٤) را می‌توان به‌صورت زیر تقریب زد

$$m(H,t+1) \geq \frac{m(H,t)f(H,t)}{f_{ave}(t)}[1-p_m o(H))] \times \{1-p_c\left[1+\frac{L(H)}{n(H)-1}\left(\frac{-m(H,t)f(H,t)}{Nf_{ave}(t)}\right)\right]\} \quad (۳۵-۷)$$

همچنین می‌توانیم با فرض کوتاه بودن شِما و در نتیجه $L(H)/(n(H)-1) \ll 1$ تقریب بیشتری را به کار برد

$$m(H,t+1) \geq \frac{m(H,t)f(H,t)}{f_{ave}(t)}[1-p_m o(H))](1-p_c) \quad (۳٦-۷)$$

این تقریب شِمای GP که برای شِمای کوتاه معتبر است مانند عبارتِ شِمای GA در معادله‌ی (٤-٦) است. نظریه شِمایی که ما در این بخش استنتاج نمودیم به جای یک تساوی، حد پایینی برای عبارت $m(H,t+1)$ به دست می‌دهد. این موضوع به دلیل تقریب‌های به کار رفته است. به همین دلیل این نظریه شِما بدبینانه خوانده می‌شود.

نظریه شِما را به روش‌های مختلفی می‌توان ارائه نمود. دیگر راه‌های موجود برای مدل‌سازی فرایندهای انتخاب، برش و جهش GP به نظریه‌های شِمایی منجر می‌شوند که متفاوت از نظریه استنتاج شده در این بخش خواهند بود. برخی از این نظریه‌ها به جای اینکه بدبینانه باشند دقیق بوده و به همین دلیل به جای یک نامساوی یک تساوی و یک معادله را به دست می‌دهند [آلتِنبرگ[1]، ۱۹۹٤]، [لانگدن و پولی، ۲۰۰۲، فصل ۳ و ۵]. اونا- می‌اوریلی[2] نظریه شِمای حد پایینی را بر پایه‌ی کار کوزا ایجاد نمود که شماتا را به‌گونه‌ای متفاوت از این فصل تعریف می‌نماید [کوزا، ۱۹۹۲]، [اوریلی و اوپچر[3]، ۱۹۹۵]. ژوستینین روسکا[4] نیز یک نظریه شِمای GP را با تعریفی دیگر از شِما به وجود آورد [روسکا، ۱۹۹۷]. همچنین برای سیستم‌های GP که از نمایندگی برنامه به جای درخت‌های نحو استفاده می‌کنند نیز نظریه‌های شِمایی به وجود آمده است. [ویگهام[5]، ۱۹۹۵].

[1] Altenberg
[2] Una-May O'Reilly
[3] Oppacher
[4] Justinian Rosca
[5] Wigham

علاوه بر نظریه شِما، روش‌های دیگری نیز برای تحلیل ریاضی *GP* وجود دارد. برای مثال، مدل‌های مارکوف برای *GP* در سال ۲۰۰۱ معرفی شدند [چ.لی و همکاران، ۲۰۰۱]، [چولی و همکاران، ۲۰۰٤]. همچنین، می‌توان از انتخاب هزینه‌ای و قضیه‌ی کوواریانس برای مدل‌سازی ریاضی *GP* استفاده نمود [لانگدن و پولی، ۲۰۰۲، فصل ۳].

۷-۷ نتیجه‌گیری

این فصل با استفاده از *Lisp* و درخت نحو به *GP* محدود گشت. علاوه بر این‌ها *GP* با ساختارها و زبان‌های برنامه‌نویسی دیگر نیز پیاده‌سازی شده است. برای مثال، برنامه‌ها را می‌توان به‌صورت توالی خطی از دستورات پیاده‌سازی نمود. این نوع برنامه‌نویسی فرمی از برنامه‌نویسی است که بیشترِ ما به آن عادت داریم. این روش *GP* خطی نام داشته [پولی و همکاران، ۲۰۰۸، فصل ۷] و به‌طور خاص مناسب برنامه‌نویسی اسمبلی است. رشد و نمود دادن برنامه‌ی کد اسمبلی با استفاده از درخت نحو برای یک سیستم جاسازی شده کار سختی است چرا که ابتدا باید یک کامپایلر درخت- به- اسمبلی ایجاد کنیم. با این حال، اگر برنامه‌ها را مستقیماً با استفاده از کد اسمبلی رشد دهیم، هزینه‌ی تکامل بسیار سرراست‌تر خواهد بود.

GP کارتزین راهی برای ارائه‌ی برنامه‌ها با مجموعه‌ای از ارائه‌ها می‌باشد. هر ارائه دارای عنصری است که عملیات آن ارائه را مشخص می‌کند و همچنین دارای ارائه‌هایی است که ورودی آن‌ها را مشخص می‌نماید [میلر و اسمیت، ۲۰۰٦]. *GP* گرافی راهی برای ارائه‌ی برنامه‌ها با گره و شاخه است [پولی و همکاران، ۲۰۰۸]. از ساختارهای بسیار دیگری نیز برای ارائه نمودن برنامه‌های کامپیوتری در *GP* استفاده شده است [بنژاف[1] و همکاران، ۱۹۹۸، فصل ۹].

استفاده از برنامه‌نویسی ژنتیک با گسترش دامنه‌ی آن به ورای برنامه‌نویسی کامپیوتر و رسیدن آن به رشد و تکامل جامع‌تر از الگوریتم‌ها و طراحی‌های مهندسی افزایش یافته است. جان کوزا لیستی از ۷٦ نتیجه‌ی به دست آمده از *GP* تهیه کرده است که با نتایج تولید شده توسط انسان رقابت می‌کنند [کوزا، ۲۰۱۰]. وی همچنین بیان می‌کند که نرخ تولید نتایج تولید شده توسط *GP* که یارای رقابت با نتایج تولید شده توسط انسان‌ها را دارد با توان محاسباتی متناسب است. این موضوع افزایشی دراماتیک در طراحی‌های مهندسی تولید شده توسط کامپیوترها و همچنین همکاری معنی‌دار میان انسان‌ها و کامپیوترها را نوید می‌دهد.

مسائلی وجود دارند که *GP* برای آن‌ها مناسب نیست. فضای جستجوی *GP* عبارتست از مجموعه‌ای از همه‌ی برنامه‌های کامپیوتری، در محدوده‌ی محدودیت‌های درخت نحو که توسط کاربر تعریف شده است.

[1] Banzhaf

این دامنه‌ی وسیع جستجو هم یک قوت و هم یک ضعف برای *GP* به حساب می‌آید. وسیع بودن این دامنه به *GP* این اجازه را می‌دهد که کاملتر و جامعتر از سایر الگوریتم‌های تکاملی به جستجوی بهینه بپردازد اما همچنین، این گستردگی به این معناست که معمولاً *GP* استفاده‌ی چندانی از اطلاعات در دسترس در مورد دامنه‌ی که توسط برنامه‌نویس فراهم شده است استفاده نمی‌کند. در کل، اگر از پیش ساختار راه‌حل را بدانیم، آنگاه یک الگوریتم تکاملی استاندارد می‌تواند بهتر از *GP* عمل نماید چرا که در این صورت جا دادن اطلاعات مخصوص مسئله در یک مسئله‌ی بهینه‌سازی پارامتری بسیار راحت‌تر از جا دادن آن در یک مسئله‌ی بهینه‌سازی برنامه‌ای است. با این حال، اگر آشکار ساختن ساختار راه‌حل کار دشواری باشد، آنگاه *GP* روشی مناسب می‌نماید. همچنین توجه داشته باشید که می‌توان جمعیت اولیه‌ی *GP* را با راه‌حل‌های نامزد خوب بذرافشانی کرد. آنگاه *GP* بر گذر زمان بر روی این راه‌حل‌ها بهبود ایجاد خواهدکرد. بنابراین، *GP* برای مسائلی که در آن‌ها بهبودهای گسسته در راه‌حل‌های موجود بسیار مطلوب است، بسیار مناسب می‌نماید [کوزا، ۲۰۱۰].

یک زمینه‌ی بسیار جالب برای کارهای آینده نسلی از برنامه‌های کامپیوتری است که برنامه‌های کامپیوتری را رشد می‌دهند که آن برنامه نیز خود برنامه‌های کامپیوتری را رشد می‌دهند. این را می‌توان یک فرا *GP*[1] خواند. یک *GP* می‌تواند برنامه‌های کامپیوتری را رشد دهد، اما چطور می‌توان یک *GP* خوب پیدا کرد؟ شاید یک فرا *GP* بتواند یک *GP* را رشد دهد. احتمال توان محاسباتی مورد نیاز برای یک فرا *GP* حداقل دو برابر توان محاسباتی مورد نیاز برای یک *GP* باشد. فرا *GP* اولین بار توسط یورگن اشمیدهوبر[2] در رساله‌ی سال ۱۹۸۷ وی ارائه شد [اشمیدهوبر، ۱۹۸۷]. فرا *GP*ها نوعی از فرایادگیری هستند (که یعنی یاد بگیریم چگونه یاد بگیریم) [اندرسون و اوتز، ۲۰۰۷]. فرا *GP* را می‌توان جستجویی برای جستجو در نظر گرفت و به همین علت در دسته‌ی قضیه‌ی *no-free-lunch* عمودی قرار خواهد گرفت [دمبسکی[3] و مارکز[4]، ۲۰۱۰].

توجه داشته باشید که می‌توان *GP* را با سایر الگوریتم‌های تکاملی ترکیب نمود. برای مثال می‌توان GP و EDAها را برای پیدا کردن توصیفی احتمالی از برنامه‌ی مؤثر با یکدیگر ترکیب نمود و بدین ترتیب جستجوی خود را به سمت برنامه‌های بهتر سوق دهیم. این کار اولین با با نام رشد برنامه‌ی گسسته‌ی احتمالی مطرح شد [سالوستوویکز[5] و اشمیدهویر، ۱۹۹۷]. در این الگوریتم، هر گره در درخت نحو دارای احتمالی برای برابر بودن با یک تابع یا ترمینال خاص دارد و این احتمالات با مقادیر برازندگی ذراتِ برنامه متناسب

[1] Meta-GP
[2] Jurgen Schmidhuber
[3] Dembski
[4] Marks
[5] Salustowicz

است. الگوریتم‌های تکاملی جدید در این نوشته مطرح شده‌اند و یک کار جالب می‌توان این باشد که تحقیق کرد که کدام یک از این الگوریتم‌های تکاملی جدید برای پیاده‌سازی به عنوان *GP* مناسب‌اند (فصل ۱۷ را ببینید).

در آخر آنکه باید توجه کنیم که اگر یک دانشجو بخواهد به‌صورت جدی در زمینه‌ی *GP* تحقیق و مطالعه کند، باید مهارت توابع خودبه‌خود تعریف‌شده (*ADF*) را به دست آورد. *ADF*ها زیرروتین‌هایی هستند که به‌صورت خودکار و پویا رشد می‌نمایند. با در نظر گرفتن این حقیقت که انسان برنامه‌نویس از زیرروتین‌ها استفاده می‌نماید، طبیعی است که فکر کنیم *GP* نیز باید زیرروتین‌هایی ایجاد کرده و از آن‌ها استفاده نماید. *ADF*ها می‌توانند هنگام اعمال یک *GP* به مسائل پیچیده به طرز قابل توجهی از تلاش محاسباتی آن بکاهند. *ADF*ها به طور مفصل در [کوزا، ۱۹۹٤، فصل ٤]، [کوزا و همکاران، ۱۹۹۹] و سایر کتاب‌های مرتبط با *GP* مورد بحث قرار گرفته‌اند.

خوانندگان می‌توانند برای مطالعه‌ی بیشتر به چندین کتاب عالی که به *GP* اختصاص یافته‌اند مراجعه نمایند. کتاب نوشته شده توسط ولفگانگ[1] بنژاف و همکاران مقدمه‌ای خواندنی را فراهم می‌سازد [بنژاف و همکاران، ۱۹۹۸]. کتاب‌های مختصر و مفید جان کوزا، که به راستی شایسته‌ی شهرت ستاره‌ای ایشان هستند، مراجعی استاندارد در این زمینه می‌باشند [کوزا، ۱۹۹۲]، [کوزا، ۱۹۹٤]، [کوزا و همکاران، ۱۹۹۹]، [کوزا و همکاران، ۲۰۰۵]. کتاب *راهنمای میدانی به برنامه‌نویسی ژنتیک* کتابی است که به‌صورت مجانی در دسترس بوده و مروری زیبا بر موضوع *GP* ارائه می‌دهد [پولی و همکاران، ۲۰۰۸]. تحلیل شِمای مفصل را نیز می‌توان در کتاب ویلیام لانگدن و ریکاردو پولی پیدا نمود [لانگدن و پولی، ۲۰۰۲].

[1] Wolfgang

مسائل نوشتاری

۷-۱ یک عبارت s و یک درخت نحو برای راه‌حل مثبت معادله‌ی درجه دو بنویسید: $(\sqrt{b^2-4ac}-b)/2a$). عمق درخت نحوتان چه قدر است؟ آیا درخت نحو شما کامل است؟

۷-۲ یک عبارت s بنویسید به طوری که اگر $x > 2$ بود ۸ را بازگردانده و در غیر این صورت ۹ را بازگرداند.

۷-۳ فرض کنید راه‌حل نامزد یک GP (f) را بر روی n ورودی مختلف $\{u_i\}$ ارزیابی می‌کنید. یک تابع برازندگی بنویسید به‌طوری که این تابع به عملکرد متوسط از f، دو برابرِ ضعیف‌ترین عملکرد آن وزن بدهد.

۷-٤ در $Lisp$ یک تابع روت توان دو محافظت شده تعریف کنید به طوری که اگر ورودی آن منفی بود صفر را برگرداند.

۷-۵ فرض کنید از روش رشد برای ضمانت تولید یک عبارت s اتفاقی با عمق بیشینه‌ی D_c استفاده کرده‌ایم. فرض کنید در هر گره احتمال انتخاب یک ترمینال یا یک تابع برابر ۵۰٪ است. در این صورت احتمال آنکه یک شاخه به ماکزیمم عمق ممکن برسد چه قدر خواهد بود؟

۷-٦ فرض کنید از روش رشد برای ضمانت تولید یک عبارت s اتفاقی با عمق بیشینه‌ی D_c استفاده کرده‌ایم. فرض کنید در هر گره احتمال انتخاب یک ترمینال یا یک تابع برابر ۵۰٪ است. فرض کنید هر تابع دو آرگومان می‌پذیرد. احتمال اینکه عبارت s نماینده‌ی یک درخت نحو کامل با عمق D_c باشد چه قدر است؟

۷-۷ لیستی از تمام برنامه‌هایی که می‌توانند از جهش جرثقیلی درخت نحو شکل ۷-۱ ایجاد شوند تهیه نمائید.

۷-۸ لیستی از برنامه‌های منحصر به فرد که می‌توانند از جهش جایگشتی درخت نحو شکل ۷-۲ ایجاد شوند تهیه نمائید.

۷-۹ معادله‌ی زیر را به‌صورت ساده شده بنویسید (برای مثال، $(+11)$ را می‌توان با ۲ جایگذاری نمود). پس از به دست آوردن نسخه‌ی ساده‌تر، آن را به روش معمولی‌تر و استانداردتری بنویسید.

$$(defun\ Pgm(x)\ (+\ (\ DIV\ (-\ (+\ (+\ 1\ 1)\ (+\ (DIV\ x\ 1)\ (+\ 1\ 1)\ )\ )\ (-\ (*\ x\ 1)\ (+\ 1\ 1)\ )\ )$$
$$(abs\ x)\ )\ (DIV(+\ (-\ x\ 1)\ (abs\ x)\ )\ (-\ (-\ (*\ x\ 1)\ (+\ 1\ 1)\ )\ (-\ (+\ 1\ 1)\ (*\ 1\ 1)\ )\ )\ )\ )\ )$$

۷-۱۰ طول تعریف، مرتبه و طول شِمای $(if\ (\#x\#)8\#)$ را بیابید. ساختار این شِما چیست؟

۷-۱۱ حد پایین تعداد نمونه‌های شِما با تغییر احتمال جهش چگونه تغییر می‌کند؟ با مرتبه‌ی شِما چگونه تغییر می‌کند؟ با احتمال برش چطور؟ جواب‌های خود را توضیح دهید.

مسائل کامپیوتری

۷-۱۲ تمارین *Lisp*:

الف) آخرین نسخه‌ی *CLISP (Common Lisp)* را دانلود کرده و نصب نمائید.

ب) محیط توسعه‌ی یکپارچه (*IDE*[1]) *CLISP* را نیز دانلود و نصب نمائید. نکته: مسئله‌ی کمترین زمان کنترل در بخش ۷-۳ توسط برنامه‌ی *LispIDE* پیاده‌سازی شده است.

ج) *Lisp IDE* را اجرا کرده و دستور زیر را در آن وارد کنید

$$(Print\ (*\ 5\ (+\ 3\ 2)))$$

با این کار *Lisp* عدد ۲۵ را دو بار در ترمینال چاپ می‌کند: یک بار به خاطر خود دستور *print* و یک بار نیز به دلیل آنکه مقدار ۲۵ از دستور *print* بازگردانده شده است.

۷-۱۳ تمرین کمترین زمان کنترل:

الف) فایل *GPcartcontrol.lisp* و سایر فایل‌های مربوطه را از سایت کتاب دانلود کرده و آن را بر روی کامپیوتر خود اجرا نمائید. پس از اجرا، نتایج ارائه شده برای مسئله‌ی کمترین زمان کنترل در بخش ۷-۳ نمایش داده می‌شوند. اگر از *LispIDE* استفاده می‌نمائید، برای انجام این کار مراحل زیر را انجام دهید:

- *LispIDE* را اجرا کنید.
- *GPcartcontrol.lisp* را از *LispIDE* اجرا نمائید.
- خط ۱۵ از *GPcartcontrol.lisp* را به‌گونه‌ای اصلاح نمائید که مسیر به دایرکتوری اشاره نماید که فایل‌های *Lisp* در کامپیوتر شما در آن قرار دارند.
- کل *GPcartcontrol.lisp* را با ماوس کامپیوتر خود انتخاب نمائید.
- گزینه‌ی *Send to Lisp* را از منوی *Edit* انتخاب نمائید. این کار تابع *GPcartcontrol* را در *Lisp* تعریف می‌نماید.
- عبارت *GPcartcontrol* را در خط فرمان *LispIDE* تایپ نمائید. این کار باعث اجرای برنامه خواهد شد.

ب) پس از اتمام اجرای *[GPcartcontro]l.lisp*، دو فایل خروجی خواهید داشت. یکی از آن‌ها *DateTimeString.txt* است که شامل شماره‌ی نسل، بهترین هزینه و هزینه‌ی میانگین است. فایل دوم فایل

[1] Integrated Development Environment

[DateTimeString].lisp است که حاوی برنامه‌ای است که توسط *GP* پیدا شده است (توجه داشته باشید که *[DateTimeString]* نوشته‌ای است که تاریخ و زمان ایجاد فایل را نشان می‌دهد).

ج) حال دستور *setf LispPgm [BestProgram]* را اجرا کنید (*[BestProgram]* نوشته‌اس است که بهترین برنامه‌ی پیدا شده توسط *GP* را نشان می‌دهد). شما باید این نوشته را از *[DateTimeString.lisp]* بگیرید. برای مثال، *(setf Pmg "(defun CartControl (x v)(if (> (* -1 x) (* v (abs v))) 1 (-1)))")*

د) تابع *(PhasePlane LispPgm)* را اجرا نمائید. این کار دو فایل ایجاد می‌کند. یکی از آن‌ها *PhasePlane.txt* است که شامل تعدادی مقادیر *(x, v, control)* است که توسط *LispPgm* ایجاد شده است. فایل دیگر *PhasePlane1.txt* است که لیستی است از مقادیر *(x, v)* که کنترل میان *1+* و *1-* جابه‌جا می‌کند. این موضوع بر پایه‌ی این فرض است که کنترل ایجاد شده توسط *LispPgm* همیشه اشباع است. اگر این فرض صحیح نباشد، *PhasePlane1.txt* بلااستفاده است.

ه) فایل *MATLAB* با نام *PlotPhasePlane.m* را با ورودی *"Phase-Plane"* اجرا کنید. با این کار نمودار صفحه‌ی فازی از کنترل با استفاده از *PlanePhase.txt* و *PlanePhase1.tct* به‌صورت تابعی از *x* و *v* ایجاد می‌شود. با این حال، این نمودار در صورتی قابل استفاده است که فرض بیان شده در بند قبلی صحیح باشد، یعنی کنترل تولید شده توسط *LispPgm* همواره اشباع باشد.

ز) برازندگی یک برنامه‌ی کنترل گاری *Lisp* را می‌توان با اجرای *EvalCartControl.lisp* محاسبه کرد. اگر شما *CartControl* را مانند آنچه در زیرمسئله‌ی ج در بالا توضیح داده شد تعریف کنید، آنگاه می‌توانید *EvalCartControl.lisp* را باز کرده، در *Lisp IDE* اررزیابی کنید تا *EvalCartControl* یک تابع تعریف شده باشد. سپس دستور زیر را بنویسید و اجرا کنید: *(EvalCartcontrol #' CartControl)*.

۷-۱٤ برخی پارامترهای موجود در *GPCartControl.lisp* را اصلاح کنید و ببینید این تغییرات چه تأثیری بر روی عملکرد دارند. برخی پارامترهایی که شما می‌توانید آن‌ها را اصلاح کنید عبارتند از:

- *Dinitial*: بیشینه‌ی عمق درخت اولیه.
- *Dcreated*: بیشینه عمق درخت.
- *Pcross*: احتمال برش.
- *Preproduce*: احتمال بازتولید.
- *Pinternal*: احتمال وقوع برش در گره‌های درونی.
- *NumEvals*: تعداد ارزیابی‌های تابع در ذره.
- *NumElites*: تعداد نخبه‌های هر نسل.

- *GenLimit*: محدوده‌ی نسل.
- *PopSize*: اندازه‌ی جمعیت.
- *SelectionMethod*: روش انتخاب.

۷-۱۵ فایل GPCartcontrol.lisp و سایر فایل‌های مرتبط را به‌گونه‌ای اصلاح کنید که GP بتواند نگاشت $\hat{y}(x)$ را به‌گونه‌ای بیابد که به‌صورت نزدیکی با $y(x)$ منطبق باشد. مقادیر $y(x)$ به شرح زیر است:

$$y(0) = 3, \quad y(1) = 5, \quad y(2) = 1, \quad y(3) = 2, \quad y(4) = 9$$
$$y(5) = 8, \quad y(6) = 3, \quad y(7) = 4, \quad y(8) = 1, \quad y(9) = 6$$

یک نمودار همگرایی GP را که پیشرفت کمترین و میانگین هزینه‌ی جمعیت را به‌عنوان تابعی از شماره‌ی نسل نشان می‌دهد، به دست دهید. همچنین بهترین برنامه‌ی پیدا شده توسط GP و همچنین نموداری مقایسه‌ای میان مقادیر دقیق $y(x)$ و مقادیر تقریبی به دست آمده از GP، $\hat{y}(x)$، را ارائه دهید.

فصل هشتم

مطالب گوناگون مرتبط با الگوریتم‌های تکاملی

گزینه‌های زیادی وجود دارد.

دیوید فوگل [فوگل، ۲۰۰۰]

فصل‌های پیشین چهار رویکرد معروف و بنیادین در زمینه‌ی محاسبات تکاملی را مورد بحث قرار دادند. با این حال، فصل‌های پیشین تنها به ارائه‌ی الگوریتم‌ها و ایده‌های اساسی و بنیادین پرداختند. تنوعات بسیاری برای پیاده‌سازی در این الگوریتم‌ها وجود دارند. این تنوعات نه تنها برای الگوریتم‌های تکاملی که پیش از این در این کتاب مورد بحث قرار گرفته‌اند، بلکه برای آن‌هایی که پس از این در این کتاب در موردشان سخن خواهد رفت نیز کاربرد دارند. بنابراین، این فصل به گستره‌ی وسیعی از الگوریتم‌های تکاملی قابل اعمال می‌باشد. برخی از تنوعاتی که در این بخش مورد بحث قرار خواهیم داد می‌توانند تفاوتی بزرگ در عملکرد الگوریتم تکاملی ایجاد کنند. معمولاً آنچه که باعث ایجاد تفاوت در عملکرد دو الگوریتم می‌شود، جزئیات پیاده‌سازی هستند تا تفاوت‌های عمده‌ی میان دو الگوریتم.

مروری بر فصل

بخش ۸–۱ به بحث در مورد راه‌های مختلف جهت مقداردهی اولیه‌ی جمعیت الگوریتم تکاملی می‌پردازد. در بخش ۸–۲ در مورد نحوه و روش‌های پایان‌دهی یک الگوریتم تکاملی صحبت خواهد شد. بخش ۸–۳ به بحث در مورد چگونگی ارائه‌ی راه‌حل‌های نامزد الگوریتم تکاملی خواهد پرداخت. این بخش همچنین نشان خواهد داد که چگونه نحوه‌ی ارائه‌ی راه‌حل‌ها تأثیری قابل توجهی بر نتایج می‌گذارد. بخش ۸–٤ در مورد نخبه‌گرایی سخن خواهد راند. نخبه‌گرایی در ابتدا برای GA ارائه شد، اما امروزه به دلیل فواید ذاتی آن در سایر الگوریتم‌های تکاملی نیز پیاده‌سازی می‌شود. بخش ۸–۵ تفاوت‌های میان الگوریتم‌های تکاملی نسلی و الگوریتم‌های تکاملی حالت– ماندگار را بیان خواهد نمود.

جمعیت‌های الگوریتم‌های تکاملی تمایل دارند به یک تک– ذره با برازندگی بسیار زیاد همگرا شوند، به بیان دیگر کل جمعیت به کلونی‌هایی از یک راه‌حل نامزد تبدیل می‌شود. این موضوع توانایی الگوریتم تکاملی برای جستجوی یک راه‌حل بهینه را شدیداً کاهش می‌دهد. از این رو است که بخش ۸–٦ به توضیح چگونگی متنوع نگه داشتن جمعیت الگوریتم تکاملی می‌پردازد.

تا به اینجا بیشتر بر انتخاب چرخ رولت برای انتخاب والدین تمرکز کرده‌ایم. اما روش‌های دیگری نیز برای عمل گزینش وجود دارند که برخی از آن‌ها در بخش ۸–۷ مورد بحث قرار خواهند گرفت. یکی از این روش‌ها روش *stud* (زیربخش ۸–۷–۷) است که در اصل برای الگوریتم‌های ژنتیک ارائه شد اما می‌توان آن را در مورد سایر الگوریتم‌های تکاملی نیز به کار برد. بخش ۸–۸ به بحث در مورد روش‌های مختلف موجود برای ترکیب والدین و بخش ۸–۹ نیز به بحث در مورد روش‌های مختلف برای پیاده‌سازی جهش می‌پردازند.

۸–۱ مقداردهی اولیه

ما معمولاً یک الگوریتم تکاملی را با یک جمعیت اتفاقی مقداردهی اولیه می‌کنیم. این روش، ساده‌ترین و همچنین محبوب‌ترین روش برای مقداردهی اولیه است. با این حال، مقداردهی اولیه می‌تواند تفاوتی عظیم در موفقیت الگوریتم تکاملی ایجاد کند. به همین جهت کمی تلاش بر روی این موضوع خالی از لطف نیست.

فرض کنید می‌خواهیم یک الگوریتم تکاملی با N ذره را به اجرا بگذاریم. یک روش آن است که در ابتدا بیشتر از N ذره تولید کنیم و سپس به راحتی بهترین N ذره را به‌عنوان جمعیت اولیه نگه داریم. برای مثال، [بِهاتاچاریا[1]، ۲۰۰۸] $5N$ ذره تولید کرده و بهترین N ذره را به‌عنوان جمعیت اولیه نگه می‌دارد.

ما همچنین می‌توانیم در ابتدا ذرات را به صورت تصادفی تولید کرده و سپس هر ذره را به صورت محلی بهینه نماییم تا به جمعیت اولیه‌ی الگوریتم تکاملی برسیم. برای مثال، می‌توان N ذره‌ی اتفاقی تولید نمود، سپس از بهینه‌سازی نزول گرادیانی بر روی یک زیرمجموعه از آن‌ها استفاده کرد و در نهایت از ذرات به دست آمده به‌عنوان جمعیت اولیه‌ی الگوریتم تکاملی استفاده نمود. این روش را می‌توان به چندین روش پیاده‌سازی نمود. برای مثال، می‌توانیم تنها از بهترین ذرات در جمعیت اولیه استفاده کنیم. همچنین می‌توان بهینه‌سازی نزول گرادیانی را تنها در مورد بهترین ذرات به کار برد اما در نهایت از تمامی ذرات در جمعیت اولیه استفاده نماییم.

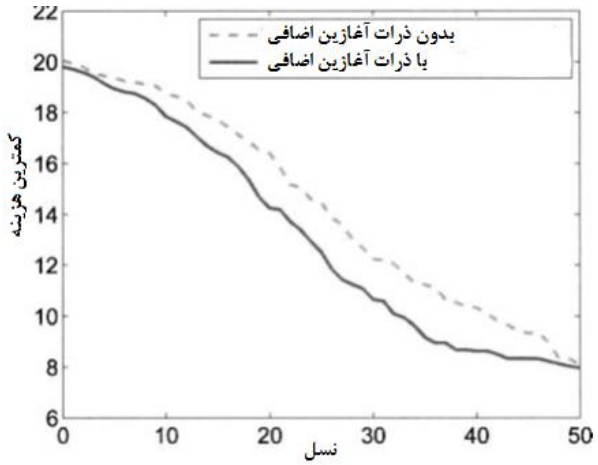
یک گزینه دیگر استفاده از راه‌حل‌های تخصصی برای مقداردهی اولیه‌ی جمعیت الگوریتم تکاملی است. برای مثال، فرض کنید می‌خواهیم از یک الگوریتم تکاملی برای میزان‌سازی یک الگوریتم کنترل استفاده نماییم. برای این کار می‌توانیم از دانش تخصصی برای تخمین راه‌حل‌های کنترل منطقی استفاده نماییم و جمعیت اولیه‌ی الگوریتم تکاملی را با این راه‌حل‌ها بذرافشانی نماییم. یا اینکه می‌توان از راه‌حل‌های نامزد به دست آمده از هر منبع دیگری (الگوریتم‌های دیگر، نتایج منتشر شده‌ی دیگر و غیره) برای بذرافشانی اولیه‌ی جمعیت الگوریتم تکاملی استفاده نمود. استفاده از اطلاعات خاصه‌ی مسئله برای بذرافشانی اولیه‌ی جمعیت الگوریتم تکاملی غالباً مقداردهی جهت‌یافته نامیده می‌شود.

مثال ۸–۱

این مثال به تأثیر ذرات اولیه‌ی اضافی بر روی یک EP برای بهینه‌سازی یک تابع روزنبروکِ ۱۰ بعدی (ضمیمه‌ی ج.٤–۱ را ببینید) می‌پردازد. ما از اندازه‌ی جمعیتی برابر ۱۰ استفاده کرده و EP را برای ۵۰ نسل

[1] Bhattacharia

اجرا می‌نماییم. اگر بخواهیم از پیاده‌سازی استاندارد *EP* استفاده کنیم، باید ۱۰ ذره‌ی اولیه را به‌صورت اتفاقی تولید نماییم. اما در پیاده‌سازی حاضر از ذرات ابتدایی اضافی استفاده می‌کنیم. برای این کار در ابتدا ۲۰ ذره‌ی اتفاقی تولید کرده و سپس از بهترین ۱۰ ذره به‌عنوان جمعیت اولیه‌ی *EP* استفاده خواهیم نمود. شکل ۸–۱ نتایج به دست آمده از دو الگوریتم را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است را نشان می‌دهد. تلاش محاسباتی مورد نیاز برای مقداردهی اولیه با ذرات اضافی در طول مرحله‌ی مقداردهی اولیه تقریباً دو برابر مقداردهی استاندارد است، اما این تلاش اضافی در عمل نتایج بسیار بهتری را به همراه خواهد داشت. این شکل نشان می‌دهد که اگر از مقداردهی اولیه با ذرات اضافی استفاده نماییم، در طول ٤٠ نسل اول نتایج بسیار بهتری را به دست خواهیم آورد، هر چند عملکرد دو الگوریتم از نسل ۵۰ به بعد تقریباً شبیه یکدیگر است.



شکل ۸–۱ مثال ۸–۱: هزینه در مقابل تعداد نسل برای یک *EP* که از آن برای بهینه‌سازی تابع روزنبروک ۱۰ بعدی استفاده شده است. نتایج بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده‌اند. به نظر می‌آید، حداقل در ٤٠ نسل ابتدایی، تولید ۱۰ ذره‌ی اولیه‌ی اضافی به زحمت آن بیارزد.

استفاده از مقداردهی اولیه‌ی جهت‌یافته یا ذرات اولیه‌ی اضافی روشی است که در مورد مسائل با تعداد نسل‌های کم و تعداد ذرات کم، منطقی است. به بیان دیگر این روش برای مسائلی مناسب است که به دلیل هرینه‌ی زیادشان، به تعداد ارزیابی‌های برازندگیِ کم محدود هستند (بخش ۲۱–۱ را ببینید).

۸–۲ معیار همگرایی

یکی از مواردی که هنگام پیاده‌سازی یک الگوریتم تکاملی باید در مورد آن تصمیم گرفت، زمان توقف آن است. ما این کار را در الگوریتم‌های فصل‌های پیشین با استفاده از عبارت عمومیِ "تا زمانی که شرایط توقف برآورده نشده است"، انجام داده‌ایم (برای مثال، شکل‌های ۳–۶، ۵–۱، ۶–۱ و ۷–۵ را ببینید). از چه معیار پایان‌دهی باید استفاده نمود؟ یک الگوریتم تکاملی تا چه مدت باید اجرا شود؟ چندین معیار وجود دارند که می‌توان از آن‌ها برای تعریف همگرایی استفاده نمود.

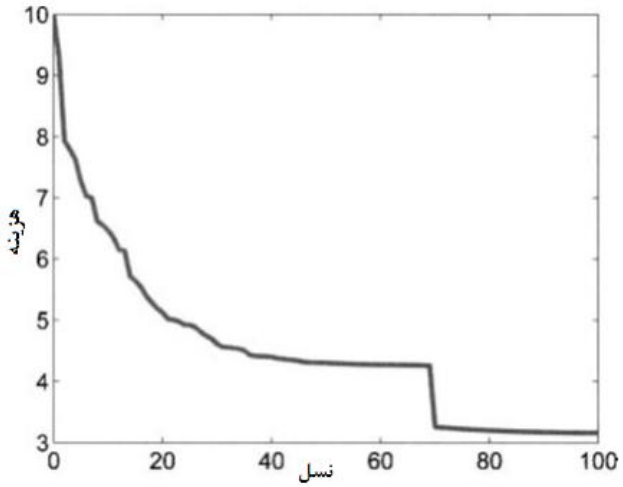
۱. می‌توان الگوریتم تکاملی را پس از تعداد معینی از نسل‌ها متوقف نمود. این روش ساده بوده، مدت زمان اجرا در آن قابل پیش‌بینی است و احتمالاً پرکاربردترین معیار پایان‌دهی است. با این حال، اگر بخواهیم چند الگوریتم تکاملی را با هم مقایسه نماییم، باید به جای متوقف نمودن آن‌ها پس از تعداد معینی نسل، اجرای آن‌ها را پس از تعداد معینی ارزیابیِ تابع هدف متوقف نماییم. علت این موضوع آن است که تعداد ارزیابی تابع در نسل برای الگوریتم‌های تکاملی مختلف، متفاوت است. بنابراین برای مقایسه‌ی عادلانه میان الگوریتم‌های تکاملی مختلف باید آن‌ها را پس از حد معینی از ارزیابیِ تابع متوقف نمود.

۲. می‌توان اجرای الگوریتم‌های تکاملی را پس از آن که راه‌حل "به اندازه‌ی کافی خوب شد"، متوقف نمود. این معیار پایان‌دهی به مسئله بستگی دارد چرا که تعریف "به اندازه‌ی کافی خوب" از یک مسئله به مسئله‌ی دیگر متفاوت است. این یک معیار پایان‌دهی جذاب و خوشایند است. اگر راه‌حلی پیدا کنیم که عملکردی رضایت‌بخش داشته باشد، چرا به دنبال راه‌حل بهتر بگردیم؟ با این حال، بسیاری از راه‌حل‌های به دست آمده برای مسائل دنیای واقعی "به اندازه‌ی کافی خوب" نیستند. ما همواره در تلاش برای بهتر شدن هستیم. از سوی دیگر، در بسیاری از موارد دو برابر کردن زمان اجرا برای بهتر کردن عملکرد به اندازه‌ای ناچیز، به هدر دادن منابع است. تقابل میان زمان اجرا و عملکرد، موضوعی است که به مسئله وابسته است و نیاز به تصمیم‌گیری مهندسی دارد.

۳. می‌توان الگوریتم تکاملی را پس از آنکه برازندگی بهترین ذره پس از تعداد معینی نسل دچار تغییر چندانی نشد، متوقف نمود. این موضوع ممکن است ناشی از گیر افتادن الگوریتم تکاملی در یک مینیمم محلی باشد. هر چند که این مینیمم محلی ممکن است مینیمم جهانی نیز باشد، اما ما قادر به تشخیص این موضوع نخواهیم بود، مگر آنکه یک مینیمم محلی بهتر پیدا کنیم.

٤. می‌توان الگوریتم تکاملی را پس از آنکه میانگین مقادیر برازندگی جمعیت پس از تعداد معینی نسل دچار تغییر چندانی نشد، متوقف نمود. این معیار نیز مانند معیار بالا است چرا که این معیار نیز نشان می‌دهد بهبودِ جمعیت در کل متوقف شده است.
٥. می‌توان الگوریتم تکاملی را پس از آنکه کاهش انحراف استاندارد برازندگی‌های جمعیت متوقف شد و یا به زیر یک مقدار آستانه رسید، متوقف نمود. این یعنی جمعیت به سطحی مشخص از یکنواختی رسیده است.
٦. می‌توان از ترکیبی از معیارهای بالا استفاده نمود.

موارد ۳–۵ نشان می‌دهند بهبود جمعیت متوقف شده است (هرچند این موضوع را تضمین نمی‌کنند)، پس شاید ما بخواهیم الگوریتم تکاملی را متوقف نماییم. با این حال، در این رویکرد خطری وجود دارد. حتی وقتی که به نظر برسد الگوریتم تکاملی همگرا شده است، یک جهش و یا یک بازترکیب غیرمحتمل می‌تواند بهبودی عظیم در راه‌حل ایجاد نماید. این موضوع در شکل ۸–۲ نشان داده شده است. از سوی دیگر، نمی‌توان الگوریتم تکاملی را تا ابد اجرا نمود. اما صرف نظر از این که چه موقع اجرا متوقف می‌شود، ما ریسک این که اجرا را درست قبل از وقوع یک جهش و یک بازترکیب مساعد متوقف نماییم، می‌پذیریم.



شکل ۸–۲ نمودار هزینه در برابر شماره‌ی نسل برای یک الگوریتم تکاملی فرضی. اگر الگوریتم تکاملی را بعد از توقف روند بهبودی هزینه متوقف نماییم، بهبودی عظیم در نسل ۷۰ اُم، که ممکن است بر اثر یک جهش غیرمحتمل رخ داده باشد، را از دست خواهیم داد. با این حال، اگر مقدار هزینه‌ی ۵ مشتری را راضی کند، آنگاه دیگر بهبودی که در نسل ۷۰ رخ می‌دهد برایمان مهم نخواهد بود.

۸–۳ نمایش مسئله با استفاده از کدگذاری Gray

این بخش به بحث در مورد پیاده‌سازی دودویی الگوریتم‌های تکاملی با استفاده از کدگذاری *Gray* می‌پردازد. یک کدِ *gray*، که گاهاً کد دودویی بازتابیده نیز نامیده می‌شود، روشی است برای نشان دادن اعداد در مبنای دودویی به‌طوری که اعدادِ همسایه فقط در یک بیت با هم تفاوت داشته باشند [دوران[1]، ۲۰۰۷]. نمایش اعداد ۰– ۷ در مبنای دودویی را در نظر بگیرید:

$$\begin{aligned} 000 &= 0, \quad 001 = 1 \\ 010 &= 2, \quad 011 = 3 \\ 100 &= 4, \quad 101 = 5 \\ 110 &= 6, \quad 111 = 7 \end{aligned} \tag{۱–۸}$$

می‌توان دید که کدهای دودویی برای اعداد همسایه می‌توانند در بیشتر از یک بیت با هم تفاوت داشته باشند. برای مثال، کد دودویی برای عدد ۳ برابر ۰۱۱ بوده و کد دودویی برای عدد ٤ برابر ۱۰۰ می‌باشد. این دو کد دودویی در هر سه بیتشان با هم فرق می‌کنند. بر عکس این حالت هم صادق است، یعنی برخی از کدهای دودویی مشابه نماینده‌ی اعدادِ بسیار دور از همی هستند. برای مثال، ۰۰۰ عدد ۰ را نشان می‌دهد، حال اگر تنها بیت سوم را از ۰ به ۱ تغییر دهیم، عدد ٤ حاصل می‌شود، که با توجه به برد اعدادی که در صدد ارائه‌شان هستیم، بسیار دور از ۰ می‌نماید.

حال نمایش اعداد ۰– ۷ را به روش کدگذاری *gray* در نظر بگیرید.

$$\begin{aligned} 000 &= 0, \quad 001 = 1 \\ 011 &= 2, \quad 010 = 3 \\ 110 &= 4, \quad 111 = 5 \\ 101 &= 6, \quad 100 = 7 \end{aligned} \tag{۲–۸}$$

می‌توان دید که کدهای *gray* برای اعداد همسایه همیشه دقیقاً در یک بیت با هم تفاوت دارند. برای مثال، کدِ *gray* برای عدد ۳ برابر ۰۱۰ بوده و کدِ *gray* برای عدد ٤ برابر ۱۱۰ است. می‌توان دید که این دو کد تنها در بیت سوم با هم تفاوت دارند.

مثال ۸–۲

تابع $y = f(x)$ از مثال ۲–۲ را در نظر بگیرید. اگر مقادیر $x \in [-4, +1]$ از ۱٦ مقدار با فضای مساوی از محور x در شکل ۲–۱ را با چهار بیت کدگذاری دودویی نماییم، خواهیم داشت

[1] Doran

کد دودویی:

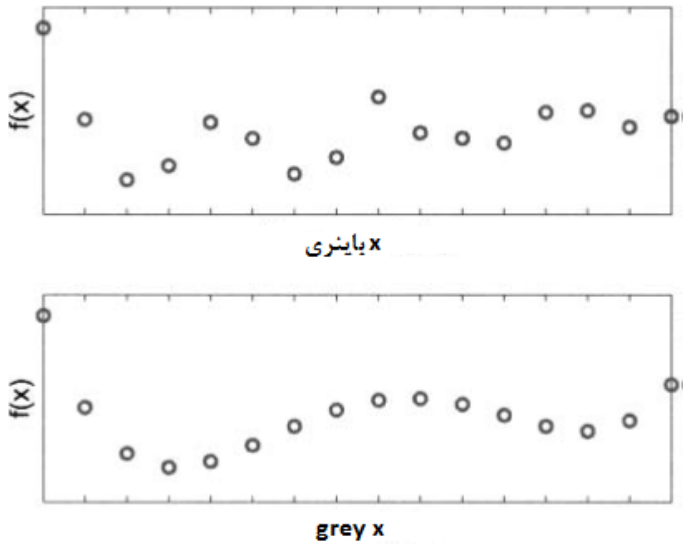
$$\begin{aligned} 0000 &= -4.00, \quad 0001 = -3.67 \\ 0010 &= -3.33, \quad 0011 = -3.00 \\ 0100 &= -2.67, \quad 0101 = -2.33 \\ 0110 &= -2.00, \quad 0111 = -1.67 \\ 1000 &= -1.33, \quad 1001 = -1.00 \\ 1010 &= -0.67, \quad 1011 = -0.33 \\ 1100 &= +0.00, \quad 1101 = +0.33 \\ 1110 &= +0.67, \quad 1111 = +1.00 \end{aligned} \tag{۸–۳}$$

از سوی دیگر می‌توان این مقادیر را با روش *gray* کدگذاری نمود. در این صورت خواهیم داشت

کد دودویی:

$$\begin{aligned} 0000 &= -4.00, \quad 0001 = -3.67 \\ 0011 &= -3.33, \quad 0010 = -3.00 \\ 0110 &= -2.67, \quad 0111 = -2.33 \\ 0101 &= -2.00, \quad 0100 = -1.67 \\ 1100 &= -1.33, \quad 1101 = -1.00 \\ 1111 &= -0.67, \quad 1110 = -0.33 \\ 1010 &= +0.00, \quad 1011 = +0.33 \\ 1001 &= +0.67, \quad 1000 = +1.00 \end{aligned} \tag{۸–٤}$$

اگر y را بر حسب x رسم کنیم، به‌طوری که مقادیر همسایه از x در کد دودویی‌شان دقیقاً در یک بیت با هم تفاوت داشته باشند، قسمت بالایی شکل ۸–۳ حاصل خواهد شد. اما اگر y را به‌گونه‌ای بر حسب x رسم کنیم که مقادیر همسایه‌ی x در کد *gray* تنها در یک بیت با هم متفاوت باشند، نمودار پایینی به دست خواهد آمد. می‌توان دید که با کدگذاری *gray*، نمودار شکل اصلی خود را نگه می‌دارد (با شکل ۲–۱ مقایسه کنید). این موضوع، بهینه‌سازی توابعِ صاف و ملایم را آسان می‌کند چرا که تغییرات کوچک در کدها، تغییرات کوچک در مقادیر تابع را نتیجه می‌دهد.



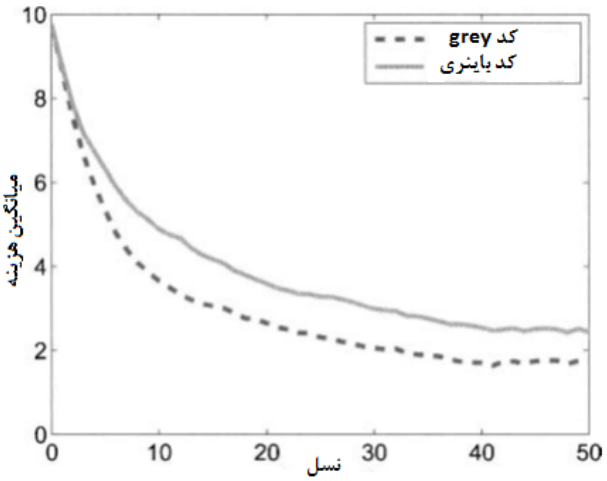
شکل ۸–۳ مثال ۸–۲: در نمودار بالایی محور *x* به‌گونه‌ای تنظیم شده است که مقادیری از *x* در کنار هم قرار گرفته باشند که در کد دودویی‌شان با هم همسایه هستند. نمودار بالایی نیز مشابه نمودار پایینی است با این تفاوت که در نمودار پایینی از کد *gray* برای مقادیر *x* استفاده شده است. اگر از کدگذای دودویی استفاده کنیم، توابع صاف، صاف بودن خود را از دست می‌دهند.

مثال ۸–۳

در این مثال تأثیر کدگذاری دودویی بر *GA* را در مقابل تأثیر کدگذاری *gray* می‌آزماییم. ما از *GA* برای بهینه‌سازی تابع دو بعدی اَکلی که در مثال ۳–۳ توضیح داده شده است استفاده می‌نماییم. هر بعد در این مثال با ٦ بیت کدگذاری شده است. ما از اندازه‌ی جمعیتی برابر ۲۰ و نرخ جهشی برابر با ۲٪ در بیت در نسل، استفاده می‌نماییم. شکل ۸–٤ میانگین هزینه‌ی ۲۰ ذره‌ی *GA* در هر نسل، که بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، را نشان می‌دهد. به دلیل سطح صاف و منظم تابع اَکلی، کدگذاری *gray* عملکرد به مراتب بهتری از کدگذاری دودویی دارد (شکل ۲–۵ را ببینید).

اگرچه به نظر می‌رسد کدگذاری *gray* در بیشتر موارد عملی بهتر از کدگذاری دودویی عمل می‌نماید، اما می‌توان ثابت نمود کدگذاری دودویی در مسائل بدترین– حالت عملکرد بهتری دارد [وایتلی، ۱۹۹۹]. یک مسئله‌ی بدترین– حالت مسئله‌ای گسسته است که در آن نیمی از نقاط فضای جستجو مینیمم محلی هستند. در آخر متذکر می‌شویم که علاوه بر کدگذاری دودویی و *gray*، می‌توان از هر نوع نمایش دیگری در الگوریتم‌های تکاملی استفاده نمود. نحوه‌ی نمایش می‌تواند تأثیر قابل توجهی بر عملکرد الگوریتم تکاملی داشته باشد. به همین دلیل، اگرچه نحوه‌ی نمایش با جزئیات بیشتری در این کتاب مورد بحث قرار نمی‌گیرید،

اما نباید در کاربردهای الگوریتم‌های تکاملی نادیده گرفته شوند. مطالعه‌ی نحوه‌ی نمایش می‌تواند کاملاً پیچیده باشد. برای مطالعه‌ی بیشتر در این زمینه خوانندگان می‌توانند به مراجعی چون [چویی[1] و مون[2]، ۲۰۰۳] و [روثلاوف[3] و گلدبِرگ، ۲۰۰۳] مراجعه نمایند.



شکل ۸-٤ مثال ۸-۳: نتایج مثال ۸-۳ برای مینیمم‌سازی تابع دو بعدی آکلی که در آن هر بعد با شش بیت کدگذاری شده است. نمودار، میانگین هزینه‌ی همه‌ی ذرات *GA* در هر نسل را که بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. یک *GA* با کدگذاری *gray* بسیار بهتر از *GA* با کدگذاری دودویی عمل می‌نماید.

مثال ۸-٤

فرض کنید مسئله‌ی بدترین- حالتی در اختیار داریم که در آن مقادیر زوج کدهای دودویی دارای هزینه‌ی ۱ بوده و مقادیر فرد از این کدگذاری دارای هزینه‌ی ۲ هستند. اگر ذرات را با کدگذاری دودویی ارائه کنیم، مقادیر هزینه برای یک مسئله‌ی بدترین- حالتِ سه بیتی به قرار زیر خواهد بود:

$$\begin{aligned} f(000) &= 1, \quad f(001) = 2 \\ f(010) &= 1, \quad f(011) = 2 \\ f(100) &= 1, \quad f(101) = 2 \\ f(110) &= 1, \quad f(111) = 2 \end{aligned} \qquad (۵-۸)$$

اگر ذرات را با کدگذاری *gray* نشان دهیم، مقادیر هزینه برای مسئله‌ی بدترین- حالتِ سه بیتی، به همان ترتیبی که برای کدگذاری دودیی نوشته شد، به قرار زیر خواهد بود:

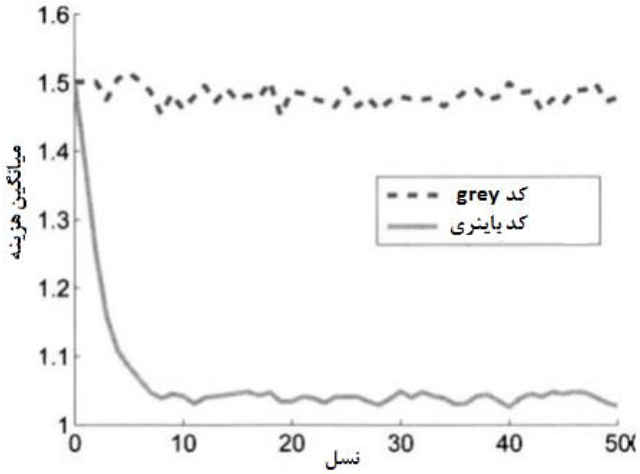
[1] Choi
[2] Moon
[3] Ruthlauf

$$\begin{aligned} f(000) &= 1, \quad f(001) = 2 \\ f(011) &= 1, \quad f(010) = 2 \\ f(110) &= 1, \quad f(111) = 2 \\ f(101) &= 1, \quad f(100) = 2 \end{aligned} \tag{۸–٦}$$

اگر به مقادیر تابع هزینه در معادله‌ی ۸–۵ نگاهی بیاندازیم، خواهیم دید که برش میان ذراتِ بسیار برازنده که به‌صورت دودویی کد شده‌اند، منجر به تولید فرزندانی می‌شود که آن‌ها نیز بسیار برازنده خواهند بود. این موضوع به دلیل آن است که همه‌ی ذراتِ بسیار برازنده دارای یک ۰ در راست‌ترین بیت خود هستند، که این موضوع باعث می‌شود فرزندان آن‌ها نیز دارای یک بیت ۰ در راست‌ترین بیت خود باشند. این یعنی فرزندان دارای کد دودویی زوج بوده و در نتیجه مانند والدینشان بسیار برازنده خواهند بود. با این حال، معادله‌ی (۸–٦) نشان می‌دهد که برش میان ذراتِ بسیار برازنده که با روش ***gray*** کدگذاری شده‌اند، ممکن است باعث ایجاد فرزندان با برازندگی کم بشود. این مثال ساده درک درستی از چگونگی بهتر عمل کردن کد دودویی نسبت به کد ***gray*** در مسائل با مینیمم‌های محلی زیاد به ما می‌دهد.

مثال ۸–۵

در این مثال، عملکرد *GA* بر روی یک مسئله‌ی بدترین– حالتِ ۲۰ بیتی که در آن مقادیر زوج دودویی دارای هزینه‌ی ۱ بوده و مقادیر فرد دارای هزینه‌ی ۲ می‌باشند را به بوته‌ی آزمایش می‌گذاریم. ما از اندازه‌ی جمعیتی برابر ۲۰ و نرخ جهشی برابر ۲٪ در بیت در نسل استفاده خواهیم نمود. شکل ۸–۵ میانگین هزینه‌ی ۲۰ ذره‌ی *GA* در هر نسل را که بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، به تصویر می‌کشد. کدگذاری دودویی بسیار بهتر از کدگذاری ***gray*** عمل می‌نماید. این موضوع به این حقیقت اشاره دارد که در مسائل با تعداد زیادی مینیمم محلی، عملکرد کدگذاری دودویی در پیدا کردن مینیمم‌های محلی بسیار بهتر از کدگذاری ***gray*** است. به خاطر داشته باشید که در بسیاری از مسائل بهینه‌سازی ترجیح بر آن است که به جای یک راه‌حل خوب، گستره‌ای از راه‌حل‌های خوب را پیدا کنیم.



شکل ۸–۵ نتایج مثال ۸–۵. نمودار، میانگین هزینه‌ی ۲۰ ذره‌ی *GA* در هر نسل از یک مسئله‌ی ۲۰ بیتی، که مینیمم‌های محلی بسیاری دارد، نشان می‌دهد. نتایج بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده‌اند. عملکرد کدگذاری دودویی در پیدا کردن مینیمم‌های محلی بسیار بهتر از کدگذاری *gray* است.

۸–۴ نخبه‌گرایی

این بخش به بحث در مورد نخبه‌گرایی می‌پردازد. نخبه‌گرایی در یک الگوریتم تکاملی به معنای حصول اطمینان از باقی ماندن بهترین ذرات در جمعیت از یک نسل به نسل دیگر می‌باشد. اگرچه نتایج الگوریتم تکاملی که در این کتاب نشان داده شده‌اند خوب به نظر می‌آیند، اما همیشه خطر از دست رفتن بهترین ذرات از یک نسل به نسل دیگر وجود دارد. نخبه‌گرایی از این خطر جلوگیری می‌کند.

GA نشان داده شده در شکل ۳–٦ را در نظر بگیرید. می‌بینیم که بهترین والدین بازترکیب شده تا فرزندان را به وجود آورند. با این حال، اگر x_e بهترین راه‌حل نامزد مسئله‌ی بهینه‌سازی در نسل iاُم باشد، هیچ تضمینی وجود ندارد که بهترین ذره در نسل $(i+1)$اُم از x_e بهتر بوده و یا حتی به خوبی آن باشد. x_e با سایر والدین بازترکیب شده تا فرزندان نسل بعد را به وجود آورد، اما خود جزئی از نسل بعد نخواهد بود. چگونه می‌توان به‌طور همزمان هم از نتایج خوب برخاسته از بازترکیب بهره برد و هم از از بین رفتن بهترین ذرات جمعیت جلوگیری نمود؟

جواب این سؤال در این است که باید بهترین ذرات الگوریتم تکاملی را از یک نسل به نسل دیگر نگه داشت. این ایده که برای اولین بار در [دِجونگ، ۱۹۷۵] ارائه شد نخبه‌گرایی نام دارد و معمولاً باعث بهبود عملکرد الگوریتم تکاملی می‌شود. نخبه‌گرایی را می‌توان به چند روش مختلف پیاده‌سازی نمود.

۱. نخبه‌گرایی را می‌توان با تولید تنها $(N-E)$ ذره پیاده‌سازی نمود، که در آن N اندازه‌ی جمعیت بوده و E تعداد ذرات نخبه است که توسط کاربر تعریف می‌شود. فرض کنید می‌خواهیم بهترین E ذره را از کل N ذره‌ی جمعیت، از یک نسل به نسل دیگر نگه داریم. در این صورت، ما از بازترکیب و جهش برای تولید $(N-E)$ فرزند استفاده کرده و سپس این فرزندان را همراه با بهترین E ذره به نسل بعد می‌فرستیم. شکل ۸-٦، که نسخه‌ی اصلاح شده‌ی شکل ۳-٦ است، این روش را برای GA نخبه‌گرا نشان می‌دهد. این ایده را می‌توان به راحتی در مورد سایر الگوریتم‌های تکاملی نیز به کار برد.

۲. می‌توان نخبه‌گرایی را با تولید N فرزند و جایگزین کردن بدترین فرزندان با بهترین E ذره از نسل قبل، پیاده‌سازی نمود. شکل ۸-۷ این روش را در مورد GA نخبه‌گرا نشان می‌دهد. این ایده را نیز می‌توان به راحتی در مورد سایر الگوریتم‌های تکاملی استفاده نمود. در کل می‌توان عملکرد بهتری را از این گزینه، نسبت به گزینه‌ی بالا انتظار داشت، اما این گزینه به یک مرحله‌ی اضافی مرتب‌سازی نیاز دارد.

۳. گزینه‌های دیگری نیز برای پیاده‌سازی نخبه‌گرایی وجود دارد. برای مثال، می‌توان N فرزند ایجاد نمود و سپس از نوعی انتخاب چرخ رولت معکوس برای انتخاب E عدد از آن‌ها استفاده کرد که در این حالت بدترین فرزندان بیشترین احتمال انتخاب شدن را خواهند داشت. سپس می‌توان این فرزندان را با بهترین E ذره از نسل قبل جایگزین نمود. در این زمینه تنوعات بسیار دیگری نیز وجود دارد.

```
{جمعیت تولید شده به صورت اتفاقی} ← والدین
تا زمانی که شرایط توقف برقرار نشده است
    برازندگی هر یک از والدین حاضر در جمعیت را حساب کن
    بهترین E والد ← نخبه‌ها
    ∅ ← فرزندان
    تا زمانی که |والدین| − E < |فرزندان|
        از مقادیر برازندگی برای انتخاب احتمالاتی یک جفت والد استفاده کن
        والدین را مزدوج کن تا فرزندان c1 و c2 به وجود آیند
        فرزندان ∪ {c1, c2} ← فرزندان
    حلقه
    برخی از فرزندان را به صورت اتفاقی دچار جهش کن
    نخبه‌ها ∪ فرزندان ← والدین
نسل بعد
```

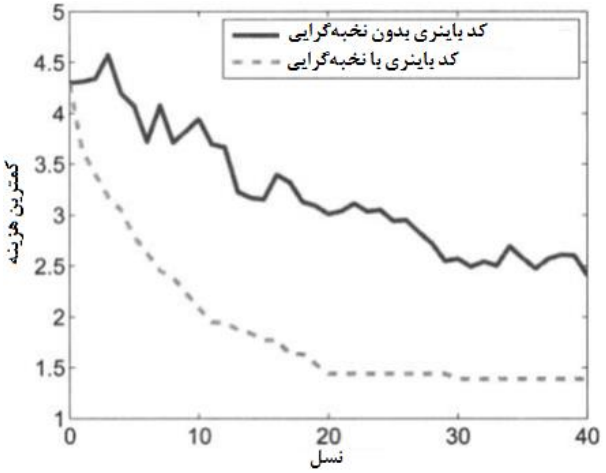
شکل ۸-٦ گزینه‌ی اول برای نخبه‌گرایی: یک الگوریتم ژنتیک ساده که جهت نخبه‌گرایی اصلاح شده است. N اندازه‌ی جمعیت بوده و E تعداد نخبه‌هایی است که از یک نسل به نسل دیگر نگه داشته می‌شوند و $(N-E)$ نیز تعداد ذراتی است که در هر نسل تولید می‌شود.

```
{جمعیت تولید شده به صورت اتفاقی} ← والدین
تا زمانی که شرایط توقف برقرار نشده است
    برازندگی هر یک از والدین حاضر در جمعیت را حساب کن
    بهترین E والد ← نخبه‌ها
    ∅ ← فرزندان
    تا زمانی که |والدین| < |فرزندان|
        از مقادیر برازندگی برای انتخاب احتمالاتی یک جفت والد استفاده کن
        والدین را مزدوج کن تا فرزندان c1 و c2 به وجود آیند
        فرزندان ∪ {c1, c2} ← فرزندان
    حلقه
    برخی از فرزندان را به صورت اتفاقی دچار جهش کن
    نخبه‌ها ∪ فرزندان ← والدین
    بهترین N والد ← والدین
نسل بعد
```

شکل ۸-۷ گزینه‌ی دوم نخبه‌گرایی: یک الگوریتم ژنتیک ساده که جهت نخبه‌گرایی اصلاح شده است. N اندازه‌ی جمعیت بوده و E تعداد نخبه‌هایی است که از یک نسل به نسل دیگر نگه داشته می‌شود و تعداد ذرات تولید شده در هر نسل نیز برابر N است.

مثال ۸–٦

در این مثال اثر نخبه‌گرایی بر عملکرد *GA* را می‌آزماییم. در اینجا نیز تابع دو بعدی اُکلی که در مثال ۳–۳ توصیف شده است و هر بعد آن با شش بیت کدگذاری می‌شود را بهینه می‌نماییم. ما از اندازه‌ی جمعیتی برابر ۲۰ و نرخ جهشی برابر با ۲٪ در بیت در نسل استفاده می‌نماییم. شکل ۸–۸ مینیمم هزینه‌ی ۲۰ ذره‌ی *GA* در هر نسل را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. *GA* نخبه‌گرا بهترین دو ذره از هر نسل را نگه داشته و از روش نخبه‌گرایی نشان داده شده در شکل ۸–٦ استفاده می‌کند. می‌توان دید که استفاده از نخبه‌گرایی بهبودی قابل توجه در عملکرد *GA* به وجود می‌آورد.



شکل ۸–۸ نتایج مثال ۸–٦ برای مینیمم‌سازی یک تابع دو بعدی اُکلی که هر بعد آن با شش بیت کدگذاری شده است. نمودار مینیمم هزینه‌ی تمام ذرات *GA* را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است نشان می‌دهد. *GA* نخبه‌گرا بسیار بهتر از *GA* عادی عمل می‌نماید.

تقریباً باید همیشه از نخبه‌گرایی در الگوریتم‌های تکاملی استفاده نمود چرا که هزینه‌ی آن اندک بوده و منفعت آن زیاد است. با این حال، برخی مسائل با توابع هزینه‌ی پویا و پرهزینه وجود دارند که الگوریتم‌های تکاملی غیرنخبه‌گرا در مورد آن‌ها عملکرد بهتری از الگوریتم‌های تکاملی نخبه‌گرا دارند (فصل ۲۱ را ببینید).

۸–۵ الگوریتم‌های نسلی و حالت– ماندگار

بیشتر الگوریتم‌های تکاملی که تا به حال در مورد آن‌ها به بحث پرداخته‌ایم، الگوریتم‌های تکاملی نسلی هستند. بدین معنا که از هر نسل به نسل دیگر کل جمعیت، به غیر از احتمالاً چند ذره‌ی نخبه، عوض می‌شود. با این حال، در طبیعت تکامل بدین صورت اتفاق نمی‌افتد. به بیان دیگر تولد و مرگ در طبیعت به‌صورت

پیوسته در حال وقوع است. این نوع تکامل، تکامل حالت- ماندگار نام دارد. مشاهده‌ی طبیعت به ما این انگیزه را می‌دهد که به دنبال پیاده‌سازی نسخه‌های حالت- ماندگار از الگوریتم‌های تکاملی برویم. شکل ۸-۹ طرحی کلی از یک GA حالت- ماندگار به دست می‌دهد.

شکل ۸-۹ نشان می‌دهد که در هر نسل تنها دو فرزند ایجاد شده و این دو فرزند جای والدین خود در جمعیت را می‌گیرند. این شکل را با شکل ۳-٦ که یک GA نسلی را نشان می‌دهد، مقایسه کنید. در شکل ۳-٦ ما همه‌ی N فرزند را پیش از آنکه جای والدینشان را بگیرند ایجاد می‌نماییم. گزینه‌های زیادی برای پیاده‌سازی در یک الگوریتم تکاملی حالت- ماندگار وجود دارند.

```
{جمعیت تولید شده به صورت اتفاقی} ← والدین
برازندگی هر یک از والدین حاضر در جمعیت را حساب کن
تا زمانی که شرایط توقف برقرار نشده است
    از مقادیر برازندگی برای انتخاب احتمالاتی یک جفت والد استفاده کن
    والدین را ترکیب کن تا فرزندان c1 و c2 به وجود آیند
    c1 و c2 را به صورت اتفاقی دچار جهش کن
    برازندگی c1 و c2 را حساب کن
    p1 ← c1 ،p2 ← c2
نسل بعد
```

شکل ۸-۹ یک الگوریتم ژنتیک حالت- ماندگار.

- ما می‌توانیم تنها در صورتی p_1 را با c_1 جایگزین نماییم که c_1 برازندگی بهتری نسبت به p_1 داشته باشد، سپس می‌توانیم این کار را در مورد p_2 و c_2 نیز انجام دهیم. این کار مانند عمل نخبه‌گرایی است که در بخش ۸-٤ توضیح داده شد چرا که در این صورت بهترین ذره هیچ‌گاه از جمعیت بیرون نخواهد افتاد. همچنین این عمل مشابه الگوریتم $(\mu + \lambda) - ES$ از شکل ٦-۱۰ نیز می‌باشد. در $(\mu + \lambda) - ES$ یک فرزند تنها در صورتی جزئی از نسل بعد خواهد بود که جز بهترین μ ذره از کل $(\mu + \lambda)$ ذره‌ی جمعیت باشد.
- ما می‌توانیم در هر نسل بیش از دو ذره را تولید و جایگزین نماییم. توجه داشته باشید که GA نسلی از شکل ۳-٦ تمام N ذره را در هر نسل جایگزین می‌نماید، این در حالی است که GA حالت- ماندگار از شکل ۸-۹ در هر نسل تنها دو ذره را جایگزین می‌نماید. می‌توان یک GA را به‌گونه‌ای طراحی نمود که در جایی بین این دو حد قرار بگیرد، یعنی برای مثال در هر نسل ٤، ٦ و یا تعدادی

تصادفی از ذرات را جایگزین نماید. کِنِت دِجونگ از عبارت "شکاف نسلی" برای اشاره به تعداد ذرات جایگزین شده در هر نسل استفاده می‌نماید [دِجونگ، ۱۹۷۵].

توجه داشته باشید که جهت مقایسه‌ی عملکرد GA نسلی از شکل ۳-٦ و GA حالت- ماندگار از شکل ۸-۹، نمی‌توان از تعدادی مساوی نسل استفاده نمود. یک نسل از شکل ۳-٦، N ذره تولید می‌نماید و این در حالی است که یک نسل از شکل ۸-۹ تنها دو ذره تولید می‌نماید. به همین دلیل برای تعدادی مساوی نسل، GA نسلی همواره عملکرد بهتری از GA حالت- ماندگار نشان خواهد داد. این مقایسه‌ای عادلانه نخواهد بود. برای مقایسه‌ی عادلانه میان این دو الگوریتم باید هر دو الگوریتم را برای تعدادی مساوی از محاسبات تابع برازندگی اجرا نماییم. بنابراین $NG/2$ نسل از GA حالت- ماندگار از شکل ۸-۹ به لحاظ محاسباتی با G نسل از GA نسلی از شکل ۳-٦ برابری می‌کند.

شکل ۳-٦ و ۸-۹ استراتژی‌های نسلی و حالت- ماندگار برای GAها را نشان می‌دهد، اما این ایده‌ها را می‌توان به‌صورت کلی در مورد هر الگورتیم تکاملی دیگر نیز به کار برد. همان‌طور که در صفحات پیشین این کتاب دیده‌ایم، و در صفحات آتی نیز خواهیم دید، برای برخی الگوریتم‌های تکاملی روش نسلی مناسب‌تر بوده و برای برخی دیگر استراتژی حالت- ماندگار مناسب‌تر است. اما در کل، پیاده‌سازی استاندارد هر الگوریتم تکاملی را می‌توان برای به دست آوردن الگوریتم نسلی و یا حالت- ماندگار اصلاح نمود.

۸-۶ تنوع جمعیت

این بخش به بحث درباره‌ی نحوه‌ی مدیریت ذرات تکراری در جمعیت و همچنین چگونگی اصلاح فرایندهای انتخاب و بازترکیب برای تقویت تنوع در مسائل چندپیمانه‌ای می‌پردازد. ابتدا در زیربخش ۸-٦-۱ مسئله‌ی ذرات تکراری را در نظر می‌گیریم. سپس به بحث در مورد دو روش برای ترویج جمعیت‌های متنوع از الگوریتم‌های تکاملی می‌پردازیم: محدودیت‌های بازترکیب در زیربخش ۸-٦-۲ و روش‌هایی برای حفظ Nicheهای جمعیت در بخش ۸-٦-۳ که شامل به اشتراک‌گذاری برازندگی، تسویه و ازدحام می‌شود.

۸-۶-۱ ذرات تکراری

معمولاً جمعیت‌هایی که مرتباً از یک نسل به نسل دیگر بازترکیب می‌شوند، دچار یکنواختی می‌گردند. این بدان معناست که جمعیت به‌صورت جمعیتی از کلون‌ها در می‌آید. یکنواختی بیشتر در مورد مسائل با دامنه‌ی گسسته روی می‌دهد، اما می‌تواند در مورد مسائل با دامنه‌ی پیوسته نیز رخ دهد. یکنواختی باعث محدودیت الگوریتم تکاملی در جستجوی بیشترِ فضای جستجو می‌شود. اگرچه معمولاً راه‌حلِ نامزدی که الگوریتم تکاملی به آن همگرا می‌شود راه‌حل خوبی است، اما همواره ممکن است راه‌حل بهتر دیگری در

سایر نواحی فضای جستجو موجود باشد. به همین دلیل ما، حتی پس از آنکه الگوریتم تکاملی راه‌حلی خوب پیدا کرد، به یافتن راه‌حلی بهتر توسط الگوریتم تکاملی امید داریم. اگر یکنواختی پیش از آنکه راه‌حلی رضایت‌بخش پیدا شود رخ دهد، به آن *همگرایی نارس* می‌گویند [رونالد[1]، ۱۹۹۸]. می‌توان با افزایش نرخ جهش از وقوع این رخداد جلوگیری نمود، اما اگر نرخ جهش را زیادی افزایش دهیم، الگوریتم تکاملی به یک جستجوی اتفاقی منحط می‌شود. یک راه مرسوم برای مقابله با همگرایی نارس آن است که پیوسته به دنبال ذرات تکراری در جمعیت باشیم و در صورت پیدا کردن ذرات تکراری، آن‌ها را جایگزین نماییم. این کار را می‌توان به چند طریق انجام داد که برخی از آن‌ها در زیر توضیح داده شده‌اند.

۱. هرگاه که یک فرزند تولید می‌نماییم، می‌توانیم جمعیت را بررسی کنیم تا مطمئن شویم در حال تولید ذره‌ای تکراری نیستیم. اگر ذره‌ای تکراری تولید نموده باشیم، می‌توانیم عملیات بازترکیب را دوباره و با والدینی متفاوت و یا با پارامترهای برشی متفاوت انجام دهیم تا فرزندی غیر- تکراری به دست آوریم. یا آنکه می‌توانیم فرزند تکراری را دچار جهش کنیم تا یک ذره‌ی غیر- تکراری حاصل شود.
۲. هرگاه که یک ذره را دچار جهش می‌نماییم، می‌توانیم جمعیت را بررسی کرده تا مطمئن شویم ذره‌ای تکراری به وجود نیاورده‌ایم. اگر چنین بود، می‌توانیم عملیات جهش را تکرار نماییم.
۳. می‌توان در انتهای هر نسل جمعیت را برای ذرات تکراری بررسی نمود. می‌توان ذرات تکراری به روش‌های مختلفی جایگزین نمود. برای مثال، می‌توان ذرات تکراری را با ذرات اتفاقی جایگزین کرد و یا می‌توان ذرات تکراری را دچار جهش نمود و یا اینکه می‌اینکه می‌توانیم از عملیات بازترکیب برای جایگزین نمودن هر ذره‌ی تکراری استفاده کنیم.
٤. می‌توان وجود تعدادی ذره‌ی تکراری را درجمعیت مجاز شمرد. البته این مقدار نباید از یک مقدار آستانه‌ی D بیشتر باشد. ذرات تکراری معمولاً ذراتی هستند که بسیار برازنده‌اند چرا که اگر نبودند احتمال وقوع آن‌ها در جمعیت بسیار کم می‌بود. بنابراین، ممکن است حضور ذرات تکراری چندان ذهن ما را مشغول نکند چرا که این ذرات احتمال شرکت کردن ذرات بسیار برازنده در فرایند بازترکیب را افزایش می‌دهند. بنابراین می‌توان تنها در صورتی ذرات تکراری را جایگزین نمود که بیشتر از D عدد از آن‌ها در جمعیت موجود باشند. یا آنکه می‌توانیم به‌صورت احتمالاتی و بسته به میزان برازندگیشان آن‌ها را جایگزین نماییم.

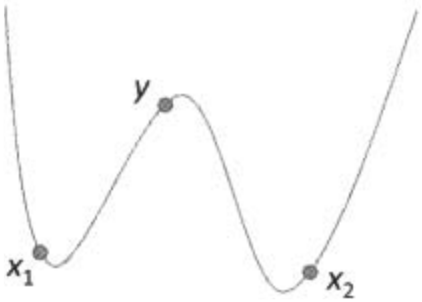
بررسی جمعیت برای پیدا کردن ذرات تکراری به لحاظ محاسباتی فرایندی پرهزینه تلقی می‌شود چرا که به یک حلقه‌ی تودرتو نیاز دارد و به همین علت برای اندازه‌ی جمعتی برابر با N به تلاشی محاسباتی از

[1] Ronald

مرتبه‌ی N^2 نیاز خواهد داشت. این مقدار، جزئی عظیم از تلاش محاسباتی مورد نیاز برای مسائل محک خواهد بود. با این حال، باید به یاد داشته باشیم که مسائل دنیای واقعی معمولاً چندین مرتبه از مسائل محک پیچیده‌تر هستند. در مسائل دنیای واقعی محاسبه‌ی تابع برازندگی قسمت اعظمی از تلاش محاسباتی را به خود اختصاص می‌دهد (فصل ۲۱ را ببینید) و به همین دلیل، در این نوع مسائل تلاش محاسباتی مورد نیاز برای پیدا نمودن ذرات تکراری اندک تلقی می‌شود. با این حال، اگر تلاش محاسباتی در یک مسئله‌ی محک یک دغدغه باشد، می‌توان بررسی جمعیت برای ذرات تکراری را به جای هر نسل هر G نسل یکبار انجام داد (G پارامتری است که توسط کاربر تعریف می‌گردد).

۸–۶–۲ بازترکیب گونه– محور و Niche– محور

پیاده‌سازی‌های موجود از الگوریتم‌های تکاملی ذرات والد را انتخاب کرده و آن‌ها را برای ایجاد فرزندان ترکیب می‌کند، بی‌آنکه میزان شباهت و یا تفاوت میان والدین را در نظر بگیرد. با این حال، در بیولوژی گاها دیده می‌شود که والدین کمی شبیه یکدیگر هستند. برای مثال، به ندرت می‌توان دید دو حیوان از دو گونه‌ی متفاوت با یکدیگر جفت‌گیری نمایند. همچنین به ندرت می‌توان دید دو شخص که با هم نسبت فامیلی دارند با یکدیگر ازدواج نمایند. شکل ۸–۱۰ مسئله‌ای را نشان می‌دهد که در آن بازترکیب میان دو ذره‌ی بسیار متفاوت از هم صورت می‌پذیرد. اول آنکه، فرزند به دست آمده می‌تواند راه‌حلی بسیار ضعیف برای مسئله‌ی بهینه‌سازی باشد، چرا که نقطه‌ی میانی دو ذره‌ی بسیار برازنده می‌تواند برازندگی بسیار ضعیفی داشته باشد. دوم آنکه، عمل برش می‌توان باعث از بین رفتن اطلاعات ژنتیکی مهم برای حل مسئله شود.



شکل ۸–۱۰ این شکل یک مسئله‌ی بهینه‌سازی چندپیمانه‌ای را نشان می‌دهد. تابع این مثال، مشکل موجود در مورد بازترکیب دو ذره‌ی بسیار متفاوت از هم را نشان می‌دهد. دو والد x_1 و x_2 دارای هزینه‌ی کمی هستند اما فرزند آن‌ها که y است دارای هزینه‌ی زیادی می‌باشد. به علاوه اگر y جای یکی از والدین خود را بگیرد (برای مثال x_2)، آنگاه پیدا کردن بهینه‌ی جهانی که در نزدیکی x_2 واقع شده است برای الگوریتم تکاملی دشوار خواهد بود.

شکل ۸-۱۱ مسئله‌ای را نشان می‌دهد که در آن بازترکیب میان دو ذره‌ی بسیار مشابه هم صورت می‌پذیرد. اگر به ذراتی که با هم تفاوت دارند اجازه‌ی بازترکیب داده نشود، ممکن است فرایند تکاملی در یک شیار به دام افتد.

شکل ۸-۱۱ این شکل یک مسئله‌ی بهینه‌سازی چندپیمانه‌ای را نشان می‌دهد. تابع این مثال مشکلی را نشان می‌دهد که اگر ذرات متفاوت با هم مجاز به بازترکیب با یکدیگر نباشند رخ می‌دهد. اگر x_1 و x_2 به دلیل تفاوتشان با یکدیگر مجاز به بازترکیب با یکدیگر نباشند، آنگاه پیدا کردن مینیمم محلی که در نزدیکی y واقع شده است دشوار خواهد بود.

مشکلات توضیح شده در بالا انگیزه‌ی استفاده از بازترکیبِ گونه- محور و Niche- محور را به ما می‌دهند.

- استراتژی‌های Niching، بازترکیب میان ذراتی که در فضای دامنه بسیار متفاوت از هم هستند را منع می‌کنند [ماهفود[1]، ۱۹۹۵b]، [ماهفود، ۱۹۹۵a]. این روش نه تنها به پیدا کردن راه‌حل بهینه یاری می‌رساند (شکل ۸-۱۰)، بلکه به ما کمک می‌کنند بهینه‌های محلی را نیز بیابیم. یافتن بهینه‌های محلی برای بسیاری از مسائل مهم هستند. Niching همچنین می‌تواند برای بهینه‌سازی چندهدفه و بهینه‌سازی پویا نیز مفید واقع شود (فصل ۲۰ و ۲۱ را ببینید).
- استراتژی‌های گونه- محور، بازترکیب میان ذراتی که در فضای دامنه بسیار مشابه هم هستند را منع می‌کنند [بِنژاف و همکاران، ۱۹۹۸، بخش ٦-٤]. این موضوع باعث تقویت شدن کاوش می‌شود.

توجه داشته باشید که دو استراتژی گونه- محور و Niche- محور، دو استراتژی متقابل برای یک مسئله مشخص می‌باشند. فلسفه‌ی بازترکیب Niche- محور این است که وقتی ذرات برازنده با یکدیگر بازترکیب می‌شوند، تنها در صورتی می‌توان انتظار فرزند برازنده را داشت که والدین شبیه یکدیگر باشند. از سوی دیگر، فلسفه‌ی بازترکیب گونه- محور این است که وقتی ما ذرات برازنده را با یکدیگر بازترکیب می‌نماییم،

[1] Mahfoud

باید از تفاوت آن‌ها با یکدیگر اطمینان حاصل نماییم تا بتوانیم به‌طور مؤثر به کاوش فضای جستجو بپردازیم. انتخاب میان این دو روش تصمیمی است که به نوع مسئله بستگی دارد.

Niching ۳–۶–۸

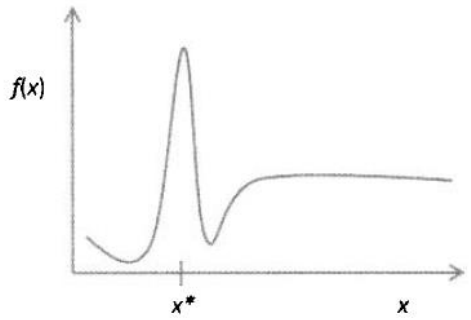
در این زیربخش ما از عبارت Niching با معنایی متفاوت از زیربخش پیشین استفاده می‌نماییم. Niching در این زیربخش روشی است که به ذرات الگوریتم تکاملی اجازه می‌دهد در پاکت‌هایی مجزا از فضای جستجو به بقای خود ادامه دهند. انگیزه‌ی Niching در این زیربخش مانند انگیزه‌ای است که در زیربخش پیشین وجود داشت. با این حال، مخاطب Niching در زیربخش قبلی انتخاب والدین بود، در حالی که در زیربخش حاضر مخاطبِ Niching، تنظیم نمودن مقادیر برازندگی می‌باشد.

انگیزه‌ی استفاده از مفهوم Niching از مسائل چندپیمانه‌ای نشئت گرفته است. مسائلی که در آن‌ها ممکن است نگه داشتن ذرات در نزدیکی تعداد زیادی از بهینه‌های محلی اهمیت داشته باشد. همچنین ممکن است در این نوع مسائل پیدا کردن چندین راه‌حل خوب نیز حایز اهمیت باشد. اولین روش Niching به اشتراک‌گذاری برازندگی است [هالند، ۱۹۷۵، صفحه‌ی ١٦٤]. ما در مورد سه استراتژی Niching بحث خواهیم نمود: به اشتراک‌گذاری برازندگی در زیربخش ۸–٦–۳–۱، تسویه در زیربخش ۸–٦–۳–۲ و ازدحام در زیربخش ۸–٦–۳–۳. بحث‌های بیشتر در مورد این ایده‌ها را می‌توان در [سارِنی[1] و کراهِنباول، ۱۹۹۸] یافت.

۸–٦–۳–۱ *به اشتراک‌گذاری برازندگی.* گاهی اوقات ذرات الگوریتم تکاملی می‌توانند در یک ناحیه‌ی خوب از فضای جستجو، جمعیت را در اختیار بگیرند. این موضوع می‌تواند باعث بروز همگرایی نارس شود. ما می‌خواهیم ذرات خوب را در جمعیت نگه داریم، اما از سویی دیگر نیز می‌خواهیم تنوع جمعیت را حفظ کرده تا با پیشرفت الگوریتم تکاملی از یک نسل به نسل دیگر، شانس کاوش نواحی جدیدِ فضای جستجو را نیز داشته باشیم. به اشتراک‌گذاری برازندگی به خصوص برای مسائل چندپیمانه‌ای که در آن‌ها به دنبال پیدا کردن چندین راه‌حل در نواحی مختلف فضای جستجو هستیم، بسیار مفید است.

شکل ۸–۱۲ را در نظر بگیرید. می‌توان دید که x^* ماکزیمم جهانی است، اما ذرات نزدیک x^* به دلیل مقادیر برازندگی کمشان نسبت به سایر ذرات موجود در فضای جستجو، از احتمال کمی برای بقا و انتقال به نسل بعد برخوردار هستند. برای تقویت تنوع می‌توان مقادیر برازندگی ذراتی که نسبتا منحصر به فرد هستند را به‌صورت مصنوعی افزایش و مقادیر برازندگی ذراتی که نسبتاً شایع هستند را به‌طور مصنوعی کاهش داد.

[1] Sareni



شکل ۸-۱۲ این تابع دارای یک ماکزیمم جهانی در x^* است، اما ذرات نزدیک x^* به دلیل مقادیر برازندگی کمشان نسبت به سایر ذرات موجود در فضای جستجو از احتمال کمی جهت انتخاب شدن برای فرایند بازترکیب برخوردار هستند.

به اشتراک‌گذاری برازندگی، برازندگی ذرات نزدیک به یکدیگر در فضای جستجو را کاهش می‌دهد [سارِنی و کراهِنباول، ۱۹۹۸]. انگیزه‌ی بیولوژیکی این ایده در این حقیقت نهفته است که ذرات مشابه بر سر منابع مشابه به رقابت می‌پردازند. بنابراین، یک ذره حتی اگر بسیار برازنده هم باشد، ممکن است به دلیل وجود تعداد زیادی ذره‌ی مشابه در محل جغرافیایی خود، قادر به بازتولید نباشد.

فرض کنید یک جمعیتِ الگوریتم تکاملیِ $\{x_i\}$ با اندازه‌ی N در اختیار داریم. همچنین فرض کنید f_i نشان‌دهنده‌ی برازندگی ذره‌ی x_i است. به اشتراک‌گذاری برازندگی، مقادیر اصلاح‌شده‌ی برازندگی را به‌صورت زیر محاسبه می‌نماید:

$$f_i' = {f_i}/{m_i} \qquad (۷-۸)$$

که در آن m_i، که تعداد Nicheهای x_i خوانده می‌شود، به تعداد ذرات مشابه x_i بستگی دارد. تعداد Niche به‌صورت زیر محاسبه می‌شود

$$m_i = \sum_{j=1}^{N} s(d_{ij}) \qquad (۸-۸)$$

که در آن $s(.)$ تابع به اشتراک‌گذاری بوده و d_{ij} فاصله‌ی میان ذرات x_i و x_j را اندازه می‌گیرد. ما معمولاً از فاصله‌ی اقلیدسی برای اندازه‌گیری d_{ij} استفاده می‌نماییم. یکی از توابعِ مرسوم به اشتراک‌گذاری تابع زیر است

$$s(d) = \begin{cases} 1 - (d/\sigma)^{\alpha} & \text{اگر } \sigma\, d < \\ 0 & \text{در غیر این صورت} \end{cases} \qquad (۹-۸)$$

که در آن σ پارامتری است که توسط کاربر تعریف شده و با نام‌های آستانه‌ی عدم تجانس، قطع فاصله و شعاع Niche شناخته می‌شود. α نیز پارامتری است که توسط کاربر تعیین می‌شود. ما معمولاً از $\alpha = 1$ استفاده می‌کنیم. این کار تابعِ به اشتراک‌گذاری مثلثی را به دست می‌دهد. محققان روش‌های متنوعی برای تعیین مقدار آستانه‌ی عدم تجانس پیشنهاد کرده‌اند [دِب و گلدبرگ، ۱۹۸۹]. برای مثال

$$\sigma = rq^{-\frac{1}{n}}$$
$$r = \frac{1}{2}\sum_{k=1}^{n}(max_i x_i(k) - min_i x_i(k))^2 \qquad (۱۰-۸)$$

که در آن n ابعاد مسئله، $x_i(k)$، kاُمین عنصرِ x_i و q مقدار منتظره‌ی مینیمم محلی در تابع برازندگی است. در روش به اشتراک‌گذاری برازندگی، از مقادیر برازندگی اصلاح شده‌ی معادله‌ی (۸-۷) برای انتخاب والدین جهت بازترکیب استفاده می‌شود. توجه داشته باشید که اگر یک مسئله‌ی مینیمم‌سازی داشته باشیم، ابتدا باید مقادیر هزینه را، پیش از اعمال به معادله‌ی (۸-۷)، به مقادیر برازندگی تبدیل نماییم، سپس باید مقادیر اصلاح شده‌ی برازندگی را که از این معادله به دست می‌آیند را به مقادیر هزینه‌ی اصلاح شده تبدیل نماییم (مسئله‌ی ۸-۷ را ببینید).

۸-۶-۳-۲ *تسویه*. روش تسویه مانند به اشتراک‌گذاری برازندگی است تنها با این تفاوت که به جای آنکه مقادیر برازندگی را بین ذراتی که Niche یکسانی دارند به اشتراک بگذاریم، مقادیر برازندگی برخی از این ذرات را کاهش می‌دهیم [پِتروسکی، ۱۹۹۶]، [سارِنی و کراهِنباول، ۱۹۹۸]. چندین راه برای پیاده‌سازی این ایده وجود دارد. برای مثال، می‌توان ابتدا یک مجموعه‌ی Niche از هر ذره‌ی موجود در جمعیت تشکیل داد (D_i):

$$D_i = \{x_j : d_{ij} < \sigma\} \qquad (۱۱-۸)$$

که در آن d_{ij} همان فاصله‌ای است که در معادله‌ی (۸-۸) وجود داشت و σ نیز پارامتری است که توسط کاربر تعیین می‌شودد. سپس باید ذرات را با توجه به مقادیر برازندگیشان در هر Niche رده‌بندی کرد:

$$r_{ki} = \text{رتبه‌ی } x_k \text{ در } D_i \qquad (۱۲-۸)$$

در معادله‌ی بالا بهترین ذره در هر Niche دارای رتبه‌ی ۱ بوده، دومین بهترین ذره دارای رتبه‌ی ۲ بوده و الی آخر. در انتها نیز باید پارامترِ R را که نشان‌دهنده‌ی تعداد ذراتی است که می‌خواهیم در هر Niche به بقای خود ادامه دهند تعیین نماییم. بدین ترتیب مقادیر اصلاح‌شده‌ی برازندگی به‌صورت زیر به دست خواهند آمد (N اندازه‌ی جمعیت است)

For $i = 1\ to\ N$
  For $k = 1\ to\ |D_i|$
    If $r_{ki} \leq R$ then
      $f'_k \leftarrow f_k$
    Else
      $f'_k \leftarrow -\infty$
    End if
  Next Niche
Next Individual

الگوریتم بالا به ما این تضمین را می‌دهد که ذرات با کمترین مقدار برازندگی در هر Niche برای فرایندهای انتخاب و یا بازترکیب در دسترس نخواهند بود. با این حال، این الگوریتم نمی‌تواند تضمین کند که برازنده‌ترین ذرات برای فرایند انتخاب در دسترس هستند چرا که هر ذره ممکن است به بیش از یک Niche تعلق داشته باشد. برای مثال، ذره‌ی x_m ممکن است در Niche خود برازنده‌ترین ذره باشد، اما ممکن است به یک Niche دیگر نیز تعلق داشته باشد و در آن Niche بهترین ذره نباشد که در این صورت ممکن است مقدار اصلاح‌شده‌ی برازندگی‌اش برابر با $-\infty$ گردد. مجموعه‌ی ذراتی که از الگوریتم بالا جان سالم به در می‌برند به مرتبه‌ی فرایند Nicheهای $\{D_i\}$ بستگی دارد.

۸-۶-۳-۳ *ازدحام*. در این روش، ذرات موجود در جمعیت با ذراتِ مشابهی که اخیراً توسط فرایند بازترکیب ایجاد شده‌اند، جایگزین می‌شوند. ازدحام اولین بار توسط کِنِث دِجونگ برای تقلید از رقابت موجود در طبیعت بر سرِ منابع معرفی گردید [دِجونگ، ۱۹۷۵]. در زیر به بحث در مورد سه نوع ازدحام می‌پردازیم: ازدحام استاندارد، ازدحام قاطع و انتخاب مسابقه‌ای محدود.

ازدحام استاندارد. ازدحام استاندارد در ارتباط با بازترکیب حالت- مانگار به کار می‌رود (بخش ۸-۵ را ببینید). ازدحام استاندارد در هر نسل M فرزند تولید می‌نماید و سپس این فرزندان را با C_f والد که به‌صورت اتفاقی انتخاب شده‌اند، مقایسه می‌نماید. M و C_f پارامترهایی هستند که توسط کاربر تعیین می‌گردند. C_f فاکتور ازدحام خوانده می‌شود. هر فرزند جای شبیه‌ترین ذره به خود از مجموعه‌ی C_f والد را می‌گیرد. مقادیر معمول برای این دو پارامتر عبارتند از $M = N/10$ و $C_f = 3$، که در آن N اندازه‌ی جمعیت است [ماهفود، ۱۹۹۲]. شکل ۸-۱۳ یک پیاده‌سازی از ازدحام استاندارد را نشان می‌دهد.

{جمعیت تولید شده به صورت اتفاقی} ← $\{p_k\}$ والدین
برازندگی هر یک از والدین حاضر در جمعیت را حساب کن
تا زمانی که شرایط توقف برقرار نشده است
از مقادیر برازندگی برای انتخاب احتمالاتی M والد جهت بازترکیب استفاده کن
والدین را ترکیب کن و M فرزند تولید کن c_i، $i \in [1, M]$
هر فرزند را به صورت اتفاقی دچار جهش کن
برازندگی هر فرزند را محاسبه کن
برای M تا $i = 1$
C_f ذره را به صورت اتفاقی از میان جمعیت والدین انتخاب کن (مجموعه I)
$p_{min} = argmin_p ||p - c_i|| : p \in \mathrm{I}$

شکل ۸-۱۳ یک الگوریتم تکاملی حالت- ماندگار با ازدحام استاندارد. M و C_f پارامترهایی هستند که توسط کاربر تعیین می‌گردند و $||p - c_i||$ یک تابع فاصله است که توسط کاربر تعیین می‌شود.

ازدحام قاطع ازدحام قاطع شامل رقابت میان فرزندان و والدین است [ماهفود، ۱۹۹۵b]. والدین با یکدیگر ترکیب شده و فرزندان را به وجود می‌آورند. اما فرزندان تنها در صورتی جای شبیه‌ترین والد به خود را می‌گیرند که دارای برازندگی بهتری از آن والد باشند. شکل ۸-۱٤ یک پیاده‌سازی از ازدحام قاطع را نشان می‌دهد.

انتخاب مسابقه‌ای محدود انتخاب مسابقه‌ای محدود دارای ویژگی‌های مشترکی هم با ازدحام استاندارد و هم با ازدحام قاطع است [هَریک[1]، ۱۹۹۵]. M والد برای تولید دو فرزند با یکدیگر بازترکیب می‌شوند. سپس این فرزندان با C_f ذره که به‌صورت اتفاقی انتخاب شده‌اند مقایسه می‌شوند. هر ذره جای شبیه‌ترین ذره از این C_f ذره را می‌گیرد، تنها در صورتی که دارای برازندگی بهتری نسبت به آن باشد. شکل ۸-۱۵، یک پیاده‌سازی از انتخاب مسابقه‌ای محدود را نشان می‌دهد.

[1] Harik

{جمعیت تولید شده به صورت اتفاقی} ← والدین
برازندگی هر یک از والدین حاضر در جمعیت را حساب کن
تا زمانی که شرایط توقف برقرار نشده است
از مقادیر برازندگی برای انتخاب احتمالاتی یک جفت والد p_1 و p_2 استفاده کن
دو والد را با یکدیگر ترکیب کن تا دو فرزند c_1 و c_2 به دست آیند
c_1 و c_2 را به صورت اتفاقی دچار جهش کن
برازندگی c_1 و c_2 را محاسبه کن
برای $i = 1$ تا 2
اگر $||p_1 - c_i|| < ||p_2 - c_i||$ و برازندگی(p_1) > برازندگی(c_i)
$p_1 \leftarrow c_i$
اگر $||p_2 - c_i|| < ||p_1 - c_i||$ و برازندگی(p_2) > برازندگی(c_i)
$p_2 \leftarrow c_i$
پایان اگر
فرزند بعد
نسل بعد

شکل ۸-۱٤ یک الگوریتم تکاملی حالت- ماندگار با ازدحام قاطع. هر فرزند جای نزدیکترین والد را تنها در صورتی که دارای برازندگی بهتری نسبت به والد باشد، می‌گیرد.

{جمعیت تولید شده به صورت اتفاقی} ← والدین
برازندگی هر یک از والدین حاضر در جمعیت را حساب کن
تا زمانی که شرایط توقف برقرار نشده است
از مقادیر برازندگی برای انتخاب احتمالاتی M والد استفاده کن
M والد را با هم ترکیب کن تا M فرزند به دست آید، c_i، $i \in [1, M]$
هر فرزند را به صورت اتفاقی دچار جهش کن
برازندگی هر فرزند را محاسبه کن
C_f ذره را به صورت اتفاقی از میان جمعیت والدین انتخاب کن (مجموعه I)
برای $i = 1$ تا M
$p_{min} = argmin_p ||p - c_i|| : p \in \mathrm{I}$
اگر برازندگی(p_{min}) > برازندگی آنگاه(c_i)
$p_{min} \leftarrow c_i$
پایان اگر
فرزند بعد
نسل بعد

شکل ۸-۱۵ یک الگوریتم تکاملی حالت- مانگار با انتخاب مسابقه‌ای محدود. M و C_f پارامترهایی هستند که توسط کاربر تعیین می‌گردند و $||p - c_i||$ یک تابع فاصله است که توسط کاربر تعیین می‌گردد.

۸–۷ گزینه‌های انتخاب

پیش از آنکه بتوانیم ذرات الگوریتم تکاملی را برای ایجاد فرزندان با یکدیگر ترکیب نماییم، باید والدین را از میان ذرات موجود انتخاب نماییم. انتخاب چرخ رولت، که در زیربخش ۳–٤–۲ در مورد آن به بحث پرداختیم، روش انتخاب استانداردی است که هم GA و هم سایر الگوریتم‌های تکاملی از آن استفاده می‌نمایند. اما الگوریتم‌های انتخاب دیگری نیز وجود دارند که در این بخش ۷ مورد از آن‌ها را بررسی خواهیم نمود. تقریباً تمامی این الگوریتم‌ها به انتخاب برازنده‌ترین ذرات جمعیت گرایش دارند. به این معنی که مهم نیست از چه الگوریتمی برای فرایند انتخاب استفاده نماییم، ذرات برازنده همواره شانس بیشتری برای انتخاب شدن دارند تا ذرات با برازندگی کمتر.

اگر الگوریتم انتخاب بیش از حد به سمت انتخاب ذرات برازندی گرایش‌دهی شده باشد، آنگاه جمعیت به سرعت به یک راه‌حل یکنواخت همگرا می‌شود بی‌آنکه به اندازه‌ی کافی به کاوش فضای جستجو پرداخته باشد. از سوی دیگر، اگر روش انتخاب به اندازه‌ی کافی به سمت ذرات برازنده گرایش داده نشده باشد، آنگاه الگوریتم تکاملی قادر نخواهد بود از اطلاعات موجود در ذرات برازنده به خوبی استفاده نماید.

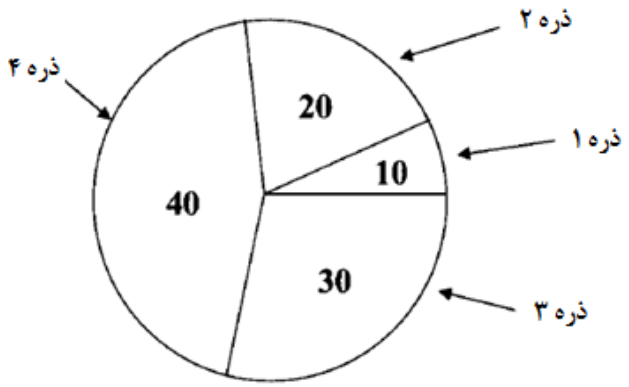
یک معیار مفید برای اندازه‌گیری تفاوت میان الگوریتم‌های انتخاب متفاوت فشار انتخاب ϕ است و به‌صورت زیر تعریف می‌شود

$$\phi = \frac{\Pr(\text{انتخاب برترین ذره})}{\Pr(\text{انتخاب ذره‌ی متوسط})} \qquad (۱۳–۸)$$

که در آن $P_r(selection\ of\ x)$ احتمال انتخاب ذره‌ی x جهت عمل بازترکیب است. فشار انتخاب، مقادیر احتمال نسبی با حضور یک ذره‌ی بسیار برازنده در عمل بازترکیب را نشان می‌دهد. در زیر به بحث در مورد ۷ نوع مختلفِ فرایند انتخاب پرداخته‌ایم.

۸–۷–۱ نمونه‌گیری اتفاقی کلی

همان‌طور که پیش از این نیز ذکر شد، روش استاندارد برای فرایند انتخاب در GA و سایر الگوریتم‌های تکاملی، انتخاب چرخ رولت است که با نام انتخاب متناسب با برازندگی نیز از آن یاد می‌شود. شکل ۳–۲، که برای راحتی بار دیگر در شکل ۸–٦ آورده شده است، انتخاب چرخ رولت برای یک جمعیت ٤ عضوه را نشان می‌دهد.

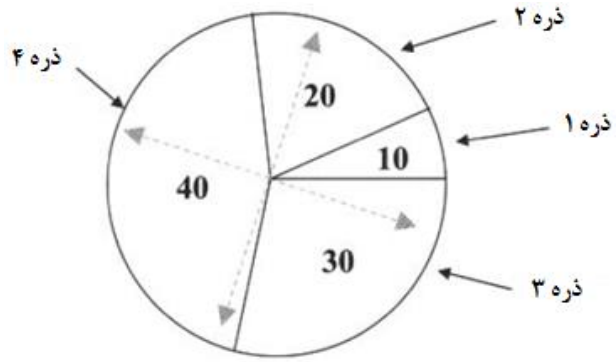


شکل ۸-۱۶ نمایش انتخاب چرخ رولت برای یک جمعیت ٤ عضوه. در این شکل به هر ذره یک تکه اختصاص داده شده است که بزرگی این تکه با برازندگی ذره متناسب است. به همین ترتیب، احتمال انتخاب هر ذره به‌عنوان یک والد به بزرگی تکه‌ی مربوط به آن ذره در چرخ رولت بستگی دارد.

یک مشکل بالقوه که در مورد انتخاب چرخ رولت وجود دارد آن است که احتمال انتخاب نشدن بهترین ذره برای فرایند بازترکیب زیاد است. برای مثال، فرض کنید ما چرخ رولت موجود در شکل ۸-۱٦ را برای انتخاب ٤ ذره جهت بازترکیب، ٤ بار می‌چرخانیم. احتمال آنکه بهترین ذره، ذره شماره‌ی ٤، در هیچ یک از ٤ بار چرخش انتخاب نشود برابر $(0.6)^4 = \%13$ است. اگر این اتفاق بیافتد ما تمامی اطلاعات موجود در بهترین ذره‌ی جمعیت را از دست خواهیم داد و به همین علت این مقدار را می‌توان غیرقابل قبول تلقی کرد.

نمونه‌گیری اتفاقی کلی [بِیکر[1]، ۱۹۸۷] در عین استفاده از انتخاب چرخ رولت، این مشکل را حل می‌نماید. در این روش ما به جای ٤ بار چرخاندن چرخاندن چرخ رولت، از یک چرخنده که دارای ٤ نشانگر با فاصله‌ی یکسان است استفاده کرده، آن را بر روی چرخ رولت قرار داده و یکبار می‌چرخانیم. بدین ترتیب با یکبار چرخش ما ٤ ذره به دست خواهیم آورد و تضمین می‌نماید حداقل یکبار ذره‌ی ۳ و حداقل یکبار ذره‌ی ٤ انتخاب گردند چرا که سهم هر دوی آن‌ها از مقدار کل برازندگی بیشتر از ۲۵٪ است. شکل ۸-۱۷ این ایده را نشان می‌دهد.

[1] Baker



شکل ۸–۱۷ انتخاب اتفاقی کلی برای یک جمعیت ٤ عضوه. به هر ذره یک تکه تخصیص یافته است که بزرگی تکه با برازندگی ذره متناسب است. یک چرخنده با ٤ نشانگر با فواصل مساوی یکبار چرخانده می‌شود تا ٤ والد به دست آید.

اگر نمونه‌گیری اتفاقی کلی را به مقادیر برازندگی شکل ۸–۱۷ اعمال کنیم یکی از حالت‌های زیر به دست خواهد آمد:

$$\begin{aligned} &\text{ذره } 4\# \text{ و } \#1, \#2, \#3 \\ &\text{ذره } 4\# \text{ و } \#1, \#3, \#4 \quad \text{یا} \\ &\text{ذره } 4\# \text{ و } \#2, \#3, \#3 \quad \text{یا} \\ &\text{ذره } 4\# \text{ و } \#2, \#3, \#4 \quad \text{یا} \end{aligned} \tag{۸–۱٤}$$

شکل ۸–۱۸ شبه کدِ نمونه‌گیری اتفاقی کلی را نشان می‌دهد. این کد را با کدِ شکل ۳–۵ که برای انتخاب چرخ رولت ارائه شده است مقایسه نمایید. شکل ۸–۱۸ تضمین می‌کند ذره‌ی x_i چیزی بین $N_{i,min}$ بار و $N_{i,max}$ بار انتخاب شود به‌طوری که

$$\begin{aligned} N_{i,min} &= \left\lfloor \frac{Nf_i}{f_{sum}} \right\rfloor \\ N_{i,max} &= \left\lceil \frac{Nf_i}{f_{sum}} \right\rceil \end{aligned} \tag{۸–۱۵}$$

که در آن $\lfloor \alpha \rfloor$ بزرگترین عدد صحیحی است که کوچکتر یا مساوی α است و $\lceil \alpha \rceil$ نیز کوچکترین عدد صحیحی است که بزرگتر یا مساوی α است.

ذره iام در جمعیت $= x_i$، $i \in [1, N]$
برازندگی $f_i \leftarrow x_i$
$$f_{sum} \leftarrow \sum_{i=1}^{N} f_i$$
یک عدد با توزیع یکنواخت $r \in [0, \frac{f_{sum}}{N}]$ تولید کن
$f_{accum} \leftarrow 0$
والدین $\leftarrow \emptyset$
$k \leftarrow 0$
تا زمانی که $|\text{والدین}| < N$
$k \leftarrow k + 1$
$f_{accum} \leftarrow f_{accum} + f_k$
تا زمانی که $f_{accum} > r$
والدین $\leftarrow$ والدین $\cup\, x_k$
$r \leftarrow r + f_{sum}/N$
پایان حلقه
والد بعد

شکل ۸–۱۸ شبه کد برای انتخاب N ذره از میان N ذره با استفاده از نمونه‌گیری اتفاقی کلی. در این کد برای تمامی $i \in [1, N]$ فرض شده است ، $f_i \geq 0$ است.

۸–۷–۲ انتخاب بیش از حد[1]

انتخاب بیش از حد روشی است که اولین بار توسط جان کوزا در متن مربوط به برنامه‌نویسی ژنتیک مطرح شد [کوزا، ۱۹۹۲، فصل ٦]. این روش با وزن‌دهی ذرات بسیار برازنده به‌صورت غیرمتناسب، روش چرخ رولت را اصلاح کرده و بدین ترتیب احتمال انتخاب ذرات بسیار برازنده را افزایش می‌دهد. در نسخه‌ی ارائه شده توسط کوزا، بهترین ذرات جمعیت که ۳۲٪ از کل جمعیت را تشکیل می‌دهند، ۸۰٪ احتمال انتخاب شدن داشته و باقی ٦۸٪ جمعیت تنها از ۲۰٪ احتمال انتخاب شدن برخوردار هستند. دقیق بودن درصدها چندان حائز اهمیت نیست، بلکه خاصیت کلیدی این روش آن است که ذرات برازنده دارای احتمال انتخاب نامتناسب با برازندگی‌شان هستند. این موضوع در واقع یک نوع از مقیاس‌دهی برازندگی است [گلدبِرگ، ۱۹۸۹a].

کوزا سه نوع مختلف از فرایند انتخاب را برای برنامه‌نویسی ژنتیک امتحان کرد و دریافت که انتخاب چرخ رولت بدترین عملکرد را داشته، انتخاب مسابقه‌ای کمی بهتر بوده و انتخاب بیش از حد بهترین عملکرد

[1] Over-Selecting

را دارد [کوزا، ۱۹۹۲، فصل ۲۵]. با این حال، این موضوع ممکن است به دلیل اندازه‌های جمعیت بزرگی باشد که معمولاً در GP به کار می‌روند. برای اندازه جمعیت‌های کوچکتر، انتخاب بیش از حد در مراحل اولیه‌ی تکامل که واریانس برازندگی جمعیت بسیار بزرگ است، فشار انتخاب زیادی را اعمال می‌نماید (معادله‌ی (۸–۱۳) را ببینید) اما در ادامه و در مراحل بعدی تکامل که واریانس برازندگی کوچک است، این موضوع مفید واقع خواهد شد.

۸–۷–۳ مقیاس‌گذاری سیگما[1]

مقیاس‌گذاری سیگما مقادیر برازندگی را نسبت انحراف استاندارد برازندگی کل جمعیت نرمالیزه می‌نماید. مقادیر مقیاس شده‌ی برازندگی عبارت خواهند بود از

$$f'(x_i) = \begin{cases} \max\left[1 + \frac{f(x_i) - \bar{f}}{2\sigma}, \epsilon\right] & \text{اگر } \sigma \neq 0 \\ 1 & \text{اگر } \sigma = 0 \end{cases} \qquad (۱۶–۸)$$

که در آن $f(x_i)$ برازندگی iاُمین ذره‌ی جمعیت، $\overline{f}$ میانگین مقادیر برازندگی، σ انحراف استاندارد مقادیر برازندگی و ϵ نیز مینیمم مقدار مجازِ مقادیر برازندگی مقیاس شده است که غیرمنفی بوده و توسط کاربر تعیین می‌شود.

توجه داشته باشید که این جمله که σ انحراف استاندارد مقادیر برازندگی است یک جمله‌ی مبهم است. اگر مقادیر برازندگی بدون نویز بوده و ما بخواهیم انحراف استاندارد یک سری از مقادیر برازندگی را که خود اندازه گرفته‌ایم، محاسبه نماییم باید از فرمول زیر استفاده نماییم

$$\sigma = \left(\frac{1}{N}\sum_{i=1}^{N}\left(f(x_i) - \bar{f}\right)^2\right)^{1/2} \qquad (۱۷–۸)$$

با این حال، اگر مقادیر برازندگی دارای نویز بوده و یا نمونه‌هایی از یک توزیع احتمالی باشند، آنگاه یک تخمین بدون گرایش از انحراف استاندارد مقادیر برازندگی را می‌توان به‌صورت زیر محاسبه نمود [سایمون، ۲۰۰۶، مسئله‌ی ۳–۶]:

$$\sigma = \left(\frac{1}{N-1}\sum_{i=1}^{N}\left(f(x_i) - \bar{f}\right)^2\right)^{1/2} \qquad (۱۸–۸)$$

[1] Sigma Scaling

مثال ۸–۷

فرض کنید یک جمعیت ٤ عضوه با مقادیر برازندگی زیر در اختیار داریم

$$f(x_1) = 10, \quad f(x_2) = 5$$
$$f(x_3) = 40, \quad f(x_4) = 15 \qquad (۱۹–۸)$$

انتخاب چرخ رولت احتمالات زیر را جهت انتخاب شدن به ذرات نسبت می‌دهد

$$\Pr(x_1) = 14\%, \quad \Pr(x_2) = 7\%$$
$$\Pr(x_3) = 57\%, \quad \Pr(x_4) = 22\% \qquad (۲۰–۸)$$

با استفاده از معادله‌ی (۸–۱۸)، مقادیر میانگین و انحراف استاندارد برابرند با $\overline{f} = 17.5$ و $\sigma = 15.5$. با استفاده از معادله‌ی (۱۸–٦) مقادیر مقیاس شده‌ی برازندگی برابر خواهند بود با:

$$f'(x_1) = 0.76, \quad f'(x_2) = 0.60$$
$$f'(x_3) = 1.72, \quad f'(x_4) = 0.92 \qquad (۲۱–۸)$$

اگر از این مقادیر مقیاس شده‌ی برازندگی استفاده نماییم، احتمالات گزینش ذرات با استفاده از انتخاب چرخ رولت برابر خواهد بود با

$$\Pr(x_1) = 19\%, \quad \Pr(x_2) = 15\%$$
$$\Pr(x_3) = 43\%, \quad \Pr(x_4) = 22\% \qquad (۲۲–۸)$$

با مقایسه‌ی مقادیر معادلات (۸–۲۰) و (۸–۲۲) می‌توان دید که مقیاس‌گذاری سیگما احتمال انتخاب ذرات با مقادیر برازندگی بسیار متفاوت را به هم نزدیک می‌نماید.

مثال ۸–۸

به‌عنوان یک مثال دیگر، فرض کنید ٤ ذره با مقادیر برازندگی زیر در اختیار داریم:

$$f(x_1) = 15, \quad f(x_2) = 25$$
$$f(x_3) = 20, \quad f(x_4) = 10 \qquad (۲۳–۸)$$

با استفاده از انتخاب چرخ رولت، مقادیر احتمال جهت گزینش ذرات برابر مقادیر زیر خواهد بود:

$$\Pr(x_1) = 21\%, \quad \Pr(x_2) = 36\%$$
$$\Pr(x_3) = 29\%, \quad \Pr(x_4) = 14\% \qquad (۲٤–۸)$$

مقدار میانگین و انحراف استاندارد برای مقادیر برازندگی معادله‌ی (۸–۲۳) عبارتند از $\overline{f} = 17.5$ و $\sigma = 6.5$. با استفاده از معادله‌ی (۸–۱٦) مقادیر مقیاس شده‌ی برازندگی برابر خواهند بود:

$$f'(x_1) = 0.81, \quad f'(x_2) = 1.58$$
$$f'(x_3) = 1.19, \quad f'(x_4) = 0.42 \qquad (۲۵–۸)$$

حال اگر از این مقادیر در الگوریتم انتخاب چرخ رولت استفاده نماییم، احتمالات گزینش زیر به دست خواهد آمد

$$\Pr(x_1) = 20\%, \quad \Pr(x_2) = 40\%$$
$$\Pr(x_3) = 30\%, \quad \Pr(x_4) = 10\% \qquad (۲۶–۸)$$

مقایسه‌ی معادلات (۸–۲٤) و (۸–۲٦) نشان می‌دهد که مقیاس‌گذاری سیگما احتمال انتخاب ذرات با برازندگی نزدیک به هم را پخش می‌نماید.

۸–۷–۴ انتخاب رتبه– محور[1]

انتخاب رتبه– محور، که با نام وزن‌دهی رتبه‌ای نیز نامیده می‌شود، ذرات موجود در جمعیت را از بهترین تا بدترین مرتب کرده و فرایند انتخاب را به جای مقدار مطلق برازندگی‌شان بر اساس رتبه‌شان انجام می‌دهد [وایتلی، ۱۹۸۹]. برای مثال، فرض کنید جمعیتی از ٤ ذره با مقادیر برازندگیِ معادله‌ی (۸–۱۹) و احتمالات گزینشِ معادله‌ی (۸–۲۰) که از انتخاب چرخ رولت به دست آمده‌اند در اختیار داریم. انتخاب رتبه– محور، ذرات را بر اساس مقادیر برازندگی‌شان رتبه‌بندی می‌نماید، بدین ترتیب که بهترین ذره دارای رتبه‌ی N بوده (N اندازه‌ی جمعیت است) و بدترین ذره دارای رتبه‌ی ۱ خواهد بود:

$$R(x_1) = 2, \quad R(x_2) = 1$$
$$R(x_3) = 4, \quad R(x_4) = 3 \qquad (۲۷–۸)$$

سپس فرایند انتخاب به جای مقادیر برازندگی، بر اساس رتبه‌ها صورت می‌پذیرد.

مثال ۸–۹

فرض کنید از انتخاب رتبه– محور به همراه انتخاب چرخ رولت برای رتبه‌های معادله‌ی (۸–۲۷) استفاده می‌نماییم. در این صورت احتمالات گزینش به‌صورت زیر به دست خواهد آمد:

$$\Pr(x_1) = 20\%, \quad \Pr(x_2) = 10\%$$
$$\Pr(x_3) = 40\%, \quad \Pr(x_4) = 30\% \qquad (۲۸–۸)$$

می‌توان دید هنگامی که مقادیر برازندگی مانند معادله‌ی (۸–۱۹) بسیار متفاوت از یکدیگر هستند، انتخاب رتبه– محور احتمالات گزینش را متعادل‌تر کرده و به هم نزدیکتر می‌سازد. این کار از غلبه‌ی ذرات بسیار

[1] Rank-Based Selection

برازنده بر جمعیت در مراحل اولیه‌ی تکامل جلوگیری کرده و در نتیجه از همگرایی نارس جلوگیری به عمل می‌آورد.

مثال ۸–۱۰

به‌عنوان یک مثال دیگر فرض کنید ٤ ذره با مقادیر برازندگی معادله‌ی (۸–۲۳) و احتمالاتِ گزینشِ چرخ رولتِ معادله‌ی (۸–۲٤) در اختیار داریم. انتخاب رتبه– محور ذرات را به‌صورت زیر رتبه‌بندی می‌نماید:

$$R(x_1) = 2, \quad R(x_2) = 4$$
$$R(x_3) = 3, \quad R(x_4) = 1 \qquad (۸–۲۹)$$

اگر از انتخاب رتبه– محور به همراه انتخاب چرخ رولت استفاده نماییم، آنگاه معادله‌ی (۸–۲۹) به مقادیر احتمال گزینش زیر منجر خواهد شد:

$$\Pr(x_1) = 20\%, \quad \Pr(x_2) = 40\%$$
$$\Pr(x_3) = 30\%, \quad \Pr(x_4) = 10\% \qquad (۸–۳۰)$$

می‌توان دید هنگامی که مقادیر برازندگی مانند معادله‌ی (۸–۲۳ بسیار نزدیک به هم هستند،) انتخاب رتبه– محور مقادیر احتمال گزینش را پخش می‌نماید. این کار باعث می‌شود پس از آنکه جمعیت شروع به همگرایی کرد، تفاوت بیشتری میان ذرات مشابه حاصل شود.

ما می‌توانیم گستردگی احتمالات گزینش را با عبور دادن رتبه‌ها از میان یک تابع غیرخطی، تنظیم نماییم. برای مثال، اگر بخواهیم تفاوت بیشتری میان احتمالات گزینش ذرات وجود داشته باشد، می‌توانیم پیش از آنکه از انتخاب چرخ رولت استفاده نماییم، مقادیر رتبه‌ها را به توان ۲ برسانیم. در این صورت معادله‌ی (۸–۲۹) به‌صورت زیر درخواهد آمد:

$$R^2(x_1) = 4, \quad R^2(x_2) = 16$$
$$R^2(x_3) = 9, \quad R^2(x_4) = 1 \qquad (۸–۳۱)$$

استفاده از مقادیر بالا در انتخاب چرخ رولت، احتمالات گزینش زیر را نتیجه خواهد داد

$$\Pr(x_1) = 13\%, \quad \Pr(x_2) = 53\%$$
$$\Pr(x_3) = 30\%, \quad \Pr(x_4) = 4\% \qquad (۸–۳۲)$$

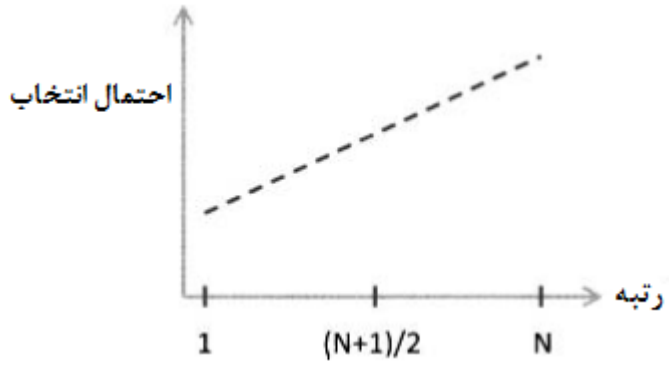
می‌توان دید که به توان ۲ رساندن مقادیر رتبه‌ها باعث گستردگی بیشتر احتمالات گزینش نسبت به معادله‌ی (۸–۳۰) خواهد شد. سایر انواع عملیات بر روی رتبه‌ها (مانند ریشه‌ی دوم گرفتن) می‌تواند باعث توزیع یکنواخت‌تر احتمالات گزینش شود.

۸–۷–۵ رتبه‌بندی خطی

رتبه‌بندی خطی تعمیمی است از انتخاب رتبه- محور. در رتبه‌بندی خطی، احتمال انتخاب ذره‌ی x_i برابرست با

$$\Pr(x_i) = \alpha + \beta R(x_i) \qquad (۳۳–۸)$$

که در آن $R(x_i)$ رتبه‌ی x_i است که در زیربخش ۸–۷–٤ تعریف نمودیم. α و β نیز پارامترهایی هستند که توسط کاربر تعیین می‌شوند. شکل ۸–۱۹ احتمال انتخاب را با اندازه‌ی جمعیتی برابر N نشان می‌دهد. هرچه شیب خط زیادتر باشد، فشار انتخاب بیشتر خواهد بود.



شکل ۸–۱۹ این شکل روش رتبه‌بندی خطی را برای فرایند انتخاب در یک الگوریتم تکاملی با اندازه‌ی جمعیتی برابر N نشان می‌دهد. بدترین ذره دارای رتبه‌ی ۱ بوده و بهترین ذره دارای رتبه‌ی N می‌باشد.

از آنجایی که بهترین ذره دارای رتبه‌ی N بوده و ذره‌ی متوسط دارای رتبه‌ی $(N+1)/2$ است، فشار انتخاب از معادله‌ی (۸–۱۳) برابر خواهد بود با

$$\phi = \frac{\alpha + \beta N}{\alpha + \beta(N+1)/2} \qquad (۳٤–۸)$$

اگر مقادیر احتمالات انتخاب را به‌گونه‌ای که جمعشان برابر با ۱ شود نرمالیزه نماییم خواهیم داشت

$$\sum_{i=1}^{N}(\alpha + \beta i) = \alpha N + \frac{\beta N(N+1)}{2} = 1 \qquad (۳۵–۸)$$

اگر فشار انتخاب (ϕ) خاصی مد نظرمان باشد، می‌توانیم معادلات (۸–۳٤) و (۸–۳۵) را برای α و β حل کرده و معادلات زیر را به دست آوریم

$$\alpha = \frac{2N - \phi(N+1)}{N(N-1)}$$
$$\beta = \frac{2(\phi - 1)}{N(N-1)} \qquad (۳۶-۸)$$

این معادلات به ما می‌گویند α و β را چگونه مقداردهی کنیم تا به یک فشار انتخاب خاص دست یابیم. از آنجا که در معادله‌ی (۸-۳۳)، $\Pr(x_i)$ تابعی است خطی از $R(x_i)$ و با فرض اینکه تمام احتمالات غیرمنفی هستند داریم:

$$\Pr\left(x \text{ متوسط}\right) = \frac{1}{2}\left[\Pr\left(x \text{ بدترین}\right) + \Pr\left(x \text{ بهترین}\right)\right]$$
$$\geq \frac{1}{2}\Pr(x \text{ بهترین}) \qquad (۳۷-۸)$$

با ترکیب معادله‌ی بالا با تعریف فشار انتخاب از معادله‌ی (۸-۱۳)، می‌توان دید که:

$$\phi = \frac{\Pr(x \text{ بهترین})}{\Pr(x \text{ متوسط})} \leq 2 \qquad (۳۸-۸)$$

اگر سعی کنیم $\phi > 2$ قرار دهیم، آنگاه Pr(x بدترین) کمتر از ۰ خواهد بود. به‌طور معمول ما در مراحل اولیه‌ی الگوریتم تکاملی به دنبال فشار انتخاب کم هستیم تا از همگرایی نارس جلوگیری به عمل آید و از سوی دیگر در مراحل پایانی الگوریتم تکاملی به دنبال فشار انتخاب بالا هستیم تا از ذرات بسیار برازنده نهایت استفاده را ببریم.

رتبه‌بندی خطی و انتخاب چرخ رولت

یکی از فواید رتبه‌بندی خطی آن است که اگر آن را به همراه انتخاب چرخ رولت استفاده نماییم، نیازی به استفاده از یک حلقه در برنامه‌ی کامپیوتری‌مان، مانند شکل ۳-۵ نخواهیم داشت. برای اینکه ببینید چگونه می‌توان از حلقه پرهیز کرد، فرض کنید یک عدد اتفاقی $r \sim U[0,1]$ برای عمل انتخاب تولید نماییم. این یعنی می‌خواهیم mاُمین ذره را در جایی انتخاب نماییم که:

$$\sum_{i=1}^{m} \Pr(x_i) \approx r$$
$$\sum_{i=1}^{m} (\alpha + \beta R(x_i)) \approx r \qquad (۳۹-۸)$$
$$\alpha m + \beta m(m+1)/2 \approx r$$

اما معادله‌ی بالا یک معادله‌ی درجه دو ساده برای m است که می‌توان آن را به‌صورت زیر حل نمود

$$m = \frac{-2\alpha - \beta + \sqrt{(2\alpha+\beta)^2 + 8\beta r}}{2\beta} \qquad (٤٠-٨)$$

صدالبته که از آنجایی که m فقط می‌تواند عدد صحیح باشد، باید سمت راست معادله‌ی (٨-٤٠) را به نزدیکترین عدد صحیح گرد نماییم. بدین ترتیب می‌توانیم رتبه‌بندی خطی را همراه با انتخاب چرخ رولت پیاده‌سازی نماییم بی‌آنکه نیازی به وجود حلقه داشته باشیم. این در حالی است که الگوریتم انتخاب چرخ رولت استاندارد خود به حلقه نیاز دارد.

یکی از مضرات رتبه‌بندی خطی آن است که باید مقادیر برازندگی را رتبه‌بندی نماییم، چیزی که در انتخاب چرخ رولت استاندارد نیازی به آن نبود. اما اگر از الگوریتم تکاملی حالت- ماندگار، که در آن در هر نسل تعداد کمی فرزند تولید می‌شود، استفاده نماییم آنگاه نیازی به مرتب‌سازی و رتبه‌بندی کامل در هر نسل نخواهد بود. به‌طور خلاصه، هیچ ضرر یا فایده‌ی روشنی در استفاده از رتبه‌بندی خطی نسبت به انتخاب چرخ رولت وجود ندارد. این موضوع بیش از هر چیز به جنبه‌های دیگرِ پیاده‌سازی الگوریتم تکاملی و همچنین ترجیح کاربر بستگی دارد.

فشار انتخاب توسط کاربر تعیین گردد $\phi \in (1,2)$
معادله‌ی (8.26) را برای α و β حل کن
جمعیت الگوریتم تکاملی که بر اساس برازندگی (صعودی) مرتب شده است $= \{x_i\}$
یک عدد با توزیع یکنواخت تولید کن $r \in (0,1)$
معادله‌ی (8.40) را برای m (اندیس والد انتخاب شده)، حل کن

شکل ۸-۲۰ شبه کد برای انتخاب یک والد از میان N ذره با استفاده از رتبه‌بندی خطی و انتخاب چرخ رولت.

۸-۷-۶ انتخاب مسابقه‌ای

انتخاب مسابقه‌ای هزینه‌ی محاسباتیِ فرایند انتخاب را کاهش می‌دهد. شکل ۳-۵ نشان می‌دهد که انتخابِ N والد از میان N ذره‌ی جمعیت با استفاده از چرخ رولت، به حلقه‌های تودرتو نیاز داشته که این حلقه‌ها برای اندازه‌های جمعیت بزرگ از لحاظ محاسباتی پرهزینه هستند. در انتخاب مسابقه‌ای ما τ ذره را به‌صورت اتفاقی از میان ذرات موجود در جمعیت انتخاب می‌کنیم. $\tau \geq 2$ پارامتری است که توسط کاربر تعیین می‌شود. سپس مقادیر برازندگی ذرات انتخاب شده را با یکدیگر مقایسه کرده و بهترین ذره را برای عمل بازترکیب برمی‌گزینیم.

برای تحلیل انتخاب مسابقه‌ای، فشار انتخاب تعریف شده در معادله‌ی (۸-۱۳) را در نظر بگیرید. اگر برازنده‌ترین ذره برای مسابقه انتخاب شود، آنگاه احتمال انتخاب شدنش برای عمل بازترکیب برابر ۱۰۰٪ خواهد بود. اگر یک ذره‌ی معمولی مانند x انتخاب گردد، آنگاه باید از $\tau - 1$ ذره‌ی دیگرموجود در مسابقه بهتر باشد تا جهت عمل بازترکیب انتخاب شود. در این صورت احتمال آنکه x از هر یک از دیگر ذرات شرکت‌کننده در مسابقه برازنده‌تر باشد تقریبا برابر ۵۰٪ خواهد بود (چرا؟). در این صورت، احتمال انتخاب شدن x جهت عمل بازترکیب برابر $(1/2)^{\tau-1}$ خواهد بود. در این صورت معادله‌ی (۸-۱۳) به معادله‌ی زیر تبدیل خواهد شد:

$$\phi = 2^{\tau-1} \qquad (۸-۴۱)$$

می‌توان دید که در انتخاب مسابقه‌ای، با افزایش τ فشار انتخاب نیز افزایش می‌یابد.

روش انتخاب مسابقه‌ای بالا، انتخاب مسابقه‌ای سخت نام دارد چرا که بهترین ذره‌ی جمعیت همیشه در مسابقه برنده خواهد بود. یک مسابقه‌ی نرم مسابقه‌ای است که در آن احتمال برنده شدن بهترین ذره در مسابقه برابر $p < 1$ است [ریوز و رو، ۲۰۰۳، بخش ۲-۳]. سایر ذرات که دارای میزان کمتری برازندگی هستند نیز دارای احتمالی برای برنده شدن در مسابقه هستند. با ثابت در نظر گرفتن اندازه‌ی مسابقه، مسابقات نرم دارای فشار انتخاب کمتری نسبت به مسابقات سخت هستند.

یکی از فواید استفاده از انتخاب مسابقه‌ای نسبت به سایر روش‌های گزینش آن است که این روش را می‌توان تنها با مقایسه‌ی درونی ذرات به کار انداخت. بدین معنا که برای به کار بستن انتخاب مسابقه‌ای نیازی به محاسبه‌ی برازندگی مطلق ذرات نیست، بلکه کافی است مقادیر نسبی برازندگی ذرات حاضر در مسابقه را بدانیم.

۸-۷-۷ الگوریتم‌های تکاملی stud

بسیاری از الگوریتم‌های تکاملی انتخاب ذرات جهت عمل بازترکیب را بر اساس مقادیر برازندگی نسبی ذرات انجام می‌دهند. تمام روش‌هایی که تاکنون در مورد آن‌ها بحث نموده‌ایم از این قانون پیروی می‌کنند. با این حال، در الگوریتم‌های تکاملی *stud*، ما در هر نسل بهترین ذره را برای هر عمل بازترکیب انتخاب می‌نماییم. بهترین ذره در هر نسل *stud* نام دارد. سپس والدین دیگر را نیز به روش عادی (برای مثال، انتخاب برازندگی- محور، انتخاب رتبه- محور و غیره) انتخاب کرده، آن‌ها را با *stud* ترکیب می‌کنیم تا فرزندان به

وجود آیند. این ایده در ابتدا به *GA*ها اعمال شد. این گونه *GA*ها را ***stud GA*** می‌نامند [خطیب[1] و فِلمینگ[2]، ۱۹۹۸]. با اضافه نمودن منطق ***stud*** به یک *GA*، باید *GA* استاندارد از شکلِ ۳-٦ را اصلاح نمود تا به ***stud*** *GA* از شکل ۸-۲۱ رسید.

{جمعیت تولید شده به صورت اتفاقی} ← والدین
تا زمانی که شرایط توقف برقرار نشده است
برازندگی هر یک از والدین حاضر در جمعیت را حساب کن
∅ ← فرزندان
برازنده‌ترین والد ← x_1
تا زمانی که $E -$ |والدین| < |فرزندان|
از مقادیر برازندگی یرای انتخاب احتمالاتی دومین والد x_2 استفاده کن $(x_1 \neq x_2)$
x_1 و x_2 را ترکیب کن تا فرزندان c_1 و c_2 به دست آیند
فرزندان ∪ $\{c_1, c_2\}$ ← فرزندان
حلقه
برخی فرزندان را به صورت اتفاقی دچار جهش کن
فرزندان ← والدین
نسل بعد

شکل ۸-۲۱ شبه کد بالا طرح کلی *stud GA* را نشان می‌دهد.

مثال ۸-۱۱

ما در این مثال به شبیه‌سازی یک *GA* پیوسته بر روی یک مجموعه از مسائل محک ۲۰ بعدی از ضمیمه‌ی ج، با و بدون گزینه‌ی ***stud*** می‌پردازیم. ما همچنین از اندازه‌ی جمعیتی برابر ۵۰، محدودیت نسلی برابر ۵۰ و نرخ جهشِ ۱٪ برای هر یک از ۲۰ خصیصه‌ی موجود در هر ذره در هر نسل استفاده می‌نماییم. پیاده‌سازی جهش نیز بدین گونه صورت می‌پذیرد که متغیر مستقل با یک متغیر دیگر که به‌صورت اتفاقی از یک توزیع یکنواخت میان بیشترین و کمترین مقدار دامنه انتخاب شده است، جایگزین می‌گردد. ما همچنین از یک پارامتر نخبه‌گرایی برابر ۲ استفاده می‌نماییم. این بدان معنی است که ما بهترین دو ذره‌ی هر نسل را برای نسل بعد حفظ می‌نماییم.

جدول ۸-۱ بهترین عملکرد *GA* استاندارد و ***stud GA*** را که بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است را نشان می‌دهد. این جدول نشان می‌دهد که ***stud GA*** کاملا بهتر از *GA* استاندارد

[1] Khatib
[2] Fleming

عمل می‌نماید. برای برخی توابع محک استفاده از گزینه‌ی *stud* باعث بهبود ناگهانی و عظیم در عملکرد می‌شود.

تا به کنون در مقالات تحقیقاتی، الگوریتم تکاملی *stud* عمدتا (و یا منحصرا) به *GA*ها اعمال شده است، اما می‌توان به سادگی آن‌ها را به سایر الگوریتم‌های تکاملی نیز اعمال نمود. همان‌طور که از مقایسه‌ی دو شکل ۳–٦ و ۸–۲۱ نیز پیدا بود، اضافه نمودن منطق *stud* به یک الگوریتم تکاملی نیازمند تغییری ساده در الگوریتم تکاملی است. پیاده‌سازی ساده و عملکرد عالی نشان داده شده در جدول ۸–۱ ما را بر آن می‌دارد تا توجهی جدی به منطق *stud* در الگوریتم‌های تکاملی‌مان داشته باشیم. یک زمینه‌ی تحقیقاتی جالب برای کارهای آینده، به دست آوردن مدل‌های ریاضی برای *GA*ها (فصل ٤) و همچنین سایر الگوریتم‌های تکاملی هنگامی که شامل منطق *stud* هستند، می‌باشد.

جدول ۸–۱ نتایج مثال ۸–۱۱ که عملکرد نسبی یک *GA* را با و بدون گزینه‌ی *stud* نشان می‌دهد. جدول مقادیر مینیمم نرمالیزه شده‌ی پیدا شده توسط دو نسخه‌ی *GA* را که بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. برای تعاریف توابع محک به ضمیمه‌ی ج مراجعه نمایید.

تابع محک	Non-Stud GA	Stud GA
آکلی	۱,٤٤	۱
فلچر	۳,۲٦	۱
گرینوانک	۳,۹٦	۱
مجازات #۱	1.05×10^5	*1*
مجازات #۲	۱٦۰,۸	۱
درجه ٤	۹,۱٤	۱
راستریجین[1]	۱,۹۲	۱
روزنبروک	۳,۹۸	۱
اشوفل ۱,۲	۱,۲٤	۱
اشوفل ۲,۲۱	۱,٦٥	۱
اشوفل ۲,۲۲	۳,۷۰	۱
اشوفل ۲,۲٦	۲,٥٦	۱
کروی	٤,٤۷	۱
پله	٤,۲۳	۱

[1] Rastrigin

۸–۸ بازترکیب

GA ساده از برش تک- نقطه‌ای استفاده می‌نماید. این بخش به بحث در مورد سایر روش‌های بازترکیب هم برای الگوریتم‌های تکاملی پیوسته و هم برای الگوریتم‌های تکاملی گسسته می‌پردازد. توجه داشته باشید که در این کتاب، بازترکیب و برش هر دو دارای یک معنی هستند. برخی از روش‌های بازترکیب ارائه شده در این فصل در [هِررا[1] و همکاران، ۱۹۹۸] به تفصیل مورد بحث قرار گرفته‌اند.

فرض کنید جمعیتی از ذرات $\{x_1, x_2, \dots, x_N\}$ در اختیار داریم. فرض کنید هر یک از این ذرات دارای n خصیصه بوده و ما خصیصه‌ی kاُم از iاُمین ذره را با نماد $x_i(k)$ نمایش می‌دهیم ($k \in [1, n]$). بنابراین می‌توان x_i را با بردار زیر نمایش داد

$$x_i = [x_i(1) \quad x_i(2) \quad \dots \quad x_i(n)] \tag{۸–۴۲}$$

ما یک ذره‌ی فرزند را، که حاصل فرایند بازترکیب است، با حرف y نمایش می‌دهیم. همچنین برای نشان دادن kاُمین خصیصه‌ی فرزند از نماد $y(k)$ استفاده می‌نماییم، پس

$$y = [y(1) \quad y(2) \quad \dots \quad y(n)] \tag{۸–۴۳}$$

۸–۸–۱ برش تک– نقطه‌ای[2] (الگوریتم‌های تکاملی دودویی یا پیوسته)

فرض کنید دو والد x_a و x_b را در اختیار داریم به‌طوری که $a, b \in [1, N]$. برش تک– نقطه‌ای، که با نام‌های برش ساده و برش گسسته نیز شناخته می‌شود، نوعی از برش است که برای اولین بار در GA دوودویی مورد استفاده قرار گرفت (فصل ۳ را ببینید):

$$y(k) \leftarrow [x_a(1) \quad \dots \quad x_a(m) \quad x_b(m+1) \quad \dots \quad x_b(n)] \tag{۸–۴۴}$$

که در آن m نقطه‌ی اتفاقی برش است، بدین معنا که: $m \sim U[0, n]$. اگر $m = 0$ باشد آنگاه y کلونی از x_b خواهد بود. اگر $m = n$ باشد آنگاه y کلونی از x_a خواهد بود. معمولاً از برش تک- نقطه‌ای برای به دست آوردن دو فرزند از دو والد استفاده می‌شود. این کار با انتخاب هر یک از ویژگی‌های فرزند دوم y_2 از والدِ مقابلِ والدی که فرزندی y_1 خصیصه‌های خود را گرفته است، قابل انجام است:

$$\begin{aligned} y_1(k) &\leftarrow [x_a(1) \quad \dots \quad x_a(m) \quad x_b(m+1) \quad \dots \quad x_b(n)] \\ y_2(k) &\leftarrow [x_b(1) \quad \dots \quad x_b(m) \quad x_a(m+1) \quad \dots \quad x_a(n)] \end{aligned} \tag{۸–۴۵}$$

[1] Herrera
[2] Single-point crossover

۸–۸–۲ برش چند– نقطه‌ای[1] (الگوریتم‌های تکاملی دودویی یا پیوسته)

برش دو– نقطه‌ای به معادله‌ی زیر منجر می‌شود

$$y(k) \leftarrow [x_a(1) \quad ... \quad x_a(m_1) \quad x_b(m_1+1) \quad ... \quad x_b(m_2) \quad x_a(m_2+1) \quad ... \quad x_a(n)] \qquad (٤٦–٨)$$

که در آن دو نقطه‌ی برش عبارتند از $m_1 \sim U[0,n]$ و $m_2 \sim U[0,n]$. اگر $m_1 = 0$ بوده و $m_2 = n$ باشد، آنگاه برش دو– نقطه‌ای به برش تک– نقطه‌ای تقلیل پیدا می‌کند. اگر $m_1 = n$ باشد آنگاه y کلونی از x_a خواهد بود. معادله‌ی (۸–٤٦) را می‌توان به برش سه– نقطه‌ای یا برش M– نقطه‌ای برای $M > 2$ تعمیم داد. مانند برش تک– نقطه‌ای، برش چند– نقطه‌ای نیز برای به دست آوردن دو فرزند از دو والد به کار می‌رود.

۸–۸–۳ برش قطعه قطعه شده[2] (الگوریتم تکاملی دودویی یا پیوسته)

برش قطعه قطعه شده [مایکلویکز، ۱۹۹٦، بخش ٤–٦] را می‌توان به‌صورت تعمیمی از برش چند– نقطه در نظر گرفت. فرزند ۱ اولین خصوصیت خود را از والد ۱ می‌گیرد. سپس خصوصیت دوم فرزند ۱ یا با احتمال ρ از والد ۲ گرفته می‌شود و یا با احتمال $\rho - 1$ از والد ۱ گرفته می‌شود. هر بار که یکی از خصوصیات فرزند از یکی از والدین به دست می‌آید، ما برای به دست آوردن خصوصیت بعدی فرزند، با احتمال ρ به والد دیگر سوئیچ می‌کنیم. فرزند ۱ و فرزند ۲ خصوصیات خود را از والدین مختلف می‌گیرند بدین معنی که اگر فرزند ۱ خصوصیت kاُم خود را برای تمامی $k \in [1,n]$ از والد ۱ گرفته باشد، آنگاه فرزند ۲ خصوصیت kاُم خود را از والد ۲ خواهد گرفت. به همین ترتیب اگر فرزند ۱ خصوصیت kاُم را از والد ۲ گرفته باشد فرزند ۲ خصوصیت kاُم خود را از والد ۱ خواهد گرفت. برش قطعه قطعه معادل برش چند– نقطه‌ای است اگر در برش چند– نقطه‌ای تعداد نقاط برش برابر یک عدد اتفاقی باشد. شکل ۸–۲۲ الگوریتمی را برای برش قطعه قطعه به تصویر می‌کشد. احتمال سوئیچ ρ معمولاً برابر با ۰٫۲ در نظر گرفته می‌شود.

[1] Multi-point crossover
[2] Segmented crossover

۸–۸–۴ برش یکنواخت (الگوریتم تکاملی دودویی یا پیوسته)[1]

فرض کنید دو والد در اختیار داریم: x_a و x_b. برش یکنواخت [اَکلی، ۱۹۸۷a]، [مایکلویکز و شونوئِر، ۱۹۹۶] فرزند y را به دست خواهد داد به‌طوری که kاُمین خصیصه‌ی آن عبارتست از

$$y(k) \leftarrow x_{i(k)}(k) \qquad (۴۷–۸)$$

برای هر $k \in [1,n]$، که در آن $i(k)$ به‌صورت اتفاقی از مجموعه‌ی $\{a,b\}$ انتخاب شده است. این یعنی ما هر خصیصه‌ی فرزند را به‌صورت اتفاقی از میان یکی از دو والدش، با احتمال ۵۰٪، انتخاب می‌کنیم.

$S \leftarrow true$
برای $k = 1$ تا n
  اگر S آنگاه
    $c_1(k) \leftarrow p_1(k)$
    $c_2(k) \leftarrow p_2(k)$
  در غیر این صورت
    $c_1(k) \leftarrow p_2(k)$
    $c_2(k) \leftarrow p_1(k)$
  پایان اگر
  $r \leftarrow U[0,1]$
  اگر $r < \rho$ آنگاه $S \leftarrow not\ S$
ویژگی راه‌حل بعدی

شکل ۸–۲۲ برش قطعه قطعه شده برای ذرات n بعدی. p_1 و p_2 دو والد بوده و c_1 و c_2 فرزندانشان می‌باشند.

۸–۸–۵ برش چند– والده[2] (الگوریتم تکاملی دودویی یا پیوسته)

برش چند– والده که هم اکنون در مورد آن به بحث خواهیم پرداخت، تعمیمی است از برش یکنواخت [اِیبِن، ۲۰۰۳]، [اِیبِن و باک، ۱۹۹۸]، [اِیبِن، ۲۰۰۰] و با نام‌های دیگری نیز شناخته می‌شود که از جمله‌ی آن‌ها می‌توان به بازترکیب استخر ژن[3] [باک، ۱۹۹۶]، [باک و همکاران، ۱۹۹۷b]، [موهلنبین[4] و وویگت[5]، ۱۹۹۵]، برش پویشی[6] [اِیبِن و شیپِر[7]، ۱۹۹۶] و بازترکیب چند– جنسی[8] [اِشوفل، ۱۹۹۵] اشاره کرد. در برش چند–

[1] Uniform crossover
[2] Multi-parent crossover
[3] Gene pool recombination
[4] Muhlenbein
[5] Voigt
[6] Scanning crossover
[7] Schipper
[8] Multi-sexual crossover

والده ما هر یک از خصوصیات فرزند را به‌صورت اتفاقی از یکی از والدینش انتخاب می‌کنیم به‌صورتی که تعداد والدین بیشتر از دو است. این ایده اولین بار توسط [بِرِمِرمن[1] و همکاران، ۱۹٦٦] ارائه شد. با استفاده از برش چند- والده داریم

$$y_k \leftarrow x_{i(k)}(k) \qquad (٤٨-٨)$$

برای هر $k \in [1,n]$ که در آن $i(k)$ را به‌صورت اتفاقی از زیرمجموعه‌ی $[1,N]$ انتخاب می‌نماییم (به خاطر آورید که N فرزند بالقوه در جمعیت داریم). هنگام پیاده‌سازی برش چند- والده باید در مورد چند مسئله تصمیم گرفت. برای مثال، در استخر والدین بالقوه چند ذره باید وجود داشته باشد؟ ذرات را با چه معیاری باید برای این استخر انتخاب نمود؟ پس از آنکه این استخر تشکیل شد، ذرات با چه معیاری باید از آن انتخاب شوند؟

۸-۸-۶ برش یکنواخت جهانی[2] (الگوریتم تکاملی دودویی یا پیوسته)

یکی از راه‌های پیاده‌سازی برش چند- والده آن است که هر خصوصیت فرزند را به‌صورت اتفاقی از یکی از والدین انتخاب کنیم به‌طوری که استخر والدین را برابر با کل جمعیت در نظر بگیریم. این کار به بازترکیب یکنواخت جهانی منجر خواهد شد:

$$y_k \leftarrow x_{i(k)}(k) \qquad (٤٨-٨)$$

برای هر $k \in [1,n]$ که در آن $i(k)$ را به‌صورت اتفاقی از زیرمجموعه‌ی $[1,N]$ انتخاب می‌نماییم. یا آنکه می‌توان $i(k)$ را بر اساس برازندگی انتخاب نمود. این یعنی احتمال $i(k)=m$ برای تمامی $k \in [1,n]$ و $m \in [1,N]$، با برازندگی x_m متناسب باشد.

۸-۸-۷ برش شافل[3] (الگوریتم تکاملی دودویی یا پیوسته)

برش شافل ویژگی‌های راه‌حل موجود در والدین را بازآرایی می‌کند [اِشلمن[4] و همکاران، ۱۹۸۹]. ما برای تمامی والدینی که در به وجود آمدن یک فرزند مشخص همکاری می‌کنند از بازآرایی یکسانی استفاده می‌نماییم. سپس از یکی از روش‌های برش (معمولاً برش تک- نقطه‌ای) که پیش از این معرفی شدند برای

[1] Bremermann
[2] Global uniform crossover
[3] Shuffle crossover
[4] Eshelman

به دست آوردن فرزندان استفاده می‌نماییم. ما سپس بازآرایی ویژگی‌های راه‌حل که پیش از این انجام داده بودیم را در فرزندان خنثی می‌کنیم. شکل ۸–۲۳ الگوریتم برش شافل را که با برش تک– نقطه‌ای ترکیب شده است را نشان می‌دهد.

جایگشتی اتفاقی از $\{r_1, \dots, r_n\} \leftarrow \{1, \dots, n\}$
m نقطه برش $\leftarrow U[1, n-1]$
برای $k = 1$ تا m
  $t_1(k) \leftarrow p_1(r_k)$
  $t_2(k) \leftarrow p_2(r_k)$
k بعدی
برای $k = m+1$ تا n
  $t_1(k) \leftarrow p_2(r_k)$
  $t_2(k) \leftarrow p_1(r_k)$
k بعدی
برای $k = 1$ تا n
  $c_1(r_k) \leftarrow t_1(k)$
  $c_2(r_k) \leftarrow t_1(k)$
k بعدی

شکل ۸–۲۳ برش شافل ترکیب شده با برش تک– نقطه‌ای برای والدین n بعدی. p_1 و p_2 والدین بوده، t_1 و t_2 فرزندان قبل از برچیدن بازآرایی و c_1 و c_2 فرزندان بعد از برچیدن بازآرایی می‌باشند.

۸–۸–۸ برش مسطح[1] و برش حسابی[2] (الگوریتم‌های تکاملی پیوسته)

برش مسطح، که با نام برش حسابی نیز شناخته می‌شود، به‌صورت زیر تعریف می‌شود:

$$y(k) \leftarrow U[x_a(k), x_b(k)] \\ = ax_a(k) + (1-\alpha)x_b(k) \qquad (۵۰–۸)$$

که در آن $\alpha \sim U[0,1]$ است. این بدان معنی است که $y(k)$ عددی اتفاقی است که از یک توزیع یکنواخت میان kاُمین ویژگی‌های دو والدش گرفته شده است. این مانند آن است که بگوییم فرزند ترکیبی است خطی از ویژگی‌های دو والدش. گاهی اوقات گفته می‌شود که تفاوت میان برش مسطح و برش حسابی در آن است که برش مسطح یک فرزند و برش حسابی دو فرزند به دست می‌دهد:

[1] Flat crossover
[2] Arithmetic crossover

برش مسطح: $y(k) = \alpha x_a(k) + (1-\alpha)x_b(k)$

برش حسابی: $\begin{cases} y_1(k) = \alpha x_a(k) + (1-\alpha)x_b(k) \\ y_2(k) = (1-\alpha)x_a(k) + \alpha x_b(k) \end{cases}$ (۵۱–۸)

ما همچنین می‌توانیم برای α از تابع چگالی احتمال مثلثی به جای تابع چگالی احتمال یکنواخت استفاده نماییم:

$$\mathrm{PDF}(\alpha) = \begin{cases} 1+\alpha & \text{اگر } -1 \le \alpha < 0 \\ 1-\alpha & \text{اگر } 0 \le \alpha \le 1 \end{cases} \quad (۵۲–۸)$$

که در این صورت معادله‌ی (۸–۵۱) بازترکیب فازی[1] خوانده می‌شود [اِشلمن و شافِر[2]، ۱۹۹۳].

۸–۸–۹ برش مخلوط شده[3] (الگوریتم‌های تکاملی پیوسته)

برش مخلوط شده که با نام‌های برش $BLX-\alpha$ و همچنین برش اکتشافی نیز شناخته می‌شود [هوک[4] و همکاران، ۱۹۹۵]، والدین x_a و x_b را به‌صورت زیر ترکیب می‌نماید:

$$\begin{aligned} &x_{min}(k) \leftarrow \min(x_a(k), x_b(k)) \\ &x_{max}(k) \leftarrow \max(x_a(k), x_b(k)) \\ &\Delta x(k) \leftarrow x_{max}(k) - x_{min}(k) \\ &y_k(k) \leftarrow U[x_{min}(k) - \alpha\Delta x(k), x_{max}(k) + \alpha\Delta x(k)] \end{aligned} \quad (۵۳–۸)$$

که در آن α پارامتری است که توسط کاربر تعیین می‌شود. اگر $\alpha = 0$ باشد آنگاه برش مخلوط شده به برش مسطح تبدیل می‌گردد. اگر $-0.5 < \alpha < 0$ باشد آنگاه برش مخلوط شده دامنه‌ی جستجو را کوچک می‌کند. این موضوع می‌تواند برای استفاده از جمعیت حاضر مفید باشد. اگر $\alpha > 0$ باشد آنگاه برش مخلوط شده دامنه‌ی جستجو را وسیع می‌نماید که این موضوع برای عمل کاوش سودمند است. [هِررا و همکاران، ۱۹۹۸]، $\alpha = 0.5$ را پیشنهاد می‌کند.

[1] Fuzzy recombination
[2] Schaffer
[3] Blended crossover
[4] Houk

۸–۸–۱۰ برش خطی[1] (الگوریتم‌های تکاملی پیوسته)

برش خطی فرزندان را از والدین x_a و x_b به‌صورت زیر به وجود می‌آورد:

$$y_1(k) \leftarrow \left(\frac{1}{2}\right) x_a(k) + (\frac{1}{2}) x_b(k)$$
$$y_2(k) \leftarrow \left(\frac{3}{2}\right) x_a(k) + (\frac{1}{2}) x_b(k) \qquad (۸–۵۴)$$
$$y_3(k) \leftarrow \left(\frac{-1}{2}\right) x_a(k) + (\frac{3}{2}) x_b(k)$$

ما بهترین یا دو ذره‌ی بهتر از سه فرزند را برای نسل بعد نگه می‌داریم. این موضوع به خصوصیات پیاده‌سازی الگوریتم تکاملی بستگی دارد.

۸–۸–۱۱ برش شبیه‌سازی شده‌ی دودویی[2] (الگوریتم‌های تکاملی پیوسته)

برش شبیه‌سازی شده‌ی دودویی (SBX) فرزندان را از دو والد x_a و x_b به‌صورت زیر به وجود می‌آورد [دِب و آگراوال[3]، ۱۹۹۵]:

$$y_1(k) \leftarrow \left(\frac{1}{2}\right) [(1-\beta_k) x_a(k) + (1+\beta_k) x_b(k)]$$
$$y_2(k) \leftarrow \left(\frac{1}{2}\right) [(1+\beta_k) x_a(k) + (1-\beta_k) x_b(k)] \qquad (۸–۵۵)$$

که در آن β_k عددی اتفاقی است که از تابع چگالی زیر تولید می‌شود:

$$PDF(\beta) = \begin{cases} 1/2(\eta+1)\beta^{\eta} \\ 1/2(\eta+1)\beta^{-(\eta+2)} \end{cases} \qquad (۸–۵۶)$$

که در آن η هر عدد حقیقی غیرمنفی است. [دِب و آگراوال، ۱۹۹۵] به بحث در مورد تأثیر η بر روی عملگر SBX می‌پردازد و در کل برای η مقداری میان ۰ و ۵ را پیشنهاد می‌کند. ما می‌توانیم β را با استفاده از الگوریتم زیر تولید نماییم:

[1] Linear crossover
[2] Simulated Binary crossover (SBX)
[3] Agrawal

$$r \leftarrow U[0,1]$$
$$\beta \leftarrow \begin{cases} (2r)^{1/(\eta+1)} & \text{اگر } r \leq 1/2 \\ (2-2r)^{-1/(\eta+1)} & \text{اگر } r > 1/2 \end{cases} \qquad (۸–۵۷)$$

توجه داشته باشید که اگر $\beta = 2\alpha - 1$ باشد، SBX با برش حسابیِ معادله‌ی (۸–۵۱) معادل خواهد شد. ما همچنین می‌توانیم SBX را با مقادیری از β_k پیاده‌سازی نماییم که دارای توزیعی غیر از معادله‌ی (۸–۵۶) هستند.

۸–۸–۱۲ خلاصه

روش‌های بازترکیبی که در این بخش مورد بحث قرار گرفتند در ابتدا برای GA ارائه شدند اما می‌توان آن‌ها را برای هر الگوریتم تکاملی دیگری نیز به کار برد. محققین همچنین روش‌های برش دیگری نیز ارائه نموده‌اند [هِررا و همکاران، ۱۹۹۸] اما روش‌های ارائه شده در بالا ایده‌های کلی را به دست می‌دهند. علاوه بر این، ما می‌توانیم برخی از این روش‌ها را با هم ترکیب کرده و الگوریتم بازترکیب مورد نظر خودمان را به دست آوریم. هیچ برتری مشخصی میان این روش‌ها وجود ندارد. یک روش برش ممکن است در مورد یک مسئله‌ی خاص بهترین عملکرد را داشته باشد در حالی که یک روش برش دیگر در مورد یک مسئله‌ی خاص دیگر بهترین عملکرد را داشته باشد. با این حال، هرچند نمی‌توانیم بگوییم کدام روش برش بهترین است، اما معمولاً می‌توانیم بگوییم برش تک- نقطه‌ای یکی از بدترین‌هاست.

۸–۹ جهش

در الگوریتم‌های تکاملی دودویی، جهش عملی سرراست است. اگر جمعیتی از N ذره در اختیار داشته باشیم که در آن هر ذره با n بیت نشان داده می‌شود و همچنین نرخ جهش ما برابر ρ باشد، آنگاه در انتهای هر نسل ما هر بیت از هر ذره را با احتمال ρ تعییر می‌دهیم (معادله‌ی (۳–۶) را ببینید).

در الگوریتم‌های تکاملی پیوسته ما گزینه‌های بیشتری برای جهش در اختیار داریم. در این الگوریتم‌ها ما همچنان نرخ جهش را با ρ نشان می‌دهیم و همچنان $x_i(k)$ را با احتمال ρ برای هر i و هر k اصلاح (جهش) می‌نماییم. اما اگر تصمیم به اصلاح $x_i(k)$ گرفته شود، باید در مورد چگونگی اصلاح آن نیز تصمیم گرفته شود. یک راه این است که $x_i(k)$ را از یک توزیع گاوسی یا یکنواخت که مقدار متوسط آن در وسط دامنه‌ی جستجو قرار دارد، تولید نماییم. یک راه دیگر ان است که $x_i(k)$ را از یک توزیع گاوسی یا یکنواخت که مقدار متوسط آن در مقداری جهش نیافته از $x_i(k)$ قرار دارد تولید نماییم. ما در ادامه به تعریف این گزینه‌ها

پرداخته و از علامت‌های $x_{min}(k)$ و $x_{max}(k)$ برای نشان دادن محدوده‌ی دامنه‌ی جستجو برای kاُمین بعد در مسئله‌ی بهینه‌سازی خود استفاده می‌نماییم.

۸–۹–۱ جهش یکنواخت متمرکز در $x_i(k)$[1]

جهش یکنواخت متمرکز در $x_i(k)$ را می‌توان برای $i \in [1,N]$ و $k \in [1,n]$ به‌صورت زیر نوشت

$$x_i(k) \leftarrow \begin{cases} r \leftarrow U[0,1] & \\ x_i(k) & \text{اگر } r \geq \rho \\ U[x_i(k) - \alpha_i(k), x_i(k) + \alpha_i(k)] & \text{اگر } r < \rho \end{cases} \quad (۵۸–۸)$$

که در آن $\alpha_i(k)$ پارامتری است که توسط کاربر تعیین شده و بزرگی جهش را نشان می‌دهد. ما معمولاً $\alpha_i(k)$ را تا حد ممکن بزرگ انتخاب می‌نماییم اما به‌گونه‌ای که مطمئن باشیم جهش در درون دامنه‌ی جستجو باقی می‌ماند

$$\alpha_i(k) = \min(x_i(k) - x_{min}(k), x_{max}(k) - x_i(k)) \quad (۵۹–۸)$$

۸–۹–۲ جهش یکنواخت متمرکز در مرکز محدوده‌ی جستجو[2]

جهش یکنواخت متمرکز در مرکز محدوده‌ی جستجو را می‌توان برای هر $i \in [1,N]$ و هر $k \in [1,n]$ به‌صورت زیر نوشت

$$x_i(k) \leftarrow \begin{cases} r \leftarrow U[0,1] & \\ x_i(k) & \text{اگر } r \geq \rho \\ U[x_{min}(k), x_{max}(k)] & \text{اگر } r < \rho \end{cases} \quad (۶۰–۸)$$

۸–۹–۳ جهش گاوسی متمرکز در $x_i(k)$[3]

جهش گاوسی متمرکز در $x_i(k)$ را می‌توان برای هر $i \in [1,N]$ و هر $k \in [1,n]$ به‌صورت زیر نوشت

[1] Uniform Mutation centered at the $x_i(k)$
[2] Uniform Mutation centered at the middle of search domain
[3] Gaussian Mutation centered at $x_i(k)$

$$r \leftarrow U[0,1]$$
$$x_i(k) \leftarrow \begin{cases} x_i(k) & \text{اگر } r \geq \rho \\ max\left[\min(x_{max}(k), N\left(x_i(k), \sigma_i^2(k)\right), x_{min}(k)\right] & \text{اگر } r < \rho \end{cases} \qquad (۶۱-۸)$$

که در آن $\sigma_i(k)$ پارامتری است که توسط کاربر تعیین شده و با بزرگی جهش متناسب است. مقادیر min و max تضمین می‌کند مقادیر جهش یافته‌ی $x_i(k)$ در محدوده‌ی جستجو باقی می‌ماند. این نوع از جهش مانند عملگرهای جستجویی است که در EP و ES از آن‌ها استفاده می‌نماییم.

۸-۹-۴ جهش گاوسی متمرکز در مرکز محدوده‌ی جستجو[1]

جهش گاوسی متمرکز در مرکز محدوده‌ی جستجو را می‌توان برای هر $i \in [1, N]$ و هر $k \in [1, n]$ به‌صورت زیر نوشت

$$r \leftarrow U[0,1]$$
$$x_i(k) \leftarrow \begin{cases} x_i(k) & \text{اگر } r \geq \rho \\ max\left[\min(x_{max}(k), N\left(c_i(k), \sigma_i^2(k)\right), x_{min}(k)\right] & \text{اگر } r < \rho \end{cases} \qquad (۶۲-۸)$$

که در آن $c_i(k) = (x_{min}(k) + x_{max}(k))/2$ بوده و مرکز محدوده‌ی جستجو را مشخص می‌نماید. $\sigma_i(k)$ نیز پارامتری است که توسط کاربر تعیین شده و با بزرگی جهش متناسب است. مقادیر min و max تضمین می‌کند مقادیر جهش یافته‌ی $x_i(k)$ در محدوده‌ی جستجو باقی می‌ماند.

۸-۱۰ نتیجه‌گیری

در این فصل ما بسیاری از تنوعات موجود در الگوریتم‌های تکاملی را بررسی نمودیم، اما در واقع بحث خود را به معمول‌ترین این تنوعات محدود نمودیم. روش‌های بسیار دیگری برای اصلاح نمودن الگوریتم‌های تکاملی وجود دارد که از جمله‌ی آن می‌توان به استفاده از اندازه‌های جمعیت متغیر [هو[2] و همکاران، ۲۰۱۰]، زیر- جمعیت‌های متعامل[3] [لی[4] و همکاران، ۲۰۰۹]، دیپلوئیدی و پلی‌پلوئیدی[5] (که در آن هر ذره نماینده‌ی چند راه‌حل نامزد است) [وانگ و همکاران، ۲۰۰۹] و مدل‌سازی جنسیت[6] (که عمل برش را به والدین با

[1] Gaussian Mutation centered at the middle of search domain
[2] Hu
[3] Interacting sub-populations
[4] Li
[5] Diploidy and polyploidy
[6] Gender Modelling

جنس مخالف محدود می‌کند) [میچل، ۱۹۹۸]، اشاره نمود. محققین همچنین اصلاحات دیگری را نیز پیشنهاد نموده‌اند که در این کتاب فضای کافی برای پرداختن به تمامی آن‌ها وجود ندارد.

با در نظر گرفتن تمامی تنوعات موجود، شاید بهتر باشد میان یک الگوریتم تکاملی و یک نمونه‌ی الگوریتم تکاملی تفاوت قایل شویم [ایبن و اسمیت، ۲۰۱۱]. یک الگوریتم تکاملی یک قالب کلی است که روش بهینه‌سازی را تعریف می‌کند و شامل جمعیتی از راه‌حل‌های نامزد، گزینش، بازترکیب و جهش می‌شود. این در حالی است که یک نمونه از الگوریتم تکاملی تحققی است از این قالب که شامل روش‌هایی خاص و پارامترهای میزان‌سازی می‌شود (برای مثال، یک *GA* میزان شده به‌گونه‌ای خاص). این دیدگاه به هر یک از نمونه‌های الگوریتم تکاملی به‌عنوان تحققی خاص از قالب کلی الگوریتم تکاملی می‌نگرد. این کار برای واحدسازی این زمینه مفید بوده و از متلاشی شدن این زمینه به تکه‌های به ظاهر غیرمتصل جلوگیری به عمل می‌آورد. این دیدگاه واحد از الگوریتم تکاملی پایه‌ی کتاب‌های [دجونگ، ۲۰۰۲] و [ایبن و اسمیت، ۲۰۱۰] می‌باشد.

توجه داشته باشید که می‌توان از دو منظر متفاوت به تنظیم نمودن پارامترهای الگوریتم تکاملی نگاه نمود. اول آنکه می‌توان پارامترها را برای بهینه ساختن عملکرد الگوریتم تکاملی میزان نمود و ثانیاً اینکه می‌توان پارامترها را برای مطالعه‌ی نحوه‌ی تغییر عملکرد با تنظیمات پارامترها، تنظیم نمود [ایبن و اسمیت، ۲۰۱۱]. دیدگاه دوم را به‌طور خلاصه در بخش ۲۱-٤ مورد بحث قرار خواهیم داد.

به‌طور قطع در آینده تنوعات خلاقانه‌ای از الگوریتم‌های تکاملی ارائه خواهد شد. در ابتدا چالش بر انگیزترین جنبه‌ی این مطالعات، کاوش تحقیقات گذشته است تا مطمئن شد ایده‌های جدید قبلاً منتشر نشده باشند. در حال حاضر شاهد نمونه‌های بسیاری از اختراع دوباره‌ی چرخ هستیم و این نتیجه‌ی این حقیقت است که بسیاری از نویسندگان و محققین کارهای گذشته را نادیده می‌گیرند. برخی اوقات الگوریتم‌هایی یکسان دوباره اختراع شده و توسط نویسندگان مختلف نام‌های متفاوتی می‌گیرند. گاهی نیز الگوریتم‌های کاملاً جدید ایجاد شده ولی اسم‌هایی مشابه با نام الگوریتم‌های موجود دریافت می‌کنند. اگر بخواهیم عادلانه با این قضیه برخورد کنیم، همگام شدن با انفجار متن‌های مربوط به الگوریتم‌های تکاملی در چند دهه‌ی اخیر کاری است بس دشوار و به همین دلیل همه‌ی ما تا حدی نسبت به تحقیقات انجام شده در گذشته نامطلع هستیم. با این وجود، هنگام مستندسازی تحقیقمان متعهد به مطالعه‌ی کامل و جامع متن‌های منتشر شده در گذشته هستیم تا بتوانیم نتیجه‌ی تحقیقات خود را به‌گونه‌ای مناسب منتشر کنیم.

مسائل

مسائل نوشتاری

۸-۱ فرض کنید جمعیتی از N ذره را که به‌صورت یکنواخت در یک دامنه‌ی یک بعدی $[x_{min}, x_{max}]$ که در آن $x_{min} = -x_{max}$ است، مقداردهی اولیه می‌نمایید.

الف) احتمال اینکه حداقل یکی از این ذرات در فاصله‌ی ε از نقطه‌ی بهینه در دامنه واقع شده باشد چه قدر است؟

ب) فرض کنید که $\varepsilon/x_{max} \ll 1$ و همچنین فرض کنید که N خیلی بزرگ نیست. از تقریب سری تیلور برای یافتن فاکتور افزایشِ جواب قسمت الف در صورت دو برابرشدن اندازه‌ی جمعیت استفاده نمایید.

۸-۲ کدهای *Gray* یکتا نیستند. معادله‌ی (۸-۲) کدگذاری *Gray* برای اعداد ۰-۷ را نشان می‌دهد. یک کدگذاری *Gray* جایگزین را برای این اعداد ارائه نمایید.

۸-۳ نخبه‌گرایی و استراتژی‌های تکامل:

الف) شباهت میان GA نخبه‌گرا و $(\mu + \lambda) - ES$ را توضیح دهید.

ب) مقادیر N و E از شکل (۸-۷) و μ و λ از شکل ۶-۱۰ را برابر با چه اعدادی قرار دهیم تا یک GA نخبه‌گرا و یک $(\mu + \lambda) - ES$ به دست آوریم که تا حد ممکن شبیه هم باشند؟

۸-۴ نخبه‌گرایی و تکامل حالت- ماندگار:

الف) چگونه می‌توانیم گزینه‌ی اول نخبه‌گرایی از شکل ۸-۶ را با تکامل حالت- ماندگار ترکیب نماییم؟

ب) چگونه می‌توانیم گزینه‌ی دوم نخبه‌گرایی از شکل ۸-۷ را با تکامل حالت- ماندگار ترکیب نماییم؟

۸-۵ فرض کنید یک الگوریتم تکاملی با فاصله‌ی نسلی برابر k در اختیار داریم. از لحاظ محاسباتی، چند نسل از این الگوریتم تکاملی با G نسل از الگوریتم تکاملی نسلی معادل است؟

۸-۶ در یک جمعیت با اندازه‌ی N چند بار مقایسه باید انجام دهیم تا جمعیت را به‌طور کامل برای ذرات تکراری بررسی کرده باشیم؟

۸-۷ یک توالی از معادلات برای تبدیل مقادیر هزینه به مقادیر هزینه‌ی اصلاح شده با استفاده از به اشتراک‌گذاری برازندگی بنویسید.

۸-۸ آیا تقسیم بر صفر در معادله‌ی (۸-۷) مشکل‌ساز می‌شود؟

۸–۹ روش تسویه از بخش ۸–٦–۳–۲ ممکن است باعث شود بهترین ذره در یک *Niche* برای عمل انتخاب و بازترکیب غیر قابل دسترس گردد. یک مثال از این موضوع را از حالتی که این موضوع در آن بتواند اتفاق بیافتد ارائه دهید.

۸–۱۰ فشار انتخاب:

الف) مقدار فشار انتخابِ چرخ رولت برای مقادیر نشان داده شده در شکل ۸–١٦ چه قدر است؟

ب) مقدار فشار انتخابِ نمونه‌گیری اتفاقی جهانی برای مقادیر نشان داده شده در شکل ۸–۱۷ چه قدر است؟

۸–۱۱ نمونه‌گیری اتفاقی جهانی:

الف) احتمال دو بار انتخاب شدن بهترین ذره در شکل ۸–۱۷ با استفاده از نمونه‌گیری اتفاقی جهانی چه قدر است؟

ب) احتمال یکبار انتخاب شدن بدترین ذره در شکل ۸–۱۷ با استفاده از نمونه‌گیری اتفاقی جهانی چه قدر است؟

۸–۱۲ فرض کنید در یک الگوریتم تکاملی چهار ذره با مقادیر برازندگی ۱۰، ۲۰، ۳۰ و ٤۰ در اختیار داریم.

الف) احتمال انتخاب هر ذره در یکبار چرخش چرخ رولت چه قدر است؟

ب) فرض کنید از روش انتخاب بیش از حد به‌گونه‌ای استفاده می‌نماییم که نیمه‌ی بهتر جمعیت دارای ۷۵٪ احتمال انتخاب شدن بوده و نیمه‌ی بدتر جمعیت دارای تنها ۲۵٪ احتمال انتخاب باشند. احتمال انتخاب هر ذره در یک بار چرخشِ چرخ رولت چه قدر است؟

ج) در صورت استفاده از مقیاس‌گذاری سیگما احتمالات انتخاب چه قدر خواهند بود؟

د) در صورت استفاده از انتخاب رتبه– محور احتمالات انتخاب چه قدر خواهند بود؟

۸–۱۳ فرض کنید در یک الگوریتم تکاملی چهار ذره با مقادیر برازندگی ۱۰، ۲۰، ۳۰ و ٤۰ در اختیار داریم. همچنین فرض کنید از رتبه‌بندی خطی استفاده می‌نماییم. از معادله‌ی (۸–۳٦) استفاده کرده و مقادیر α و β را برای مقادیر ϕ داده شده در زیر محاسبه نمایید.

الف) $\phi = 1.4$.

ب) $\phi = 1.6$.

ج) $\phi = 1.8$.

۸-۱٤ فرض کنید از یک مسابقه‌ی نرم با اندازه‌ی مسابقه‌ی ۳ برای فرایند انتخاب استفاده می‌نمایید. همچنین فرض کنید در این مسابقه احتمال انتخاب شدن بهترین ذره ۷۰٪، احتمال انتخاب شدن دومین بهترین ذره برابر ۲۰٪ و احتمال انتخاب شدن بدترین ذره برابر ۱۰٪ است. فشار انتخاب این مسابقه چه قدر است؟

۸-۱۵ فرض کنید می‌خواهید از یک الگوریتم تکاملی برای حل یک مسئله‌ی ۲۰ بعدی استفاده نمایید.

الف) احتمال آنکه فرزندان ایجاد شده توسط برش تک- نقطه‌ای کلونی از والدین باشند چه قدر است؟

ب) احتمال آنکه فرزندان ایجاد شده توسط برش دو- نقطه‌ای کلونی از والدین باشند چه قدر است؟

ج) احتمال آنکه فرزندان ایجاد شده توسط برش قطعه قطعه با $\rho = 0.2$ کلونی از والدین باشند چه قدر است؟

د) احتمال آنکه فرزندان ایجاد شده توسط برش یکنواخت کلونی از والدین باشند چه قدر است؟

مسائل کامپیوتری

۸-۱٦ یک *GA* پیوسته‌ی نخبه‌گرا را برای مینیمم کردن تابع ۱۰ بعدی آکلی پیاده‌سازی نمایید. جمعیت *GA* را برابر ۵۰ قرار داده و آن را برای ۵۰ نسل پیاده‌سازی نمایید و همچنین احتمال جهش را برابر ۱٪ قرار دهید. *GA* را ۲۰ بار اجرا کرده و میانگین کمترین هزینه را به‌عنوان تابعی از شماره‌ی نسل رسم کنید. این کار را برای ۰ نخبه، ۲ نخبه، ۵ نخبه و ۱۰ نخبه انجام دهید. چهار منحنی را در یک ننمودار رسم کنید تا بتوان بین آن‌ها مقایسه انجام داد.

۸-۱۷ یک *GA* پیوسته را با سه نوع مختلف ازدحام که در زیربخش ۸-٦-۳-۳ مورد بحث قرار گرفتند، برای مینیمم‌سازی تابع ۱۰ بعدی آکلی پیاده‌سازی نمایید. اندازه‌ی جمعیت را برابر ٤۰، نرخ جهش را برابر ۲٪، پارامتر نخبه‌گرایی را برابر ۲ فرض کرده و در انتهای هر نسل ذرات تکراری را با ذرات تولید شده‌ی اتفاقی جایگزین نمایید. هر *GA* را برای ۱۰۰۰ بار ارزیابی تابع اجرا نمایید (توجه داشته باشید که روش‌های ازدحام مختلف، تعداد ارزیابی‌های تابع در نسلِ متفاوتی به دست می‌دهند. بنابراین باید *GA*ها را برای تعداد نسل‌های متفاوتی اجرا نماییم تا بتوانیم مقایسه‌ای عادلانه به دست آوریم). مقدار میانگین هزینه‌ی به دست آمده با میانگین‌گیری بر روی ۲۰ شبیه‌سازی مونت کارلو را برای *GA* بدون ازدحام و همچنین *GA*های همراه با سه نوع مختلف ازدحام، گزارش نمایید.

۸-۱۸ برنامه‌ای بنویسید که به‌صورت عددی جواب‌هایتان برای مسئله‌ی ۸-۱۱ را تأیید نماید.

۸-۱۹ فرض کنید ذرات مسئله‌ی ۸-۱۳ را به همراه رتبه‌بندی خطی و همچنین فشار انتخاب $\phi = 1.6$ در اختیار دارید.

الف) معادله‌ی (۸-٤۰) را چندهزار بار اجرا کرده و درصد تعداد دفعات انتخاب شدن هر ذره را ثبت نمایید. از عمل گرد کردن در معادله‌ی (۸-٤۰) برای به دست آوردن اندیس صحیح ذرات انتخاب شده استفاده نمایید. نتایج به دست آمده از شبیه‌سازی خود را با احتمالات انتخاب نظری مقایسه نمایید و توضیح دهید.

ب) بزرگترین مقدار ممکنِ معادله‌ی (۸-٤۰) (بدون عمل گرد کردن) برای این مسئله چه قدر است؟ چگونه می‌توان از این مقدار برای توضیح دادن اختلاف موجود میان نتایج نظری و نتایج به دست آمده از شبیه‌سازی استفاده نمود؟

فصل نهم

ذوب فلزات[1]

[1] Simulated Annealing (SA)

فکر می‌کنیم قیاس با ترمودینامیک می‌تواند دیدی جدید در مورد مسائل بهینه‌سازی به دست دهد و الگوریتم‌هایی بهینه برای حل آن‌ها معرفی کند.

کرنی[1] [کرنی، ۱۹۸۵]

ذوب فلزات (SA)، یک الگوریتم بهینه‌سازی است که بر پایه‌یه‌ی سرد شدن و کریستالیزه شدن مواد شیمیایی قرار دارد. معمولاً میان SA و الگوریتم‌های تکاملی تفاوت قائل می‌شوند چرا که SA شامل جمعیتی از راه‌حل‌های نامزد نمی‌شود. SA یک الگوریتم اتفاقی تک- ذره‌ای است. با این حال، $(1+1)-ES$ یک حالت خاص از از یک الگوریتم SA است [دروسته[2] و همکاران، ۲۰۰۲] و به همین دلیل از این دیدگاه می‌توان SA را یک الگوریتم تکاملی در نظر گرفت.

SA اولین بار در سال ۱۹۸۳ و توسط اسکات کِرک‌پاتریک[3]، چارلز گِلَت[4] و ماریو وِچی[5] برای ارائه‌ی راه‌حل بهینه مرتبط با برخی طراحی‌های کامپیوتری مانند قرار دادن اجزای کامپیوتری و سیم‌کشی، ارائه شد [کِرک‌پاتریک و همکاران، ۱۹۸۳]. کرنی دیگر محققی بود که در سال ۱۹۸۵ به‌طور جداگانه به این الگوریتم دست یافت و از آن برای حل مسئله‌ی فروشنده‌ی دوره‌گرد استفاده نمود [کرنی، ۱۹۸۵]. یک الگوریتم بسیار مشابه SA نیز در اواخر دهه‌ی ۱۹۶۰ توسط مارتین پینکوس[6] ارائه شد [پینکوس، ۱۹٦۸a]، [پینکوس، ۱۹٦۸b]. گاهاً SA را الگوریتم متروپولیس[7] نیز می‌نامند چرا که در ارتباط تنگاتنگی با کار نیکولاس[8] متروپولیس قرار دارد [متروپولیس و همکاران، ۱۹۵۳]. کار وی، که ایجاد یک الگوریتم برای تحقیق در مورد خواص ذرات متعامل[9] بود، به زیرساختی برای SA مبدل شد. در آخر، SA گاهاً الگوریتم متروپولیس- هِیستینگ[10] نیز خوانده می‌شود. دبلیو کِیث هِیستینگ[11] محققی بود که نتایج متروپولیس و همکاران را تعمیم داد [هِیستینگ، ۱۹۷۰].

مروری بر فصل

بخش ۹-۱ بحثی مختصر در مورد مکانیک آماری که اصل بنیادی SA است را به دست خواهد داد. بخش ۹-۲ یک الگوریتم ساده‌ی SA را ارائه خواهد کرد. بخش ۹-۳ به بحث در مورد زمانبندی‌های مختلف فرایند

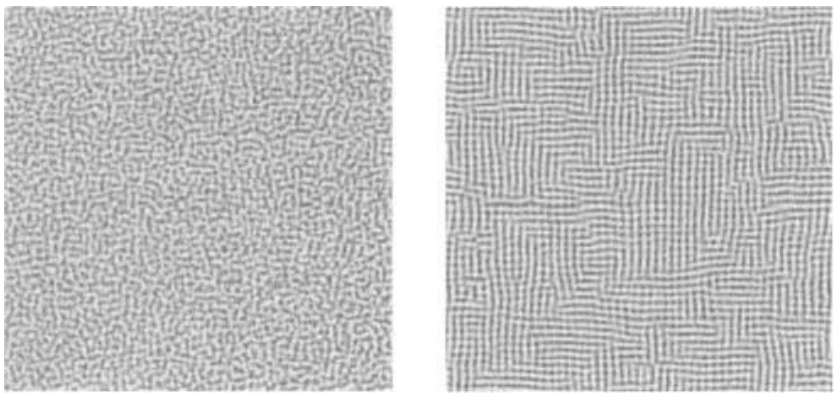
---

[1] Cerny
[2] Droste
[3] Scott Kirkpatrick
[4] Charles Gelatt
[5] Mario Vecchi
[6] Martin Pincus
[7] Metropolis
[8] Nicholas
[9] Interacting Particles
[10] Metropolis-Hasting
[11] W. Keith Hastings

سردشدن، که یکی از پارامترهای اولیه‌ی میزان‌سازی SA است خواهد پرداخت و همچنین بیان خواهد نمود کدام یک بیشترین تأثیر را بر عملکرد SA دارد. بخش ۹-٤ نیز به‌طور مختصر چندی از موارد پیاده‌سازی را مورد بحث قرار می‌دهد. از جمله‌ی این موارد می‌توان به راه‌های در دسترس برای تولید راه‌حل‌های نامزد جدید در SA، زمانِ دوباره‌ی مقداردهی دمای سرد شدن و دلیل دنبال نمودن بهترین راه‌حل نامزد، اشاره نمود.

۹-۱ ذوب در طبیعت

شبکه‌های کریستالی مثال‌هایی فوق‌العاده از توانایی طبیعت در بهینه‌سازی هستند. یک شبکه‌ی کریستالی ترتیبی از اتم‌ها و مولکول‌ها در یک مایع یا یک جامد هستند. برخی مثال‌های آشنا که مردم به‌طور روزمره با آن‌ها سروکار دارند عبارت است از ساختارهای کریستالی یخ و نمک. در دماهای بالا، مواد کریستالی چنین ساختاری را از خود به نمایش نمی‌گذارند. دمای بالا انرژی زیادی به مواد داده و باعث ایجاد لرزش و بی‌نظمی می‌شود. با این حال، با کاهش دما، مواد کریستالی به حالتی منظم‌تر در می‌آیند. این حالت منظم همیشه یکسان نیست. یک ماده که چندین بار گرما داده شده و سپس خنک شده است، هر بار حالت تعادل متفاوتی را اتخاذ می‌نماید، اما هر یک از این حالات تعادل دارای انرژی کمی می‌باشند. شکل ۹-۱ دو ساختار کریستالی را با هم مقایسه می‌نماید. یکی از آن‌ها دارای آنتروپی بالا (بی‌نظمی زیاد) در دمای بالا و دیگری دارای آنتروپی پایین (بی‌نظمی کم) در دمای پایین می‌باشد. فرایند گرم و سرد کردن ماده جهت دوباره کریستالیزه شدن، فرایند بازپخت (ذوب) نام دارد.



شکل ۹-۱ این شکل دیدی مفهومی از فرایند بازپخت به دست می‌دهد. شکل سمت چپ ساختار کریستالی در حالت پرانرژی، دمای زیاد و با بی‌نظمی زیاد را نشان می‌دهد. شکل سمت راست ساختار کریستالی را پس از سردشدن نشان می‌دهد. این ساختار دارای نظم بوده و از انرژی کمی برخوردار است (این شکل از http://en.wikipedia.org/wiki/Simulated_annealing گرفته شده است).

همان‌طور که پیش از این نیز گفته شد، پایه‌ی SA مکانیک آماری است. مکانیک آماری علم مطالعه‌ی رفتار تعداد عظیمی از ذرات متعامل، مانند اتم‌ها در یک گاز است. تعداد اتم‌های موجود در یک ماده از مرتبه‌ی 10^{23} در سانتیمتر مکعب است، بنابراین هنگام بررسی خواص یک ماده، تنها خواصی را مشاهده می‌نماییم که احتمال وقوعشان بسیار زیاد است. همچنین هنگام مطالعه‌ی مواد متوجه می‌شویم که اگرچه پیکربندی‌های انرژی- متعادل[1] و سایر پیکربندی‌های مشابه آن‌ها تنها جزئی از پیکربندی‌های ممکن را تشکیل می‌دهند، اما بیشتر از سایرین رخ می‌دهند.این بدان دلیل است که مواد تمایل دارند به سمت حالت کمترین- انرژی همگرا شوند. به عبارت دیگر طبیعت یک بهینه‌ساز است.

فرض کنید از عبارت $E(s)$ برای نشان دادن انرژی یک پیکربندی خاص s از اتم‌های یک ماده استفاده می‌نماییم. احتمال قرار گرفتن سیستم اتم‌ها در پیکربندی s برابرست با

$$P(s) = \frac{\exp[-\frac{E(s)}{kT}]}{\sum_w \exp[-\frac{E(w)}{kT}]} \qquad (۹-۱)$$

که در آن k ثابت بولتزمن[2]، T دمای سیستم در حالت تعادل بوده و جمع موجود در مخرج کسر بر روی تمام پیکربندی‌های ممکن ω صورت پذیرفته است [دیویس[3] و استین‌استراپ[4]، ۱۹۸۷]. حال فرض کنید سیستم در پیکربندی q قرار دارد و ما یک پیکربندی r را، که نامزد پیکربندی سیستم در گام زمانی بعدی است، به‌صورت اتفاقی انتخاب می‌نماییم. اگر $E(r) < E(q)$ باشد، آنگاه r را با احتمال ۱ به‌عنوان پیکربندی بعدی می‌پذیریم:

$$P(r|q) = 1 \text{ اگر } E(r) < E(q) \qquad (۹-۲)$$

این بدان معنی است که اگر پیکربندی نامزد r دارای انرژی کمتری نسبت به s باشد آنگاه ما به‌صورت خودکار در گام زمانی بعدی به r می‌رویم. با این حال اگر $E(r) \geq E(q)$ باشد، آنگاه در گام زمانی بعدی ما با احتمالی متناسب با انرژی‌های q و r به r می‌رویم:

$$P(r|q) = \exp\left[\frac{E(q) - E(r)}{kT}\right] \text{ اگر } E(r) \geq E(q) \qquad (۹-۳)$$

[1] Equilibrium-energy configurations
[2] Boltzmann's constant
[3] Davis
[4] Steenstrup

این بدان معنی است که ممکن است سیستم با یک احتمال غیرصفر $P(r|q)$ به پیکربندی با انرژی بالاتر برود. اگر $E(r) > E(q)$ باشد آنگاه معادله‌ی (۹–۳) نشان می‌دهد که احتمال انتقال سیستم از حالت q به حالت r کمتر از ۱ بوده اما با افزایش T افزایش می‌یابد. اگر از قوانین انتقال موجود در معادلات (۹–۲) و (۹–۳) پیروی کنیم، آنگاه با میل کردن زمان به سمت بی‌نهایت، احتمال قرار گرفتن سیستم در یک پیکربندی مانند s، به سمت توزیع بولتزمن از معادله‌ی (۹–۱) همگرا می‌شود.

۹–۲ یک الگوریتم ساده‌ی ذوب فلزات

از آنجا که عمل بازپخت در طبیعت پیکربندی‌های کم- انرژی از کریستال‌ها را نتیجه می‌دهد، می‌توان آن را در یک الگوریتم برای مینیمم‌سازی توابع هزینه شبیه‌سازی نمود. ما این کار را با یک راه‌حل نامزد برای یک مسئله‌ی مینیمم‌سازی آغاز می‌نماییم. همچنین در ابتدا از یک دمای بالا استفاده می‌نماییم تا احتمال تبدیل راه‌حل نامزد به یک پیکربندی دیگر زیاد باشد. ما یک راه‌حل نامزد دیگر مانند r را به‌صورت اتفاقی تولید کرده و هزینه‌ی آن را اندازه می‌گیریم. این هزینه را می‌توان چیزی مشابه انرژی ساختار کریستالین در نظر گرفت. اگر هزینه‌ی r کمتر از هزینه‌ی s باشد، آنگاه راه‌حل نامزدمان را طبق آن و مانند آنچه که در معادله‌ی (۹–۲) نشان داده شده است، جایگزین می‌نماییم. اگر هزینه‌ی r بیشتر و یا مساوی s باشد، آنگاه راه‌حل نامزد را با احتمالی کمتر و یا مساوی ۱، و بنابر معادله‌ی (۹–۳) جایگزین می‌نماییم. SA را گاهاً به دلیل استفاده‌ی آن از معادلات (۹–۲) و (۹–۳)، بازپخت بولتزمن نیز می‌نامند. با افزایش تعداد دوره‌ها و با جلو رفتن زمان، دما را کاهش می‌دهیم. این موضوع باعث گرایش راه‌حل‌های نامزد به ساکن شدن در یک حالت کم- هزینه می‌شود. تشابهات موجود بین فرایند بازپخت در طبیعت و الگوریتم SA در زیر خلاصه شده‌اند.

ذوب در طبیعت	ذوب فلزات شبیه‌سازی شده
پیکربندی اتمی	راه‌حل نامزد
دما	تمایل برای کاوش فضای جستجو
سرد شدن	کاهش تمایل برای کاوش
تغییر پیکربندی اتمی	تغییر راه‌حل‌های نامزد

می‌توان دید که SA بسیاری از رفتارهای استاندارد الگوریتم‌های تکاملی را شامل می‌شود. اگرچه برای SA از فرایند بازپخت موجود در طبیعت الهام گرفته‌ایم، اما می‌توان SA را بدون هیچ الهامی از طبیعت نیز

بسط داد [میشیلز[1] و همکاران، ۲۰۰۷]. در هر یک از این روش‌ها فوایدی وجود دارد. یک الگوریتم ساده‌ی SA در شکل ۹-۲ نشان داده شده است.

$T =$ دمای اولیه > 0
$\alpha(T) =$ تابع سرد شدن: $\alpha(T) \in [0, T]$
یک راه‌حل نامزد مانند x_0 را برای مساله‌ی مینیمم‌سازی $f(x)$ مقداردهی اولیه کن
تا زمانی که شرایط توقف برآورده نشده است
  یک راه‌حل نامزد مانند x تولید کن
  اگر $f(x) < f(x_0)$
    $x_0 \leftarrow x$
  در غیر این صورت
    $r \leftarrow U[0,1]$
    اگر $r < \exp\left[(f(x_0) - \frac{f(x)}{T}\right]$
      $x_0 \leftarrow x$
    پایان اگر
  پایان اگر
  $T \leftarrow \alpha(T)$
نسل بعد

شکل ۹-۲ یک الگوریتم پایه‌ای و ساده‌ی ذوب فلزات برای مینیمم‌سازی $f(x)$. تابع $U[0,1]$ یک عدد اتفاقی با توزیع یکنواخت بر روی بازه‌ی [0,1] تولید می‌نماید.

شکل ۹-۲ نشان می‌دهد که الگوریتم پایه‌ی SA ویژگی‌های مشترکی با بیشتر الگوریتم‌های تکاملی دارد. اول آنکه این الگوریتم ساده و بسیار جذاب است. ثانیاً آنکه این الگوریتم بر اساس یک فرایند بهینه‌سازی در طبیعت است. ثالثاً، این الگوریتم دارای چندین پارامتر میزان‌سازی است که تأثیر قابل توجهی بر روی عملکرد آن دارند:

- دمای اولیه‌ی T یک کران بالا برای اهمیت نسبی میان کاوش و بهره‌وری (انتفاع) فراهم می‌کند. اگر دمای اولیه بسیار کم باشد، آنگاه الگوریتم نخواهد توانست به‌طور مؤثر به کاوش فضای جستجو بپردازد. اگر دمای اولیه بسیار زیاد باشد، آنگاه الگوریتم بسیار دیر همگرا خواهد شد.
- زمانبندی سردشدن $\alpha(T)$ نرخ همگرایی را کنترل می‌نماید. در ابتدای الگوریتم میزان کاوش زیاد و میزان بهره‌وری کم است. از سوی دیگر، در انتهای الگوریتم میزان کاوش کم و میزان بهره‌وری زیاد است. زمانبندی سردشدن، انتقال از کاوش به بهره‌وری را کنترل می‌نماید. اگر $\alpha(T)$ زیادی شدید

[1] Michiels

باشد، مانند ساختارهای کریستالی که به سرعت سرد می‌شوند، فرایند بازپخت به حالتی بی‌نظم (هزینه‌ی زیاد) همگرا می‌گردد. اگر $\alpha(T)$ زیادی تدریجی باشد، آنگاه فرایند بازپخت بسیار دیر همگرا می‌شود. در بخش ۹-۳ به بحث در مورد این پارامتر خواهیم پرداخت.

- استراتژی استفاده شده در هر دوره برای تولید یک راه‌حل نامزد x می‌تواند تأثیر عظیمی بر عملکرد SA داشته باشد. تولید اتفاقی x ممکن است جواب دهد، اما یک روش هوشمندانه‌تر که بتواند یک x بهتر از x_0 تولید نماید احتمالا عملکرد بهتری خواهد داشت. بحث در مورد استراتژی‌های تولید راه‌حل‌های نامزد را به بخش ۹-٤-۱ موکول می‌نماییم.

یک الگوریتم SA ساده شده را می‌توان با جایگزین کردن آزمایش پذیرش[1] در شکل ۹-۲ با عبارت زیر، پیاده‌سازی نمود:

"اگر $r < \exp[(f(x_0) - f(x))/T]$" را با "اگر $r < \exp[-c/T]$" جایگزین کن (۹-٤)

که در آن c *ثابت احتمال پذیرش*[2] نام دارد. این عبارت نشان‌دهنده‌ی آن است که اگر راه‌حل نامزد x دارای هزینه‌ی بیشتری نسبت به x_0 باشد، احتمال جایگزین کردن x_0 با x، مستقل از هزینه‌ی آن خواهد بود. ثابت احتمال پذیرش تعادل میان کاوش و بهره‌وری را کنترل می‌نماید. اگر c بسیار بزرگ باشد آنگاه الگوریتم با شدت لازم به جستجوی فضای جستجو نمی‌پردازد. اگر c بسیار کوچک باشد، آنگاه الگوریتم به شدت به کاوش فضای جستجو پرداخته و بدین ترتیب قادر نخواهد بود از راه‌حل‌های خوبی که پیش از این یافته است بهره‌ی لازم را ببرد.

۹-۳ زمانبندی سردشدن

این بخش به بحث در مورد زمانبندی‌های سردشدن متفاوت که می‌توانند در الگوریتم SA شکل ۹-۲ به کار روند می‌پردازد. زمانبندی سردشدن می‌توان اثر عظیمی بر عملکرد SA داشته باشد. اگر یک پیاده‌سازی از SA بر روی مسئله مؤثر واقع نشود، ممکن است به خاطر آن باشد که زمانبندی سردشدن اعمال شده برای آن مسئله مناسب نباشد. برخی از زماندبندی‌های سرد شدن متداول عبارتند از سردشدن خطی، سردشدن نمایی، سردشدن معکوس، سردشدن لگاریتمی و سردشدن خطی معکوس که درمورد هر یک از آن‌ها در بخش‌های بعدی سخن خواهیم راند. همچنین باید توجه نمود که برخی مسائل بهینه‌سازی دارای مقیاس‌های

[1] Acceptance test
[2] Acceptance probability constant

متفاوتی در طول ابعاد مختلف هستند و به همین دلیل در مورد سردشدن وابسته به ابعاد نیز در بخش ۹-۳-٦ صحبت خواهیم نمود.

۹-۳-۱ سردشدن خطی

سردشدن خطی ساده‌ترین نوع سردشدن است و از زمانبندی زیر تبعیت می‌نماید:

$$\alpha(T) = T_0 - \eta k \qquad (۵-۹)$$

که در آن T_0 دمای اولیه، k شماره‌ی دوره‌ی SA و η یک ثابت است. باید اطمینان حاصل نمود که برای تمامی $T > 0$ ها k است، بنابراین باید η را به‌گونه‌ای انتخاب نماییم که که دما در بزرگترین شماره‌ی نسل مثبت باشد. متناوباً می‌توان از فرم اصلاح شده‌ی زیر برای سردشدن خطی استفاده نمود:

$$\alpha(T) = \max(T_0 - \eta k, T_{min}) \qquad (٦-۹)$$

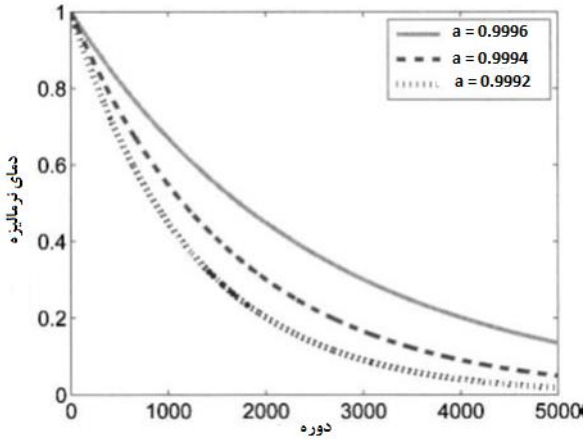
که در آن T_{min} یک دمای مینیمم تعریف شده توسط کاربر است.

۹-۳-۲ سردشدن نمایی

سرد شدن نمایی از زمانبندی زیر طبیعت می‌نماید:

$$\alpha(T) = aT \qquad (۷-۹)$$

که در آن معمولاً $a \in (0.8,1)$ است. مقادیر بزرگتر از a سردشدن آهسته‌تری را نتیجه می‌دهد. شکل ۹-۳ مقادیر نرمالیزه شده‌ی دما در زمانبندی نمایی را برای مقادیر مختلف a نشان می‌دهد. می‌توان دید که a باید به ۱ نزدیک باشد چرا که در غیر این صورت نرخ سردشدن بسیار شدید خواهد یود.



شکل ۹-۳ دمای نرمالیزه شده به‌عنوان تابعی از a برای زمانبندی سردشدن نمایی. پارامتر a برای این زمانبندی معمولاً بسیار نزدیک ۱ انتخاب می‌شود. نرخ سردشدن نسبت به تغییر a حساس است.

۹-۳-۳ سردشدن معکوس

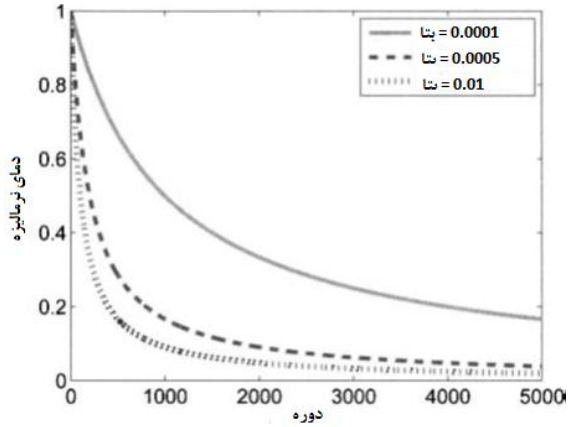
سردشدن معکوس از زمانبندی زیر تبعیت می‌نماید

$$\alpha(T) = T/(1 + \beta T) \qquad (۸-۹)$$

که در آن β یک ثابت کوچک از مرتبه‌ی ۰,۰۰۱ است. مقادیر کوچکتر β زمانبندی سردشدن آهسته‌تری را نتیجه می‌دهند. این زمانبندی سردشدن اولین بار در [لاندی[1] و میز[2]، ۱۹۸٦] پیشنهاد گردید. شکل ۹-٤ مقادیر نرمالیزه شده‌ی دما در سردشدن معکوس را برای مقادیر مختلف β نشان می‌دهد. می‌توان دید که β باید بسیار کوچک باشد چرا که در غیر این صورت نرخ سردشدن بسیار شدید خواهد بود.

با مقایسه‌ی شکل‌های ۹-۳ و ۹-٤ می‌توان دید که با انتخاب مقادیر مناسب برای a و β، دو زمانبندی نمایی و معکوس بسیار شبیه هم خواهند بود.

[1] Lundy
[2] Mees



شکل ۹–٤ دمای نرمالیزه شده به‌عنوان تابعی از β برای زمانیندی سردشدن معکوس. پارامتر β برای این زمانبندی معمولاً بسیار کوچک انتخاب می‌شود. نرخ سردشدن نسبت به تغییرات β بسیار حساس است.

مثال ۹–۱

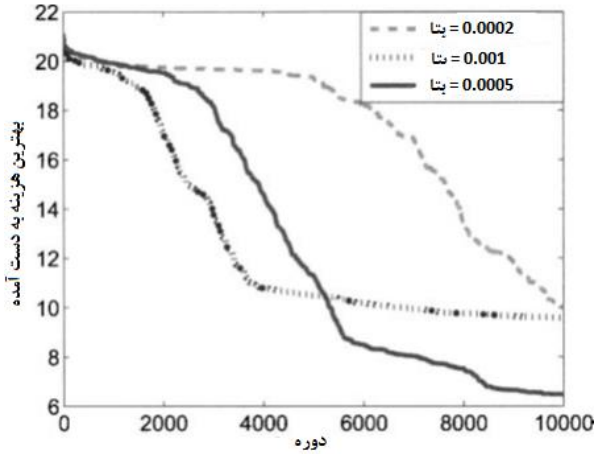
در این مثال تابع ۲۰ بعدی اَکلی را که در ضمیمه‌ی ج.۲–۱ تعریف شده است را با استفاده از الگوریتم SA شکل ۹–۲ بهینه می‌کنیم. در اینجا از تابع سردشدن معکوسِ وصف شده در معادله‌ی (۹–۸) بهره می‌بریم: $T_{k+1} = {T_k}/{1 + \beta T_k}$. در این معادله، k شماره‌ی دوره، β پارامتر زمانبندی سردشدن، T_k دما در دوره‌ی kاُم بوده و $T_0 = 100$ می‌باشد. ما از یک عدد اتفاقی گاوسی متمرکز در x_0 برای تولید نمودن راه‌حل نامزد جدید در هر دوره استفاده می‌نماییم:

$$x \leftarrow x_0 + N(0, T_k I) \qquad (۹–۹)$$

که در آن $N(0, T_k I)$ یک بردار اتفاقی گاوسی با میانگین ۰ و کوواریانس $T_k I$ می‌باشد و I نیز یک ماتریس همانی 20×20 است. در نهایت از تست پذیرش معادله‌ی (۹–٤) با $c = 1$ استفاده می‌نماییم.

شکل ۹–۵ بهترین راه‌حل پیدا شده را، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، به‌عنوان تابعی از شماره‌ی دوره‌ی SA و برای سه مقدار متفاوت از β نشان می‌دهد. می‌توان دید که اگر β بسیار کوچک انتخاب شود (۰,۰۰۰۲)، سردشدن بسیار آهسته صورت گرفته و الگوریتم SA به شدت در اطراف فضای جستجو پرش کرده بدون آنکه از راه‌حل‌های خوبی که پیش از این یافته است بهره‌ی لازم را ببرد. اگر β بسیار بزرگ باشد (۰,۰۰۱)، آنگاه سرد شدن بسیار سریع صورت پذیرفته و الگوریتم SA به سمت مینیمم محلی متمایل شده و در آنجا به دام می‌افتد. اگر β مقداری درست و مناسب داشته باشد (۰,۰۰۰۵)، آنگاه سرد شدن با نرخی صورت می‌پذیرد که بهترین همگرایی را نتیجه می‌دهد. با این حال توجه داشته

باشید که در اواخر نمودار، منحنی $\beta = 0.0002$ به سرعت به منحنی $\beta = 0.0005$ نزدیک می‌شود. این موضوع به این نکته اشاره دارد که اگرچه $\beta = 0.0002$ برای به دست دادن همگرایی خوب در محدوده‌ی تعداد دوره‌های استفاده شده در این مثال بسیار کوچک است، اما اگر تعداد دوره‌ها به اندازه‌ی کافی بزرگ در نظر گرفته شود، در نهایت سرد شدن به میزان کافی صورت گرفته و SA به یک نتیجه‌ی خوب همگرا خواهد شد.



شکل ۹–۵ نتایج شبیه‌سازی مثال ۹–۱ از یک الگوریتم SA برای بهینه‌سازی تابع ۲۰ بعدی آکلی. نتایج بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده‌اند. پارامتر زمانبندی سردشدن معکوس β تأثیر عظیمی بر عملکرد SA دارد.

مثال ۹–۱ نشان می‌دهد که اگر سرشدن بسیار سریع یا بسیار کند باشد، عملکرد SA خوب نخواهد بود. همین حرف را می‌توان در مورد دمای اولیه T_0 نیز زد. در مثال ۹–۱، ما به‌طور دلخواه از $T_0 = 100$ استفاده نمودیم. متأسفانه هیچگونه خط مشی برای چگونگی انتخاب T_0 وجود ندارد و انتخاب آن کاملاً به مسئله‌ی بهینه‌سازی بستگی دارد.

۹–۳–۴ سرد شدن لگاریتمی

سرد شدن لگاریتمی از زمانبندی زیر استفاده می‌نماید

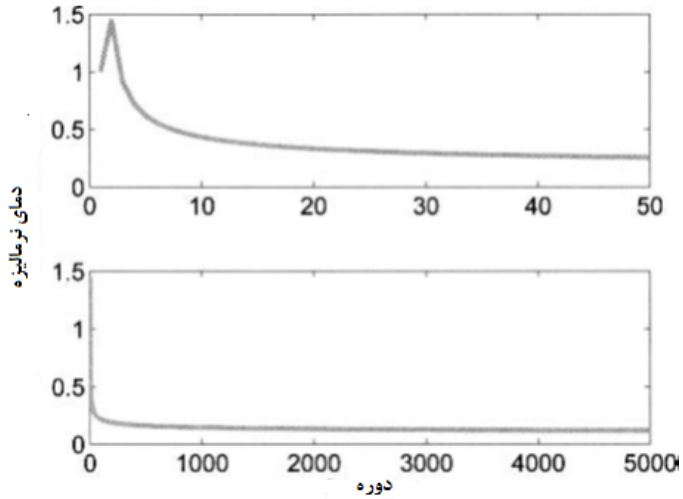
$$\alpha(T) = \mathrm{c}/\ln k \qquad (۹–۱۰)$$

که در آن c یک ثابت بوده و k شماره‌ی دوره‌ی SA است. این روش ابتدا در [گِمان[1] و گِمان، ۱۹۸٤] پیشنهاد شد. این معادله گاهی به‌صورت زیر تعمیم داده می‌شود

$$\alpha(T) = \mathrm{c}/\ln(k+d) \qquad (۹-۱۱)$$

که در آن d یک ثابت بوده و معمولاً برابر ۱ در نظر گرفته می‌شود [نورانی[2] و آندرسِن[3]، ۱۹۹۸]. همان‌طور که از شکل ۹-٦ نیز مشهود است، سردشدن لگاریتمی از نظر کیفی با سرد شدن نمایی و معکوس متفاوت است. دما در طول دوره‌های ابتدایی به سرعت و در طول دوره‌های انتهایی به آهستگی کاهش می‌یابد. این کاهش آهسته بدین معنی است که معمولاً همگرایی SA با سردشدن لگاریتمی بسیار ضعیف است. به همین دلیل، زمانبندی سردشدن لگاریتمی برای کاربردهای عملی پیشنهاد نمی‌شود.

با این حال، زمانبندی سردشدن لگاریتمی از لحاظ نظری جذاب بوده و در میان جامعه‌ی SA به خوبی شناخته شده است. دلیل این موضوع آن است که ثابت شده است تحت شرایط معینی، این روش مینیمم جهانی را به دست می‌دهد [گِمان و گِمان، ۱۹۸٤].



شکل ۹-٦ دمای نرمالیزه شده برای زمانبندی سردشدن لگاریتمی. نمودار بالا دما را برای ۵۰ دوره‌ی اول و نمودار پایین دما را برای ۵۰۰۰ دوره‌ی ابتدایی نشان می‌دهد. دما در طول چند دوره‌ی اول به سرعت کاهش پیدا کرده و سپس بسیار آهسته کاهش پیدا می‌کند و به همین دلیل برای کاربردهای عملی مناسب نمی‌باشد.

[1] Geman
[2] Nourani
[3] Andresen

به‌عنوان یک نمایش ساده از این اثبات، فرض کنید مسئله‌ای گسسته در اختیار داریم و اندازه‌ی فضای جستجو محدود است. ما راه‌حل نامزد x را از یک توزیع گاوسی تولید می‌نماییم به‌طوری که احتمال تولید x، در صورتی که x_k راه‌حل نامزد موجود در دوره‌ی kاُم باشد برابرست با

$$g_k = P(x|x_k) = (2\pi T_k)^{D/2}\exp[-\frac{||x - x_k||_2^2}{2T_k}] \qquad (۹–۱۲)$$

که در آن D ابعاد مسئله است. به عبارت دیگر، احتمال شرطی تولید x به شرط آنکه x_k راه‌حل نامزد کنونی باشد، یک احتمال گاوسی با میانگین x_k و کوواریانس $T_k I$ است که در آن T_k دما در دوره‌ی kاُم بوده و I ماتریس همانی است. به‌منظور دیدن تمامی راه‌حل‌های نامزد موجود در فضای جستجو کافی است نشان دهیم با میل نمودن شماره‌ی دوره‌ها به سمت بی‌نهایت، احتمال دیده نشدن x به سمت صفر میل می‌کند:

$$\lim_{N\to\infty}\prod_{k=1}^{N}(1 - g_k) = 0 \qquad (۹–۱۳)$$

اگر از عبارت بالا در مبنای طبیعی لگاریتم بگیریم به عبارت زیر می‌رسیم

$$\ln[\lim_{N\to\infty}\prod_{k=1}^{N}(1 - g_k) = \lim_{N\to\infty}[\ln\prod_{k=1}^{N}(1 - g_k)] = -\infty \qquad (۹–۱٤)$$

با استفاده از یک بسط سری تیلور از لگاریتم حول نقطه‌ی $g_1 = g_2 = \cdots = 0$ خواهیم داشت

$$\ln[(1 - g_1)(1 - g_2)\ldots] = \ln\ 1 - g_1 - g_2 - \cdots \qquad (۹–۱۵)$$

ترکیب دو معادله‌ی قبل شرط کافی برای به دست آوردن احتمال ۱۰۰٪ برای دیدن تمامی xها با میل کردن شماره‌ی دوره‌ها به سمت بی‌نهایت، حاصل می‌شود:

$$\lim_{N\to\infty}\prod_{k=1}^{N} g_k = \infty \qquad (۹–۱٦)$$

اگر g_k را از معادله‌ی (۹–۱۲) جایگزین کرده و $T_k = {T_0}/{k}$ را قرار دهیم، آنگاه معادله‌ی بالا به‌صورت زیر درخواهد آمد

$$\lim_{N\to\infty}\sum_{k=1}^{N}(2\pi T_0/\ln k)^{D/2}\exp[-||x - x_k||_2^2/(2T_0/\ln k)] \geq \qquad (۹–۱۷)$$

$$\sum_{k=1}^{\infty} \exp(-\ln k) = \sum_{k=1}^{\infty} \frac{1}{k} = \infty$$

که در آن نامساوی در صورتی که T_0 به اندازه‌ی کافی بزرگ باشد برقرار خواهد بود.

۹–۳–۵ سردشدن خطی معکوس

سردشدن خطی معکوس از زمانبندی زیر تبعیت می‌کند

$$\alpha(T) = T_0/k \qquad (۱۸–۹)$$

که در آن T_0 دمای اولیه و k شماره‌ی دوره‌ی SA می‌باشد. همان‌طور که در شکل ۹–۷ مشاهده می‌شود، سرد شدن خطی معکوس در چند دوره‌ی اول همان رفتار سرد شدن سریعِ سرد شدن لگاریتمی را از خود به نمایش می‌گذارد، اما از دماهای غیرصفر و سرد شدن آهسته در دوره‌های بعدی خودداری می‌کند. دما به سرعت کاهش پیدا کرده و به سرعت به صفر می‌رسد. این بدان معنی است که سردشدن خطی معکوس برای مسائلی که نیاز به کاوش زیاد دارند مؤثر واقع نخواهد شد، اما برای مسائلی که در آن‌ها جمعیت اولیه‌ی راه‌حل‌ها به راه‌حل بهینه نزدیک باشند مناسب خواهد بود.

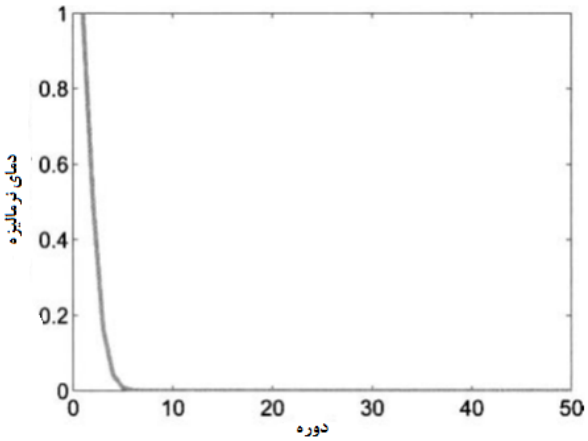
سرد شدن خطی معکوس نیز مانند سردشدن لگاریتمی از لحاظ نظری بسیار جذاب بوده و در میان جامعه‌ی SA به خوبی شناخته شده است چرا که این روش نیز تحت شرایطی معین جواب بهینه را به دست خواهد داد [سزو[1] و هارتلی[2]، ۱۹۸۷]. به‌عنوان یک نمایش ساده مانند آنچه که در مورد سردشدن لگاریتمی به کار رفت [اینگبر[3]، ۱۹۹۶]، فرض کنید یک مسئله‌ی گسسته در اختیار داریم و بدین ترتیب اندازه‌ی فضای جستجو محدود است. ما x را از یک توزیع کوشی تولید می‌نماییم به‌گونه‌ای که احتمال حاصل شدن x به شرط آنکه راه‌حل نامزد حاضر در دوره‌ی kاُم x_k باشد برابرست با

$$g_k \equiv P(x|x_k) = \frac{T_k}{(||x - x_k||_2^2 + T_k^2)^{(D+1)/2}} \qquad (۱۹–۹)$$

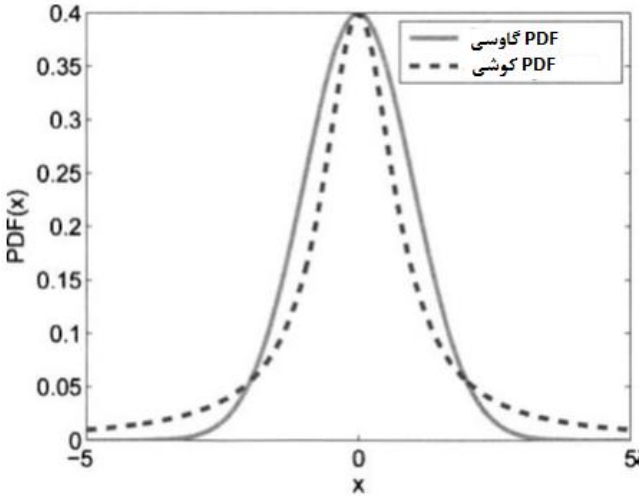
که در آن D ابعاد مسئله است. توجه داشته باشید که در زیربخش قبل ما از توزیع گاوسی برای تولید راه‌حل‌های نامزد استفاده نمودیم در حالی که در این زیربخش از توزیع کوشی برای این کار استفاده نمودیم. شکل ۹–۸، یک PDF کوشی را با یک PDF گاوسی مقایسه می‌نماید. می‌توان دید که کناره‌های نمودار مربوط

[1] Szu
[2] Hartley
[3] Ingber

به PDF کوشی بسیار پهن‌تر از کناره‌های مربوط به PDF گاوسی است. این بدین معناست که با استفاده از توزیع کوشی، احتمال تولید راه‌حل‌های نامزد دورتر از راه‌حل نامزد حاضر بیشتر خواهد بود.



شکل ۹-۷ دماهای نرمالیزه شده برای زمانبندی سردشدن خطی معکوس. دما به سرعت به صفر می‌رسد.



شکل ۹-۸ مقایسه‌ای میان توابع چگالی احتمال کوشی و گاوسی یک بعدی.

اگر g_k را از معادله‌ی (۹-۱۹) جایگزین کرده و $T_k = {}^{T_0}/_{k}$ را قرار دهیم، سمت چپ معادله‌ی (۹-۱۶) به‌صورت زیر درخواهد آمد

$$\lim_{N\to\infty} \sum_{k=1}^{N} \frac{T_0/k}{(||x - x_k||_2^2 + T_k^2)^{(D+1)/2}} \geq \sum_{k=1}^{\infty} \frac{1}{k} = \infty \qquad (۹-۲۰)$$

که در آن نامساوی به ازای مقادیر مناسب T_0 برقرار خواهد بود (مسئله‌ی ۹–۶ را ببینید).

حال به مقایسه‌ی نتایج همگرایی به دست آمده از زمانبندی سرد شدن لگاریتمی و خطی معکوس می‌پردازیم. به یاد آورید که کناره‌های PDF کوشی بسیار پهن‌تر از کناره‌های PDF گاوسی است. همچنین به خاطر آورید که تابع تولید راه‌حل‌های نامزد معادله‌ی (۹–۱۹) که از PDF کوشی استفاده می‌نماید، با زمانبندی خطی معکوس از شکل ۹–۷ ترکیب می‌شود. در نهایت، به یاد آورید که تابع تولید راه‌حل‌های نامزد معادله‌ی (۹–۱۲) که از PDF گاوسی استفاده می‌نماید، با زمانبندی سرد شدن گاوسی از شکل ۹–۶ استفاده می‌نماید. از آنجا که تابع تولید کوشی دارای کناره‌های پهن‌تری نسبت به تابع تولید گاوسی است، این زمانبندی بسیار سریع‌تر از زمانبندی سرد شدنی که از تابع تولید گاوسی استفاده می‌نماید، همگرا می‌شود.

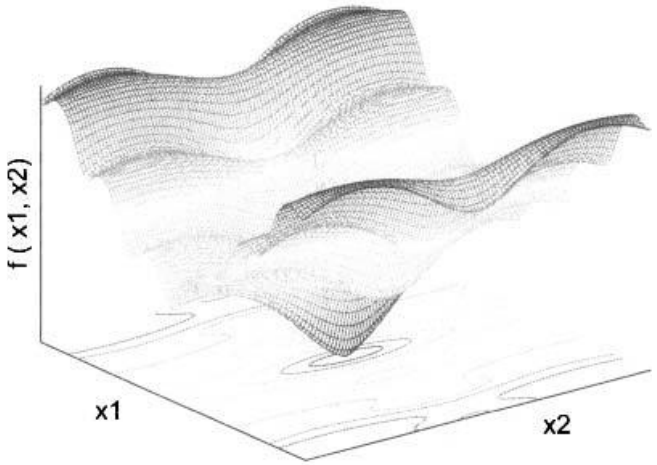
۹–۳–۶ سرد شدن وابسته به ابعاد

در کاربردهای دنیای واقعی و همچنین برخی توابع محک، تابع هزینه در طول ابعاد مختلف دارای ظاهری متفاوت است. برای مثال تابع زیر را در نظر بگیرید

$$f(x) = 20 + e - 20\exp\left(-0.2\sum_{i=1}^{n}\frac{y_i^2}{n}\right) - \exp(\sum_{i=1}^{n}(cos2\pi y_i)/n) \qquad (۹–۲۱)$$

$$y_i = \begin{cases} x_i & \text{برای مقادیر فرد } i \\ \dfrac{x_i}{4} & \text{برای مقادیر زوج } i \end{cases}$$

این تابع یک تابع مقیاس شده از تابع اکلی است که در ضمیمه‌ی ج.۲–۱ تعریف شده است. این حقیقت که x_i برای مقادیر زوج i مقیاس شده است بدان معناست که تابع در طول این ابعاد "کشیده" شده است. شکل ۹–۹ یک نمودار دو بعدی از این تابع را نشان می‌دهد. به دلیل مقیاس شدن ابعاد زوج، تابع در طول بعدِ x_2 هموارتر و صاف‌تر است تا در طول بعدِ x_1.



شکل ۹-۹ نسخه‌ی مقیاس شده از تابع دو بعدی اکلی. شکل نمودار در طول بعد x_2 بسیار هموارتر و صاف‌تر از شکل آن در طول بعد x_1 است. این یعنی الگوریتم SA باید از زمانبندی سردشدن آهسته‌تری در طول آن بعد استفاده نماید.

برای توابعی که دارای پیکربندی متفاوتی در طول ابعاد مختلف هستند، باید از زمانبندی‌های مختلف برای ابعاد مختلف استفاده نماییم. برای تابع مقیاس شده‌ی اکلی از معادله‌ی (۹-۲۱) باید از سردشدن‌های آهسته‌تر در طول ابعاد زوج و سردشدن‌های سریع‌تر در طول ابعاد فرد استفاده نماییم. این کار به الگوریتم SA این اجازه را می‌دهد تا به تدریج به سمت مقادیر بهینه‌ی هر بعد همگرا شود. یک زمانبندی سردشدن سریع در طول ابعادِ به تدریج در حال تغییر، از پایین رفتن SA در شیب‌های ملایم ممانعت می‌کند. با این حال، برای ابعاد پویاترِ تابع به یک زمانبندی سردشدن سریع‌تر نیاز است. حتی با نرخ سردشدن سریع نیز الگوریتم SA در طول ابعاد پویا در سراشیبی‌ها پایین خواهد رفت، اما نرخ سردشدن آهسته باعث پرش‌های پراکنده‌ی زیادی خواهد شد. به عبارت دیگر می‌توان گفت برای ابعاد با حساسیت کمتر نیاز به جستجوی پرتکاپوتر (دمای بالاتر) و برای ابعاد با حساسیت بیشتر به جستجوی کم‌تکاپوتر (دمای پایین‌تر) نیاز داریم.

اگر از زمانبندی سردشدن معکوسِ معادله‌ی (۹-۸) استفاده نماییم، برای ابعاد زوج به β کوچکتر و برای ابعاد فرد به β بزرگتر نیاز خواهیم داشت. این بدان معناست که هر بعدِ مسئله دارای دمای مخصوص خود خواهد بود. با توجه به این موضوع، می‌توان با اعمال یک اصلاح، الگوریتم SA پایه‌ی شکل ۹-۲ را به الگوریتم SA وابسته به ابعاد تبدیل نمود (شکل ۹-۱۰)

$i \in [1,n]$ ،$T_i =$ دمای اولیه > 0

$i \in [1,n]$ ،$\alpha_i(T_i) = i$:تابع سرد شدن برای بعد $\alpha_i(T_i) \in [0, T_i]$

یک راه‌حل نامزد مانند x_0 را برای مساله‌ی مینیمم‌سازی $f(x)$ مقداردهی اولیه کن:

$x_0 = [x_{01}, x_{02}, \dots, x_{0m}]$

تا زمانی که شرایط توقف برآورده نشده است

یک راه‌حل نامزد مانند $x_1 = [x_{11}, x_{12}, \dots, x_{1n}]$ تولید کن

اگر $f(x_1) < f(x_0)$

$x_0 \leftarrow x_1$

در غیر این صورت

برای $i = 1, \dots, n$

$r \leftarrow U[0,1]$

اگر $r < \exp\left[(f(x_0) - \frac{f(x_1)}{T_i}\right]$ آنگاه

$x_{0i} \leftarrow x_{1i}$

پایان اگر

i بعدی

پایان اگر

$T \leftarrow \alpha(T)$

نسل بعد

شکل ۹–۱۰ الگوریتم ذوب فلزات وابسته به ابعاد برای مینیمم‌سازی تابع n بعدی $f(x)$. تابع $U[0,1]$ یک عدد اتفاقی با توزیع یکنواخت بر روی $[0,1]$ برمی‌گرداند. این الگوریتم یک تعمیم از الگوریتم SA پایه از شکل ۹–۲ است. در اینجا ما به هر بعد اجازه داده‌ایم دما و زمانبندی سردشدن مخصوص به خود را داشته باشد.

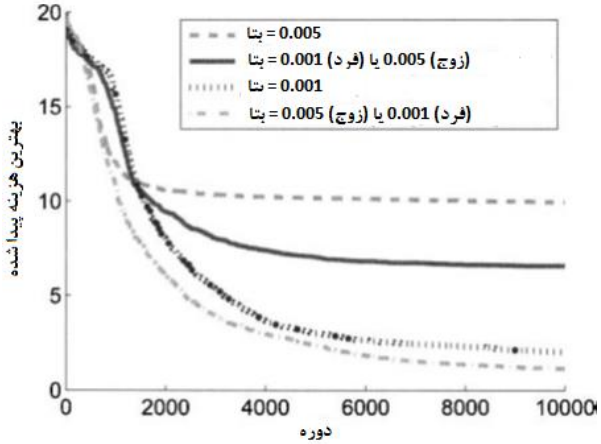
مثال ۹–۲

در این مثال ما یک تابع آکلی ۲۰ بعدی را که در معادله‌ی (۹–۲۱) نشان داده شده است با استفاده از الگوریتم SA وابسته به ابعاد از شکل ۹–۱۰ بهینه می‌نماییم. از تابع سردشدن معکوس وصف شده در معادله‌ی (۹–۸) برای هر یک از ابعاد استفاده می‌نماییم: $T_{k+1,i} = {T_{ki}}/{(1 + \beta_i T_{ik})}$، که در آن i شماره‌ی بعد، k شماره‌ی دوره‌ی SA، β_i پارامتر زمانبندی سردشدن برای iاُمین بعد و برای تمامی iها $T_{0i} = 100$ خواهد بود. از یک عدد اتفاقی گاوسی متمرکز در x_0 برای تولید راه‌حل‌های نامزد جدید در هر دوره استفاده می‌نماییم.

$$x_{1i} \leftarrow x_{0i} + N(0, T_{ki}) \qquad (۹–۲۲)$$

که در آن $N(0, T_{ki})$ یک عدد تصادفی گاوسی با میانگین ۰ و واریانس T_{ki} است. ما از تست ساده‌ی پذیرشِ معادله‌ی (۹–٤) با $c = 1$ استفاده می‌نماییم. شکل ۹–۱۱ بهترین راه‌حل پیدا شده را که بر روی ۲۰ شبیه‌سازی

مونت کارلو میانگین‌گیری شده است را برای چهار مقدار متفاوت از β، به‌عنوان تابعی از شماره‌ی دوره نشان می‌دهد. می‌توان دید که اگر β خیلی کوچک باشد (۰٫۰۰۱)، آنگاه سردشدن بسیار آهسته صورت گرفته و الگوریتم SA در فضای جستجو به شدت به اطراف پرش کرده، بی‌آنکه از راه‌حل‌های خوبی که تا کنون یافته شده بهره‌ی لازم را ببرد. از سوی دیگر اگر β خیلی بزرگ باشد (۰٫۰۰۵)، سردشدن بسیار سریع صورت پذیرفته و SA تمایل به گیر افتادن در مینیمم محلی خواهد داشت. با این حال، اگر β برای ابعاد فرد بزرگ بوده و برای ابعاد زوج کوچک باشد، سردشدن با نرخی صورت خواهد پذیرفت که بهترین همگرایی را نتیجه خواهد داد. این ترکیب، سردشدن سریع برای ابعاد فرد بسیار پویا و سردشدن آهسته برای ابعاد زوج را نتیجه خواهد داد.



شکل ۹-۱۱ نتایج شبیه‌سازی الگوریتم SA وابسته به ابعاد در مثال ۹-۲ برای بهینه‌سازی تابع ۲۰ بعدیِ مقیاس شده‌ی اَکلی. نتایج بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده‌اند. پارامترهای زمانبندی سردشدن $\{\beta_i\}$ را می‌توان برای هر یک از ابعاد تنظیم نمود تا بهترین نتایج حاصل شود.

۹-۴ موارد اجرایی

این بخش به بحث در مورد پاره‌ای از موارد اجرایی می‌پردازد. از جمله‌ی این موارد می‌توان به چگونگی تولید راه‌حل‌های نامزد، زمان بازمقداردهی دمای سردشدن و چرایی نیاز به دنبال نمودن بهترین راه‌حل نامزد اشاره نمود.

۹-۴-۱ تولید راه‌حل نامزد

عبارت "تولید راه‌حل نامزد" به کار رفته در الگوریتم‌های SA شکل ۹-۲ و ۹-۱۰ به طرز فریب‌آمیزی ساده است. راه‌های متفاوت زیادی برای پیاده‌سازی این عبارت وجود دارد و انتخاب نحوه‌ی پیاده‌سازی می‌تواند تأثیر عظیمی بر عملکرد SA داشته باشد. یک روش تولید راه‌حل‌های نامزد آن است که یک نقطه‌ی تصادفی را در فضای جستجو انتخاب نمود. با این حال، پس از آنکه الگوریتم SA شروع به همگرایی به سمت یک راه‌حل خوب می‌نماید، انتظار خواهیم داشت راه‌حل نامزد کنونی (x_0) از بسیاری از نقاط دیگر موجود در فضای جستجو بهتر باشد. به همین دلیل احتمالاً تولید راه‌حل نامزد به‌صورت اتفاقی چندان مؤثر نخواهد بود. به‌عنوان یک قانون کلی، باید تولید راه‌حل‌های نامزد را به سمت راه‌حل نامزد کنونی x_0 گرایش‌دهی نمود. به همین دلیل است که معادلات (۹-۹) و (۹-۲۲) از یک متغیر گاوسی اتفاقی متمرکز در x_0 برای تولید راه‌حل‌های نامزد استفاده می‌نمایند. همچنین، واریانس متغیر اتفاقی گاوسی برابر با دما است. دما با گذشت زمان کاهش می‌یابد و بدین ترتیب با گذشت زمان و جلو رفتن دوره‌های SA جستجو باریک‌تر خواهد شد. می‌توان از معادله‌ی (۹-۱۹) و توزیع کوشی به‌عنوان یک روش پرتکاپوتر برای تولید راه‌حل‌های نامزد استفاده نمود و در عین حال در x_0 متمرکز باقی ماند. گرایش دادن تولید راه‌حل‌های نامزد به سمت x_0 نه تنها باعث کنار گذاشته شدن راه‌حل‌های ضعیف، بلکه باعث کنار گذاشته شدن راه‌حل‌های خیلی خوب نیز خواهد شد. با این حال، معمولاً تعداد نقاط ضعیف در فضای جستجو بسیار بیشتر از تعداد نقاط خیلی خوب بوده و به همین دلیل معمولاً این نوع گرایش‌دهی مؤثر واقع خواهد شد.

۹-۴-۲ مقداردهی اولیه دوباره

همان‌طور که پیش از این نیز گفته شد، زمانبندی سردشدن پارامتر مهمی در عملکرد SA است. اگر دما را سریع کم کنیم، SA در بهینه‌ی محلی به دام افتاده و عملکرد ضعیفی خواهد داشت. با این حال، معمولاً نمی‌توان از پیش دانست بهترین زمانبندی سردشدن چیست. بنابراین، معمولاً باید بهبود الگوریتم SA را رصد کرده و اگر برای L دوره راه‌حل بهتری نیافتیم، دما را دوباره با T_0 مقداردهی اولیه خواهیم نمود تا میزان کاوش را افزایش دهیم.

۹-۴-۳ دنبال نمودن بهترین راه‌حل نامزد

از شکل ۹-۲ به خاطر آورید که ممکن است یک راه‌حل نامزد x جای راه‌حل نامزد کنونی x_0 را بگیرد، هرچند x از x_0 بدتر باشد. این ریسکی است که باید برای جستجوی کافی فضای جستجو به جان خرید، اما

این ریسک ممکن است منجر به از دست رفتن راه‌حل‌های خوب شود. بنابراین، ما معمولاً یک آرشیو را پیاده‌سازی کرده و با استفاده از آن بهترین راه‌حل‌های به دست آمده تا به حال را دنبال می‌نماییم. این موضوع مانند نخبه‌گرایی بخش ۸-٤ است. با این حال، در آن بخش بهترین راه‌حل نامزد را در میان جمعیت نگه می‌داشتیم. این کار را نمی‌توان مستقیماً در SA انجام داد، مگر آنکه اندازه‌ی جمعیت را به بیش از ۱ ذره افزایش دهیم، موضوعی که در این فصل آن را مورد بحث قرار ندادیم. با این حال، صرف نظر از اندازه‌ی جمعیت، می‌توان یک آرشیو ایجاد نمود و بهترین راه‌حل پیدا شده تا به حال را در آن نگه داشت. بدین ترتیب، حتی اگر به دلیل طبیعت کاوش‌گر SA، یک راه‌حل خوب با یک راه‌حل ضعیف جایگزین شود، بهترین راه‌حل پیدا شده تا کنون را پیگیری خواهیم نمود. بهترین راه‌حل نامزد موجود در آرشیو هیچگاه با یک راه‌حل بدتر جایگزین نخواهد شد.

۹-۵ نتیجه‌گیری

ذوب فلزات یکی از الگوریتم‌های تکاملی قدیمی است که در سال ۱۹۸۳ ایجاد شده است، اما ما آن را در این فصل از کتاب مورد بحث قرار دادیم چرا که نمی‌توان این الگوریتم را صرفاً یک الگوریتم کلاسیک در نظر گرفت. این الگوریتم یک الگوریتم جمعیت محور نیست، اما واضح است که برخی از الگوریتم‌های تکاملی کلاسیک نیز جمعیت محور نیستند. بدین ترتیب این دلیل خوب و کافی برای جدا نمودن این الگوریتم از الگوریتم‌های تکاملی نخواهد بود. از آنجا که SA بر اساس یک فرایند طبیعی بوده و یک الگوریتم بهینه‌سازی دوره‌ای است، معمولاً آن را یک الگوریتم تکاملی در نظر می‌گیرند. بلوغ و ریشه‌های علمی این الگوریتم، مقالات، کتاب‌ها و کاربردهای بسیاری را نتیجه داده است. خوانندگانی که به دنبال پوششی جامع‌تر و کامل‌تر از SA هستند می‌توانند به [وان‌لورهوون[1]، و آرتز[2]، ۲۰۱۰]، [اوتِن[3] و وان‌گینکِن[4]، ۱۹۸۹] و [آرتز و کُرست[5]، ۱۹۸۹] مراجعه نمایند. فصل‌های آموزشی را می‌توان در [آرتز و همکاران، ۲۰۰۳] و [هِندرسون[6] و همکاران، ۲۰۰۳] یافت.

مانند بسیاری دیگر از الگوریتم‌های تکاملی بحث شده در این کتاب، SA را نیز می‌توان برای گستره‌ی وسیعی از مسائل بهینه‌سازی با دامنه‌ی پیوسته و گسسته به کار برد. جهت‌های تحقیقاتی کنونی در زمینه‌ی SA تأکید موجود در تحقیقات کلی الگوریتم‌های تکاملی را بازتاب می‌دهد: SA برای مسائل چندهدفه

[1] Van Laarhoven
[2] Aarts
[3] Otten
[4] Van Ginneken
[5] Korst
[6] Henderson

[باندیوپودهیی[1] و همکاران، ۲۰۰۸]، پیوندزنی SA با سایر الگوریتم‌های تکاملی [کاکیر[2] و همکاران، ۲۰۱۱]، موازی‌سازی [زیمرمن[3] و لینچ[4]، ۲۰۰۹] و بهینه‌سازی مقید [سینگ[5] و همکاران، ۲۰۱۰].

این فصل به بحث در مورد پیشینه و کاربردهای SA پرداخته است، اما جنبه‌های بسیار دیگری از SA وجود دارد که وقت کافی برای بحث در مورد آن‌ها در این کتاب در دسترس نیست. برای مثال، یک مدل مارکوف و برخی اثبات‌های نظری برای همگرایی در [میچیلز[6] و همکاران، ۲۰۰۷] ارائه شده است، هرچند همچنان جای زیادی برای مدل‌سازی‌ها و نتایج نظری وجود دارد. محققین نه تنها به همگرایی، بلکه به عملکرد الگوریتم در طول بازه‌های زمانی متناهی علاقه‌مندند. این موضوع در [هِندرسون و همکاران، ۲۰۰۳] و [وُروِرک[7] و همکاران، ۲۰۰۹] ارائه شده است.

مسائل

مسائل نوشتاری

۹-۱ واحد دما در معادله‌ی (۹-۱) چیست؟

۹-۲ یک نمودار کیفی صحیح از احتمال $P(r|q)$ از بخش ۹-۱ را به‌عنوان تابعی از $\Delta E = E(r) - E(q)$ رسم کنید.

۹-۳ اگر راه‌حل نامزد جدید x دارای هزینه‌ی بیشتری نسبت به راه‌حل نامزد کنونی x_0 باشد، آنگاه چه مقادیری از ثابت احتمال پذیرش c یک احتمال پذیرش p را نتیجه خواهد داد؟

۹-٤ فرض کنید می‌خواهید SA را برای ۱۰۰۰۰ دوره و با استفاده از سردشدن خطی اجرا نمایید. از چه مقداری از η باید استفاده نمود تا دما در دوره‌ی نهایی به صفر برسد؟

۹-۵ "بزرگ به اندازه‌ی کافی" در اثبات همگرایی زمانبندی سردشدن لگاریتمی چه قدر است؟

۹-٦ "مقادیر مناسب T_0" در اثبات همگرایی زمانبندی سردشدن خطی معکوس چیست؟

۹-۷ زمانبندی سردشدن خطی و خطی معکوس را به فرم $T_{k+1} = \alpha(k, T_k)$ که در آن k شماره‌ی دوره است، بنویسید.

۹-۸ این مسئله، سردشدن نمایی و سردشدن معکوس را مقایسه می‌نماید.

---

[1] Bandyopodhyay
[2] Cakir
[3] zimmerman
[4] Lynch
[5] Singh
[6] Michiels
[7] Vorwerk

الف) زمانبندی سردشدن نمایی و معکوس را به فرم $T_{k+1} = f(k, T_0)$ که در آن k شماره‌ی دوره بوده و T_0 دمای اولیه است، بنویسید.

ب) در زمانبندی سردشدن نمایی، a را به‌گونه‌ای بیابید که دما بعد از N دوره برابر دما در زمانبندی سردشدن معکوس باشد.

پ) اگر $T_0 = 100$ باشد چه مقادیری از a بعد از ۱۰۰۰۰ دوره دمای معادل را به دست می‌دهد اگر: ۱) $\beta = 0.01$، ۲) $\beta = 0.001$، ۳) $\beta = 0.0001$؟

مسائل کامپیوتری

۹–۹ این مسئله به کاوش استراتژی بازمقداردهی بحث شده در بخش ۹–٤–۲ می‌پردازد. یک SA برای مینیمم‌سازی تابع ۲۰ بعدی آکلی پیاده‌سازی نمایید. از پارامترهای زیر استفاده نمایید:

- سردشدن معکوس با $\beta = 0.001$
- دمای اولیه = ۱۰۰
- محدودیت دوره = ۱۰۰۰۰
- تست پذیرش با معادله‌ی (۹–٤) با $c = 1$
- تولید راه‌حل‌های نامزد به‌صورت $x \leftarrow x_0 + r$ که در آن r یک عدد اتفاقی با میانگین صفر و توزیع نرمال و واریانس T^2 است.

بهترین راه‌حل یافت شده تاکنون (x_k^*) را به‌عنوان تابعی از شماره‌ی دوره نگه‌داری کرده و نمودار آن را به‌صورت میانگین‌گیری شده بر روی ۲۰ شبیه‌سازی مونت کارلو رسم نمایید. این نمودار را برای استراتژی‌های بازمقداردهی زیر مقایسه نمایید:

- هر گاه x بدتر از x_0 بود، T را برای ۱۰ دوره‌ی پشت سر هم بازمقداردهی کنید.
- هر گاه x بدتر از x_0 بود، T را برای ۱۰۰ دوره‌ی پشت سر هم بازمقداردهی کنید.
- هر گاه x بدتر از x_0 بود، T را برای ۱۰۰۰ دوره‌ی پشت سر هم بازمقداردهی کنید.
- هیچگاه T را بازمقداردهی نکنید.

از این مقایسه چه نتیجه‌ای می‌گیرید.

۹–۱۰ این مسئله به کاوش روش‌های به کار رفته برای تولید راه‌حل‌های نامزد می‌پردازد. یک SA برای مینیمم‌سازی تابع ۲۰ بعدی آکلی پیاده‌سازی نمایید. از همان پارامترهای مسئله‌ی ۹–۹ در اینجا نیز استفاده کنید. بهترین راه‌حل یافت شده تا کنون (x_k^*) را به‌عنوان تابعی از شماره‌ی دوره نگه‌داری کرده و نمودار آن

را به‌صورت میانگین‌گیری شده بر روی ۲۰ شبیه‌سازی مونت کارلو رسم نمایید. . این نمودار را برای استراتژی‌های تولید راه‌حل‌های نامزد مقایسه نمایید:

- $x \leftarrow x_0 + r_1$ که در آن r_1 یک عدد اتفاقی با میانگین صفر و توزیع نرمال و واریانس T^2 است.
- $x \leftarrow x_0 + r_2$ که در آن r_2 یک عدد اتفاقی با توزیع یکنواخت بر روی دامنه‌ی جستجو است.

از این مقایسه چه نتیجه‌ای می‌گیرید؟

فصل دهم

بهینه‌سازی کلونی مورچگان

مورچه‌ها موجودات کوچکی هستند اما می‌توانند با همکاری با یکدیگر کارهای زیادی انجام دهند. مورچه‌ها هم نمونه‌ی سخت‌کوشی و هم نمونه‌ی همکاریِ غیر خودخواهانه‌اند. یک مورچه به تنهایی نمی‌تواند کار زیادی انجام دهد. یک مورچه‌ی تنها ممکن است آنقدر بر روی یک دایره به دور خود بچرخد تا از خستگی بمیرد [دِلسوک[1]، ۲۰۰۳]. یک مورچه‌ی معمولی دارای حدود ۱۰۰۰۰ نورون در مغز خود است. با این مقدار نورون نمی‌توان کارهای زیادی را از مورچه انتظار داشت. اما مورچه‌ها در کلونی‌هایی از رده‌ی میلیون به یکدیگر می‌پیوندند. یک کلونی با اندازه جمعیت ۱ میلیون دارای نورون جمعی از رده‌ی ۱۰ بیلیون (۱۰ میلیارد است) که این مقدار می‌تواند رقیبی برای تعداد نورون‌های یک انسان عادی به حساب بیابد. مورچه‌ها در ظاهر مانند یک موجود واحد عمل می‌نمایند و به همین دلیل گاهاً از آن‌ها با عنوان سوپرارگانیسم‌ها یاد می‌شود [هولدوبلر[2] و ویلسون[3]، ۲۰۰۸]. در سال ۱۹۷۹ یک کلونی از مورچگان با اندازه‌ی جمعیتی برابر ۳۰۰ میلیون در جزیره‌ی هوکایدو[4] در ژاپن گزارش شد که در ٤٥۰۰۰ لانه‌ی متصل به هم با یکدیگر زندگی می‌کردند [هولدوبلر و ویلسون، ۱۹۹۰، صفحه‌ی ۱]. مورچه‌ها تقریباً در هر شرایط محیطی موجود بر روی کره‌ی زمین رشد می‌کنند و به‌طور تخمینی حدود ۱۵٪ جرم کل موجودات خشکی روی زمین را تشکیل می‌دهند [شولتز[5]، ۲۰۰۰]. بنا بر گفته‌های مورچه‌شناسان، حدود ۸۸۰۰ گونه‌ی مختلفِ مورچه بر روی زمین وجود دارد و تعداد کل مورچه‌های روی زمین به قدری است که به ازای هر انسان حدود ۱۵۰۰۰۰ مورچه وجود دارد این اعداد ما را به این فکر می‌اندازد که این موجودات بسیار کوچک چگونه توانسته‌اند چنین مدت مدیدی را در شرایط مختلف محیطی دوام بیاورند؟ دانشمندان این موفقیت را به قابلیت وفق‌پذیری و چیرگی اجتماعی مورچگان نسبت می‌دهند. از این منظر، شاید دلیل آنکه ما انسان‌ها نیز چیره‌ترین پستانداران موجود بر روی کره‌ی زمین هستیم، همان قابلیت چیرگی اجتماعی مطلقمان است.

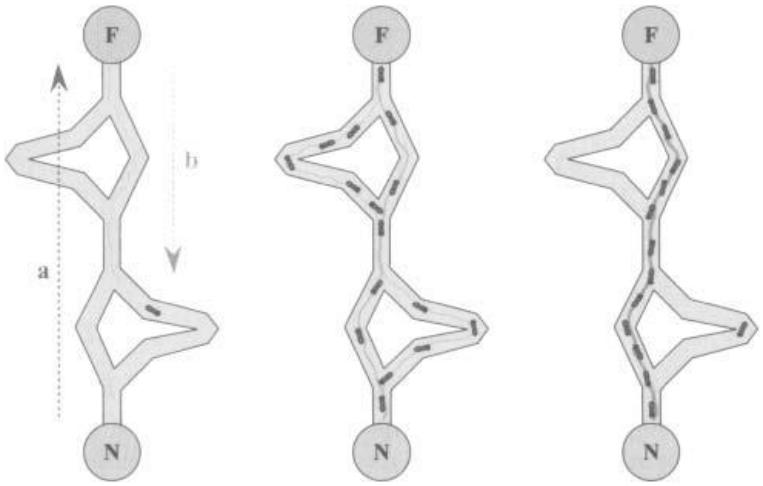
مورچه‌ها از طریق فرومون، نوعی ماده‌ای شیمیایی که مورچه‌ها از خود ترشح می‌کنند، با یکدیگر ارتباط برقرار می‌کنند. هنگامی که مورچه‌ها در یک مسیر به سمت یک منبع غذایی حرکت می‌کنند و برای کلونی خود غذا می‌آورند، یک رد از فرومون از خود به جای می‌گذارند. مورچه‌های دیگر با استفاده از آنتن‌های خود این فرومون را حس کرده و آن را به سمت منبع غذایی دنبال کرده و غذای بیشتری را برای کلونی می‌آورند. در این فرایند، مورچه‌ها به ترشح کردن فرومون بر روی این مسیر ادامه می‌دهند تا مورچه‌های دیگر آن را حس کنند و بدین ترتیب فرومون موجود بر روی مسیر منتهی به منبع غذایی دائماً تقویت می‌شود.

---

[1] Delsuc
[2] Holldobler
[3] Wilson
[4] Hokkaido
[5] Schultz

به همین دلیل و به دلیل وجود فیدبک مثبت، کوتاه‌ترین مسیر منتهی به منبع غذایی به مرور دارای فرومون بیشتری شده و برای مورچه‌ها جذاب‌تر می‌شود.

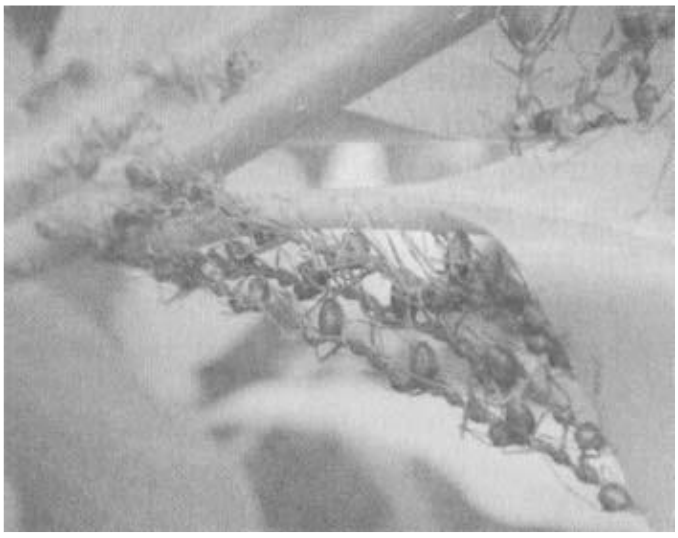
در این میان گاهی منبع غذایی تمام شده و یا یک شی مانع از رفتن به سمت منبع غذایی می‌شود. وقتی مورچه‌ها بر روی یک مسیر حرکت کرده و هیچ غذایی پیدا نمی‌کنند، آنقدر به گشت‌زنی ادامه می‌دهند تا بالاخره یک منبع غذایی پیدا کنند. اگر آن‌ها از طریق راه اولیه به کلونی خود باز نگردند، از ترشح فرومون اضافی بر روی آن مسیر خودداری خواهند کرد. با گذشت زمان، فرومون موجود بر روی مسیر اولیه تبخیر شده و به همین دلیل مورچه‌های کمتری از مسیر ابتدایی استفاده کرده و بیشتر مورچه‌ها از مسیر جدید برای رسیدن به منبع غذایی استفاده کرده و بدین ترتیب راه‌های بهینه‌ی جدیدی توسط مورچه‌ها کشف می‌شود. این فرایند کلی در شکل ۱۰-۱ نشان داده شده است.



شکل ۱۰-۱ ترشح و دنبال کردن فرومون توسط مورچه‌ها. (۱) مورچه‌ی اول بر روی مسیر a حرکت کرده و منبع غذایی F را پیدا می‌کند، سپس با استفاده از مسیر b به سمت لانه N برمی‌گردد و در تمام طول این مسیر از خود فرومون به جای می‌گذارد. (۲) مورچه‌ها یکی از چهار مسیر موجود را از N به F دنبال می‌کنند اما فرایند تقویت فرومون باعث می‌شود تا مسیر کوتاه‌تر جذاب‌تر شود. (۳) مورچه‌ها تمایل بیشتری به دنبال کردن مسیری که دارای فرومون بیشتری است از خود نشان می‌دهند و بدین ترتیب مطلوبیت آن را تقویت می‌کنند. این در حالی است که فرومون موجود بر روی مسیر طولانی‌تر به مرور تبخیر می‌شود (این شکل توسط ژوهان دِروو[1] ایجاد شده و از http://en.wikipedia.org/wiki/File:Aco_branches.svg کپی شده است).

[1] Johann Dreo

مورچه‌ها نه تنها می‌توانند کوتاه‌ترین مسیر به سمت منبع غذایی را بیابند بلکه می‌توانند کارهای مؤثر بسیاری را با کمک یکریگر انجام دهند. آن‌ها می‌توانند شبکه‌های پیچیده‌ای از تونل‌ها را در زیر زمین و یا درون درختان به وجود آورند. کلونی‌های آن‌ها دارای اتاق‌های مخصوصی برای جمع‌آوری غذا، جفت‌گیری و مراقبت از لاروها می‌باشند. آن‌ها می‌توانند گیاه بکارند تا منبع غذایی خود را کشت کنند [شولتز، ۱۹۹۹]. آن‌ها می‌توانند برای عبور از حفره‌های موجود بر روی زمین یا برای عبور از آب زنجیره‌هایی طویل ایجاد کنند (شکل ۱۰-۲ را ببینید). آن‌ها همچنین می‌توانند برای عبور از آب و یا برای جان به در بردن از سیل قایق‌های مسطحی ایجاد کنند.



شکل ۱۰-۲ مورچه‌ها در حال تشکیل یک پل میان برگ‌ها. مورچه‌ها از این پل‌ها نه تنها برای جابه‌جایی بلکه برای نزدیک کردن برگ‌ها هنگام ساخت و ساز لانه‌ها استفاده می‌نمایند. عکس‌های بسیار دیگری مانند این عکس را می‌توان در [هولدوبلر و ویلسون، ۱۹۹٤] یافت (این عکس توسط شان هویلند[1] گرفته شده است و از http://en.wikipedia.org/wiki/File:SSL11903.jpg کپی شده است).

مروری بر فصل

ما در این فصل به بحث در مورد الگوریتم کلونی مورچگان (ACO) می‌پردازیم. این الگوریتم از فرایند ترشح فرومون توسط مورچه‌ها الهام گرفته شده است. بسیاری از محققان فعال در زمینه‌ی ACO بر این موضوع تأکید دارند که از آنجا که در این الگوریتم راه‌حل‌های نامزد مستقیماً به تبادل اطلاعات نمی‌پردازند، نمی‌توان این الگوریتم را در زمره‌ی الگوریتم تکاملی قرار داد. ما در این کتاب به بحث در مورد ACO

[1] Sean Hoyland

می‌پردازیم نه از آن جهت که بخواهیم وارد بحث در مورد الگوریتم تکاملی بودن یا نبودن آن بشویم، بلکه صرفا به این دلیل که این الگوریتم، یک الگوریتمِ بهینه‌سازی جمعیت- محور و الهام گرفته شده از زیست‌شناسیِ بسیار جالب و مؤثر است.

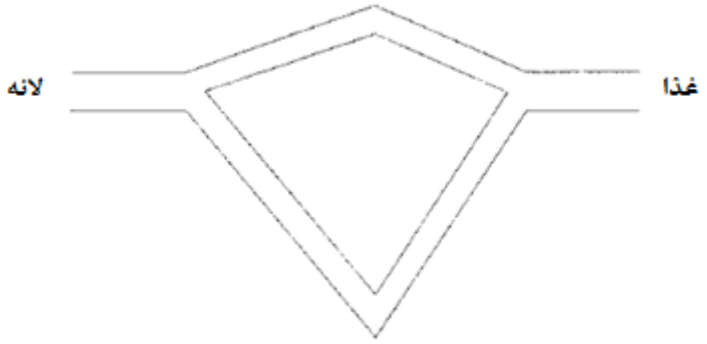
الگوریتم ACO ابتدا توسط مارتین دوریگو[1] و در غالب رساله‌ی دکتری معرفی گردید و برای اولین بار در سال ۱۹۹۱ به چاپ رسید [کولورنی[2] و همکاران، ۱۹۹۱]. در بخش ۱۰-۱ به بحث در مورد نحوه‌ی تولید فرومون توسط مورچه‌های واقعی و همچنین نحوه‌ی تبخیر آن پرداخته و مدل‌های ریاضی این فرایندها را معرفی خواهیم کرد. در بخش ۱۰-۲، سیستم مورچه (AS[3]) را که در اواسط دهه‌ی ۱۹۹۰ برای اولین بار ارائه گردید و اولین الگوریتم ACO به شمار می‌رفت، معرفی خواهیم کرد. الگوریتم ACO در ابتدا برای پیدا کردن بهترین مسیر ارائه شد اما به زودی به نحوی اصلاح شد که بتوان از آن برای حل مسائل بهینه‌سازی با دامنه‌ی پیوسته استفاده نمود. این موضوع در بخش ۱۰-۳ مورد بحث قرار خواهد گرفت. در بخش ۱۰-٤ برخی اصلاحات مشهورِ انجام گرفته بر روی الگوریتم سیستم مورچه ارائه خواهند گردید. از جمله‌ی این اصلاحات می‌توان به سیستم مورچه‌ی کمینه- بیشینه (MMAS[4]) و سیستم کلونی مورچه (ACS[5]) اشاره نمود. در انتها و در بخش ۱۰-۵ برخی از تحقیقات موجود در زمینه‌ی نظری و مدل‌سازیِ ACO ارائه خواهند شد.

۱۰-۱ مدل‌های فرومون

فرض کنید در حال رصد یک لانه‌ی مورچه و یک منبع غذا هستیم. همچنین فرض کنید تنها دو مسیر ممکن جهت رسیدن به منبع غذایی وجود دارد. یکی از این مسیرها طولانی‌تر از دیگری است. این دو مسیر در شکل ۱۰-۳ نشان داده شده‌اند. گاس[6] و همکارانش آزمایش‌های بسیاری از این دست را بر روی مورچه‌های آرژانتین[7] انجام دادند و دریافتند که در ۹۵٪ از آزمایش‌های آن‌ها حدود ۸۰٪ ترافیک مورچه‌ها بر روی مسیر کوتاه‌تر قرار دارد [گاس و همکاران، ۱۹۸۹]. در ابتدا هنگامی که مورچه‌ها به محل انشعاب مسیرها می‌رسیدند، به‌صورت اتفاقی یکی از مسیرها را انتخاب می‌نمودند. در این میان مورچه‌هایی که مسیر کوتاه‌تر را انتخاب کرده بودند زودتر از مورچه‌هایی که مسیر طولانی‌تر را برگزیده بودند به لانه می‌رسیدند. این موضوع باعث می‌شد تا مورچه‌های بیشتری در واحد زمان از مسیر کوتاه‌تر استفاده نمایند. این خود باعث می‌شد تا فرومون

[1] Martin Dorigo
[2] Colorny
[3] Ant System
[4] Max-Min Ant System
[5] Ant Colony System
[6] Goss
[7] Argentine Ant

بیشتری بر روی مسیر کوتاه‌تر قرار بگیرد. در نهایت، مقدار زیاد فرومون موجود بر روی مسیر کوتاه‌تر باعث می‌گردید مورچه‌های بعدی انگیزه‌ی بیشتری برای استفاده از مسیر کوتاه‌تر داشته باشند.



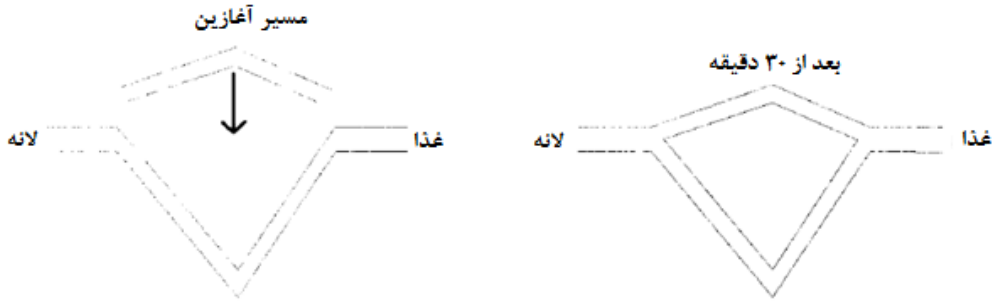
شکل ۱۰-۳ یک طرح‌ریزی تجربی برای دریافت آنکه چگونه مورچه‌ها کوتاه‌ترین مسیر به سمت منبع غذایی را می‌یابند. در ۹۵٪ از آزمایش‌ها، ۸۰٪ ترافیک مورچه‌ها بر روی مسیر کوتاه‌تر قرار دارد. برگرفته شده از [گاس و همکاران، ۱۹۸۹].

می‌توان دید که حرکت مورچه‌ها، حداقل تا یک نقطه‌ی مشخص، نوعی فیدبک مثبت است. همواره تعدادی از مورچه‌ها هستند که مسیر طولانی‌تر را انتخاب می‌نمایند چرا که فرایند انتخاب مسیر در کنه یک فرایند اتفاقی است. با این حال و در کل، هرچه تعداد مورچه‌هایی که مسیر کوتاه‌تر را انتخاب می‌نمایند افزایش می‌یابد، مسیر کوتاه‌تر فرومون بیشتری دریافت کرده و هرچه مسیر کوتاه‌تر فرومون بیشتری دریافت می‌نماید، مورچه‌های بیشتری آن را انتخاب می‌نمایند (فیدبک مثبت).

این پدیده‌ی فیدبک مثبت از ویژگی‌های الگوریتم‌های تکاملی است. برای مثال، در GAها ذراتی که در نسل اول دارای ویژگی‌های سودمند ژنتیکی هستند از شانس بیشتری جهت انتخاب شدن برای عمل بازترکیب برخوردار هستند. این بدین معنی است که احتمال وجود این ویژگی‌های ژنتیکی در میان ذرات نسل دوم بیشتر است. این افزایش شیوع ویژگی‌های مذکور در میان نسل دوم خود باعث می‌شود تا احتمال انتقال این ویژگی‌ها به نسل سوم نیز افزایش یابد. این فیدبک مثبت نه تنها در ACO و GAها، بلکه در تمامی الگوریتم‌های تکاملی دیده می‌شود.

همان‌طور که پیش از این نیز گفتیم، فرومون‌ها تبخیر می‌شوند. گاس یک آزمایش دیگر نیز انجام داده است که در آن تنها یک مسیر برای کلونی مورچه‌ها در دسترس است. مورچه‌ها بر روی این مسیر از لانه به سمت منبع غذایی و برعکس حرکت می‌کردند چرا که گزینه‌ی دیگری نداشتند. پس از گذشت مدتی، گاس یک مسیر کوتاه به سمت منبع غذایی اضافه کرد. این موضوع در شکل ۱۰-٤ نشان داده شده است. مورچه‌ها حال یک مسیر دیگر نیز در اختیار داشتند، اما همه‌ی فرومون بر روی مسیر طولانی‌تر قرار داشت. با این حال،

هنگامی که مورچه‌ها به محل انشعاب مسیر می‌رسند، به‌طور خودکار مسیر اشباع شده از فرومون را انتخاب نمی‌کنند. هرچند که مورچه‌ها تمایل به انتخابِ مسیر با فرومون بیشتر را دارند، اما در عین حال یک عنصر اتفاقی در رفتار آن‌ها نیز وجود دارد. بنابراین، برخی از مورچه‌ها مسیر جدید کوتاه‌تر را در پیش گرفتند. با انتخاب شدن این مسیر توسط مورچه‌ها، فرومون بیشتری بر روی این مسیر قرار گرفته و به همین دلیل به مسیری جذاب‌تر برای مورچه‌های بعدی تبدیل خواهد شد. در حدود ۲۰٪ از آزمایش‌ها، بیشتر مورچه‌ها از مسیر کوتاه‌تر استفاده نمودند هرچند که هیچ فرومون اولیه‌ای بر روی این مسیر وجود نداشت. این موضوع مبین این حقیقت است که فرومون تبخیر می‌شود. با این حال، در این آزمایش سرعت تبخیر شدن فرومون به اندازه‌ای نبود که باعث شود مورچه‌ها مسیر کوتاه‌تر را خیلی بیشر از مسیر طولانی‌تر انتخاب نمایند.



شکل ۱۰-٤ یک طرح‌ریزی تجربی برای دریافت نحوه‌ی رفتار مورچه‌ها هنگامی یک مسیر کوتاه جدید به مسیرهای در دسترس اضافه می‌گردد. در ۲۰٪ از آزمایش‌ها حدود ۵۰٪ ترافیک مورچه‌ها به سمت مسیر کوتاه‌تر همگرا می‌گردد. برگرفته شده از [گاس و همکاران، ۱۹۸۹].

دنیوبُرگ[1] و همکارانش یک مدل ریاضی برای ترشح و تبخیر فرمون، بر پایه‌ی این آزمایش‌های تجربی ارائه دادند [دنیوبُرگ و همکاران، ۱۹۹۰]. اگر دو مسیر برای انتخاب وجود داشته باشد، احتمال انتخاب مسیر اول توسط مورچه توسط معادله‌ی زیر به دست می‌آید

$$p_1 = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h} \qquad (۱۰-۱)$$

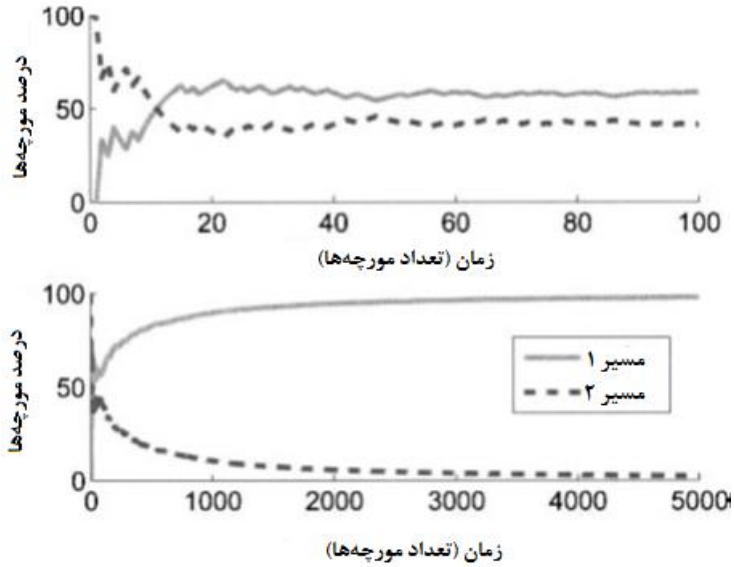
که در آن m_i برابر تعداد دفعاتی است که مسیر iاُم قبلاً توسط مورچه‌ها انتخاب شده است و h و k نیز پارامترهای هستند که به‌صورت تجربی تعیین شده‌اند. این مدل اولیه، تبخیر فرومون را نادیده می‌گیرد. مقادیر معمول برای h و k عبارتند از

[1] Deneubourg

$$k \approx 20, \quad h \approx 2 \qquad (۲-۱۰)$$

شکل ۱۰-۵ نتایج شبیه‌سازیِ معادله‌ی ۱۰-۱ را برای دو مسیر با طول یکسان نشان می‌دهد. نمودار بالایی نشانگر این موضوع است که رفتار ۱۰۰ مورچه‌ی اول قابل پیش‌بینی نیست. رفتار مورچه‌ها بیشتر حالتی اتفاقی دارد و احتمال انتخاب شدن مسیر ۱ یا ۲ هر کدام حدود ۵۰٪ است. نمودار پایینی نشانگر آن است که هرچه یک مسیر فرمون بیشتری دریافت می‌نماید، برای مورچه‌ها جذاب‌تر شده و به فیدبک مثبت فرومون که پیش از این درباره‌ی آن بحث کردیم منتهی می‌شود. در نهایت ۱۰۰٪ ترافیک مورچه‌ها بر روی یکی از این دو مسیر قرار خواهد گرفت.

با توجه به آنچه گفته شد می‌توان دید که مورچه‌ها قادرند راه‌حل‌هایی بهینه برای مسائل روزمره‌ی خود بیابند. این موضوع ما را بر آن می‌دارد تا برای پیدا کردن راه‌حل‌های بهینه برای مسائل مهندسی از شبیه‌سازی مورچه‌های مصنوعی استفاده نماییم.



شکل ۱۰-۵ نتایج شبیه‌سازی معادله‌ی ۱۰-۱. در ابتدا احتمال انتخاب شدن هر یک از مسیرها توسط مورچه‌ها حدود ۵۰٪ است. پس از مدتی، یکی از مسیرها فرومون بیشتری نسبت به دیگری دریافت کرده که باعث به وجود آمدن فیدبک مثبت می‌شود و ۱۰۰٪ ترافیک مورچه‌ها بر روی یکی از این دو مسیر قرار خواهد گرفت.

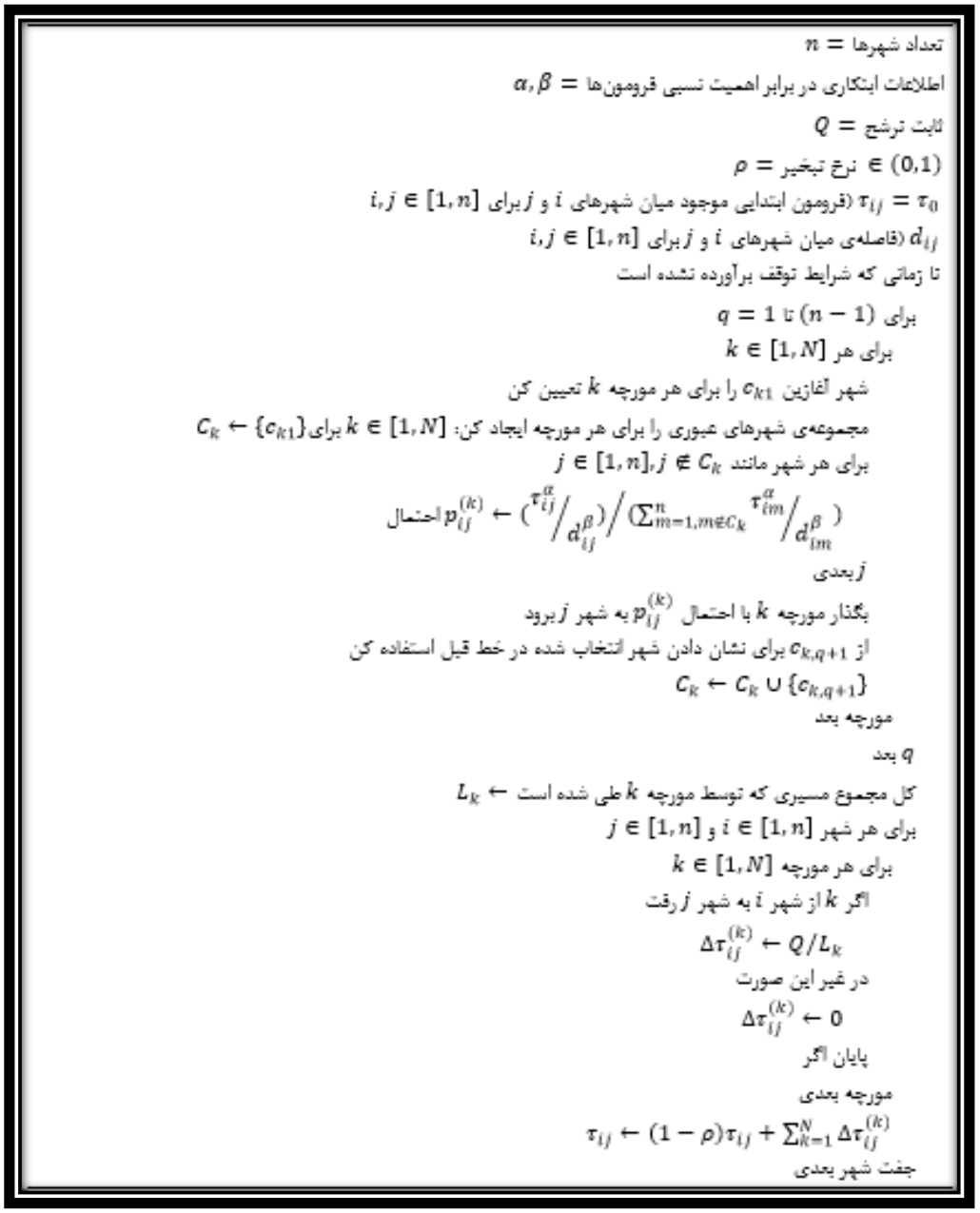
۱۰–۲ سیستم مورچه

سیستم مورچه اولین الگوریتم ACO منتشر شده است [کولورنی[1] و همکاران، ۱۹۹۱]، [دوریگو[2] و همکاران، ۱۹۹٦]. این الگوریتم را می‌توان بر روی مسئله‌ی فروشنده‌ی دوره‌گرد نشان داد (TSP؛ بخش ۲–۵ و فصل ۱۸ را ببینید). در شبیه‌سازی ACO هر مورچه از یک شهر به یک شهر دیگر سفر کرده و پس از تکمیل سفر، شبیه‌سازی فرومون را بر روی مسیر قرار می‌دهد. فرومون‌ها علاوه بر ترشح شدن، تبخیر نیز می‌شوند. احتمال رفتن یک مورچه از شهر حال حاضر خود به شهر دیگر، به مقدار فرومون موجود بین دو شهر بستگی دارد. همچنین فرض می‌شود مورچه‌ها دارای اطلاعاتی در مورد مسئله‌ی مورد بهینه‌سازی می‌باشند که این اطلاعات به آن‌ها برای گرفتن برخی تصمیمات در طول سفر کمک می‌نماید. آن‌ها از فاصله‌ی بین شهر حال حاضر خود و سایر شهرها مطلع هستند و همچنین تمایل دارند به شهرهای نزدیک‌تر سفر نمایند چرا که هدف الگوریتم پیدا کردن کوتاه‌ترین مسیر است. الگوریتم سیستم مورچه در شکل ۱۰–٦ نشان داده شده است.

شکل ۱۰–٦ نشان می‌دهد که احتمال رفتن مورچه از شهر i به شهر j با مقدار فرومون موجود میان دو شهر رابطه‌ی مستقیم و با فاصله‌ی میان دو شهر رابطه‌ی نسبت عکس دارد. نسبت α/β اهمیت نسبی اطلاعات مربوط به فرومون نسبت به اطلاعات مربوط به فاصله را هنگام تصمیم‌گیری در مورد شهر مقصد، تعیین می‌نماید. هنگامی که یک مورچه از شهر i به شهر j می‌رود، میزان فرومون موجود بر روی آن مسیر متناسب با کیفیت راه‌حل آن مورچه (که با مجموع مسافت‌های طی شده توسط آن مورچه نسبت عکس دارد) افزایش می‌یابد.

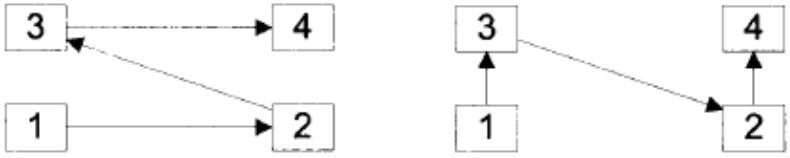
شکل ۱۰–٦ یک الگوریتم نسبتاً کامل است اما برخی جزئیات پیاده‌سازی به برنامه‌نویس واگذار شده است. برای مثال، آیا $\tau_{ij} = \tau_{ji}$ است؟ در سیستم زیستی مورچه، میزان فرومون موجود میان دو گره‌ی i و j با میزان فرومون موجود میان j و i برابر است اما لزومی ندارد در شبیه‌سازی سیستم مورچه نیز چنین باشد. می‌توان به راحتی تصور نمود که چگونه ممکن است حرکت از گره‌ی i به گره‌ی j منجر به یک راه‌حل خوب شده در حالی که حرکت از گره‌ی j به گره‌ی i به یک راه‌حل بد منجر شود. این موضوع به معنی آن است که $\tau_{ij} \neq \tau_{ji}$ بوده و با TSP نامتقارن متناظر است (شکل ۱۰–۷ را ببینید).

[1] Colorni
[2] Dorigo



شکل ۱۰-۶ یک سیستم مورچه‌ی (AS) ساده برای حل یک TSP.در هر نسل مقداری از فرومون موجود میان شهرهای i و j تبخیر می‌شود. همچنین با حرکت مورچه‌ها از یک شهر به شهر دیگر میزان فرومون موجود میان دو شهر افزایش می‌یابد.

برخی جزئیات پیاده‌سازی دیگر که می‌توان در شکل ۱۰-٦ لحاظ نمود شامل مقداردهی اولیه‌ی هوشمندانه، نخبه‌گرایی و جهش می‌شود. اول آنکه، عملکرد ACO مانند عملکرد سایر الگوریتم‌های تکاملی می‌تواند وابستگی شدیدی به مقداردهی اولیه‌ی مناسب داشته باشد (بخش ۸-۱ را ببینید). برای حل TSP می‌توان از یک الگوریتم ساده‌ی ابتکاری برای مقداردهی اولیه‌ی ذرات استفاده نمود. برای مثال، می‌توان در نسل اول یکی از مورچه‌ها را مجبور به رفتن به نزدیکترین شهر در هر نقطه‌ی تصمیم‌گیری نمود. در مورد مقداردهی اولیه‌ی TSP در بخش ۱۸-۲ بیشتر بحث خواهیم کرد. دوم آنکه، در ACO نیز می‌توان مانند هر الگوریتم تکاملی دیگر از نخبه‌گرایی استفاده نمود (بخش ۸-٤ را ببینید). برای دخالت دادن نخبه‌گرایی می‌توان رد بهترین مورچه‌ها را در هر نسل گرفت و آن‌ها را مجبور به تکرار همان مسیر قبلی در نسل بعد نمود. با این کار می‌توان اطمینان حاصل نمود که بهترین مسیر از یک نسل به نسل بعد از دست نمی‌رود. یک الگوریتم سیستم مورچه با نخبه‌گرایی معمولاً سیستم مورچه‌ی نخبه‌گرا نامیده می‌شود [دوریگو و همکاران، ۱۹۹٦]، [بلوم[1]، ۲۰۰۵a]. سوم آنکه می‌توان مانند سایر الگوریتم‌های تکاملی از جهش در ACO نیز استفاده نمود (بخش ۸-۹ را ببینید). برای این کار می‌توان برخی از مسیرها را به‌صورت اتفاقی و با یک احتمال جهش دستخوش تغییر نمود. محققین تاکنون چندین مکانیزم برای جهش دادن مسیرهای TSP ارائه داده‌اند. این مکانیزم‌ها در بخش ۱۸-٤ مورد بررسی قرار خواهند گرفت.



شکل ۱۰-۷ در این مثال فرض نموده‌ایم که سفر در گره‌ی ۱ آغاز می‌شود. مسیر سمت چپ بسیار بدتر از مسیر سمت راست است اما هر دو مسیر شامل مسیر میان گره‌های ۲ و ۳ می‌شوند. از آنجایی که مسیر سمت چپ طولانی بوده و مسیر سمت راست کوتاه است می‌توان انتظار داشت که در یک ACO مؤثر، τ_{23} از τ_{32} کمتر باشد، بدین معنی که رفتن از گره‌ی ۲ به ۳ باید از رفتن از گره‌ی ۳ به ۲ از جذابیت کمتری برخوردار باشد..

با توجه به شکل ۱۰-٦، چند پارامترِ میزان‌سازی برای سیستم مورچه وجود دارد. این پارامترها شامل موارد زیر می‌شوند:

- تعداد مورچه‌ها N، که نشان دهنده‌ی اندازه‌ی جمعیت است؛
- α و β که اهمیت نسبی مقادیر فرومون و اطلاعات ابتکاری هستند؛

[1] Blum

- Q که ثابت ترشح است؛
- ρ، که ثابت تبخیر است؛
- τ_0، که بیانگر مقدار فرومون اولیه میان هر دو شهر دلخواه است.

تأثیر این پارامترها بر روی این الگوریتم توسط چندین محقق مورد مطالعه قرار گرفته است. برای مثال، [دوریگو و همکاران، ۱۹۹۶]، مقادیر زیر را برای این پارامترها توصیه می‌کند:

- $N = n$ (یعنی تعداد مورچه‌ها با تعداد شهرها برابر است).
- $\alpha = 1$ و $\beta = 5$.
- $Q = 100$ هرچند که این پارامتر دارای تأثیر چندانی نیست.
- $\rho \in [0.5,0.99]$.
- $\tau_0 \approx 10^{-6}$.

مثال ۱۰-۱

این مثال، سیستم مورچه‌ی شکل ۱۰-۶ را به TSP برلین ۵۲ که شامل ۵۲ مکان در شهر برلین در آلمان می‌شود اعمال می‌کند [رینلت[1]. ۲۰۰۸]. برلین۵۲ یک TSP متقارن است بدین معنا که ما مجموعه‌ای از گره‌ها با فواصل مشخص از یکدیگر در اختیار داریم و می‌خواهیم یک مسیر رفت و برگشت با طول مینیمم پیدا کنیم به‌طوری که از هر گره دقیقاً یکبار عبور نماییم. در یک TSP متقارن مانند برلین۵۲، فاصله‌ی شهر i از j با فاصله‌ی شهر j از i برابر است. برای حل این مسئله از پارامترهای زیر استفاده می‌نماییم:

- $N = 53$[2]؛
- $\alpha = 1$ و $\beta = 5$.
- $Q = 20$.
- $\rho = 0.9$.
- $\tau_0 = 10^{-6}$.
- به‌طور کلی $\tau_{ij} \neq \tau_{ji}$[3].
- مقداردهی اویه‌ی اتفاقی.

---

[1] Reinelt

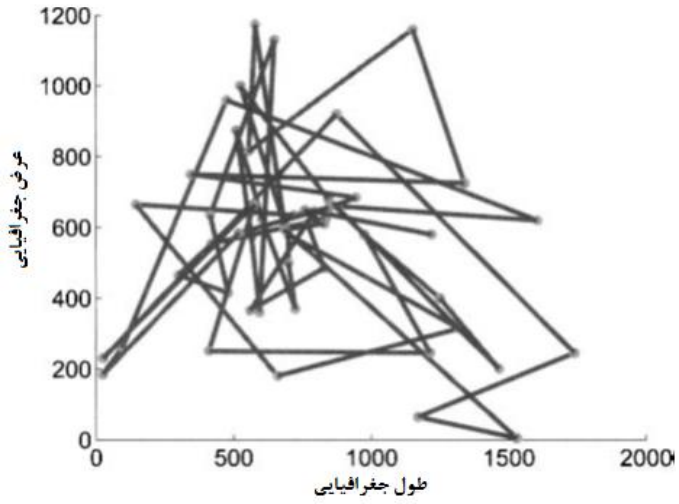
[2] مقدار استاندارد جمعیت اولیه‌ی سیستم مورچه از یک مقاله به مقاله‌ی دیگر متفاوت است. بسیاری از مقالات از $N = n$ استفاده می‌نمایند و این در حالی است که بسیاری دیگر از $N = n + 1$ استفاده می‌نمایند.

[3] معمولاً برای TSP متقارن $\tau_{ij} = \tau_{ji}$ توصیه می‌شود اما لزومی به استفاده از این تساوی نیست.

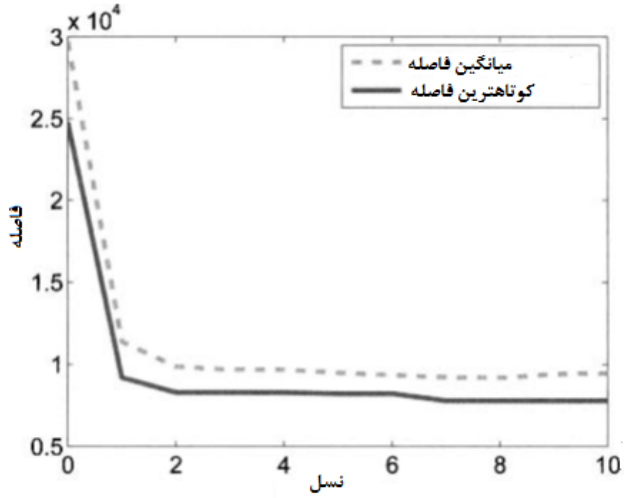
- دو مورچه‌ی نخبه در هر نسل.
- عدم استفاده از جهش.

شکل ۱۰–۸ بهترین مسیر جمعیت اولیه با طولی برابر ۲٤۷۸۰ را نشان می‌دهد. می‌توان دید که مسیر اولیه بسیار بد و ناامیدکننده است. شکل ۱۰–۹ همگرایی AS را در طول جستجوی آن برای یافتن بهترین مسیر را نشان می‌دهد. می‌توان دید که AS بسیار سریع همگرا شده و نخبه‌گرایی به کار رفته در این الگوریتم از عدم افزایش طول بهترین مسیر از یک نسل به نسل دیگر اطمینان حاصل می‌سازد. شکل ۱۰–۱۰ بهترین مسیر پیدا شده توسط AS بعد از ۱۰ نسل را با طولی برابر ۷۷۹٦ نشان می‌دهد. می‌توان دید که ACO توانسته مسیری بسیار بهتر از بهترین مسیرِ جمعیت اولیه بیابد و مجموع کلی مسیر ٦۹٪ کاهش یافته است.

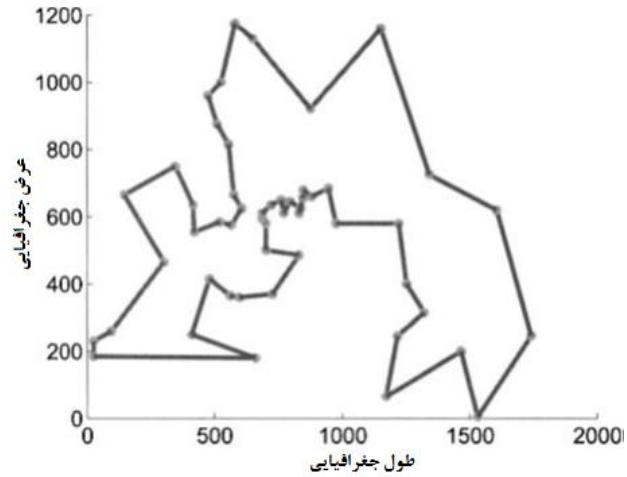
در نهایت، شکل ۱۰–۱۱ مسیر بهینه‌ی کلی را که بهینه بودن آن اثبات شده است را با طولی برابر ۷٥٤۲ نشان می‌دهد. با مقایسه‌ی شکل ۱۰–۱۰ و ۱۰–۱۱ می‌توان دید که ACO مسیری را یافته است که بسیار نزدیک به بهینه‌ترین مسیر بوده و تنها ۳٪ از آن بدتر است. توجه داشته باشید هر بار بهینه‌سازی ACO نتیجه‌ی متفاوتی را به دست می‌دهد چرا که ACO یک الگوریتم اتفاقی است. اما با توجه به آنکه $51! = 1.6 \times 10^{66}$ راه‌حل ممکن برای این مسئله وجود دارد، می‌توان گفت که ACO با پیدا کردن راه‌حلی که تنها ۳٪ از بهینه‌ترین راه‌حل بدتر است، بسیار خوب عمل کرده است.



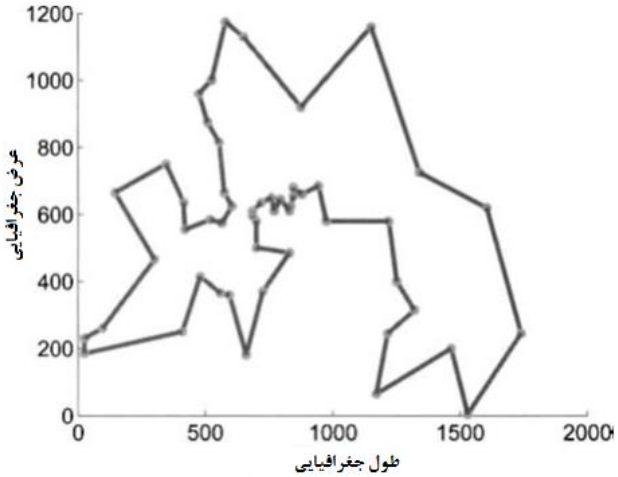
شکل ۱۰–۸ بهترین مسیر اولیه از میان ۵۳ مسیر اتفاقی ممکن برای مثال ۱۰–۱. مجموع طول این مسیر برابر با ۲٤۷۸۰ است.



شکل ۱۰-۹ همگرایی سیستم مورچه برای مثال ۱۰-۱.



شکل ۱۰-۱۰ بهترین مسیر پیدا شده توسط سیستم مورچه بعد از ۱۰ نسل در مثال ۱۰-۱. طول کلی این مسیر برابر با ۷۷۹۶ است. این مسیر ۶۹٪ از بهترین مسیر اولیه‌ی پیدا شده در شکل ۱۰-۸ بهتر است و ۳٪ از بهینه‌ترین مسیر (که بهینه بودن آن ثابت شده است . در شکل ۱۰-۱۱ نشان داده شده است) بدتر است.



شکل ۱۰-۱۱ مسیر بهینه‌ی کلی برای مثال ۱۰-۱ با طولی برابر ۷۵۴۲.

۱۰-۳ بهینه‌سازی پیوسته

در ابتدا ACO برای بهینه‌سازی مسائلی همچون TSP به کار گرفته شد اما همواره اصلاحاتی برای به کار بردن آن برای حل مسائل بهینه‌سازی با دامنه‌ی پیوسته، بر روی آن صورت گرفته است [سوچا[1] و دوریگو، ۲۰۰۸]، [تسوتسوی[2]، ۲۰۰٤]، [دفرانکا[3] و همکاران، ۲۰۰۸]. یکی از راه‌های ممکن برای اعمال یک الگوریتم بهینه‌سازی گسسته مانند ACO به یک مسئله با دامنه‌ی پیوسته آن است که هر بعد i از فضای جستجو را به بازه‌های گسسته تقسیم نماییم. در این صورت مسئله به مینیمم‌سازی تابع n بعدی $f(x)$ که در آن $x = [x_1, x_2, \dots, x_n]$ بوده، تبدیل می‌شود. در این صورت خواهیم داشت

$$x_{i,min} = b_{i1} < b_{i2} < \cdots < b_{i,B_i} = x_{i,max} \quad x_i \in [x_{i,min}.x_{i,max}] \qquad (۳-۱۰)$$

که در آن $B_i - 1$ برابر با تعداد بازه‌هایی است که هر بعد i به آن تقسیم می‌شود. در هر نسل اگر بعد iام راه‌حل نامزد بین b_{ij} و $b_{i,j+1}$ قرار بگیرد، آنگاه فرومون مرتبط با آن بازه را بنابر الگوریتم استاندارد سیستم مورچه به روزرسانی می‌نماییم:

$$\text{اگر } x_i \in [b_{ij}, b_{i,j+1}] \text{ آنگاه } \tau_{ij} \leftarrow \tau_{ij} + Q/f(x) \qquad (٤-۱۰)$$

[1] Socha
[2] Tsutsui
[3] De Franca

در معادله‌ی بالا Q ثابت ترشح سیستم استاندارد مورچه بوده و فرض بر آن است که برای تمامی xها $f(x) > 0$ است. معادله‌ی (۱۰-٤) با عبارت $\Delta\tau_{ij}^{(k)} \leftarrow {Q}/{L_k}$ در شکل ۱۰-٦ متناظر است. ما از مقادیر فرومون برای ساخت راه‌حل‌های جدید در ابتدای هر نسل به‌صورت احتمالاتی استفاده می‌نماییم. بدین ترتیب، اگر بازه‌ی $[b_{ij}, b_{i,j+1}[$ دارای مقدار زیادی فرومون باشد، احتمال ساخت یک راه‌حل نامزد به‌طوری که بعد iاُم آن در این بازه قرار بگیرد زیاد خواهد بود. یک راه ممکن برای انجام این کار آن است که بعد iام راه‌حل نامزد را برابر یک عدد اتفاقی $r \in [b_{ij}, b_{i,j+1}[$ قرار دهیم.

شکل ۱۰-۱۲ طرح کلی الگوریتم سیستم مورچه‌ی پیوسته را نشان می‌دهد. در این شکل فرض بر آن است که تابع هزینه، که با L_k نمایش داده شده است همواره برای تمامی kها مثبت است. اگر این ویژگی برای یک مسئله برآورده نشود، آنگاه مقادیر هزینه باید به‌گونه‌ای شیفت داده شود که این ویژگی برآورده شود. شکل ۱۰-۱۲ نخبه‌گرایی را شامل نمی‌شود اما می‌توان (و باید) به راحتی و نخبه‌گرایی تعریف شده برای الگوریتم‌های تکاملی در بخش ۸-٤ را برای ایم شکل لحاظ نمود. به عنوان گزینه‌ای دیگر، AS پیوسته را می‌توان با جستجوی محلی ترکیب نمود.

استفاده از بازه‌های گسسته‌سازی شده برای هر بعد مسئله، راهی ساده برای تعمیم AS به مسائل پیوسته است. یک راه دیگر برای پیاده‌سازی الگوریتم سیستم مورچه‌ی پیوسته آن است که از مقادیر فرومون برای ارائه‌ی تخمین‌های پیوسته از PDF گسسته استفاده نماییم [سایمون، ۲۰۰٦، فصل ۱۵]، [بلوم، ۲۰۰۵a].

تعداد ابعاد $= n$

بعد iاُم را به $B_i - 1$ بازه مطابق معادله‌ی (۱۰-۳) تقسیم کن

اهمیت مقدار فرومون‌ها $= \alpha$

ثابت ترشح $= Q$

نرخ تبخیر $= \rho \in (0,1)$

$\tau_{ij_i} = \tau_0$ (فرومون ابتدایی برای $i \in [1,n]$ و $j_i \in [1, B_i - 1]$

جمعیتی آغازین از مورچه‌ها (راه‌حل‌های نامزد) را به‌صورت اتفاقی مقدار دهی اولیه کن؛ a_k برای $k \in [1, N]$

تا زمانی که شرایط توقف برآورده نشده است

برای هر مورچه a_k

برای هر بعد $i \in [1,n]$

برای هر بازه‌ی گسسته شده‌ی $[b_{ij}, b_{i,j+1}[$، $j \in [1, B_i - 1]$

$p_{ij}^{(k)} \leftarrow \tau_{ij}^{\alpha} / \sum_{m=1}^{B_i-1} \tau_{im}^{\alpha}$ احتمال

مجموعه‌ی شهرهای عبوری را برای هر مورچه ایجاد کن: $k \in [1, N]$ برای $C_k \leftarrow \{c_{k1}\}$

برای هر شهر مانند $j \in [1,n], j \notin C_k$

$p_{ij}^{(k)} \leftarrow \left(\tau_{ij}^{\alpha} \middle/ d_{ij}^{\beta}\right) \middle/ \left(\sum_{m=1, m \notin C_k}^{n} \tau_{im}^{\alpha} \middle/ d_{im}^{\beta}\right)$ احتمال

j بعدی

بگذار مورچه k با احتمال $p_{ij}^{(k)}$ به شهر j برود

از $c_{k,q+1}$ برای نشان دادن شهر انتخاب شده در خط قبل استفاده کن

$C_k \leftarrow C_k \cup \{c_{k,q+1}\}$

مورچه بعد

هزینه راه‌حل تولید شده توسط a_k $\leftarrow L_k$

برای هر بُعد $i \in [1,n]$

برای هر بازه‌ی گسسته شده‌ی $[b_{ij}, b_{i,j+1}]$، $j \in [1, B_i - 1]$

برای هر مورچه a_k، $k \in [1, N]$

اگر $a_k(x_i) \in [b_{ij}, b_{i,j+1}]$

$\Delta\tau_{ij}^{(k)} \leftarrow Q/L_k$

در غیر این صورت

$\Delta\tau_{ij}^{(k)} \leftarrow 0$

پایان اگر

مورچه بعدی

$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{k=1}^{N} \Delta\tau_{ij}^{(k)}$

بازه‌ی گسسته بعدی

بُعد بعدی

نسل بعد

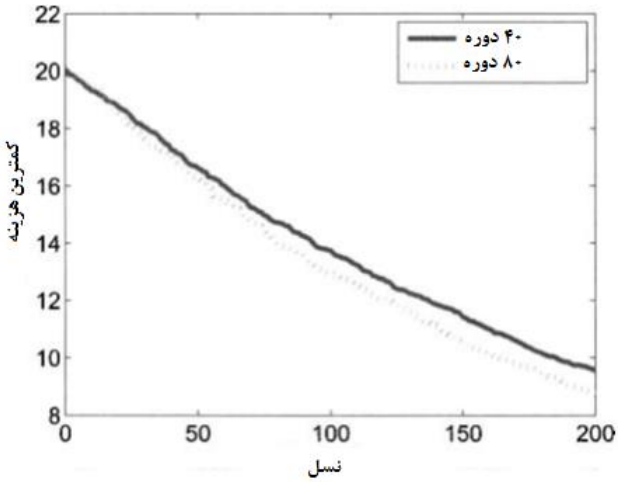
شکل ۱۰-۱۲ یک سیستم مورچه‌ی ساده برای حل مسئله‌ی مینیمم‌سازی پیوسته. $\alpha_k(x_i)$ نمایش‌گر iاُمین عنصر از kاُمین راه‌حل نامزد است. در هر نسل، فرومون موجود در هر بازه تبخیر می‌شود اما از سوی دیگر نیز فرومون موجود در هر بازه متناسب با تعداد مورچه‌های به وجودآورنده‌ی راه‌حل نامزد در آن بازه افزایش می‌یابد. $U[b_{ij}, b_{i,j+1}]$ یک عدد اتفاقی با توزیع یکنواخت میان b_{ij} و $b_{i,j+1}$ است.

مثال ۱۰–۲

در این مثال می‌خواهیم تابع ۲۰ بعدی آکلی را مینیمم‌سازی نماییم (ضمیمه‌ی ج.۲–۱ را ببینید). برای این کار از الگوریتم شکل ۱۰–۱۲ با پارامترهای زیر استفاده می‌نماییم:

- $N = 50$
- $\alpha = 1$
- $Q = 20$
- $\rho = 0.9$
- $\tau_0 = 10^{-6}$
- دو راه‌حل نخبه در هر نسل
- نرخ جهشی برابر با ۱٪ در هر بعد در هر ذره در هر تسل
- تعداد بازه‌ها: 80 یا $B_i = 40$ برای هر $i \in [1, n]$

شکل ۱۰–۱۳ بهترین راه‌حل یافت شده در هر نسل را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است را برای دوره‌های ٤٠ و ۸۰ تایی نشان می‌دهد. می‌توان دید که همگرایی با افزایش تعداد بازه‌ها در هر بعد بهبود می‌یابد اما از سویی نیز مدت زمان محاسبات با افزایش تعداد بازه‌ها، افزایش می‌یابد. دو علت برای این پدیده وجود دارد که می‌توان با نگاه به شکل ۱۰–۱۲ به این دو علت پی برد. علت اول، وجود حلقه‌ی for برای هر بازه‌ی گسسته‌سازی شده است. علت دوم نیز آن است که تصمیم‌گیری در مورد قرار دادن $\alpha_k(x_i)$ در کدام بازه پیچیده‌تر است. با این حال، باید توجه داشت که در بسیاری از مسائل بهینه‌سازی موجود در دنیای عملی، محاسبات تابع هزینه در اولویت قرار داشته و محاسبات اضافی مورد نیاز برای بازه‌های گسسته‌سازی شده مسئله‌ای چندان جدی به شمار نخواهد آمد (فصل ۲۱ را ببینید).



شکل ۱۰-۱۳ مثال ۱۰۲: همگرایی سیستم مورچه‌ی پیوسته که به تابع ۲۰ بعدی آکلی اعمال شده است. دو منحنی بهترین راه‌حل یافت شده در هر نسل را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهند. با پخش کردن فرومون بر روی تعداد بیشتری بازه در هر بعد، می‌توان به عملکرد بهتری دست پیدا کرد.

۱۰-۴ دیگر سیستم‌های مورچه

تا کنون اصلاحات زیادی بر روی الگوریتم سیستم مورچه‌ی استاندارد صورت گرفته است. این بخش به بحث در مورد دو اصلاحیه‌ی پایه می‌پردازد: سیستم مورچه‌ی ماکزیمم- مینیمم در بخش ۱۰-٤-۱ و سیستم کلونی مورچگان در بخش ۱۰-٤-۲.

۱۰-۴-۱ سیستم مورچه‌ی ماکزیمم- مینیمم

سیستم مورچه‌ی ماکزیمم- مینیمم (MMAS) با اعمال یک اصلاحیه‌ی ساده بر روی الگوریتم سیستم مورچه‌ی استاندارد ایجاد می‌شود [دوریگو و همکاران، ۲۰۰٦]، [استاتزل[1] و هوس[2]، ۲۰۰۰]. این سیستم دارای دو ویژگی عمده است. اول آنکه، فرومون در هر نسل فقط توسط بهترین مورچه افزایش می‌یابد. این کار باعث کاهش کاوش و افزایش بهره‌وری از بهترین راه‌حل موجود می‌گردد. دوم آنکه، مقدار فرومون دارای حد بالا و پایین است. این کار اثری معکوس نسبت به ویژگی اول دارد چرا که با این کار حتی بدترین مسیرها نیز دارای مقداری فرومون هستند. از سوی دیگر، بهترین مسیرها نیز نمی‌توانند آنقدر فرومون داشته باشند که به کلی تصمیمات مورچه‌ها را تحت‌الشعاع خود قرار دهند.

[1] Stutzle
[2] Hoos

اولین تفاوت موجود میان الگوریتم مورچه‌ی استاندارد و MMAS را می‌توان در معادلات زیر، که در شکل‌های ۱۰-۶ و ۱۰-۱۲ جایگذاری شده‌اند، مشاهده نمود:

$$\text{استاندارد } AS: \quad \tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{k=1}^{N} \Delta\tau_{ij}^{(k)} \qquad (۱۰-۵)$$

$$MMAS: \quad \tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \Delta\tau_{ij}^{best}$$

که در آن best، اندیس بهترین راه‌حل نامزد است. در مسئله‌ی فروشنده‌ی دوره‌گرد شکل ۱۰-۶، $\Delta\tau_{ij}^{(best)}$ توسط معادله‌ی زیر تعیین می‌گردد

$$\Delta\tau_{ij}^{(best)} \leftarrow \begin{cases} \frac{Q}{L_{best}} & \text{اگر } i \text{ شهر} \rightarrow j \text{ به بهترین مسیر تعلق دارد} \\ 0 & \text{در غیر این صورت} \end{cases} \qquad (۱۰-۶)$$

و در AS پیوسته‌ی موجود در شکل ۱۰-۱۲، $\Delta\tau_{ij}^{(best)}$ توسط معادله‌ی زیر تعیین می‌گردد

$$\Delta\tau_{ij}^{(best)} \leftarrow \begin{cases} \frac{Q}{L_{best}} & \text{اگر } i \in [b_{ij}, b_{i,j+1}] \text{ اُمین بعد بهترین ذره} \\ 0 & \text{در غیر این صورت} \end{cases} \qquad (۱۰-۷)$$

دومین تفاوت میان الگوریتم استاندارد مورچه و MMAS بعد از به‌روزرسانی شدن τ_{ij} و توسط معادلات زیر پیاده‌سازی می‌شود

$$\begin{aligned} \tau_{ij} &\leftarrow \max(\tau_{ij}, \tau_{min}) \\ \tau_{ij} &\leftarrow \min(\tau_{ij}, \tau_{max}) \end{aligned} \qquad (۱۰-۸)$$

که در آن τ_{min} و τ_{max} برای مسئله‌ی مورد بهینه‌سازی تنظیم می‌گردند.

با کمی ابتکار می‌توان چندین راه را برای تعمیم MMAS متصور شد. برای مثال، می‌توان اجازه‌ی تولید فرومون را به جای تنها بهترین مورچه، برای M مورچه‌ی بهتر متصور شد. در این صورت M یک پارامتر میزان‌سازی خواهد بود. یا آنکه می‌توان به mاُمین مورچه‌ی برتر اجازه تولید فرومون با احتمال p_m را داد، p_m با افزایش هزینه‌ی مربوطه کاهش می‌یابد. با فرض آنکه ما در ابتدای فرایند بیشتر به دنبال کاوش بوده و در انتهای فرایند بیشتر به دنبال انتفاع و بهره‌وری هستیم، می‌توانیم مقدار $(\tau_{max} - \tau_{min})$ را با افزایش شماره‌ی نسل افزایش دهیم. با کمی تجربه و همچنین ابتکار، می‌توان تعمیم‌های دیگری از MMAS جهت بهبود عملکرد آن بر روی مسائل مختلف یافت.

مثال ۱۰–۳

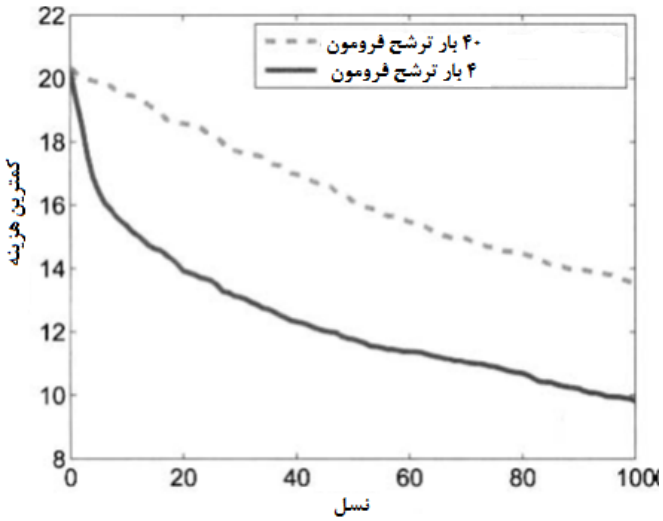
در این مثال ما مسئله‌ی مثال ۱۰–۲ را تکرار کرده و این بار از پارامترهای زیر استفاده می‌نماییم:

- $N = 40$
- $\alpha = 1$
- $Q = 20$
- $\rho = 0.9$
- $\tau_0 = 10^{-6}$
- دو راه‌حل نخبه در هر نسل
- نرخ جهشی برابر با ۱٪ در هر بعد در هر ذره در هر تسل
- تعداد بازه‌ها: $B_i = 20$ برای هر $i \in [1, n]$
- $\tau_{min} = 0$ و $\tau_{max} = \infty$

ما در این مثال تنها به بهترین M مورچه اجازه‌ی تولید فرومون می‌دهیم:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \Delta\tau_{ij}^{(best_m)} \qquad (۹–۱۰)$$

در معادله‌ی بالا $m \in [1, M]$ بوده و $best_m$ اندیس mاُمین ذره‌ی برتر در هر نسل است. این بدان معناست که تنها M مورچه‌ی برتر در دامنه‌ی مورد جستجویشان فرومون ترشح خواهند کرد. به غیر از این تغییر، الگوریتم مورد استفاده در این مثال همان الگوریتم مثال ۱۰–۲ است. شکل ۱۰–۱٤ بهترین راه‌حل هر نسل را برای $M = 4$ و $M = 40$، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، را نشان می‌دهد. می‌توان دید که هرچه به تعداد مورچه‌های کمتری اجازه‌ی تولید فرومون داده شود، همگرایی سریعتر صورت می‌پذیرد.



شکل ۱۰-۱٤ مثال ۱۰-۳: همگرایی سیستم مورچه‌ی پیوسته که بر روی تابع ۲۰ بعدی آکلی اعمال شده است. نمودار بهترین راه‌حل یافت شده در هر نسل را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین گیری شده است نشان می‌دهد. اگر تنها به بهترین مورچه‌ها اجازه‌ی تولید فرومون داده شود عملکرد بهتری حاصل می‌گردد.

۱۰-۴-۲ سیستم کلونی مورچگان

سیستم کلونی مورچگان (ACS[1]) تعمیمی است از AS [دوریگو و گامباردلا[2]، ۱۹۹۷a]، [دوریگو و گامباردلا، ۱۹۹۷b]، [دوریگو و همکاران، ۲۰۰٦]. اگرچه AS و ACS دارای ریشه‌ی مشترک هستند، اما دارای رفتار و عملکرد بسیار متفاوتی نسبت به هم می‌باشند. ACS نسبت به AS دارای دو تعمیم عمده است. اول آنکه هر مورچه برای ایجاد راه‌حل از یک تجدید فرومون محلی استفاده می‌نماید. به محض آنکه مورچه از شهر i به شهر j برود، فرومون موجود بر روی آن مسیر به طریق زیر تجدید می‌گردد:

$$\tau_{ij} \leftarrow (1-\phi)\tau_{ij} + \phi\tau_0 \qquad (۱۰-۱۰)$$

که در آن $\phi \in [0,1]$، ثابت زوال فرومون محلی بوده و τ_0 مقدار فرومون اولیه است. اگر $\phi = 0$ باشد، آنگاه τ_{ij} تغییر نکرده و همان سیستم مورچه‌ی اولیه حاصل می‌شود. معادله‌ی (۱۰-۱۰) نشان می‌دهد که فرومون موجود میان دو شهر i و j با عبور مورچه‌ها از آن مسیر زوال می‌یابد. این موضوع به لحاظ بیولوژیکی درست نیست اما در این الگوریتم، این موضوع باعث دلسرد شدن مورچه‌ها از مسیر شده و بدین ترتیب باعث

[1] Ant Colony System
[2] Gambardela

افزایش کاوش و تنوع می‌شود. پس از آنکه تمامی مورچه‌ها راه‌حل نامزد خود را ایجاد نمودند، یکی از تجدید فرومون‌های جهانی استاندارد موجود در معادله‌ی (۱۰–۵) را پیاده‌سازی می‌نماییم. تعمیمِ دوم ACS نسبت به AS، استفاده از قانون نسبی شبه اتفاقی برای ساختن راه‌حل‌های نامزد است. در اینجا ما از نماد $(a_k \to j)$ برای نشان دادن ساخته شدن راه‌حل نامزد توسط مورچه‌ی kاُم در حین سفر به شهر jاُم استفاده می‌نماییم. همچنین از نماد $\Pr(a_k \to j)$ برای نشان دادن احتمال $(a_k \to j)$ استفاده می‌نماییم. تفاوت میان ساخت راه‌حل‌های نامزد در ACS و AS را می‌توان به‌صورت زیر نشان داد:

$$AS: \quad \Pr(a_k \to j) = p_{ij}^{(k)}$$

$$ACS: \quad \Pr(a_k \to j) = \begin{cases} 1 & \text{اگر } j = argmax_J p_{ij}^{(k)} \text{ و اگر } r < q_0 \\ 0 & \text{در غیر این صورت و اگر } r \geq q_0 \\ p_{ij}^{(k)} & \text{اگر } r \geq q_0 \end{cases} \qquad (۱۱–۱۰)$$

که در آن r یک عدد اتفاقی با توزیع یکنواخت بر روی $[0,1]$ بوده و $q_0 \in [0,1]$ یک پارامتر میزان‌سازی است. در AS استاندارد، احتمالات با توجه به مقادیر فرومون به دست می‌آیند و مورچه‌ی kاُم با توجه به مقادیر احتمالات در مورد شهر مقصد تصمیم‌گیری می‌نماید (شکل ۱۰–۶ و ۱۰–۱۲ را ببینید). با این حال، در ACS مورچه‌ی kاُم ممکن است با احتمال q_0 به شهر با بیشترین مقدار احتمال برود (شهر با بیشترین احتمال شهری است که دارای بیشترین مقدار فرومون در مسیر میان خود و شهر حال حاضر مورچه است)، همچنین مورچه‌ی kاُم ممکن است با احتمال $(1 - q_0)$ از AS استاندارد برای تصمیم‌گیری در مورد شهر مقصد استفاده نماید. این موضوع باعث می‌شود مورچه‌ها در هنگام ساخت راه‌حل‌های نامزد خود به سمتِ کاوش گزینه‌های نوید دهنده‌تر گزایش داده شوند. این موضوع به‌صورت مفهومی با افزایش احتمال مسیرهای دارای فرومون بیشتر و یا افزایش α در شکل‌های ۱۰–۶ و ۱۰–۱۲ معادل است.

احتمالات ACS در معادله‌ی (۱۰–۱۱) هنگامی که $r \geq q_0$ است، تنها به‌صورت تقریبی درست هستند. برای دقت بهتر می‌توان این مقادیر را به‌صورتی نرمالیزه نمود که جمعشان برابر با ۱ شود (مسئله‌ی ۱۰–۷ را ببینید).

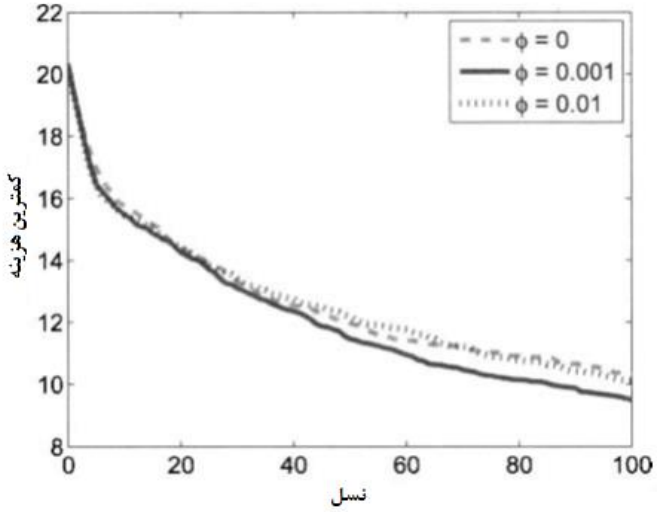
مثال ۱۰–٤

در این مثال استفاده از ثابت زوال فرومون محلی ϕ را مورد بررسی قرار خواهیم داد. در اینجا نیز از تابع ۲۰ بعدی آکلی استفاده می‌نماییم. پارامترهای مورد استفاده در این مثال به شرح زیر می‌باشند:

- $N = 40$

- $\alpha = 1$
- $Q = 20$
- $\rho = 0.9$
- $\tau_0 = 10^{-6}$
- دو راه‌حل نخبه در هر نسل
- نرخ جهشی برابر با ۱٪ در هر بعد در هر ذره در هر تسل
- تعداد بازه‌ها: $B_i = 20$ برای هر $i \in [1, n]$
- $\tau_{min} = 0$ و $\tau_{max} = \infty$
- ثابت کاوش $q_0 = 0$

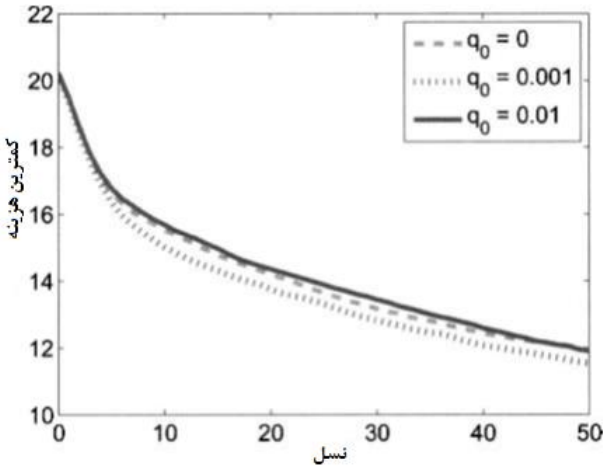
شکل ۱۰–۱۵ بهترین راه‌حل هر نسل را برای 0.01 و $\phi = 0.001$، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است نشان می‌دهد. می‌توان دید که عملکرد الگوریتم برای مقادیر غیرصفر ثابت زوال فرومون محلی، بسیار بهتر است. یک مقدار مثبت برای ϕ باعث افزایش کاوش شده و به همگرایی سریعتر منجر می‌شود. با این حال، اگر ϕ بیش از حد بزرگ باشد، سایر مورچه‌ها به شدت از کاوش مسیرهای قبلا استفاده شده دلسرد شده و به همین دلیل عملکرد الگوریتم بدتر می‌شود. برای به دست آوردن نتایج قاطع‌تر باید نتایج به دست آمده از شکل ۱۰–۱۵ را مورد ارزیابی آماری قرار داد (ضمیمه‌های ب.٤–۲ و ب.۵–۲ را ببینید).



شکل ۱۰–۱۵ مثال ۱۰–٤: عملکرد سیستم کلونی مورچگان بر روی تابع ۲۰ بعدی اَکلی. منحنی‌ها بهترین راه‌حل یافت شده در هر نسل را برای مقادیر مختلف ثابت زوال فرومون محلی، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است را نشان می‌دهند.

مثال ۱۰–۵

در این مثال به بررسی استفاده از ضریب کاوش q_0 در ACS می‌پردازیم. در این مثال نیز مانند مثال‌های قبلی تابع ۲۰ بعدی اَکلی را مینیمم خواهیم ساخت. در این مثال نیز از همان پارامترهای مثال ۱۰–٤ استفاده می‌نماییم تنها با این تفاوت که در اینجا ثابت زوال فرومون محلی $\phi = 0$ قرار داده و مقادیر مختلف q_0 را آزمایش می‌نماییم. شکل ۱۰–٦ بهترین راه‌حل به دست آمده در هر نسل را به ازای 0.01 و $q_0 = 0.001$، که بر روی ۱۰۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. می‌توان دید که به ازای مقادیر غیرصفر از ثابت کاوش عملکرد الگوریتم کمی بهبود می‌یابد. قرار دادن یک مقدار مثبت برای q_0 باعث می‌شود مورچه‌ها تمایل بیشتری به استفاده از ویژگی‌های راه‌حل مطلوب داشته باشند. با این حال، اگر مقدار q_0 بیش از حد بزرگ باشد آنگاه ACS به اندازه‌ی کافی از کاوش برخوردار نبوده و کارایی الگوریتم کاهش می‌یابد. برای به دست آوردن نتایج قاطع‌تر باید نتایج شکل ۱۰–٦ را از لحاظ آماری مورد آزمون قرار داد (ضمائم ب.٤–۲ و ب.۵–۲ را ببینید). همچنین توجه داشته باشید که این نتایج به شدت به مسئله‌ی مورد حل و همچنین به پارامترهای مورد استفاده برای الگوریتم بستگی دارند. بیشتر ACSها از مقادیر بزرگ برای q_0 استفاده می‌نمایند (برای مثال $q_0 = 0.9$) [دوریگو و گامباردِلا، ۱۹۹۷b].



شکل ۱۰–۱٦ مثال ۱۰–۵: عملکرد سیستم کلونی مورچگان بر روی تابع ۲۰ بعدی اَکلی. منحنی‌های موجود در این شکل بهترین راه‌حل یافت شده در هر نسل را که بر روی ۱۰۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده‌اند را برای مقادیر مختلف ثابت کاوش نشان می‌دهند. به ازای مقادیر مثبت، کارایی الگوریتم افزایش می‌یابد، اما مقادیرِ بیش از حد بزرگ q_0 می‌تواند باعث کاهش کارایی الگوریتم شود.

۱۰-۴-۳ دیگر سیستم‌های مورچه

به دلیل محدودیت موجود برای این کتاب نمی‌توانیم به تمام جزئیات مربوط به سایر سیستم‌های مورچه بپردازیم. با این حال، چند نوع قابل توجه از سیستم‌های مورچه وجود دارد که در اینجا به‌طور خلاصه به آن‌ها اشاره می‌کنیم. در سیستم مورچه‌ی نخبه‌گرا، بهترین راه‌حل موجود هر بار که سایر مورچه‌ها فرومون تولید کنند، به تولید فرومون می‌پردازد [دوریگو و استاتزل، ۲۰۰٤، فصل ۳]. بنابراین، محاسبات $\Delta\tau$ موجود در شکل ۱۰-۱۲ به‌صورت زیر اصلاح می‌شود:

$$\text{استاندارد } AS\text{: } \Delta\tau_{ij}^{(k)} \leftarrow \delta_{ij}^{(k)} Q/L_k$$
$$\text{نخبه‌گرا } AS\text{: } \Delta\tau_{ij}^{(k)} \leftarrow \frac{\delta_{ij}^{(k)} Q}{L_k} + \frac{\delta_{ij}^{(best)} Q}{L_{best}} \qquad (۱۰-۱۲)$$

که در آن اگر iاُمین بعد از kاُمین راه‌حل در jاُمین بازه‌ی گسسته‌سازی شده قرار بگیرد، آنگاه $\delta_{ij}^{(k)} = 1$ بوده و $best$ اندیس بهترین ذره‌ی موجود در جمعیت است.

Ant-Q ترکیبی است از AS و Q-learning [گامباردِلا و دوریگو، ۱۹۹۵]. در AS رتبه- محور، مقدار فرومون تولید شده توسط هر مورچه نه تنها به کیفیت راه‌حل مورچه بستگی دارد، بلکه به رتبه‌ی آن نسبت به سایر مورچه‌ها نیز وابسته است [دوریگو و استاتزِل، ۲۰۰٤، فصل ۳]. درخت جستجوی غیرقاطع تخمینی (ANTS[1]) مکانیزم‌های مشخصی برای تعیین میزان جذابیت یک حرکت و به روزرسانی فرومون فراهم می‌آورد [مانیزو[2] و همکرانریال ۲۰۰٤]. AS بهترین- بدترین به بهترین راه‌حل فرومون بیشتر اضافه کرده و همچنین میزان تبخیر بیشتری را به بدترین راه‌حل اعمال می‌نماید. این AS همچنین از جهش برای افزایش کاوش استفاده می‌نماید [کوردون[3] و همکاران، ۲۰۰۰]. الگوریتم ACO ابرمکعب از محدودسازی مقدار فرومون به بازه‌ی $[0,1]$ برای مرتب و منظم‌سازی رفتار ACO در مورد مسائل با هدف‌های مختلف استفاده می‌نماید [بلوم و دوریگو، ۲۰۰٤]. الگوریتم ACO جمعیت- محور به جای نگه‌داری تمام اطلاعات در یک نقشه‌ی فرومون، جمعی از پیشینه‌ی فرومون را نگه می‌دارد و از این پیشینه‌ها برای اصلاح الگوریتمِ به‌روزرسانی استفاده می‌نماید [گانسچ[4] و میدِندورف[5]، ۲۰۰۲]. الگوریتم ACO پرتو[6] نیز ترکیبی است از ACO و جستجوی پرتو (جستجوی پرتو یک الگوریتم جستجوی محبوب است) [بلوم، ۲۰۰۵b].

[1] Approximated non-deterministic Tree Search
[2] Maniezzo
[3] Cordon
[4] Gunstch
[5] Middendorf
[6] Beam ACO

۱۰–۵ نتایج نظری

از همان زمانی که نتایج تجربی حاکی از کار کردن ACO بودند، محقیقن بر آن شدند تا نظریه ACO را بسط داده و توضیح دهند ACO کِی، چرا و چگونه عمل می‌نماید. اولین اثبات همگرایی ACO در [گوجار[1]، ۲۰۰۰] ارائه شده است. پس از آن اثبات‌های همگرایی مختلفی برای انواع مختلف الگوریتم‌های ACO ارائه شده است [دوریگو و استاتزل، ۲۰۰٤]. بسیاری از این اثبات‌ها به جمله‌ای مانند این ختم می‌شوند "با گذشت مدت زمان کافی، ACO در نهایت بهترین راه‌حل را برای یک مسئله‌ی بهینه‌سازی ترکیبی خواهد یافت". نتایج همگرایی مانند این از لحاظ ریاضی جذاب‌اند اما به لحاظ عملی چندان جذاب نخواهند بود. تا زمانی که میزان فرومون بر روی هر شاخه و مسیر در یک محدوده با حد بالا و پایین قرار گیرد (مانند آنچه که در MMAS رخ می‌دهد)، همواره یک احتمال غیرصفر برای هر مورچه جهت کاوش هر شاخه و مسیر از فضای جستجو وجود دارد. بنابراین با گذشت زمان کافی، همه‌ی شاخه‌ها و مسیرها مورد جستجو قرار خواهند گرفت. این بدین معناست که راه‌حل بهینه در نهایت پیدا خواهد شد. پس کاملا واضح است که هر الگوریتم جستجوی اتفاقی که دارای یک احتمال غیرصفر برای جستجوی هر یک از راه‌حل‌های نامزد باشد، در نهایت همگرا خواهد شد. حتی ساده‌ترین الگوریتم جستجوی اتفاقی نیز در نهایت همگرا خواهد شد [باک، ۱۹۹٦].

دیگر نتایج نظری جذاب مانند احتمال همگرایی در یک بازه‌ی زمانی مشخص، مقیاس‌پذیری با ابعاد مسئله و مدل‌های ریاضی توصیفی مانند مدل‌های مارکوف یا مدل‌های سیستم پویا می‌باشند (فصل ٤ را برای مدل‌های ریاضی GA ببینید) [گوجار، ۲۰۰۸]، [نیومن و ویت، ۲۰۰۹]. توجه داشته باشید که نتایج نظری برای مسائل ترکیبی بسیار متفاوت از نتایج نظری برای مسائل با دامنه‌ی پیوسته می‌باشد. همچنین، اگر بتوان نشان داد که ACO معادل سایر الگوریتم‌های بهینه‌سازی با اثبات‌های همگرایی جذاب‌تر می‌باشد، آنگاه می‌توان آن اثبات‌های همگرایی را برای ACO اقتباس کرد و بدین ترتیب پایه‌های نظری ACO را تقویت نمود.

پیش از این نشان داده‌ایم که تحت شرایط مشخصی، ACO مشابه الگوریتم‌های بهینه‌سازی شیب صعودی اتفاقی (SGA[2]) و آنتروپی متقاطع (CE[3]) می‌باشد [میولیو۴۰۲[4] و دوریگو، ۲۰۰۲]، [ازلوچین[5] و همکاران، ۲۰۰٤]، [دوریگو و استاتزل، ۲۰۰٤]. SGA و CE الگوریتم‌های بهینه‌سازی مدل- محوری هستند که راه‌حل‌ها

[1] Gutjahr
[2] Stochastic Gradient Ascent
[3] Cross Entropy
[4] Meuleau
[5] Zlochin

را بر پایه‌ی توزیعِ احتمالِ پارامتری بر روی فضای جستجو، ایجاد می‌کنند. از ارزیابی راه‌حل‌های نامزد برای اصلاح توزیع احتمال جهتِ گرایش دادن آن به سمت راه‌حل‌های نامزد بهتر استفاده می‌شود.

۱۰-۶ نتیجه‌گیری

بسیاری از محققین بر این باورند که ACO یک الگوریتم نبوده، بلکه به دلیل تنوع بسیار زیاد آن، یک فرا ابتکار محسوب می‌شود. با این حال، می‌توان گفت که هر یک از الگوریتم‌های مورد بحث در این کتاب (GAها، EP، ES، GP، و غیره)، همگی دارای تنوعات بسیار زیادی بوده و بنابراین تمامی آن‌ها فرا ابتکار محسوب می‌شوند. تفاوت میان یک الگوریتم و یک فراالگوریتم تنها یک جنبه است، بنابراین تفاوت مانند تفاوت میان سیاه و سفید نیست. بسیاری از محققین فعال در زمینه‌ی ACO بر این باورند که ACO یک الگوریتم تکاملی نیست چرا که در این الگوریتم، ذرات به تبادل اطلاعات با یکدیگر نمی‌پردازند؛ آنچه که در الگوریتم‌های تکاملی سنتی رواج دارد.

بسیاری از مباحث موجود در این فصل بر روی فرومون‌های مسیریابی متمرکز بود. با این حال، مورچه‌ها از دیگر انواع فرومون برای مقاصدی غیر از علامت‌گذاری مسیرها نیز استفاده می‌نمایند. یک کلونی مورچگان معمولی از ۲۰ نوع فرومون مختلف استفاده می‌نماید [هولدابلر و ویلسون، ۱۹۹۰، فصل ۷]. برای مثال، مورچه‌ها در هنگام مواجه با یک مهاجم از خود فرومون زنگ خطر تولید می‌نمایند، که این موضوع باعث می‌شود سایر مورچه‌ها برانگیخته شده و در برابر مهاجم ایستادگی کنند [سوبوتنیک[1] و همکاران، ۲۰۰۸]. انواع مختلف فرومون را می‌توان با شبیه‌سازی در ACO جای داد تا یک راه‌حل ضعیف اطلاعات خود را به سایرین مخابره کند و بدین ترتیب سایر ذرات از تکرار آن استراتژی ضعیف بر حذر داشته شوند. این مانند تقویت منفی PSO[2] است که در بخش ۱۱-۶ مورد بحث واقع خواهد شد.

مورچه‌های ماده هنگام تخم‌گذاری از خود نوعی فرومون نمایشی تولید می‌نمایند تا از این طریق به سایر مورچه‌های ماده از گونه‌ی مشابه اعلام دارند تا در مکان دیگری تخم‌گذاری نمایند [گومز[3] و همکاران، ۲۰۰۵]. به‌طور کلی هر گونه از حیوانات از نوعی فرومون قلمرویی برای مشخص نمودن قلمرو خود استفاده می‌نمایند [هورن[4] و جایگر[5]، ۱۹۸۸]. برای مثال، فرومون قلمرویی در ادرار سگ و گربه وجود دارد و آن‌ها از ادرار خود برای مشخص کردن ناحیه‌ی قلمرو خود استفاده می‌نمایند. همچنین حیوانات از فرومون جنسی

[1] Sobotnik
[2] Negative reinforcement PSO
[3] Gomez
[4] Horne
[5] Jaeger

برای نشان دادن توانایی تولید مثل خود استفاده می‌نمایند [ویات[1]، ۲۰۰۳]. مورچه‌ها همچنین برای جمع کردن سایر مورچه‌ها در محلی که نیاز به نیروی کار است، از نوعی فرومون خاص استفاده می‌نمایند [هولدابلر و ویلسون، ۱۹۹۰]. این نوع فرومون‌ها را می‌توان در ACO جهتِ جلوگیری از کاوشِ مناطق قبلاً کاوش شده توسط سایر ذرات، و یا جهتِ تشویق ذرات به کاوش مناطقِ نویدبخشِ فضای جستجو، شبیه‌سازی نمود. برای انجام این شبیه‌سازی‌ها می‌توان ذرات را مجبور به تبادل اطلاعات خود با سایر ذره‌ها نمود. مورچه‌ها همچنین می‌توانند فرومون‌هایی از خود تولید کنند که مرتبط با وظیفه‌ی خاصی هستند [گرین و گوردون، ۲۰۰۷]. می‌توان از شبیه‌سازی این ویژگی در ACO جهت حل مسائل چند- هدفه استفاده نمود. می‌توان دید که گزینه‌های زیادی برای تعمیمِ زیستی ACO وجود دارد.

برای مطالعه‌ی بیشتر در زمینه‌ی ACO می‌توانید به کتاب‌های [بونابیو[2] و همکاران، ۱۹۹۹]، [دوریگو و استاتزل، ۲۰۰٤]، [سولنُن[3]ف ۲۰۱۰] و فصل‌هایی از کتاب‌های [مانیزو و همکاران، ۲۰۰٤]، [دوریگو و استاتزل، ۲۰۱۰] و همچنین مقاله‌های آموزشی [بلوم، ۲۰۰۵a] و [بلوم، ۲۰۰۷] مراجعه نمایید. جهت‌گیری‌های آتی برای تحقیقات در زمینه‌ی ACO مانند همان اولویت‌های موجود برای سایر الگوریتم‌های بهینه‌سازی می‌باشد [دوریگو و همکاران، ۲۰۰٦]. در اینجا چند سؤال باقی می‌ماند: چگونه می‌توان ACO را به مسائل بهینه‌سازی پویا، که در آن‌ها فضای جستجو با زمان تغییر می‌نماید، اعمال نمود و همچنین چگونه می‌توان ACO را به مسائل بهینه‌سازی اتفاقی با ارزیابی توابع برازندگی نویزی اعمال نمود (فصل ۲۲۱ را ببینید)؟ چگونه می‌توان ACO را به مسائل بهینه‌سازی چند- هدفه اعمال نمود (فصل ۲۰ را ببینید)؟ چگونه می‌توان ACO را با سایر الگوریتم‌های بهینه‌سازی ترکیب نمود؟

مسائل نوشتاری

۱۰-۱ یک مثال از دنیای واقعی بیاورید که در آن هزینه‌ی سفر از گره‌ی A به گره‌ی B با هزینه‌ی سفر از گره‌ی B به گره‌ی A برابر نباشد.

۱۰-۲ فرض کنید t تعداد کل مورچه‌هاست به‌طوری که در معادله‌ی (۱۰-۱) $m_1 \approx p_1 t$ و $m_2 \approx p_2 t$.

الف) نسبت تعادل p_1/p_2 چه قدر است؟

ب) چه مقادیری از نسبت تعادل پایدار بوده و چه مقادیری از آن ناپایدارند؟

[1] Wyatt
[2] Bonabeau
[3] Solnon

۳–۱۰ در سیستم مورچه‌ی شکل ۱۰–۶ فرض کنید $\beta = 1$. اگر دو مسیر ۱ و ۲ دارای مقدار مساوی فرومون بوده و مسیر ۱ نصف مسیر ۲ باشد، احتمال آنکه یک مورچه مسیر ۱ را انتخاب نماید چه قدر بیشتر از احتمال انتخاب شدن مسیر ۲ است؟ در مورد $\beta = 2$ چه طور؟ در مورد $\beta = 3$ چه طور؟

۴–۱۰ در سیستم مورچه‌ی شکل ۱۰–۶، فرومون تولید شده توسط kاُمین مورچه برابر با $\Delta\tau_{ij}^{(k)} = \delta_{ij}^{(k)} Q/L_k$ می‌باشد که در آن اگر kاُمین مورچه از شهر i به j رفته باشد، آنگاه $\delta_{ij}^{(k)} = 1$ بوده و در غیر این صورت $\delta_{ij}^{(k)} = 0$ می‌باشد. در این شکل ϵ یک پارامتر میزان‌سازی است.

الف) چه مقادیری از ϵ معادله‌ی به‌روزرسانی فرومونِ پایداری را نتیجه می‌دهد؟

ب) در این مورد، مقدار تعادل τ_{ij} چه قدر است؟ آیا این مقدار یک مقدار مطلوب است؟

۵–۱۰ در AS استاندارد با دامنه‌ی پیوسته از شکل ۱۰–۱۲، مقدار ترشح فرومون mاُمین مورچه برابرست با $\Delta\tau_{ij}^{(m)} = Q/L_m$. حال، همان‌طور که در بخش ۱۰–۴–۱ بیان شد، فرض کنید ما به mاُمین مورچه با احتمال p_m اجازه‌ی تولید فرومون می‌دهیم. p_m با افزایش هزینه کاهش می‌یابد:

$$p_m \leftarrow \frac{1}{L_m}\sum_{\tau=1}^{N} L_r$$
$$r \leftarrow U[0,1]$$
اگر $r < p_m$ آنگاه
$$\Delta\tau_{ij}^{(m)} \leftarrow Q_1/L_m$$
در غیر این صورت
$$\Delta\tau_{ij}^{(m)} \leftarrow 0$$
پایان اگر

در الگوریتم بالا، از چه مقداری برای Q_1 باید استفاده نماییم تا مقدار میانگین فرومون تولید شده توسط mاُمین مورچه با مقدار فرومون ترشح شده در AS استاندارد از شکل ۱۰–۱۲ برابر باشد؟

۶–۱۰ تلاش محاسباتی موجود در سیستم مورچه‌ی دامنه پیوسته از شکل ۱۰–۱۲ با افزایش اندازه جمعیت، چگونه افزایش می‌یابد؟ با افزایش ابعاد مسئله چطور؟ با افزایش تعداد بازه‌های گسسته‌سازی شده در هر بعد چطور؟

۷–۱۰ احتمالات سیستم کلونی مورچگان:

الف) فرض کنید یک ACS با چهار شهر و $q_0 = \frac{1}{2}$ در اختیار داریم. فرض کنید kاُمین مورچه در شهر ۱ قرار دارد به‌طوری که

$$p_{11}^{(k)} = 0$$
$$p_{12}^{(k)} = 1/4$$
$$p_{13}^{(k)} = 1/4$$
$$p_{14}^{(k)} = 1/2$$

با توجه به معادله‌ی (۱۰–۱۱)، احتمال رفتن kاُمین مورچه به هر یک از چهار شهر چه قدر است؟ آیا مجموع این احتمالات برابر ۱ است؟

ب) مقادیر احتمالات ACS ($\Pr(a_k \to j)$) از معادله‌ی (۱۰–۱۱) را به‌گونه‌ای نرمالیزه نمایید که مجموع آن‌ها به ازای n تا $j = 1$ برابر ۱ شود.

ج) از جواب خود در بخش ب برای محاسبه‌ی احتمالات جدید در سناریوی مطرح شده در بخش الف استفاده نمایید. آیا مجموع احتمالات جدید برابر ۱ است؟

۱۰–۸ یک راه مانند آنچه در بخش ۱۰–٤–۳ ذکر شد برای پیاده‌سازی یک AS رتبه– محور ارائه دهید.

۱۰–۹ یک راه مانند آنچه در بخش ۱۰–٤–۳ ذکر شد برای پیاده‌سازی یک AS بهترین– بدترین ارائه دهید.

مسائل کامپیوتری

۱۰–۱۰ این مسئله به بررسی تأثیر β، که میزان حساسیت ابتکاری سیستم مورچه می‌باشد، اختصاص دارد. سیستم مورچه‌ی مثال ۱۰–۱ را برای $\beta = 0.1, 1$ و 10، ۲۰ بار شبیه‌سازی کرده و بهترین هزینه در هر نسل را ثبت نمایید. نمودار میانگینِ ۲۰ شبیه‌سازی مونت کارلو را بر حسبِ شماره‌ی نسل رسم کرده و نتایج به دست آمده را مورد بحث قرار دهید.

۱۰–۱۱ مثال ۱۰–۳ را با $M = 40$ تکرار نمایید. برای هر یک از مقادیرِ $\tau_{min} = 0.001, 0.01$ و 0.1، ۲۰ شبیه‌سازی مونت کارلو انجام دهید. نمودار نتایج به دست آمده را رسم نمایید. در مورد نحوه‌ی تأثیر τ_{min} بر عملکرد AS چه نظری دارید؟

۱۰–۱۲ مثال ۱۰–۳ را با $M = 40$ تکرار نمایید. برای هر یک از مقادیرِ $\tau_{max} = 1, 10, 100$ و ∞، ۲۰ شبیه‌سازی مونت کارلو انجام دهید. نمودار نتایج به دست آمده را رسم نمایید. در مورد نحوه‌ی تأثیر τ_{max} بر عملکرد AS چه نظری دارید؟

فصل یازدهم

بهینه‌سازی تجمع ذرات

الگوریتم تجمع ذرات از رفتار انسان تقلید می‌نماید.

جیمز کندی[1] و راسل اِبرهارت[2]

هوش جمعی را در بسیاری از سیستم‌های طبیعی می‌توان مشاهده نمود. برای مثال، مورچه‌ها دارای سطح خارق‌العاده‌ای از هوش جمعی می‌باشند. این موضوع را در ابتدای فصل ۱۰ مورد بحث قرار دادیم. در این گونه سیستم‌ها، هوش به یک ذره و فرد منحصر نیست، بلکه میان گروهی از ذرات توزیع شده است. این موضوع را می‌توان در رفتار حیوانات هنگام تشکیل یک گله برای مقابله با خطراتِ حیواناتِ مهاجم، یافتن غذا و دیگر رفتارهای آن‌ها مشاهده نمود.

حیوانات با تشکیل گروه و گله راحت‌تر می‌توانند از خطرات دیگر حیوانات مهاجم دوری نمایند. برای مثال، تشخیص یک گورخر تنها برای یک شیر کار بسیار ساده‌ای است چرا که رنگ پوست گورخر تضاد زیادی با محیط اطرافش دارد، اما تشخیص گروهی از گورخرها برای شیر کار ساده‌ای نیست به این دلیل که آن‌ها با هم مخلوط شده و تشخیص یک عدد از آن‌ها را برای شیر مشکل می‌سازد [استون[3]، ۲۰۰۹]. همچنین یک گروه از حیوانات، هیبت بزرگتری را تشکیل داده و صدای بلندتری تولید می‌نماید و به همین دلیل در نظر سایر حیوانات خطرناک‌تر از یک حیوان تنها به نظر می‌رسد. در نهایت آنکه، هنگامی که حیوانات در گله‌هایی با یکدیگر حرکت می‌کنند، تمرکز کردن بر روی تنها یکی از آن‌ها برای حیوانِ شکارچی دشوار می‌شود. این پدیده اثرِ سردرگمی حیوان درنده[4] نام دارد [میلینسکی[5] و هِلر[6]، ۱۹۷۸]. [هاینریچ[7]، ۲۰۰۲] توضیح بسیار جالبی از چگونگی استفاده‌ی غزال‌ها از اثر سردرگمی حیوان درنده ارائه می‌دهد.

یکی دیگر از روش‌هایی که گله‌های حیوانات از آن برای در امان نگه داشتن خود از حیوانات درنده استفاده می‌نمایند، با استفاده از فرضیه‌ی چشمان- بیشمار[8] توضیح داده می‌شود. هنگامی که یک گروه بزرگ از حیوانات به دنبال غذا و یا آب هستند، اثرات اتفاقی حکم می‌کنند که همواره تعداد کمی از آن‌ها مراقب حیوانات درنده باشند. این همکاری نه تنها به مراقبت بیشتر در برابر حیوانات درنده کمک می‌نماید، بلکه بدین طریق مدت زمانِ بیشتری برای هر یک از حیوانات جهت خوردن و آشامیدن فراهم می‌شود.

---

[1] James Kennedy
[2] Russel Eberhart
[3] Stone
[4] Predator Confusion Effect
[5] Millinski
[6] Heller
[7] Heinrich
[8] Many-eye Hypothesis

در نهایت آنکه، حیوانات به دلیل اثر رقت برخورد[1] از خود در برابر حیوانات درنده به‌صورت گروهی محافظت می‌نمایند [کراوس[2] و راکستون[3]، ۲۰۰۲]. این کار می‌تواند به چندین شکل صورت بپذیرد. اول آنکه یک حیوانِ تنها ممکن است تحت یک حرکت خودخواهانه، به دنبال استفاده از پوشش و محافظت یک گروه باشد [هَمیلتون[4]، ۱۹۷۱]. دوم آنکه، هنگامی که حیوانات به‌صورت گروهی در حرکت باشند، احتمال برخورد گروه با حیوان درنده بسیار کمتر از حالتی است که هر یک از اعضای گروه به‌صورت انفرادی در حال گشت و گذار باشد [تِرنِر[5] و پیچر[6]، ۱۹۸۶].

حیوانات همچنین در صورت قرار گرفتن در گله و گروه از شانس بیشتری برای پیدا کردن غذا و آب برخوردار هستند. در نگاه اول این موضوع درست به نظر نمی‌آید چرا که وقتی نفرات در یک گروه قرار دارند، نمی‌توانند به‌صورت مخفیانه به شکار خود نزدیک شوند و هنگامی هم که شکار خود را صید می‌کنند، باید آن را با سایر افراد گروه به اشتراک بگذارند. با این حال، میزان موفقیت گروه‌ها در پیدا کردن غذا به فرضیه‌ی چشمان- بیشمار مرتبط است. هنگامی که تعداد چشمان بیشتری به دنبال غذا باشد، گروه از شانس بیشتری نسبت به جستجوی انفرادی برای پیدا کردن غذا برخوردار خواهد بود [پیچر و پاریش[7]]. همچنین، گروه می‌تواند شانس موفقیت خود را با محاصره کردن طعمه افزایش دهد.

حیوانات همچنین با قرار گرفتن در گروه و گله می‌توانند سریعتر از حالت انفرادی حرکت نمایند. این موضوع را می‌توان در مورد دوچرخه‌سوارانی که در یک خط با هم حرکت می‌نمایند مشاهده نمود. دوچرخه سواران انتهایی نسبت به دوچرخه‌سوار ابتدای خط حدود ٤٠٪ کمتر انرژی مصرف می‌نماید، چرا که در ابتدای خط میزان مقاومت هوا بیشتر از انتهای خط می‌باشد [بِرک[8]، ۲۰۰۳]. چنین اثری را، ولو با شدت کمتر، می‌توان در مورد ورزش‌های دیگری همچون اسکیت، دو و میدانی و شنا مشاهده نمود. در دنیای حیوانات چنین تأثیری را می‌توان در پرواز غازها [مَکنَب[9]، ۲۰۰۲]، حرکت مرغابی‌ها بر روی آب [فیش[10]، ۱۹۹۵] و شنای ماهیان در کنار هم [نورن[11] و همکاران، ۲۰۰۸] مشاهده نمود.

---

[1] Encounter Dilution Effect
[2] Krause
[3] Ruxton
[4] Hamilton
[5] Turner
[6] Pitcher
[7] Parrish
[8] Burke
[9] McNab
[10] Fish
[11] Noren

بهینه‌سازی تجمع ذرات (PSO[1]) بر پایه‌ی مشاهدات گروه‌هایی از ذرات انفرادی هنگام تعامل با یکدیگر جهتِ بهبود عملکرد جمعی و فردی، قرار دارد. قواعد PSO را می‌توان به روشنی در رفتار حیوانات و حتی انسان مشاهده نمود. ما انسان‌ها هنگام تلاش برای بهبود عملکرد خود، از چندین ایده‌ی اساسی استفاده می نماییم.

- اینرسی[2]. ما معمولاً تمایل داریم از روش‌های قدیمی که کارایی و موفقیت آن‌ها قبلاً ثابت شده است استفاده نماییم. "این روشی است که من تا به حال این کار را انجام می‌داده‌ام و از این پس نیز همین روش را دنبال خواهم کرد"
- اثر اجتماعی. ما گاهاً در کتاب‌ها، روزنامه‌ها و یا اینترنت در مورد سایر افراد موفق چیزهایی می‌شنویم و سعی به تقلید از روش آن‌ها می‌نماییم. "اگر این روش برای آن‌ها کارایی داشته، پس شاید برای من نیز داشته باشد."
- اثر همسایگان. ما چیزهای بسیاری را از افراد نزدیک به خود می‌آموزیم. ما بیش از آنکه از جامعه تأثیر بپذیریم، از دوستان خود متأثر هستیم. در مکالمات خود با سایرین، اغلب از موفقیت‌ها و شکست‌های خود می‌گوییم و رفتار خود را بر اساس آن مکالمات اصلاح می‌نماییم. برای مثال، اگر نصیحت‌هایی در مورد سرمایه‌گذاری را از دوست و یا فامیلِ میلیونر خود بشنویم، تأثیر آن بسیار بیشتر از حالتی است که آن نصیحت را از طرف شخصی میلیونر که نمی‌شناسیم و تنها در اینترنت در موردِ آن می‌خوانیم، بشنویم.

مرور بر فصل

بخش ۱۱-۱ طرحی کلی و پایه‌ای را از PSO به همراه چند مثال ساده به دست می‌دهد. بخش ۱۱-۲ به بحث در مورد راه‌های مختلف برای محدود نمودن سرعت ذرات در PSO می‌پردازد، امری که تأثیر بسیاری بر روی عملکرد PSO دارد. بخش ۱۱-٤ دو ویژگی مهم از PSO را که نقش غیرمستقیم در محدود نمودن سرعت ذرات دارند را بررسی می‌نماید؛ وزن‌دهی اینرسی و ضرایب انقباض. بخش ۱۱-٤ در مورد الگوریتم PSO سراسری بحث خواهد نمود. این الگوریتم تعمیمی از PSO است که در آن از بهترین ذره‌ی هر نسل برای به‌هنگام‌سازی سرعت هر ذره استفاده می‌شود. بخش ۱۱-۵ الگوریتم PSO کاملاً آگاه را مورد بررسی قرار می‌دهد. در این الگوریتم سرعتِ هر ذره در تعیین سرعت هر ذره‌ی دیگر در هر نسل دخالت دارد. بخش

[1] Particle Swarm Optimization (PSO)
[2] Inertia

۱۱-٦ نگاهی متفاوت را نسبت به یادگیری PSO به دست می‌دهد؛ اگر می‌توان از موفقیت‌های دیگران آموخت، پس می‌توان از اشتباهات آنان نیز درس گرفت.

۱۱-۱ الگوریتم بنیادین بهینه‌سازی تجمع ذرات

فرض کنید یک مسئله‌ی بهینه‌سازی در اختیار داریم که بر روی دامنه‌ای d بعدی و پیوسته تعریف شده است. همچنین جمعیتی از N راه‌حل نامزد، که آن را با $\{x_i\}, i \in [1, N]$ نشان می‌دهیم، در اختیار داریم. به علاوه، فرض کنید هر ذره‌ی x_i با یک سرعت v_i در حال حرکت در فضای جستجو است. این حرکت میان فضای جستجو، جوهره‌ی PSO بوده و همین حرکت است که آن را از سایر الگوریتم‌های تکاملی متمایز می‌سازد. بیشترِ دیگر الگوریتم‌های تکاملی بسیار ایستاتر از PSO می‌باشند چرا که مدل‌سازی آن‌ها از راه‌حل‌های نامزد تنها به نشان دادن تکامل آن‌ها از یک نسل به نسل دیگر محدود شده و از نشان دادن پویایی و دینامیکِ حرکتِ راه‌حل‌های نامزد در میان فضای جستجو عاجزند.

یک ذره‌ی PSO هنگام حرکت در میان فضای جستجو دارای مقداری اینرسی است بنابراین تمایل دارد جهت و مقدار سرعت خود را حفظ نماید. با این حال، این سرعت به چندین دلیل می‌تواند دستخوش تغییر شود.

- اول آنکه، ذره‌ی مذکور بهترین مکانی که پیش از این در آن قرار داشته را به خاطر سپرده است بنابراین می‌خواهد سرعت خود را به نحوی تغییر دهد که بتواند به آن محل بازگردد. این موضوع مانند تمایل انسان به تکرار روزهای خوب گذشته است. در PSO، یک ذره در میان فضای جستجو حرکت کرده و از هر نسل به نسل مکان خود در فضای جستجو را تغییر می‌دهد. با این حال، ذره عملکرد خود در نسل‌های قبلی و همچنین محلی را که در آن بهترین عملکرد را داشته به خاطر می‌سپارد.
- دوم آنکه، یک ذره از بهترین مکانِ همسایه‌های خود در هر نسل خبر دارد. این یعنی باید یک اندازه‌ی همسایگی وجود داشته باشد و ذرات باید در مورد عملکرد خود بر روی مسئله‌ی بهینه‌سازی با یکدیگر به تبادل اطلاعات بپردازند.

این دو اثر به‌صورت اتفاقی بر روی سرعت یک ذره تأثیر می‌گذارند. این موضوع مانند کنش و واکنش‌های اجتماعی ما انسان‌ها می‌باشد. برخی اوقات ما بسیار یکدنده و لجوج هستیم و به همین دلیل تأثیر چندانی از همسایگان خود نمی‌پذیریم. گاهی نیز دچار احساسات نوستالژیک می‌شویم و به شدت از

موفقیت‌های گذشته‌مان تأثیر می‌پذیریم. با توجه به آنچه گفته شد، الگوریتم بنیادین PSO را می‌توان مانند آنچه که در شکل ۱۱-۱ نشان داده شده است خلاصه نمود.

جمعیت اتفاقی از ذرات را مقداردهی کن $\{x_i\}, i \in [1, N]$

بردار سرعت nتایی هر ذره را مقداردهی کن $v_i, i \in [1, N]$

بهترین مکان حال حاضر هر ذره را مقداردهی کن: $b_i \leftarrow x_i, i \in [1, N]$

اندازه‌ی همسایگی را تعریف کن: $\sigma < N$

مقادیر بیشینه تأثیر $\phi_{1,max}$ و $\phi_{2,max}$ را تعریف کن

مقدار بیشینه سرعت، v_{max} را تعریف کن

تا زمانی که شرایط توقف برآورده نشده است

برای هر ذره $x_i, i \in [1, N]$

$H_i \leftarrow \{$نزدیکترین σ همسایه به $x_i\}$

$h_i \leftarrow arg\min_x \{f(x): x \in H_i\}$

یک بردار اتفاقی ϕ_1 با $\phi_1(k) \sim U[0, \phi_{1,max}]$ برای $k \in [1, n]$ تولید کن

یک بردار اتفاقی ϕ_2 با $\phi_2(k) \sim U[0, \phi_{2,max}]$ برای $k \in [1, n]$ تولید کن

$v_i \leftarrow v_i + \phi_1 \text{o } (b_i - x_i) + \phi_2 \text{o } (h_i - x_i)$

اگر $|v_i| > v_{max}$ آنگاه

$v_i \leftarrow v_i v_{max} / |v_i|$

پایان اگر

$x_i \leftarrow x_i + v_i$

$b_i \leftarrow argmin\{f(x_i), f(b_i)\}$

ذره‌ی بعد

نسل بعد

شکل ۱۱-۱ یک الگوریتم بنیادین بهینه‌سازی تجمع ذرات برای مینیمم‌سازی تابع n بعدیِ $f(x)$ که در آن x_i، iاُمین راه‌حل نامزد بوده و v_i بردارِ سرعت آن می‌باشد. علامت $a \cdot b$ به معنای ضرب عنصر- به- عنصر میان بردارهای a و b می‌باشد.

با توجه به شکل ۱۱-۱ می‌توان دریافت که الگوریتم PSO دارای چند پارامتر میزان‌سازی است.

- در این الگوریتم، ما نه تنها باید جمعیت اولیه ذرات را مقداردهی اولیه نماییم، بلکه باید بردارِ سرعت ذرات را نیز مقداردهی اولیه نماییم. چندین راه برای مقداردهی اولیه سرعت‌ها وجود دارد. برای مثال، می‌توان سرعت‌های اولیه را برابر صفر قرار داد و یا از مقداردهی اولیه اتفاقی استفاده نمود.

- اندازه‌ی همسایگی σ یکی دیگر از پارامترهایی است که باید در این الگوریتم مشخص گردد. توجه داشته باشید که در اینجا عبارت "اندازه‌ی همسایگی" دوپهلو و دارای ابهام است. این عبارت گاهی بدین معنی است که هر ذره دارای σ همسایه‌ی نزدیک است و گاهی نیز بدین معنی است که از آنجا که σ ذره در هر همسایگی موجود است، پس هر ذره دارای $(\sigma - 1)$ همسایه‌ی نزدیک است. یکی از اولین مقاله‌های منتشر شده در زمینه‌ی PSO به این نکته اشاره دارد که هر چه اندازه‌ی همسایگی کوچکتر باشد، عملکرد کلی بهتر شده و از مینیمم محلی پرهیز می‌شود. این در صورتی است که هر چه اندازه‌ی همسایگی بزرگتر باشد، همگرایی سریعتر اتفاق می‌افتد [ابرهارت و کندی، ۱۹۹۵].
- یکی دیگر از پارامترهای میزان‌سازی نرخ‌های یادگیری $\phi_{1,max}$ و $\phi_{2,max}$ می‌باشد. پارامترهای ϕ_1 و ϕ_2 که به ترتیب نرخ یادگیری شناختی و نرخ یادگیری اجتماعی نام دارند، اعدادی اتفاقی با توزیع یکنواخت بر روی $[0, \phi_{1,max}]$ و $[0, \phi_{2,max}]$ می‌باشند. این پارامترها را در بخش ۱۱-۳-۳ بیشتر مورد بحث قرار خواهیم داد اما به خاطر داشته باشید که قاعده‌ی سرانگشتی برای $\phi_{1,max}$ و $\phi_{2,max}$ این است که هر دو را برابر ۲٫۰۵ قرار دهیم.
- دیگر پارامتر میزان‌سازی، سرعت بیشینه v_{max} است. شواهد تجربی نشان می‌دهند که هر عنصر از v_{max} باید به بردِ پویای متناظر از فضای جستجو محدود شود [ابرهارت و شی[1]، ۲۰۰۰]. می‌توان به‌صورت حسی و شهودی درک نمود که اگر v_{max} از بردِ پویای فضای جستجو بزرگتر باشد، آنگاه یک ذره می‌تواند به راحتی و در یک نسل فضای جستجو را ترک نماید. سایر نتایج قرار دادن v_{max} با ۱۰٪ تا ۲۰٪ بردِ فضای جستجو را پیشنهاد می‌نمایند [ابرهارت و شی، ۲۰۰۱]. گاهاً پیش می‌آید که در مسئله‌ای از بردِ فضای جستجو آگاه نباشیم، بدین معنی که از پیش، هیچ ایده‌ای در مورد محل بهینه در فضای جستجو نداریم. حتی در این شرایط نیز بهتر است از مقداری متناهی برای v_{max} استفاده نماییم [کارلیسل[2] و دوزیر[3]، ۲۰۰۱].
- می‌توان به‌هنگام‌سازی سرعت در شکل ۱۱-۱ را به‌صورت زیر ساده‌سازی نمود:

$$v_i \leftarrow v_i + \phi_1(b_i - x_i) + \phi_2(h_i - x_i) \qquad (۱-۱۱)$$

[1] Shi
[2] Carlisle
[3] Dozier

که در آن ϕ_1 و ϕ_2 دو پارامتر اسکالر بوده به‌صورتی که $\phi_1 \sim U[0, \phi_{1,max}]$ و $\phi_2 \sim U[0, \phi_{2,max}]$. در این حالت، که PSO خطی نام دارد [پاکِی[1] و انگلبرکت[2]، ۲۰۰۳]، هر عنصر از بردار سرعت v_i با مقادیر یکسانی از ϕ_1 و ϕ_2 به‌هنگام‌سازی می‌شود. با این حال، PSO خطی عملکرد بدتری نسبت به الگوریتم استانداردِ شکل ۱۱-۱ به دست می‌دهد.

- همانند سایر الگوریتم‌های تکاملی، نخبه‌گرایی عموماً باعث بهبود عملکرد PSO می‌گردد. نخبه‌گرایی در الگوریتم شکل ۱۱-۱ لحاظ نشده است، اما می‌توان به سادگی و همان‌گونه که در بخش ۸-٤ عنوان شد، آن را پیاده‌سازی نمود.
- معادله‌ی به‌هنگام‌سازی $x_i \leftarrow x_i + v_i$ در شکل ۱۱-۱ ممکن است باعث بیرون رفتن x_i از دامنه‌ی فضای جستجو شود. ما معمولاً از نوعی عملیات محدودسازی برای نگه داشتن x_i در داخل فضای جستجو استفاده می‌نماییم. برای نمونه، می‌توان از دو معادله‌ی زیر پس از معادله‌ی به‌هنگام‌سازی جهت عملیات محدودسازی استفاده نمود

$$x_i \leftarrow \min(x_i, x_{max})$$
$$x_i \leftarrow \max(x_i, x_{min}) \qquad (۲-۱۱)$$

که در آن $[x_{min}, x_{max}]$ حدود فضای جستجو را تعیین می‌نماید.

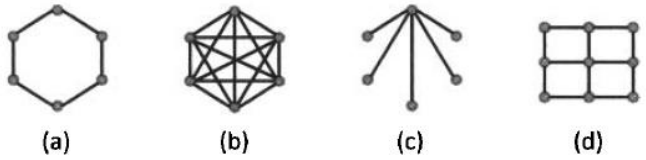
توپولوژی‌های تجمع ذرات

شکل ۱۱-۱ نشان می‌دهد که هر ذره تحت تأثیر σ ذره‌ی همسایه‌ی خود می‌باشد. ترتیب و نحوه‌ی قرار گرفتن این همسایه‌ها، توپولوژیِ تجمع نام دارد. از آنجایی که همسایگی هر ذره در هر نسل تغییر می‌نماید این توپولوژی، توپولوژیِ پویا نام دارد.

روش‌های متعددی برای تعیین همسایگی هر ذره وجود دارد [آکات[3] و گازی[4]، ۲۰۰۸]. برای مثال، می‌توان همسیگی‌ها را در ابتدای الگوریتم تعریف نمود به طوری که همسایگی‌ها ایستا بوده و از یک نسل به نسل دیگر تغییری ننمایند. یا اینکه می‌توان در صوت بروز رکود در فرایند بهینه‌سازی، همسایگی‌ها را به‌صورت اتفاقی بازتعریف نمود [کلرک[5] و پولی، ۲۰۰٦]. در حادترین شرایط، یک همسایگی کل جمعیت را فرا می‌گیرد. این بدین معنی است که H_i در شکل ۱۱-۱ برای تمامی iها برابر با کل جمعیت بوده و h_i از

[1] Paquet
[2] Engelbrecht
[3] Akat
[4] Gazi
[5] Clerc

i مستقل بوده و با بهترین ذره در میان جمعیت برابر می‌باشد. این توپولوژی، توپولوژیِ همه[1] و یا توپولوژی $gbest$ نام دارد. در ابتدا PSO بر اساس این توپولوژی ایجاد شد و امروزه نیز این توپولوژی پرکاربردترین توپولوژی می‌باشد. یک توپولوژی معمولِ دیگر، توپولوژی حلقه[2] نام دارد که در آن هر ذره با دو ذره‌ی دیگر همسایه است. توپولوژیِ خوشه‌ای[3] توپولوژی است که در آن هر ذره به تمام ذرات موجود در خوشه‌ی خود متصل است، در حالی که چند ذره از هر خوشه با چند ذره از خوشه‌های دیگر نیز متصل می‌باشند. در توپولوژی چرخ[4] یک ذره، ذره‌ی کانونی است و به تمام ذرات دیگر متصل است. این در حالی است که تمامیِ ذرات دیگر تنها به ذره‌ی کانونی متصل می‌باشند. توپولوژی مربعی[5] نیز نوعی دیگر از توپولوژی تجمع ذرات است که در آن هر ذره به چهار ذره‌ی همسایه متصل است. این توپولوژی با نامِ توپولوژی وُن نیومن[6] نیز شناخته می‌شود. شکل ۱۱-۲ برخی از این توپولوژی‌ها را به تصویر می‌کشد. توپولوژی می‌تواند تأثیر فراوانی بر عملکرد PSO داشته باشد. به همین دلیل محققین تاکنون توپولوژی‌های بسیاری را مورد آزمایش قرار داده‌اند که موارد ذکر شده تا به اینجا تنها چند نمونه از این توپولوژی‌ها می‌باشند [مِندِس و همکاران، ۲۰۰٤]، [دِل واله[7] و همکاران، ۲۰۰۸].



شکل ۱۱-۲ برخی توپولوژی‌های PSO. (a) توپولوژی حلقه. (b) توپولوژی همه. (c) توپولوژی چرخ. (d) توپولوژی مربع. توجه داشته باشید که در توپولوژی مربعی، ذرات واقع شده بر روی اضلاع مربع در واقع تشکیل یک شکل توروئیدی را می‌دهند. برای مثال ذرات واقع شده بر روی ضلع بالایی با ذرات متناظر خود در ضلع پایینی و همچنین ذرات واقع شده بر روی ضلع چپ با ذرات متناظر خود در ضلع راست در ارتباط هستند، هرچند که شکل این موضوع را نشان نمی‌دهد. بدین ترتیب هر ذره دارای ٤ همسایه است.

[1] All Topology
[2] Ring Topoogy
[3] Cluster Topology
[4] Wheel Topology
[5] Square Topology
[6] Von Neumann
[7] Del Valle

۱۱–۲ محدودسازی سرعت

در بسیاری از کاربردهای PSO این موضوع روشن شده است که در صورت استفاده از v_{max} ذرات به‌صورت شدیدی در اطراف فضای جستجو پرش می‌نمایند [اِبرهارت و کندی، ۱۹۹۵]. برای یافتن دلیل این موضوع، الگوریتم بنیادین PSO از شکل ۱۱–۱ را با $\phi_2 = 0$ (جهت ساده‌سازی) در نظر بگیرید. بدین ترتیب به‌هنگام‌سازی مکان و سرعت برابر خواهد بود با

$$\begin{aligned} v_i(t+1) &= v_i(t) + \phi_1(b_i + x_i) \\ x_i(t+1) &= x_i(t) + v_i(t+!) \end{aligned} \qquad (۳–۱۱)$$

که در آن t شماره‌ی نسل بوده و ϕ_1 نرخ یادگیری شناختی می‌باشد. به بیان دیگر:

$$\begin{bmatrix} x_i(t+1) \\ v_i(t+1) \end{bmatrix} = \begin{bmatrix} 1-\phi_1 & 1 \\ -\phi_1 & 1 \end{bmatrix} \begin{bmatrix} x_i(t) \\ v_i(t) \end{bmatrix} + \begin{bmatrix} \phi_1 \\ \phi_1 \end{bmatrix} b_i \qquad (٤–۱۱)$$

مقادیرِ ویژه‌ی ماتریسِ سمت راست در معادله‌ی بالا به شرح زیراند:

$$\lambda = \frac{2 - \phi_1 \pm \sqrt{\phi_1^2 - 4\phi_1}}{2} \qquad (۵–۱۱)$$

این مقادیر ویژه پایداری سیستم را کنترل می‌نمایند[۱]. اگر $\phi_1 \in [0,4]$ باشد، آنگاه اندازه‌ی این مقادیر ویژه برابر ۱ خواهند بود و این بدین معنی است که سیستم دارای پایداری حاشیه‌ای است و $x_i(t)$ و $v_i(t)$ با میل کردن t به سمت بی‌نهایت و بسته به شرایط اولیه، بیکران خواهند شد. اگر $\phi_1 > 4$ باشد آنگاه اندازه‌ی یکی از مقادیر ویژه بزرگتر از ۱ و این به معنای عدم پایداری سیستم می‌باشد. همچنین در این صورت $x_i(t)$ و $v_i(t)$ تقریباً برای هر شرایط اولیه‌ای به‌صورت نامحدود افزایش می‌یابند. این مثال ساده، دلیلِ نیاز به استفاده از v_{max} را برای محدود ساختن اندازه‌ی v_i نشان می‌دهد.

با این حال، در این تحلیل فرض شده است که b_i تابعی از x_i نمی‌باشد. علاوه بر این، پیاده‌سازی‌های عملی از PSO بسیار پیچیده‌تر از معادله‌ی (۱۱–۳) بوده و به همین دلیل ممکن است این تحلیل به‌صورت کلی برای الگوریتم‌های PSO صادق نباشد. اگر $\phi_2 > 0$ باشد و یا از یک وزن اینرسی کوچکتر از ۱ استفاده شود (معادله‌ی (۱۱–۹) را ببینید)، آنگاه شاید نیاز نباشد برای دستیابی به عملکرد خوب از محدودسازی سرعت استفاده نماییم [کارلیسل و دوزیر، ۲۰۰۱]، [کلرک و کندی، ۲۰۰۲].

---

[۱] برای کسب اطلاعات در مورد پایداری می‌توانید به هر کتابی در زمینه‌ی سیستم‌های خطی مراجعه نمایید.

اگر بخواهیم از محدودسازی سرعت استفاده نماییم، می‌توانیم به چند طریق این کار را انجام دهیم. یک راه آن است که اندازه‌ی v_i را بررسی نموده و اگر این اندازه بزرگتر از v_{max} بود، عناصرِ v_i را به‌گونه‌ای مقیاس نماییم که $|v_i| = v_{max}$:

$$\text{اگر } |v_i| > v_{max} \text{ آنگاه } v_i \leftarrow \frac{v_i v_{max}}{|v_i|} \qquad (۶-۱۱)$$

در شکل ۱۱-۱ نیز از این معادله استفاده شده است. یک راه دیگر محدود کردن عناصر v_i می‌باشد. به یاد آورید که هر ذره در جمعیت دارای n بعد است، بنابراین $v_i = [v_i(1) \dots v_i(n)]$[1]. با این رویکرد، هر بعد از v_i دارای یک مقدار بیشینه است. به عبارت دیگر در این رویکرد $v_{\max}(j)$ برای هر $j \in [1,n]$ وجود دارد. بدین ترتیب، محدودسازی سرعت برای iاُمین ذره به‌صورت زیر صورت می‌پذیرد:

$$j \in [1,n] \text{ برای } v_i(j) \leftarrow \begin{cases} v_i(j) & \text{اگر } |v_i(j)| \le v_{max}(j) \\ v_{max}(j) sign\big(v_i(j)\big) & \text{اگر } |v_i(j)| > v_{max}(j) \end{cases} \qquad (۷-۱۱)$$

محدودسازی سرعت را می‌توان نوعی کنترل تعادل میان کاوش و انتفاع در نظر گرفت. هرچه $v_{\max}$ بزرگتر باشد، میزان تغییرات هر ذره از یک نسل به نسل دیگر بیشتر بوده و بدین ترتیب به معنای کاوش بیشتر می‌باشد. از سوی دیگر هر چه $v_{\max}$ کوچکتر باشد، تغییرات هر ذره کمتر بوده و این به معنای انتفاع یا بهره‌وری بیشتر می‌باشد.

۱۱-۳ وزن‌دهی اینرسی و ضرایب انقباض

به‌جای استفاده از محدودسازی سرعت، می‌توان معادله‌ی به‌هنگام‌سازی در شکل ۱۱-۱ را به‌گونه‌ای اصلاح نمود که از افزایش بی‌محدودیت سرعت جلوگیری کند. در این بخش ابتدا به بحث در مورد وزن‌دهی اینرسی و سپس به بحث در مورد ضرایب انقباض می‌پردازیم. در نهایت نیز، برخی شرایطِ پایداری برای الگوریتم PSO را نیز ارائه می‌نماییم.

---

[1] توجه داشته باشید که در اینجا عبارت $v_i(j)$ به معنای jاُمین عنصر از بردار v_i می‌باشد در حالی که t در عبارت $v_i(t)$ در معادله‌ی (۱۱-۳) به معنای مقدار v_i در tاُمین نسل می‌باشد.

۱۱–۳–۱ وزن‌دهی اینرسی

معمولاً در PSO از یک وزن اینرسی استفاده می‌شود. همان‌طور که معادله‌ی به‌هنگام‌سازی از شکل ۱۱–۱ نشان می‌دهد، ذرات از یک نسل به نسل دیگر تمایل به حفظ سرعت خود دارند. هرچند کمی تغییر در سرعت به دلیل وجود نرخ‌های یادگیری مجاز است:

$$v_i(k) \leftarrow v_i(k) + \phi_1\big(b_i - x_i(k)\big) + \phi_2(k)\big(h_i(k) - x_i(k)\big) \text{ برای } k \in [1,n] \qquad (۸–۱۱)$$

که در آن n ابعاد مسئله است. با این حال، به‌صورت تجربی مشاهده شده است که با کاهش اینرسی در طول فرایند بهینه‌سازی، عملکرد بهبود می‌یابد. بنابراین معادله‌ی (۱۱–۸) به‌صورت زیر اصلاح می‌گردد:

$$v_i(k) \leftarrow wv_i(k) + \phi_1(k)\big(b_i - x_i(k)\big) + \phi_2\big(h_i(k) - x_i(k)\big) \qquad (۹–۱۱)$$

که در آن w وزن اینرسی بوده و از مقدار ۰٫۹ در نسل اول به حدود ۰٫٤ در نسل آخر کاهش می‌یابد [ابرهارت و شی، ۲۰۰۰]. این موضوع به کاهشِ سرعتِ هر ذره با افزایش شماره‌ی نسل منجر شده و بدین ترتیب همگرایی را بهبود می‌بخشد.

[کلرک و پولی، ۲۰۰٦] برای به‌هنگام‌سازی سرعت‌هایی که به فرم معادله‌ی (۱۱–۹) می‌باشند، مقادیری را برای پارامترهای PSO پیش‌بینی نموده است. در این مقاله، اندازه‌ی جمعیت برابر ۳۰ در نظر گرفته شده، اندازه‌ی همسایگی برابر ٤ بوده و ثابت لحاظ شده‌اند. در صورت بروز رکود در فرایند PSO همسایگی‌ها به‌صورت اتفاقی بازتولید می‌شوند. به‌هنگام‌سازی سرعت به فرم معادله‌ی (۱۱–۹) و با مقادیر پیشنهادی زیر برای پارامترها پیاده‌سازی شده است [کلرک و پولی، ۲۰۰٦، معادله‌ی ۱۹]:

$$\begin{aligned} &w = 0.72 \\ &\phi_1(k) \sim U[0,1.108] \text{ برای } k \in [1,n] \\ &\phi_2(k) \sim U[0,1.108] \text{ برای } k \in [1,n] \end{aligned} \qquad (۱۰–۱۱)$$

[کلرک و پولی، ۲۰۰٦] همچنین انواع دیگری از PSO برای بهبود عملکرد ارائه کرده است. پایداریِ PSO با معادله‌ی به‌هنگام‌سازی سرعت به فرمِ معادله‌ی (۱۱–۹) در [پولی، ۲۰۰۸] بررسی شده است.

مجموعه‌ی جامع‌تری از مقادیر توصیه شده برای پارامترهای PSO جهت استفاده از معادله‌ی به‌هنگام سازی (۱۱–۹) را می‌توان در [پِدِرسِن[1]، ۲۰۱۰] یافت. این مقادیر در جدول ۱۱–۱ نشان داده شده‌اند. این

[1] Pedersen

مقادیرِ توصیه شده هنگامی کاربرد دارند که همسایگیِ هر ذره برابر با کل تجمع می‌باشد. بدین ترتیب، h_i برای تمامی iها در معادله‌ی (۱۱-۹) برابر با بهترین ذره در جمعیت می‌باشد.

جدول ۱۱-۱ مقادیر توصیه شده برای پارامترهای PSO برای ابعادِ مسئله‌ی مختلف و ارزیابی‌های تابع برازندگی [پدرسن، ۲۰۱۰]. در این جدول N اندازه‌ی جمعیت بوده و w، ϕ_1 و ϕ_2 پارامترهای توصیه شده برای معادله‌ی (۱۱-۹) در حالتی می‌باشند که همسایگی هر ذره از کل جمعیت تشکیل شده باشد. با توجه به جدول می‌توان دید که برخی مسائل دارای چندین مجموعه‌ی ممکن از پارامترهای توصیه شده می‌باشند.

ابعاد مسئله	ارزیابی تابع	N	w	ϕ_1	ϕ_2
2	400	25	0.3927	2.5586	1.3358
		29	-0.4349	-0.6504	2.2073
2	4000	156	0.4091	2.1304	1.0575
		237	-0.2887	0.4862	2.5067
5	1000	63	-0.3593	-0.7238	2.0289
		47	-0.1832	0.5287	3.1913
5	10000	223	-0.3699	-0.1207	3.3657
		203	0.5069	2.5524	1.0056
10	2000	63	0.6571	1.6319	0.6239
		204	-0.2134	-0.3344	2.3259
10	20000	53	-0.3488	-0.2746	4.8976
20	40000	69	-0.4438	-0.2699	3.3950
20	400000	149	-0.3246	-0.1136	3.9789
		60	-0.4739	-0.9700	3.7904
		256	-0.3499	-0.0513	4.9087
30	600000	95	-0.6031	-0.6485	2.6475
50	100000	106	-0.2256	-0.1564	3.8876
100	200000	161	-0.2089	-0.0787	3.7637

۱۱-۳-۲ ضرایب انقباض

وزن‌دهی اینرسی اغلب به‌جای استفاده از معادله‌ی (۱۱-۹)، با استفاده از ضرایب انقباض پیاده‌سازی می‌شود. این نوع پیاده‌سازی، که در واقع همان هدفِ وزن‌دهی اینرسی را دنبال می‌نماید، معادله‌ی به‌هنگام‌سازی سرعت را به شکل زیر در می‌آورد

$$v_i \leftarrow K[v_i + \phi_1(b_i - x_i) + \phi_2(h_i - x_i)] \qquad (۱۱-۱۱)$$

که در آن K ضریب انقباض نام دارد [کلرک، ۱۹۹۹]، [ابرهارت و شی، ۲۰۰۰]، [کلرک و کندی، ۲۰۰۲]. ما جهت ساده‌سازی تحلیل خود، از به‌هنگام‌سازی سرعت خطی در معادله‌ی (۱۱-۱۱) استفاده نموده‌ایم. اگر $K = w$ باشد و ϕ_1 و ϕ_2 از معادله‌ی (۱۱-۱۱) به ترتیب با عبارات ${\phi_1}/{K}$ و ${\phi_2}/{K}$ جایگزین شوند، آنگاه

معادله‌ی (۱۱-۱۱) با معادله‌ی (۱۱-۱۹)، معادل خواهد بود. برای تحلیل این رویکرد، از t برای نشان دادن شماره‌ی نسل استفاده کرده و معادله‌ی (۱۱-۱۱) را به‌صورت زیر بازنویسی می‌نماییم:

$$v_i(t+1) = K[v_i(t) + (\phi_1 + \phi_2)(\frac{\phi_1 b_i(t) + \phi_2 h_i(t)}{\phi_1 + \phi_2} - x_i(t))] \quad (۱۱-۱۲)$$
$$= K[v_i(t) + \phi_T\big(p_i(t) - x_i(t)\big)]$$

که ϕ_T و $p_i(t)$ توسط معادله‌ی بالا تعریف شده‌اند. حال معادله‌ی زیر را تعریف می‌نماییم

$$y_i(t) = p_i(t) - x_i(t) \quad (۱۱-۱۳)$$

با فرض آنکه $p_i(t)$ تغییرناپذیر با زمان می‌باشد، می‌توانیم معادلات (۱۱-۱۲) و (۱۱-۱۳) را برای به دست آوردن معادله‌ی زیر ترکیب نماییم

$$\begin{aligned} v_i(t+1) &= Kv_i(t) + K\phi_T y_i(t) \\ y_i(t+1) &= p_i - x_i(t+1) \\ &= p_i - x_i(t) - v_i(t+1) \\ &= y_i(t) - Kv_i(t) - K\phi_T y_i(t) \\ &= -Kv_i(t) + (1 - K\phi_T)y_i(t) \end{aligned} \quad (۱۱-۱۴)$$

معادلات بالا را می‌توان ترکیب نمود و به معادله‌ی ماتریسی زیر دست یافت

$$\begin{bmatrix} v_i(t+1) \\ y_i(t+1) \end{bmatrix} = \begin{bmatrix} K & K\phi_T \\ -K & 1 - K\phi_T \end{bmatrix} \begin{bmatrix} v_i(t) \\ y_i(t) \end{bmatrix} \quad (۱۱-۱۵)$$

ماتریس سمت راست در معادله‌ی بالا که پایداری سیستم را کنترل می‌نماید، دارای مقدیر ویژل زیر می‌باشد

$$\lambda = \frac{1}{2}[1 - K(\phi_T - 1) \pm \sqrt{1 + K^2(\phi_T - 1)^2 - 2K(\phi_T + 1)}] \quad (۱۱-۱۶)$$
$$= \frac{1}{2}\big[1 - K(\phi_T - 1) \pm \sqrt{\Delta}\big]$$

ما مقادیر ویژه را با λ_1 و λ_2 نشان می‌دهیم:

$$\lambda_1 = \frac{1}{2}\big[1 - K(\phi_T - 1) + \sqrt{\Delta}\big] \quad (۱۱-۱۷)$$
$$\lambda_2 = \frac{1}{2}\big[1 - K(\phi_T - 1) - \sqrt{\Delta}\big]$$

سیستم پویای توصیف شده در معادله‌ی (۱۱-۱۵) در صورتی که $|\lambda_1| < 1$ و $|\lambda_2| < 1$ باشد، پایدار خواهد بود. در این تحلیل فرض بر آن است که ϕ_T ثابت است. در معادله‌ی (۱۱-۱۱)، ϕ_iها اتفاقی فرض

شده بودند، در حالی که در اینجا جهت ساده‌سازی تحلیل، فرض بر آن است که ϕ_iها ثابت‌اند. برای بحث پیرامونِ استفاده از K تغییرپذیر با زمان و یا تغییرناپذیر با زمان در PSO، مسئله‌ی ۱۱–۸ را ببینید. در بخش بعدی، رفتار λ_1 و λ_2 را به‌عنوان تابعی از ضریب انقباض K مورد بررسی قرار می‌دهیم. این دو مقدار ویژه نقشی تعیین‌کننده در پایداری الگوریتم PSO ایفا می‌کنند.

۱۱–۳–۳ پایداری PSO

می‌توان از معادله‌ی (۱۱–۱۶) برای انجام مشاهدات زیر استفاده نمود.

مشاهده‌ی ۱۱–۱. هنگامی که $K = 0$ باشد، $\Delta = 1$، $\lambda_1 = 1$ و $\lambda_2 = 0$ خواهد بود.

حال مقادیر λ_1 و λ_2 را با افزایش K در نظر می‌گیریم. با توجه به معادله‌ی (۱۱–۱۶) داریم

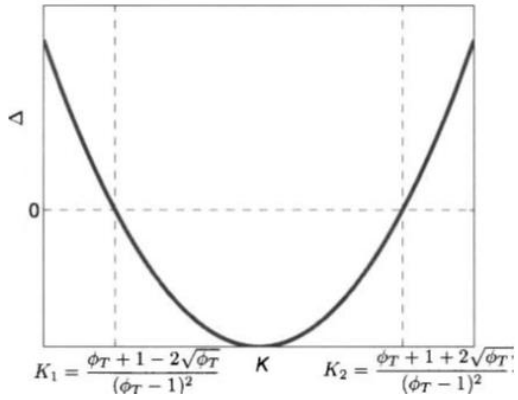
$$\lim_{|k|\to\infty} \Delta = \infty$$

$$\Delta = 0 \text{ برای } K = \frac{\phi_T + 1 \pm 2\sqrt{\phi_T}}{(\phi_T - 1)^2} = \{K_1, K_2\} \qquad (۱۱–۱۸)$$

$$\Delta < 0 \text{ برای } K \in (K_1, K_2)$$

که با توجه به معادلات بالا تعاریف K_1 و K_2 واضح است. فرض می‌کنیم $\phi_T > 0$ تا $\sqrt{\phi_T}$ مقداری حقیقی باشد. شکل ۱۱–۳ نموداری از Δ بر حسب تابعی از K را نشان می‌دهد. این موضوع مشاهده‌ی بعدی را نتیجه می‌دهد.

مشاهده‌ی ۱۱–۲. λ_1 و λ_2 برای $K < K_1$ حقیقی می‌باشند.



شکل ۱۱–۳ نمودار Δ از معادله‌ی (۱۱–۱۶) بر حسب تابعی از ضریب انقباض K. به ازای $K > K_2$ و $K < K_1$، $\Delta > 0$ بوده و این بدین معنی است که هر دو مقدار λ_1 و λ_2 حقیقی خواهند بود. به ازای $K \in (K_1, K_2)$ نیز $\Delta < 0$ بوده که این نیز به معنای مختلط بودنِ λ_1 و λ_2 خواهد بود.

هنگامی که $K = K_1$ باشد، $\Delta = 0$ بوده و این به معنی $\lambda_1 = \lambda_2$ می‌باشد. مشاهدات بعدی را می‌توان به راحتی از معادله‌ی (۱۱-۱۶) دریافت.

مشاهده‌ی ۱۱-۳. در صورتی که $K = K_1$ باشد، $\lambda_1 = \lambda_2 = (1 - \sqrt{\phi_T})/(1 - \phi_T)$ به دست می‌آید که این مقدار برای تمامی $\phi_T \neq 1$ها بین ۰ و ۱ واقع خواهد شد.

حال رفتار λ_1 را هنگامی که K از ۰ به K_1 میل می‌کند در نظر بگیرید. با گرفتن مشتق از λ_1 نسبت به K خواهیم داشت

$$\frac{d\lambda_1}{dK} = \frac{1 - \phi_T}{2} - \frac{\phi_T - K(\phi_T - 1)^2 + 1}{\sqrt{K^2(\phi_T - 1)^2 - 2K(\phi_T + 1) + 1}} \qquad (۱۱-۱۹)$$

با به کارگیری عملیات جبری می‌توان نشان داد که اگر $\phi_T > 1$ باشد، آنگاه این مشتق به ازای $K < K_1$ منفی می‌باشد. به همین ترتیب می‌توان نشان داد که مشتق λ_2 به ازای $K < K_1$ مثبت خواهد بود. این موضوع ما را به سمت مشاهده‌ی بعدی سوق می‌دهد.

مشاهده‌ی ۱۱-۴. اگر $\phi_T > 1$ باشد، آنگاه λ_1 و λ_2 هر دو به ازای $K \in (0, K_1)$ میان ۰ و ۱ واقع خواهند شد.

حال رفتار λ_1 و λ_2 را هنگامی که K از ۰ به K_1 میل می‌کند در نظر بگیرید. از شکل ۱۱-۳ می‌توان مشاهده نمود که هنگامی که $K \in (K_1, K_2)$ است، λ_1 و λ_2 هر دو مقداری مختلط با اندازه‌ی برابر خواهند داشت. این اندازه برابرست با

$$|\lambda| = \frac{1}{2}\sqrt{[1 - K(\phi_T - 1)]^2 + 2K(\phi_T + 1) - 1 - K^2(\phi_T - 1)^2} \qquad (۱۱-۲۰)$$

با کمی عملیات جبری می‌توان نشان داد که $|\lambda| = \sqrt{K}$. مشتق این عبارت به ازای تمامی $K > 0$ مثبت خواهد شود. این موضوع ما را به سمت مشاهده‌ی بعدی سوق می‌دهد.

مشاهده‌ی ۱۱-۵. هنگامی که $K \in (K_1, K_2)$ می‌باشد، λ_1 و λ_2 مختلط بوده و دارای اندازه‌ی برابر می‌باشند. این اندازه به‌صورت یکنواخت با افزایش K افزایش می‌یابد.

حال مقادیر λ_1 و λ_2 را هنگامی که $K = K_2$ است در نظر بگیرید. از شکل ۱۱-۳ می‌توان دید که اگر $K = K_2$ باشد، λ_1 و λ_2 دارای مقادیر حقیقی می‌باشند. در حقیقت، می‌توان به سادگی از معادله‌ی (۱۱-۱۶) مشاهده نمود که هنگامی که $K = K_2$ است، $\lambda_1 = \lambda_2 = (1 + \sqrt{\phi_T})/(1 - \phi_T)$ خواهد بود. این عبارت برای تمامیِ $\phi_T > 4$ها بین ۰ و ۱- واقع خواهد شد. این موضوع را می‌توان در غالب مشاهده‌ی بعدی بیان نمود.

مشاهده‌ی ۱۱-٦. اگر $K = K_2$ باشد، $\lambda_1 = \lambda_2 = (1 + \sqrt{\phi_T})/(1 - \phi_T)$ خواهد بود و این مقدار برای تمامی $\phi_T > 4$ها بین ۰ و ۱- واقع خواهد شد.

حال مقادیر λ_1 و λ_2 را هنگامی که $K > K_2$ است در نظر بگیرید. هم λ_1 و هم λ_2 برای این محدوده از K دارای مقداری حقیقی خواهند بود. معادله‌ی (۱۱-۱۹) مشتقِ λ_1 نسبت به K را هنگامی که λ_1 حقیقی است به دست می‌دهد. با کمی عملیات جبری می‌توان نشان داد که برای $\phi_T > 1$ و $K > K_2$، مشتق λ_1 مثبت و مشتق λ_2 منفی می‌باشد. با ترکیب این استدلال و مشاهده‌ی ۱۱-٦ می‌بینیم که اندازه‌ی λ_1 برای تمامی Kهای بزرگتر از K_2 کمتر از ۱ خواهد بود. با این حال، همان‌طور که از معادله‌ی (۱۱-۱۷) نیز پیداست، با میل کردن K به سمت ∞، λ_2 به سمت $-\infty$ میل خواهد کرد. این موضوع به مشاهده‌ی بعدی منجر می‌گردد.

مشاهده‌ی ۱۱-۷. هنگامی که $K > K_2$ باشد، λ_1 حقیقی و منفی بوده و اندازه‌ای کمتر از ۱ خواهد داشت. λ_2 نیز حقیقی و منفی بوده و با میل کردن K به سمت ∞، به سمت $-\infty$ میل خواهد کرد.

حدِ λ_1 را هنگامی که $K \to \infty$، می‌توان از معادله‌ی (۱۱-۱٦) به دست آورد:

$$\lambda_1 = \frac{1}{2}[1 - K(\phi_T - 1) + \sqrt{1 + K^2(\phi_T - 1)^2 - 2K(\phi_T + 1)}$$

$$= \frac{1/K - (\phi_T - 1) + \sqrt{1/K^2 + (\phi_T - 1)^2 - 2(\phi_T + 1)/K}}{2/K} \qquad (۲۱-۱۱)$$

$$= \frac{N(K)}{D(K)}$$

حد دو عبارت $N(K)$ و $D(K)$ هنگام $K \to \infty$ برابر ۰ می‌باشد، بنابراین برای محاسبه‌ی عبارت بالا می‌توان از قاعده‌ی هوپیتال استفاده نمود:

$$\frac{dN(K)}{dK} = -K^{-2} + \frac{-2K^{-3} + 2(\phi_T + 1)K^{-2}}{2\sqrt{K^{-2} + (\phi_T - 1)^2 - 2(\phi_T + 1)K^{-1}}}$$

$$\frac{dD(K)}{dK} = -2K^{-2}$$

$$\frac{dN(K)/dK}{dD(K)/dK} = \frac{1}{2} + \frac{K^{-1} - \phi_T - 1}{2\sqrt{K^{-2} + (\phi_T - 1)^2 - 2(\phi_T + 1)K^{-1}}} \qquad (۲۲-۱۱)$$

$$\lim_{K\to\infty} \lambda_1 = \lim_{K\to\infty} \frac{dN(K)/dK}{dD(K)/dK}$$

$$= \frac{1}{2} - \frac{\phi_T + 1}{2(\phi_T - 1)}$$

$$= \frac{1}{1 - \phi_T}$$

بنابراین حد بالا هنگامی که $\phi_T > 2$ باشد، اندازه‌ای کمتر از ۱ خواهد داشت. این موضوع ما را به سوی مشاهده‌ی بعدی سوق می‌دهد. این مشاهده تعمیمی است از مشاهدات ۱۱–۶ و ۱۱–۷.

مشاهده‌ی ۱۱–۸ با افزایش K از K_2 تا ∞، λ_1 به‌صورت یکنواخت از $(1+\sqrt{\phi_T})/(1-\phi_T)$ تا $1/(1-\phi_T)$ افزایش پیدا کرده و λ_2 نیز به‌صورت یکنواخت از $(1+\sqrt{\phi_T})/(1-\phi_T)$ تا $-\infty$ کاهش پیدا می‌کند.

از آنجا که λ_2 از $(1+\sqrt{\phi_T})/(1-\phi_T)$، که دارای مقداری بزرگتر از ۱– است، تا $-\infty$ کاهش پیدا می‌کند، باید به ازای مقادیری از K، که در اینجا با K_3 نشان می‌دهیم، با ۱– برابر باشد. بنابراین، برای به دست آوردن K_3 می‌توان از معادله‌ی (۱۱–۱۷) استفاده نمود

$$-1 = \frac{1}{2}[1 - K_3(\phi_T - 1) - \sqrt{1 + K^2(\phi_T - 1)^2 - 2K(\phi_T + 1)}] \qquad (۱۱–۲۳)$$

با حل این معادله برای K_3 به $K_3 = 2/(\phi_T - 2)$ دست پیدا خواهیم کرد. با ترکیب این نتیجه با تمامی مشاهدات بالا، قضیه‌ی زیر به دست خواهد آمد.

قضیه‌ی ۱۱–۱ اگر در معادله‌ی به‌هنگام‌سازی سرعت (۱۱–۱۱)، b_i و h_i ثابت باشند، و همچنین اگر $\phi_T = \phi_1 + \phi_2 > 4$ باشد، آنگاه PSO به ازای معادله‌ی زیر پایدار خواهد بود.

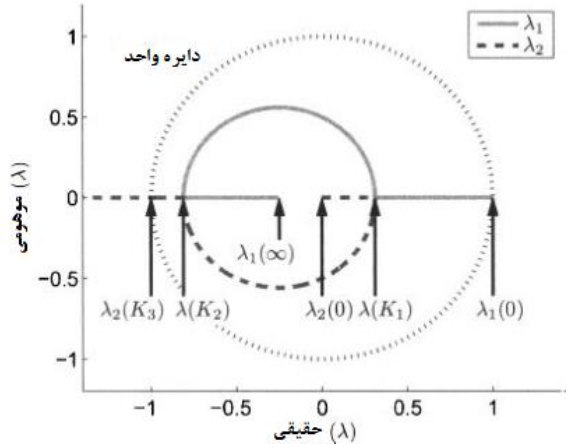
$$K < \frac{2}{\phi_T - 2} \qquad (۱۱–۲٤)$$

شکل ۱۱–٤ نحوه‌ی تغییر مقادیرِ ویژه‌ی معادله‌ی (۱۱–۱۵) در صفحه‌ی مختلط با افزایش K از ۰ تا ∞ را نشان می‌دهد. شکل ۱۱–۱۵ نیز نحوه‌ی تغییر اندازه‌ی این مقادیر ویژه را نشان می‌دهد.

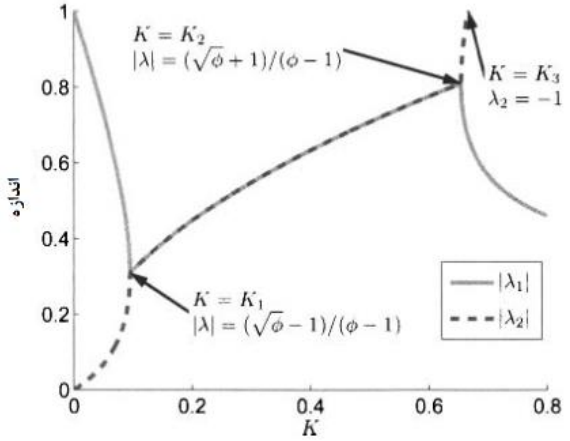
می‌توان K را برای الگوریتم پایدار PSO به‌صورت زیر نوشت

$$K = \frac{2\alpha}{\phi_T - 2} \quad \text{که در آن} \quad \phi_T = \phi_{1,max} + \phi_{2,max} \qquad (۱۱–۲۵)$$

با $\alpha \in (0,1)$. α میزان نزدیکی ضریب انقباض به مقدارِ ماکزیمم نظری را پیش از اینکه الگوریتم PSO ناپایدار گردد نشان می‌دهد. α بزرگتر به معنای کاوش بیشتر و α کوچکتر به معنای انتفاع و بهره‌وری بیشتر می‌باشد.



شکل ۱۱-٤ مقادیر ویژه‌ی معادله‌ی (۱۱-۱۵) هنگامی که ضریب انقباض K از ۰ تا ∞ تغییر می‌کند. این شکل به ازای $\phi_T = 5$ رسم شده است. به ازای $K = 0$، $\lambda_1 = 1$ و $\lambda_2 = 0$ می‌باشند. در $K = K_1$، $\lambda_1 = \lambda_2 > 0$ می‌باشد. برای $K \in (K_1, K_2)$، λ_1 و λ_2 دارای مقدار مختلط می‌باشند. در $K = K_2$، $\lambda_1 = \lambda_2 < 0$ بوده و در $K = K_3$، $\lambda_2 = -1$ می‌باشد. به ازای $K \to \infty$ نیز $\lambda_1 \to 1/(1-\phi_T)$ و $\lambda_2 \to -\infty$ خواهد بود.



شکل ۱۱-۵ اندازه‌ی مقادیر ویژه‌ی معادله‌ی (۱۱-۱۵) هنگامی که ضریب انقباض K از ۰ افزایش می‌یابد. این شکل به ازای $\phi_T = 5$ رسم شده است. این نمودار اندازه‌ی مقادیر ویژه‌ی رسم شده در شکل ۱۱-٤ را به تصویر می‌کشد.

به‌صورت معمول، الگوریتم PSO با توصیه‌ی زیر ارائه می‌شود [کارلیسل و دوزیر، ۲۰۰۱]، [کلرک و کندی، ۲۰۰۲]، [ابرهارت و شی، ۲۰۰۰]، [پولی و همکاران، ۲۰۰۷]:

توصیه‌ی معمول: $\phi_T > 4$

$$K < \frac{2}{\phi_T - 2 + \sqrt{\phi_T(\phi_T - 4)}} \qquad (۱۱-۲۶)$$

این توصیه با قضیه‌ی ۱۱-۱، هنگامی که $\phi_T \to 4$، معادل است. البته قضیه‌ی ۱۱-۱ کلی‌تر است. معادله‌ی (۱۱-۲۶) هیچ سرنخی در مورد حد بالایِ ϕ_T و یا این موضوع که چه مقداری از ϕ_T باید به $\phi_{1,max}$ و چه مقدار از آن باید به $\phi_{2,max}$ اختصاص داده شود، به دست نمی‌هد. معمولاً توصیه می‌شود که ϕ_T کمی بزرگتر از ٤ انتخاب شده و به‌صورت تقریباً مساوی میان $\phi_{1,max}$ و $\phi_{2,max}$ تقسیم شود. توجه داشته باشید که تحلیل ما یک روش خاص است. روش‌های دیگر، دارای فرضیات متفاوتی نسبت به روش ما بوده و به شرطِ پایداری متفاوتی منجر خواهند شد [کلرک و پولی، ۲۰۰٦].

۱۱-۴ به‌هنگام‌سازی سرعت سراسری

معادله‌ی به‌هنگام‌سازی سرعتِ (۱۱-۱۱) را می‌توان به‌صورت زیر تعمیم داد

$$v_i \leftarrow K[v_i + \phi_1(b_i - x_i) + \phi_2(h_i - x_i) + \phi_3(g - x_i)] \qquad (۱۱-۲۷)$$

که در آن g بهترین ذره‌ی پیدا شده از زمانِ اولین نسل می‌باشد. اگر فرض کنیم $\phi_T = \phi_{1,max} + \phi_{2,max} + \phi_{3,max}$ و همچنین اگر فرض کنیم $b_i + h_i + g$ ثابت است، آنگاه تحلیل بخش قبل برای معادله‌ی (۱۱-۲۷) برقرار خواهد بود. عبارتِ $\phi_3(g - x_i)$ در معادله‌ی به‌هنگام‌سازی سرعتِ جدید، هر ذره را به سمت بهترین ذره‌ی پیدا شده از زمانِ اولین نسل سوق می‌دهد. این موضوع از لحاظ مفهومی مانند الگوریتم تکاملی stud است، الگوریتمی که در آن در هر نسل از بهترین ذره برای عملیات بازترکیب استفاده می‌شود (بخش ۸-۷-۷). تنها تفاوت این است که در معادله‌ی (۱۱-۲۷)، g بهترین ذره‌ی پیدا شده از زمانِ اولین نسل تا کنون است، در حالی که در بخش ۸-۷-۷، stud بهترین ذره در نسل حاضر است. این تفاوت و شباهت ممکن است باعثِ ایجاد ایده‌ی استفاده از عملیات شبه-g در الگوریتم تکاملی stud و یا استفاده از عملیات شبه-stud در الگوریتم سراسری PSO شود.

مثال ۱۱-۱

در این مثال از الگوریتم PSO با معادله‌ی به‌هنگام‌سازی سرعت سراسری (۱۱-۲۷) برای مینیمم‌سازی تابع آکلی ۱۰ بعدی استفاده می‌نماییم. برای این منظور از اندازه‌ی جمعیتی برابر با ۵۰، اندازه‌ی همسایگی

$\sigma = 4$ و پارامترِ نخبه‌گرایی برابر ۲ استفاده خواهیم نمود. همچنین از مقادیر نامی برای پارامترهای زیر استفاده خواهیم نمود

$$\phi_{1,max} = \phi_{2,max} = \phi_{3,max} = 2.1$$
$$\phi_T = \phi_{1,max} + \phi_{2,max} + \phi_{3,max} \qquad (۲۸-۱۱)$$
$$K = \frac{2\alpha}{\phi_T - 2}, \qquad \alpha = 0.9$$

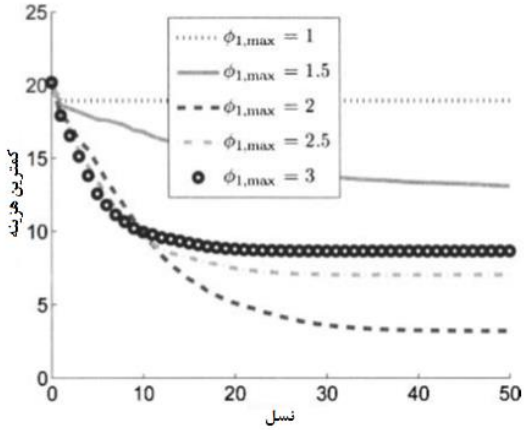
توجه داشته باشید که ϕ_T را می‌توان بر حسب K به دست آورد:

$$\phi_T = \frac{2(\alpha + K)}{K} \qquad (۲۹-۱۱)$$

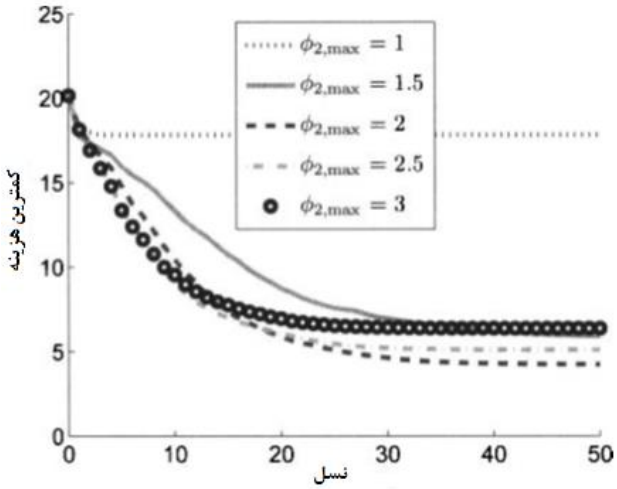
اشکال ۱۱-۱۶ تا ۱۱-۱۹ میانگینِ عملکرد PSO را به ازای مقادیر مختلف $\phi_{1,max}$، $\phi_{2,max}$ و $\phi_{3,max}$ و α، هنگامی که سایر پارامترها دارای مقادیر نامی خود می‌باشند، نشان می‌دهند. می‌توان دید که به راستی مقادیر نامی موجود در معادله‌ی (۱۱-۲۸) به‌صورت تقریبی برای تابع ۲۰ بعدی آکلی بهینه می‌باشند.

اشکال ۱۱-۶ تا ۱۱-۸ حاکی از آن هستند که هنگامی که مقادیر ϕ_{max} بسیار کوچک هستند، ذرات به شیوه‌ای بدون جهت سرگردانند. این در حالی است که برای مقادیر بسیار بزرگ ϕ_{max}، ذرات زیادی در حرکات خود محدود بوده و قادر به کاوش مؤثر فضای جستجو نخواهند بود.

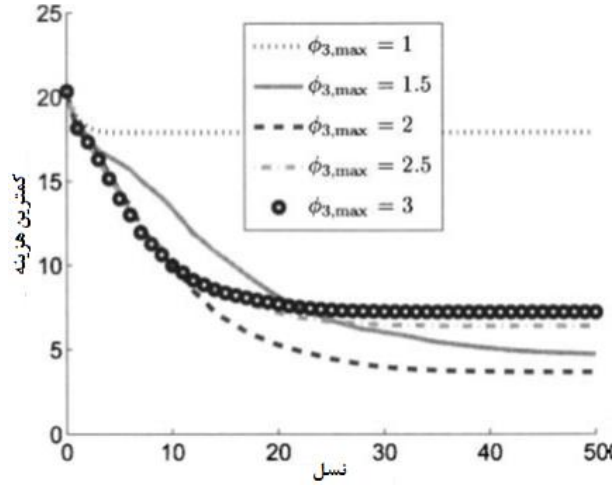
شکل ۱۱-۹ نشان می‌دهد در صورتی که α بیش از حد کوچک باشد، ذرات به دلیل سرعت‌های کم دچار رکود می‌شوند. هنگامی که α بسیار بزرگ باشد، ذرات به طرز شدیدی در فضای جستجو به اطراف پرش می‌کنند.



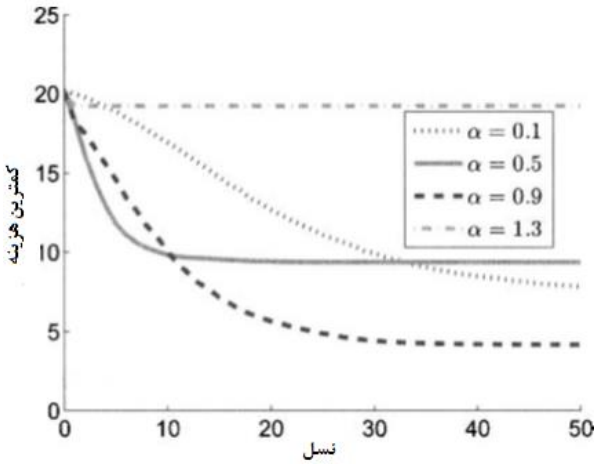
شکل ۱۱-۶ مثال ۱۱-۱: عملکرد PSO بر روی تابع ۲۰ بعدی آکلی به ازای مقادیر مختلف $\phi_{1,max}$، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است. $\phi_{1,max} = 2$ به‌صورت تقریبی برای این تابع محک مقدار بهینه می‌باشد.



شکل ۱۱–۷ مثال ۱۱–۱: عملکرد PSO بر روی تابع ۲۰ بعدی اَکلی به ازای مقادیر مختلف $\phi_{2,max}$، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است. $\phi_{2,max} = 2$ به‌صورت تقریبی برای این تابع محک مقدار بهینه می‌باشد.



شکل ۱۱–۸ مثال ۱۱–۱: عملکرد PSO بر روی تابع ۲۰ بعدی اَکلی به ازای مقادیر مختلف $\phi_{3,max}$، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است. $\phi_{3,max} = 2$ به‌صورت تقریبی برای این تابع محک مقدار بهینه می‌باشد.



شکل ۱۱–۹ مثال ۱۱–۱: عملکرد PSO بر روی تابع ۲۰ بعدی آکلی به ازای مقادیر مختلفِ ضریب انقباض $K = \alpha K_{max}$، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است. $K = 0.9K_{max}$ به‌صورت تقریبی برای این تابع محک مقدار بهینه می‌باشد.

۱۱–۵ تجمع ذرات کاملاً آگاه

معادلات (۱۱–۱۲) و (۱۱–۲۷) نشان می‌دهند کلی‌ترین فرم معادله‌ی به‌هم‌رسانی سرعت (تا کنون) به‌صورت زیر است

$$\begin{aligned} v_i(t+1) &= K[v_i(t) + \phi_T(p_i(t) - x_i(t))] \\ \phi_T &= \phi_{1,max} + \phi_{2,max} + \phi_{3,max} \\ p_i(t) &= \frac{\phi_1 b_i(t) + \phi_2 h_i(t) + \phi_3 g(t)}{\phi_1 + \phi_2 + \phi_3} \end{aligned} \qquad (۳۰–۱۱)$$

با توجه به این معادلات می‌توان دید که سه مکان در به‌هم‌رسانی سرعت دخالت دارند: بهترین مکانِ ذره تا کنون یا $b_i(t)$، بهترین مکانِ کنونی همسایه یا $h_i(t)$ و بهترین مکانِ جمعیت تا کنون یا $g(t)$. این موضوع، این ایده را به ما می‌دهد که تمامی ذرات موجود در جمعیت را در به‌هم‌رسانی سرعت دخالت دهیم. یک نوع تعمیم از معادله‌ی (۱۱–۳۰) را می‌توان به‌صورت زیر نوشت:

$$\begin{aligned} v_i(t+1) &= K[v_i(t) + \phi_T(p_i(t) - x_i(t))] \\ \phi_T &= \frac{1}{N}\sum_{j=1}^{N} \phi_{j,max} \\ p_i(t) &= \frac{\sum_{j=1}^{N} w_{ij}\phi_j b_j(t)}{\sum_{j=1}^{N} w_{ij}\phi_j} \end{aligned} \qquad (۳۱–۱۱)$$

که در آن $b_j(t)$ بهترین راه‌حل پیدا شده تاکنون توسطِ ذره‌ی jاُم می‌باشد:

$$b_j(t) = arg\min_x f(x): x \in \{x_j(0), \dots, x_j(t)\} \quad (۳۲-۱۱)$$

توجه داشته باشید که در تعریف ϕ_T یک فاکتور $1/N$ وجود دارد که روشی موقت برای ایجاد تعادلی منطقی میان میزان دخالت $v_i(t)$ و $(p_i(t) - x_i(t))$ در سرعت جدید $v_i(t+1)$ می‌باشد. پارامترهای ϕ_j موجود در معادله‌ی (۱۱-۳۱) فاکتورهای تأثیر اتفاقی با توزیع یکنواخت بر روی $U[0, \phi_{j,max}]$ می‌باشند. همان‌طور که در مثال ۱۱-۱ نیز اشاره شد، ما معمولاً از مقادیر زیر استفاده می‌نماییم

$$\begin{aligned} &\phi_{j,max} \approx 2 \\ &K = 2\alpha/(3\phi_T - 2) \end{aligned} \quad (۳۳-۱۱)$$

که در آن $\alpha \in (0,1)$ می‌باشد. وجود فاکتور ۳ موجود در معادله‌ی K در بالا به دلیل این حقیقت است که در معادله‌ی (۱۱-۲۷)، ϕ_T مجموع سه عبارتِ $\phi_{j,max}$ بوده در حالی که در معادله‌ی (۱۱-۳۱) با میانگین سه عبارت $\phi_{j,max}$ برابر می‌باشد. وزن‌های w_{ij} در معادله‌ی (۱۱-۳۱) فاکتورهای قطعی هستند که میزان تأثیر jاُمین ذره بر روی سرعت iاُمین ذره را تعیین می‌نمایند. گاهاً ما از مقادیر ثابت برای w_{ij} استفاده می‌نماییم. گاهی نیز تمایل داریم برای مقادیری از j که متناظر با ذرات x_j بهتر و یا نزدیکتر به x_i می‌باشند، از w_{ij} بزرگتر استفاده نماییم. برای نمونه، اگر مسئله‌ی مورد نظر یک مسئله‌ی مینیمم‌سازی باشد، می‌توان از معادله‌ی زیر برای w_{ij} استفاده نمود

$$w_{ij} = \left[\max_k f(x_k) - f(x_j)\right] + \left[\max_k |x_i - x_k| - |x_i - x_j|\right] \quad (۳٤-۱۱)$$

که در آن $|.|$ اپراتور اندازه‌گیری فاصله است. همچنین ممکن است نیاز باشد همکاری هزینه و برازندگی را نیز وزن‌دهی کنیم تا هر دو از مرتبه‌ی یکسانی در w_{ij} دخالت داشته باشند. برای مثال،

$$\begin{aligned} S_i &= \frac{\max_k f(x_k) - \min_k f(x_k)}{\max_k |x_i - x_k|} \\ w_{ij} &= \left[\max_k f(x_k) - f(x_j)\right] + S_i \left[\max_k |x_i - x_k| - |x_i - x_j|\right] \end{aligned} \quad (۳۵-۱۱)$$

S_i فاکتوری است که میزان دخالت دو عبارت در w_{ij} را تقریبا یکسان می‌سازد. از آنجا که معادله‌ی (۱۱-۳۱) اجازه‌ی تأثیرگذاری بر روی تمام ذرات را به همه‌ی ذرات می‌دهد، این نوع تجمع ذرات با نام

تجمع ذرات کاملاً آگاه (FIPS)[1] شناخته می‌شود [مندس و همکاران، ۲۰۰٤]. این ایده، بازترکیب جهانی یکنواخت در الگوریتم‌های تکاملی را به یاد می‌آورد (بخش ۸–۸–٦).

مثال ۱۱–۲

در این مثال ما از تجمع ذرات کاملاً آگاه و وزن‌ها تعریف شده در معادله‌ی (۱۱–۳۵) برای مینیمم‌سازی تابع آکلی ۲۰ بعدی استفاده می‌نماییم. برای این کار اندازه‌ی جمعیت را برابر ٤۰ قرار داده و از دو پارامتر نخبه استفاده می‌نماییم. همچنین از مقادیر نامی زیر استفاده می‌نماییم

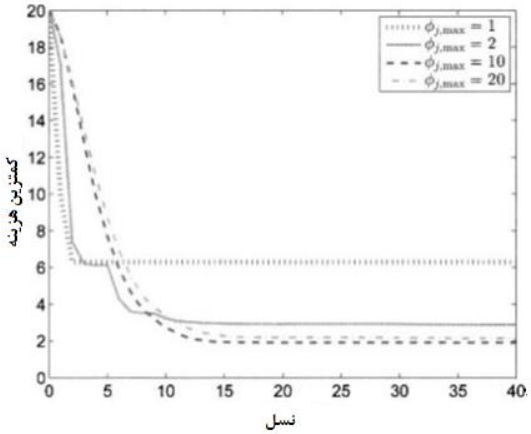
$$\phi_{j,max} = \phi_{max} = 2 \text{ برای } j \in [1,20]$$
$$K = 2\alpha/(3\phi_{max} - 2), \quad \alpha = 0.9 \qquad (۳٦–۱۱)$$

اشکال ۱۱–۱۰ و ۱۱–۱۱ میانگین عملکرد PSO را برای مقادیر مختلف ϕ_{max} و α، هنگامی که سایر پارامترها دارای مقادیر نامی خود می‌باشند، نشان می‌دهد.

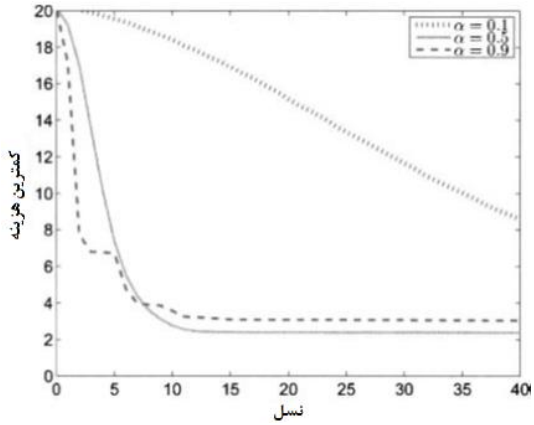
با توجه به شکل ۱۱–۱۰، هنگامی که ϕ_{max} بسیار کوچک است، الگوریتم بسیار سریع همگرا شده اما راه‌حل حاصل شده پس از همگرایی بسیار ضعیف خواهد بود. با افزایش ϕ_{max} سرعت همگرایی کاهش یافته اما راه‌حل یافت شده در انتهای همگرایی بهتر می‌شوند. این موضوع به ما انگیزه‌ی استفاده از نوعی ϕ_{max} تطبیق‌پذیر را می‌دهد که در ابتدا کوچک بوده اما با گذشت زمان افزایش می‌یابد. شکل ۱۱–۱۱ نشان می‌دهد که برای مقادیر کوچک α همگرایی بسیار کند است. از سوی دیگر به ازای $\alpha = 0.9$ همگرایی بیشترین سرعت را داشته اما بهترین راه‌حل به ازای $\alpha = 0.5$ حاصل می‌شود.

این نتایج بسیار خاص هستند بدین معنی که برای نوعی خاص از تابع محک و ابعادی خاص و پارامترهای نخبه‌گرایی خاص و شکل خاصی از پارامترهای وزن‌دهی w_{ij} کاربرد دارند. برای جواب به این سؤال که آیا می‌توان این نتایج را به گستره‌ی وسیع‌تری از مسائل تعمیم داد باید به آزمایش‌های بیشتری دست زد.

[1] Fully Informed Particle Swarm



شکل ۱۱-۱۰ مثال ۱۱-۲: عملکرد تجمع ذرات کاملاً آگاه به ازای مقادیر مختلف ϕ_{max} که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است. $\phi_{max} = 1$ بهترین عملکرد کوتاه- مدت را به دست می‌دهد در حالی که ϕ_{max}های بزرگتر، عملکرد بلند- مدت بهتری را رقم خواهند زد.



شکل ۱۱-۱۱ مثال ۲: عملکرد تجمع ذرات کاملاً آگاه به ازای مقادیر مختلف α که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است- $\alpha = 0.9$ بهترین عملکرد کوتاه- مدت را به دست داده در حالی که $\alpha = 0.5$ بهترین عملکرد بلند- مدت را نتیجه خواهد داد.

گاهاً PSO به‌گونه‌ای متفاوت از معادله‌ی (۱۱-۳۱) نوشته می‌شود. برای مثال، معادله‌ی (۱۱-۳۱) را می‌توان با معادله‌ی زیر جایگزین نمود [پولی و همکاران، ۲۰۰۷]:

$$v_i(t+1) = K\left[v_i(t) + \frac{1}{n_i}\sum_{j=1}^{n_i}\phi_j\left(b_{i,j}(t) - x_i(t)\right)\right] \qquad (۱۱-۳۷)$$

که در آن n_i اندازه‌ی همسایگی iاُمین ذره بوده، ϕ_j از توزیع یکنواخت بر روی $U[0, \phi_{j,max}]$ گرفته شده و $b_{i,j}(t)$ نیز بهترین راه‌حل پیدا شده توسط jاُمین همسایه‌ی iاُمین ذره می‌باشد. در این فرمول، هر ذره دارای یک همسایگی ثابت و مشخص بوده و بهترین راه‌حلِ هر همسایه $b_{i,j}(t)$، همگی دارای وزن همکاریِ یکسانی در به هم‌رسانی سرعتِ iاُمین ذره می‌باشند. توجه داشته باشید که تحت شرایط خاص معادله‌ی (۱۱–۳۷) با معادله‌ی (۱۱–۱۱) برابر خواهد بود. یافته‌های برخی مقالات حاکی از عملکرد ضعیف PSO کاملاً آگاه می‌باشد [دِ اوکا[1] و استاتزل، ۲۰۰۸].

۱۱–۶ یادگیری از اشتباهات

ایده‌ی اصلی موجود در الگوریتم PSO آن است که ارگانیسم‌های زیستی، تمایل به استفاده از استراتژی‌هایی دارند که در گذشته موفقیت و اثربخشی‌شان ثابت شده است. این استراتژی‌ها شامل استراتژی‌های مفیدی است که خودِ ارگانیسم‌ها و یا سایر ارگانیسم‌ها مورد استفاده قرار داده‌اند. معادله‌ی اصلی به‌هم‌رسانی سرعت، همان‌طور که از معادله‌ی (۱۱–۲۷) مشاهده می‌شود، برابرست با

$$v_i(t) \leftarrow K[v_i(t) + \phi_1(b_i - x_i) + \phi_2(h_i - x_i) + \phi_3(g - x_i)] \qquad (۱۱–۳۸)$$

که در آن x_i و v_i به ترتیب مکان و سرعت ذره‌ی iاُم بوده و h_i بهترین مکان فعلی موجود در همسایگی ذره‌ی iاُم می‌باشد. g بهترین مکان قبلی در میان کل جمعیت بوده و K، $\phi_{1,max}$ و $\phi_{2,max}$ پارامترهای میزان سازی مثبت می‌باشند.

با این حال، ارگانیسم‌های زیستی نه تنها از موفقیت‌ها، بلکه از اشتباهات نیز یاد می‌گیرند. ما معمولاً می‌خواهیم از تکرار اشتباهات گذشته پرهیز کنیم. این شامل استراتژی‌های مضری می‌شود که ما خود استفاده کرده و یا در رفتار دیگران مشاهده نموده‌ایم. یک تعمیم از PSO نیز این است که پرهیز از رفتارهای منفی را در الگوریتم اصلی PSO دخالت دهیم. در [یانگ و سیمون، ۲۰۰۵] و [سِلواکومار[2] و تانوشکودی[3]، ۲۰۰۷] از عبارتِ "PSO جدید" برای اشاره به این الگوریتم استفاده شده است، اما از آنجایی که این عبارت بی مسماست، ما در این بخش از عبارت PSO تقویت منفی (NPSO)[4] برای اشاره به این الگوریتم استفاده خواهیم نمود.

---

[1] De Oca
[2] Selvakumar
[3] Thanushkodi
[4] Negative Reinforcement PSO

در NPSO هر ذره نه تنها سرعت خود را به‌گونه‌ای تنظیم می‌نماید که در راستای بهترین محل خود و همسایگانش قرار گیرد، بلکه این تنظیم‌سازی سرعت به‌گونه‌ای اجرا می‌شود که باعث حرکت ذره در جهت مخالفِ بدترین محل پیشین خود و سایر همسایگانش شود. بنابراین می‌توان معادله‌ی (۱۱–۳۸) را به‌صورت زیر اصلاح نمود.

$$v_i(t) \leftarrow K\big[v_i(t) + \phi_1(b_i - x_i) + \phi_2(h_i - x_i) + \phi_3(g - x_i) - \phi_4(\bar{b}_i - x_i) - \phi_5(\bar{h}_i - x_i) - \phi_6(\bar{g} - x_i)\big] \qquad (۱۱–۳۹)$$

که در آن $\bar{b}_i$ بدترین مکان در میان مکان‌های پیشین ذره‌ی iاُم بوده، $\bar{h}_i$ بدترین مکان فعلی در همسایگی iاُم بوده، $\bar{g}$ بدترین مکان در میان مکان‌های پیشین کل جمعیت بوده، هر ϕ_j از توزیع یکنواخت بر روی $(0, \phi_{j,\max})$ گرفته شده و هر $\phi_{j,\max}$ یک پارامتر میزان‌سازی مثبت می‌باشد.

در این میان ما باید میان میزان‌سازی سرعت در جهت راه‌حل‌های مثبت که در PSO استاندارد وجود داشت و همچنین میزان‌سازی سرعت در خلاف جهت راه‌حل‌های مضر که به NPSO اضافه نمودیم، تعادل برقرار نماییم. این تعادل چیزی است که حتی خود ما در زندگی روزمره با آن سروکار داریم. برخی ممکن است معتقد باشند تقویت مثبت از تقویت منفی مؤثرتر است، اما باید توجه داشت که هم تقویت مثبت و هم تقویت منفی در یادگیری مهم و مؤثرند.

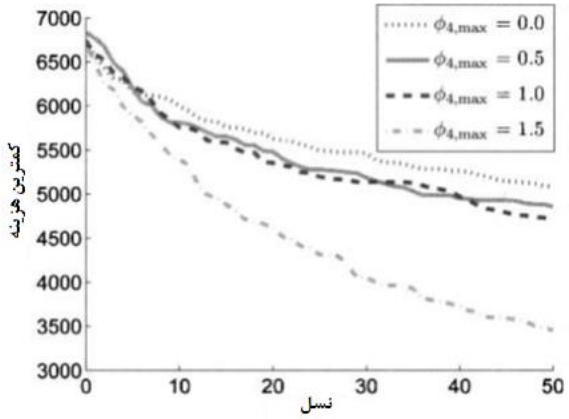
مثال ۱۱–۳

در این مثال ما از NPSO موجود در معادله‌ی (۱۱–۳۹) برای بهینه‌سازی تابع ۲۰ بعدی اشوفلِ ۲,۲٦ استفاده خواهیم نمود. در این مثال ما از اندازه‌ی جمعیتی برابر با ۲۰ استفاده کرده و پارامتر نخبه‌گرایی را برابر ۲ قرار می‌دهیم. همچنین از مقادیر نامی زیر استفاده خواهیم نمود

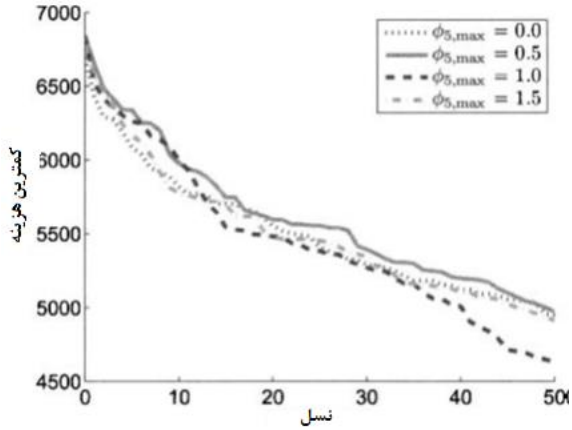
$$\begin{aligned} &\phi_{1,max} = \phi_{2,max} = \phi_{3,max} = 2 \\ &\phi_{4,max} = \phi_{5,max} = \phi_{6,max} = 0 \\ &K = \frac{2\alpha}{\phi_{1,max} + \phi_{2,max} + \phi_{3,max} - 2}, \quad \alpha = 0.9 \end{aligned} \qquad (۱۱–٤۰)$$

اشکال ۱۱–۱۲ تا ۱۱–۱٤ میانگین عملکرد NPSO را به ازای مقادیر مختلف $\phi_{4,\max}$، $\phi_{5,\max}$ و $\phi_{6,\max}$، هنگامی که سایر پارامترها دارای مقادیر نامی خود هستند نشان می‌دهد. شکل ۱۱–۱۲ نشان‌دهنده‌ی آن است که هنگامی که $\phi_{4,\max}$ (که میزان پرهیز ذره از بدترین مکان قبلی را نشان تعیین می‌کند) از مقدار نامی خود که برابر ۰ است بیشتر می‌شود، بهبود قابل توجهی در عملکرد الگوریتم حاصل می‌شود. شکل ۱۱–۱۳ نیز نتیجه‌ای مشابه را برای $\phi_{5,\max}$، که میزان پرهیز ذره از بدترین مکان فعلی موجود در میان همسایه‌های خود

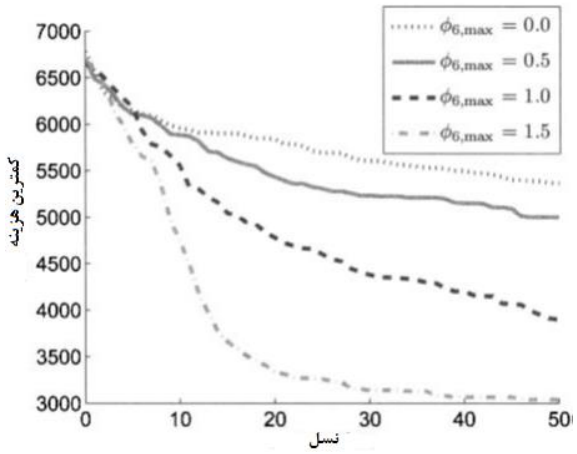
را تعیین می‌کند، نشان می‌دهد. در نهایت شکل ۱۱-۱٤ نشان‌دهنده‌ی آن است که با افزایش $\phi_{6,\max}$ از مقدار نامی خود، که برابر ۰ است، عملکرد الگوریتم بهبود می‌یابد. $\phi_{6,\max}$ میزان پرهیز ذره از بدترین مکان در میان مکان‌های قبلی کل جمعیت را تعیین می‌نماید. با توجه به این سه شکل می‌توان دریافت که $\phi_{6,\max}$ بیشترین تأثیر را بر روی عملکرد NPSO دارد.



شکل ۱۱-۱۲ مثال ۱۱-۳: عملکرد NPSO بر روی تابع ۲۰ بعدی اشوفل ۲,۲٦ به ازای مقادیر مختلف $\phi_{4,\max}$ که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است. ذراتی که از بدترین مکان قبلی خود پرهیز می‌نمایند بسیار بهتر از سایر ذرات عمل می‌نمایند.



شکل ۱۱-۱۳ مثال ۱۱-۳: عملکرد NPSO بر روی تابع ۲۰ بعدی اشوفل ۲,۲٦ به ازای مقادیر مختلف $\phi_{5,\max}$ که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است. ذراتی که از بدترین مکان فعلی در میان همسایگان خود پرهیز می‌نمایند بسیار بهتر از سایر ذرات عمل می‌نمایند.



شکل ۱۱-۱٤ مثال ۱۱-۳: عملکرد NPSO بر روی تابع ۲۰ بعدی اشوفل ۲,۲٦ به ازای مقادیر مختلف $\phi_{6,max}$ که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است. ذراتی که از بدترین مکان قبلی در میان تمامی جمعیت پرهیز می‌نمایند بسیار بهتر از سایر ذرات عمل می‌نمایند.

مثال ۱۱-۳ حاکی از آن است که NPSO می‌تواند بسیار بهتر از PSO استاندارد عمل نماید. توجه داشته باشید که در مثال ۱۱-۳ ما هر بار تنها یکی از عبارات تقویتِ منفی را تغییر دادیم و دو عبارت دیگر را برابر ۰ در نظر گرفتیم. همچنین توجه داشته باشید که می‌توان ایده‌ی تقویت منفی را با ایده‌ی PSO کاملاً آگاه ترکیب نمود. این کار را به عهده‌ی خواننده می‌گذاریم. در نهایت نیز متذکر می‌شویم که می‌توان نتایج پایداری در بخش ۱۱-۳ را برای NPSO نیز تکرار نمود. این کار نیز به عنوان فعالیتی برای تحقیقات آتی توسط خواننده در نظر گرفته شده است.

۱۱-۷ نتیجه‌گیری

می‌توان دید که PSO یک الگوریتم تکاملی مؤثر برای گستره‌ی وسیعی از مسائل می‌باشد. به همین دلیل هر الگوریتم تکاملی جدید باید با PSO مقایسه شود. همانند کلونی مورچگان، برخی از محققین PSO را در زمره‌ی الگوریتم‌های تکاملی قرار نداده و آن را نوعی هوش تجمعی به حساب می‌آورند. در PSO ذرات اطلاعات مربوط به راه‌حل‌های نامزد را مستقیماً با یکدیگر به اشتراک نمی‌گذارند. با این حال، PSO شامل انتخاب برازندگی- محور بوده و ذرات اطلاعات مربوط به سرعت‌ها را با یکدیگر به اشتراک گذاشته و سرعت به‌صورت مستقیم بر روی راه‌حل حاصله تأثیرگذار است. بنابراین، در این کتاب ما PSO را یک الگوریتم تکاملی به حساب می‌آوریم.

PSO گربه‌ماهی[1] اصلاحیه‌ای از PSO است که برای مبارزه با رکود در PSO معرفی شده است [یانگ و همکاران، ۲۰۱۱]. یک آکواریوم بزرگ را در نظر بگیرید که دارای تعداد زیادی ساردین است. ساردین‌ها اغلب به یک رفتار و مکان که بهینه‌ی محلی هستند خو می‌گیرند اما پس از مدتی بی‌حال و سست شده و سلامت خود را به سرعت از دست می‌دهند. اگر یک گربه‌ماهی به آکواریوم اضافه شود، ساردین‌ها دچار برانگیختگی شده و به حرکت افتاده و بدین ترتیب برای مدت زمان بیشتری سلامت خود را حفظ می‌نمایند. PSO نیز از این ایده بهره برده و جمعیت PSO را هنگامی که رکود رخ می‌دهد، برانگیخته می‌نماید. اگر در بهترین ذره‌ی موجود در جمعیت PSO برای m نسل متوالی (m معمولاً بین ۳ و ۷ است) بهبودی حاصل نشود، آنگاه هر متغیر مستقل از ۱۰٪ بدترین ذراتِ جمعیت، با یکی از حدود فضای جستجو برابر قرار داده می‌شود. دلیل قرار دادن ذرات در مرزهای فضای جستجو، ماکزیمم کردن فضای مورد کاوش است. همچنین، مسائل بهینه‌سازی مقید، معمولاً دارای راه‌حل‌هایی هستند که در مرز قید واقع شده‌اند [برناشتاین[2]، ۲۰۰۶].

تمامی تمرکز این فصل بر روی استفاده از PSO در مسائل با دامنه‌ی پیوسته قرار داشت. چندین تعمیم مختلف از PSO برای استفاده از آن در مسائل بهینه‌سازی ترکیبی ارائه شده است [کندی و ابرهارت، ۱۹۹۷]، [یوشیدا[3] و همکاران، ۲۰۰۱] و [کلرک، ۲۰۰٤]. سایر تحقیقات در این زمینه شامل ساده‌سازی PSO [پدرسون و چیپرفیلد[4]، ۲۰۱۰]، ترکیب آن با سایر الگوریتم‌های تکاملی [نیکنَم[5] و امیری[6]، ۲۰۱۰]، اضافه نمودن اپراتورهای جهش مانند برای جلوگیری از همگرایی نارس [ژینچاو[7]، ۲۰۱۰]، استفاده از چندین تجمع متعامل [چن[8] و مونتگومری[9]، ۲۰۱۱]، زدودن اتفاقی بودن از الگوریتم PSO [کلرک، ۱۹۹۹]، استفاده از توپولوژی‌های پویا و تطبیق‌پذیر [ریتشر[10] و همکاران، ۲۰۱۰]، کاوش استراتژی‌های مقداردهی اولیه [گوتیرز[11] و همکاران، ۲۰۰۲] و تطبیق پارامترهای PSO بدون تأخیر زمانی [ژان[12] و همکاران، ۲۰۰۹]، می‌شود. همچنین توجه داشته باشید که همان‌طور که می‌توان سرعت ذرات را در PSO مدل نمود، شتاب

[1] Catfish PSO
[2] Bernstein
[3] Yoshida
[4] Chipperfield
[5] Niknam
[6] Amiri
[7] Xinchao
[8] Chen
[9] Montgomery
[10] Ritscher
[11] Gutierrez
[12] Zhan

آن‌ها را نیز می‌توان مدل نمود [تریپاتی[1] و همکاران، ۲۰۰۷]. سایر تحقیقات آتی می‌توانند بر روی تحلیل رفتار و همگرایی تجمع ذرات، با در نظر گرفتن اتفاقی بودن الگوریتم و همچنین با در نظر گرفتن ارتباطات موجود میان ذرات، متمرکز شوند.

برای مطالعه‌ی بیشتر در زمینه‌ی PSO می‌توانید به کتاب‌های [کندی و ابرهارت، ۲۰۰۱]، [کلرک، ۲۰۰۶]، [سان و همکاران، ۲۰۰۱]، مقاله‌های [برتون[2] و کندی، ۲۰۰۷]، [بنکز و همکاران، ۲۰۰۷] و [بنکز و همکاران، ۲۰۰۸] و همچنین وب‌سایت‌های [*PSC*، ۲۰۱۲] و [کلرک، ۲۰۱۲*a*] مراجعه نمایید.

[1] Tripathi
[2] Bratton

مسائل

تمارین نوشتاری

۱۱-۱ تفاوت میان همسایگی ایستا و پویا در چیست؟ توضیح دهید.

۱۱-۲ شتاب در PSO:

الف) الگوریتم PSO موجود در شکل ۱۱-۱ را چگونه می‌توان اصلاح نمود تا شامل شتاب نیز بشود؟

ب) اگر اصلاحیه‌ی قسمت الف به الگوریتم PSO اعمال شود، معادله‌ی (۱۱-٤) چه تغییری خواهد کرد؟ در این صورت مقادیر ویژه چه خواهند بود؟

۱۱-۳ در معادله‌ی (۱۱-٤) فرض کنید $\phi_1 = 4$.

الف) مقادیر ویژه‌ی ماتریس چه خواهند بود؟

ب) آیا سیستم پایدار است؟

ج) شرایط اولیه و مقدار ورودی b_i را به‌گونه‌ای انتخاب نمایید که x_i و v_i با $t \to \infty$، دارای حد باشند.

د) شرایط اولیه و مقدار ورودی b_i را به‌گونه‌ای انتخاب نمایید که x_i و v_i با $t \to \infty$، دارای حد نباشند.

۱۱-٤ معادله‌ی (۱۱-۳۵) از هزینه و فاصله‌ی x_i برای محاسبه‌ی w_{ij} استفاده می‌نماید. چه ویژگی‌های دیگری از x_i را می‌توان برای محاسبه‌ی w_{ij} در نظر گرفت؟

۱۱-۵ با فرض ثابت بودن $p_i(t)$ در معادله‌ی (۱۱-۳۰)، معادلات پویای حالت- فضا را برای $x_i(t+1)$ و $v_i(t+1)$ بنویسید. مقادیر ویژه‌ی سیستم چه خواهند بود؟

۱۱-٦ تحت چه شرایطی معادلات (۱۱-۱۱) و (۱۱-۳۷) معادل خواهند بود؟

۱۱-۷ معادله‌ی به‌هم‌رسانی NPSO (۱۱-۳۹) را به‌گونه‌ای تعمیم دهین که معادله‌ی به‌هم‌سانی NPSO کاملاً آگاه حاصل شود.

۱۱-۸ معادله‌ی (۱۱-۲۵) مقدار زیر را برای ضریب انقباض پیشنهاد می‌کند:

$$K = \frac{2\alpha}{\phi_T - 2}$$

که در آن $\alpha \in (0,1)$ می‌باشد. ما می‌توانیم ϕ_T را برابر با جمع مقادیر بیشینه‌ی ϕ_iها در نظر بگیریم که در این صورت ϕ_T ثابت خواهد بود و یا آنکه می‌توانیم ϕ_T را برابر با جمع مقادیر ϕ_iها در نظر بگیریم، که در این صورت ϕ_T برای هر به‌هم‌رسانی سرعت مقداری متفاوت خواهد داشت. با فرض آنکه از معادله‌ی (۱۱-۲۷) برای به‌هم‌رسانی سرعت استفاده خواهیم نمود، این دو گزینه‌ی محتمل را می‌توان به‌صورت زیر نشان داد:

$$K_1 = \frac{2\alpha_1}{\phi_{1,max} + \phi_{2,max} + \phi_{3,max} - 2}$$

$$K_2 = \frac{2\alpha_2}{\phi_1 + \phi_2 + \phi_3 - 2}$$

که در آن هر ϕ_i دارای توزیعی یکنواخت بر روی $[0, \phi_{i,max}]$ خواهد بود. به ازای چه مقادیری از α_2، K_1 و K_2 به‌صورت میانگین برابر خواهند بود؟

تمارین کامپیوتری

۱۱–۹ اندازه‌ی همسایگی: الگوریتم PSO از شکل ۱۱–۱ را برای ٤٠ نسل جهت مینیمم‌سازی تابع ۱۰ بعدی کروی شبیه‌سازی نمایید (ضمیمه‌ی ج.۱–۱ را ببینید). اندازه‌ی جمعیت را برابر ۲۰ قرار داده و از معادله‌ی به‌هم‌رسانی سرعت کلی (۱۱–۲۷) استفاده نمایید. همچنین از مقادیر $\phi_{1,max} = \phi_{2,max} = \phi_{3,max} = 2$، $v_{max} = \infty$ و $\alpha = 0.9$ استفاده نمایید (مقدار K را از روی مقدار α تعیین نمایید). به ازای هر یک از اندازه‌های همسایگی 10 و $\sigma = 0,5$، ۲۰ شبیه‌سازی مونت کارلو انجام دهید. میانگین عملکرد هر یک از مجموعه شبیه‌سازی‌های مونت کارلو را به فرم تابعی از شماره‌ی نسل رسم نمایید. در مورد اهمیت همسایگی محلی در PSO چه نتیجه‌ای می‌گیرید؟

۱۱–۱۰ وزن‌دهی مسافت در تجمع ذرات کاملاً آگاه: معادله‌ی (۱۱–۳۵) را می‌توان به‌صورت زیر نوشت:

$$w_{ij} = w_{ij}(c) + Sw_{ij}(d)$$

که در آن $w_{ij}(c) = \max_k f(x_k) - f(x_j)$

$$w_{ij}(d) = \max_k |x_i - x_k| - |x_i - x_j|$$

$w_{ij}(c)$ و $w_{ij}(d)$ به ترتیب میزان دخالت هزینه و مسافت x_i در w_{ij} را مشخص می‌کنند. معادله‌ی بالا را می‌توان به‌صورت زیر تعمیم داد

$$w_{ij} = (w_{ij}(c) + DSw_{ij}(d))/(1 + D)$$

که در آن D میزان اهمیت دخالت مسافت نسبت به میزان دخالت هزینه می‌باشد. از این فرمول وزن‌دهی برای شبیه‌سازی PSO کاملاً آگاه جهیت بهینه‌سازی تابع ۲۰ بعدی رستریجین استفاده نمایید (ضمیمه‌ی ج.۱۱–۱ را ببینید). اندازه‌ی جمعیت را برابر ۲۰ قرار داده و محدودیت نسل را برابر ٤٠ در نظر بگیرید. به ازای هر یک از مقادیر 1000 و $D = 0, 0.5, 1, 2$، ۲۰ شبیه‌سازی مونت کارلو انجام دهید. در هر مورد، میانگین عملکرد الگوریتم را به‌صورت تابعی از شماره‌ی نسل رسم نمایید. در مورد نتایج به دست آمده کمی توضیح دهید.

۱۱-۱۱ اندازه‌ی همسایگی تجمع ذرات کاملاً آگاه: معادله‌ی به‌هم‌رسانی سرعت (۱۱-۳۷) را در یک شبیه‌سازی PSO جهت بهینه‌سازی تابع ۲۰ بعدی روزنبروک، پیاده‌سازی نمایید (ضمیمه‌ی ج.٤-۱ را ببینید). اندازه‌ی جمعیت را برابر ۲۰ در نظر گرفته و تعداد نسل‌ها را به ٤۰ محدود نمایید. K و ϕ_{max} را جهت عملکرد خوب تنظیم نمایید. به ازای هر یک از مقادیر همسایگی ۲، ۵، ۱۰ و ۲۰، ۲۰ شبیه‌سازی مونت کارلو انجام داده و میانگین عملکرد الگوریتم در هر مورد را به‌صورت تابعی از شماره‌ی نسل رسم نمایید. در مورد نتایج حاصله کمی توضیح دهید.

فصل دوازدهم

تکامل تفاضلی (دیفرانسیلی)

نسبت به بسیاری از الگوریتم‌های تکاملی دیگر، DE را می‌توان بسیار ساده و سرراست پیاده‌سازی نمود ... سادگی در برنامه‌نویسی برای متخصصین دیگر زمینه‌ها مهم می‌باشد، چرا که ممکن است آن‌ها تخصصی در برنامه‌نویسی نداشته باشند ...

س. داس[1]، پ. سوگانتان[2]، ک. کوئلو[3] کوئلو

تکامل تفاضلی (DE)[4] حدود سال ۱۹۹۵ توسط رینر استورن[5] و کنث و. پرایس[6] ایجاد شد. مانند بسیاری از الگوریتم‌های بهینه‌سازی، DE نیز از مسائل دنیای واقعی نشئت گرفته است: راه‌حل ضرایب چندجمله‌ای چبی‌چف و و بهینه‌سازی ضرایب فیلترهای دیجیتال. DE ورودی مؤثر و سریع به دنیای الگوریتم‌های تکاملی داشته به‌طوری که توانسته به‌عنوان یکی از بهترین‌های اولین و دومین مسابقات بین‌المللی محاسبات تکاملی انتخاب شود [استورن و پرایس، ۱۹۹٦]. اولین نگارش‌های DE به‌عنوان مقاله‌ی کنفرانس به چاپ رسید [استورن، ۱۹۹٦a]، [استورن، ۱۹۹٦b] و اولین مقاله‌ی ژورنال آن نیز یک سال بعد انتشار یافت [استورن و پرایس، ۱۹۹۷]. با این حال، اولین نگارش DE که به‌صورت گسترده‌ای مورد استقبال قرار گرفت در یک مجله‌ی بدون- داوری انتشار یافت [پرایس و استورن، ۱۹۹۷]. از آنجا که DE از زیست‌شناسی الهام نگرفته است، یک الگوریتم تکاملی منحصربه‌فرد به شمار می‌آید.

مروری بر فصل

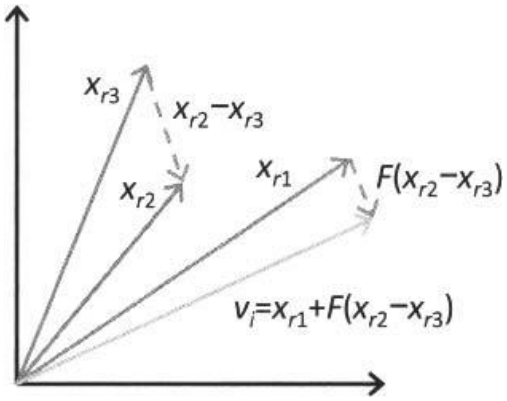
بخش ۱۲-۱ طرحی کلی از DE پایه‌ای برای بهینه‌سازی مسائل با دامنه‌ی پیوسته به دست می‌دهد. برخی از تنوعات DE که بعدها توسط محققین ارائه شدند، در بخش ۱۲-۲ مورد بحث واقع خواهند شد. پس از اثبات مؤثر بودن DE در مورد مسائل با دامنه‌ی پیوسته، محققین بر آن شدند تا آن را به مسائل با دامنه‌ی گسسته نیز تعمیم دهند. این تعمیم را در بخش ۱۲-۳ ارائه خواهیم نمود. همچنین، DE در ابتدا نه به‌صورت یک الگوریتم تکاملی مجزا بلکه به‌صورت گونه‌ای از الگوریتم ژنتیک ارائه شد، بنابراین بررسی آن از دریچه‌ی الگوریتم ژنتیک مفید خواهد بود. این کار را در بخش ۱۲-٤ انجام خواهیم داد.

---

[1] Das
[2] Suganthan
[3] Coello
[4] Differential Evolution (DE)
[5] Rainer Storn
[6] Kenneth V. Price

۱۲–۱ الگوریتم بنیادین تکامل تفاضلی

DE یک الگوریتم جمعیت– محور بوده که برای بهینه‌سازی توابع در یک دامنه‌ی پیوسته‌ی n بعدی طراحی شده است. بنابراین، هر ذره در جمعیت یک بردار n بعدی بوده و نماینده‌ی یک راه‌حل نامزد خواهد بود. ایده‌ی DE، گرفتن تفاضل برداری از دو ذره و اضافه نمودن نسخه‌ی مقیاس شده‌ی این تفاضل به ذره‌ی سوم برای به دست آوردن راه‌حل نامزد جدید می‌باشد. این فرآیند در شکل ۱۲–۱ نشان داده شده است.



شکل۱۲–۱ ایده‌ی اصلی تکامل تفاضلی که برای یک مسئله‌ی بهینه‌سازی دو بعدی ($n = 2$) نشان داده شده است. x_{r1}، x_{r2} و x_{r3} راه‌حل‌های نامزد می‌باشند. نسخه‌ای مقیاس شده از تفاضل دو بردار x_{r2} و x_{r3} به بردار x_{r1} اضافه شده تا بردار جهش‌یافته‌ی v_i ایجاد شود. این بردار، راه‌حل نامزد جدید است. توجه داشته باشید که v_i با اندیس i نشان داده شده است چرا که ما در هر نسل n بردار جهش‌یافته‌ی مجزا تولید می‌نماییم، که در آن n اندازه‌ی جمعیت است.

شکل ۱۲–۱ DE را در یک فضای جستجوی ۲ بعدی به تصویر می‌کشد. دو ذره‌ی x_{r2} و x_{r3} به‌صورت اتفاقی انتخاب می‌شوند ($r_2 \neq r_3$). نسخه‌ی مقیاس شده‌ای از تفاوت میان این دو ذره به ذره‌ی سومی، که آن هم به‌صورت اتفاقی انتخاب شده است، اضافه می‌گردد. در شکل بالا ذره‌ی سوم x_{r1} می‌باشد ($r_1 \neq r_2, r_3$). این فرآیند باعث ایجاد ذره‌ای جهش‌یافته مانند v_i خواهد شد که ممکن است به‌عنوان یک راه‌حل نامزد جدید در جمعیت پذیرفته شود.

پس از آنکه بردار جهش‌یافته‌ی v_i ایجاد شد، با یک ذره‌ی دیگر در DE مانند x_i که $i \neq r_1, r_2, r_3$ ترکیب شده (عمل برش) و یک بردار آزمایش u_i را ایجاد می‌کند. عمل برش را به فرم ریاضی می‌توان به‌صورت زیر نوشت:

$$u_{ij} = \begin{cases} v_{ij} & \text{اگر } (r_{cj} < c) \text{ یا } (j = \jmath_r) \\ x_{ij} & \text{در غیر این صورت} \end{cases} \qquad (۱–۱۲)$$

که در آن $j \in [1,n]$ بوده و n برابر با ابعاد مسئله و همچنین ابعاد u_i، v_i و x_i می‌باشد. بدین ترتیب u_{ij}، jاُمین عنصر u_i بوده؛ v_{ij}، jاُمین عنصر v_i بوده و x_{ij} نیز jاُمین عنصر x_i می‌باشد. r_{cj} نیز یک عدد اتفاقی با توزیع یکنواخت بر روی $[0,1]$ می‌باشد، c ثابت نرخ برش بوده و به بازه‌ی $[0,1]$ تعلق دارد. در نهایت J_r یک عدد صحیح اتفاقی است که از بازه‌ی $[0,n]$ گرفته شده است. می‌توان دید که بردار آزمایش u_i ترکیبی عنصر- به- عنصر از ذره‌ی حال حاضرِ x_i در DE و بردار جهش‌یافته‌ی v_i می‌باشد. هدف J_r، حصول اطمینان از این موضوع است که u_i کلونی از x_i نخواهد بود. در بسیاری از مسائل می‌توان از J_r جهت سادگی صرف نظر نمود (مسئله‌ی ۱۲-۳ را ببینید). نرخ برش c احتمال گرفته شدن هر عنصر u_i از بردار جهش‌یافته‌ی v_i را تعیین می‌کند.

پس از آنکه N بردار آزمایش ایجاد شد (N اندازه‌ی جمعیت است)، بردارهای u_i و x_i با یکدیگر مقایسه می‌شوند. برازنده‌ترین بردار در هر زوج (x_i, v_i) جهت نسل بعدیِ DE حفظ شده و نابرازنده‌ترین نیز حذف می‌شود. الگوریتم بنیادین DE برای یک مسئله‌ی n بعدی در شکل ۱۲-۲ خلاصه شده است.

$F =$ اندازه‌ی گام $\in [0.4,0.9]$

$c =$ نرخ برش $\in [0.1,1]$

جمعیتی از راه‌حل‌های نامزد را مقداردهی کن، $\{x_i\}$ برای $i \in [1,N]$

تا زمانی که شرایط توقف برآورده نشده است

برای هر ذره x_i، $i \in [1,N]$

$r_1 \leftarrow$ عدد صحیح اتفاقی $\in [1,N] : r_1 \neq i$

$r_2 \leftarrow$ عدد صحیح اتفاقی $\in [1,N] : r_1 \neq \{i, r_1\}$

$r_3 \leftarrow$ عدد صحیح اتفاقی $\in [1,N] : r_1 \neq \{i, r_1, r_2\}$

$v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3})$ (بردار جهش)

$\mathcal{J}_r \leftarrow$ عدد صحیح اتفاقی $\in [1,n]$

برای هر بعد $j \in [1,n]$

$r_{cj} \leftarrow$ عدد اتفاقی $\in [0,1]$

اگر $(r_{cj} < c)$ و یا $(j = \mathcal{J}_r)$، آنگاه

$u_{ij} \leftarrow v_{ij}$

در غیر این صورت

$u_{ij} \leftarrow x_{ij}$

پایان اگر

بعد بعدی

برای هر اندیس جمعیت $i \in [1,N]$

اگر $f(u_i) < f(x_i)$ آنگاه $x_i \leftarrow u_i$

اندیس جمعیت بعدی

نسل بعدی

شکل ۱۲–۲ یک الگوریتم ساده‌ی تکامل تفاضلی برای مینیمم‌سازی تابع n بعدی $f(x)$. این الگوریتم DE کلاسیک و یا DE/rand/1/bin نام دارد.

همان‌طور که از شکل ۱۲–۲ نیز پیداست، DE دارای پارامترهای زیادی است که نیاز به میزان‌سازی دارند. مانند تمام الگوریتم‌های تکاملی دیگر، اندازه‌ی جمعیت باید انتخاب شود. پارامترهای مختص DE شامل

اندازه‌ی گام F، که با نام فاکتور مقیاس نیز شناخته می‌شود، و نرخ برش c می‌شود. این پارامترها به مشخصات مسئله وابسته هستند اما معمولاً (نه همیشه) از بازه‌های $F \in [0.4,0.9]$ و $c \in [0.1,1]$ انتخاب می‌شوند. مقدار بهینه‌ی F با مربع اندازه‌ی جمعیت N رابطه‌ی عکس دارد. مقدار بهینه‌ی c نیز با تفکیک‌پذیری تابع هدف رابطه‌ی عکس دارد [پرایس، ۲۰۱۳].

الگوریتم شکل ۱۲–۲ معمولاً با نام DE کلاسیک نیز شناخته می‌شود. همچنین این الگوریتم گاهاً الگوریتم DE/rand/1/bin نیز خوانده می‌شود چرا که بردار پایه، x_{r1} به‌صورت اتفاقی انتخاب شده (rand)؛ یک بردار تفاضل که همان $F(x_{r2} - x_{r3})$ می‌باشد به x_{r1} اضافه شده (1)؛ و تعداد عناصر بردار جهش‌یافته که در تشکیل بردار آزمایش دخالت دارد به‌صورت کاملاً نزدیک از توزیع دوجمله‌ای پیروی می‌کند (bin).

اگر کمی در مورد الگوریتم DE کلاسیک از شکل ۱۲–۲ تفکر نماییم، علت کارکرد آن را در خواهیم یافت [پرایس، ۲۰۱۳]. اول آنکه، اختلالات وآشفتگی‌هایی که دارای فرم و شکل $(x_{r2} - x_{r3})$ می‌باشند، با نزدیک شدن جمعیت به راه‌حل مسئله کاهش می‌یابند. دوم آنکه، اندازه‌ی این آشفتگی‌ها با توجه به مقیاس مسئله از یک بعد به بعد دیگر متفاوت خواهد بود. این بدین معنی است که اندازه‌ی pاُمین عنصرِ $(x_{r2} - x_{r3})$ به میزان نزدیکی جمعیت به راه‌حل مسئله در طول بعد pاُم بستگی دارد. سوم آنکه، گام‌های آشفتگی میان ابعاد دارای همبستگی بوده و این موضوع باعث می‌شود جستجو حتی برای مسائل با درجه‌ی تفکیک‌ناپذیری بالا مؤثر واقع شود (ضمیمه‌ی ج.۲–۷ را ببینید). این خاصیت DE، تطبیق شکل[1] نام دارد و بدین معنی است که جمعیت DE خود را در طول شکلِ تابع هدف توزیع کرده و سعی می‌کند خود را با شکل تابع هدف تطبیق دهد.

۱۲–۲ انواع تکامل تفاضلی

در این بخش به برخی از انواع DE نگاهی می‌اندازیم. بخش ۱۲–۲–۱ راه‌هایی مختلف برای تولید بردار آزمایش را نشان می‌دهد، بخش ۱۲–۲–۲ برخی راه‌های متنوع برای ایجاد بردار جهش‌یافته‌ی v را نشان داده و بخش ۱۲–۲–۳ نیز برخی گزینه‌های محتمل برای استفاده از فاکتور مقیاس اتفاقی F را مورد بحث قرار خواهد داد.

[1] Contour Matching

۱۲-۲-۱ بردارهای آزمایش

توجه داشته باشید که روش موجود در شکل ۱۲-۲ هیچ مکانیزمی برای با هم نگه داشتن ویژگی‌های راه‌حل از v_i به x_i ارائه نمی‌دهد. این بدین معنی است که احتمال کپی شدن v_{ij} به u_{ij}، چه $v_{i,j-1}$ به $u_{i,j-1}$ کپی شده باشد و چه نشده باشد، یکی خواهد بود. با این حال، در بسیاری از مسائل برازندگی به ترکیبات ویژگی‌های راه‌حل بستگی داشته و به همین دلیل مطلوب است ویژگی‌های راه‌حل را در کنار هم نگه داشت. DE/rand/1/L بدین ترتیب عمل می‌کند که ابتدا یک عدد اتفاقی از بازه‌ی $[1,n]$ که با حرف L نشان می‌دهیم تولید شده و سپس L ویژگی متوالی از v_i به u_i کپی شده و باقی ویژگی‌ها نیز از x_i به u_i کپی می‌شود [استورن و پرایس، ۱۹۹۶].

برای مثال، فرض کنید یک مسئله‌ی ۷ بعدی در اختیار داریم ($n=7$). عملکرد الگوریتم DE/rand/1/L بدین ترتیب است که ابتدا یک عددی اتفاقی $L \in [1,n]$ تولید می‌شود. فرض کنید L به‌صورت اتفاقی برابر ۳ انتخاب شده است. سپس یک نقطه‌ی آغازین $s \in [1,n]$ به‌صورت اتفاقی تولید می‌شود. فرض کنید s به‌صورت اتفاقی برابر ٦ انتخاب شده است. با توجه به این اطلاعات داده شده، ویژگی‌های راه‌حل از v_i و x_i به‌صورت زیر به u_i کپی می‌شوند:

$$
\begin{aligned}
&u_{i1} \leftarrow v_{i1}\\
&u_{i2} \leftarrow x_{i2}\\
&u_{i3} \leftarrow x_{i3}\\
&u_{i4} \leftarrow x_{i4}\\
&u_{i5} \leftarrow x_{i5} \text{ (نقطه انتهایی)}\\
&u_{i6} \leftarrow v_{i6} \text{ (نقطه ابتدایی } s)\\
&u_{i7} \leftarrow v_{i7}
\end{aligned}
\qquad (۲-۱۲)
$$

می‌توان دید که کپی شدن عناصر v_i به u_i از بعد ششم آغاز می‌شود (ششمین ویژگی راه‌حل). از آنجا که $L=3$ است، سه عنصر متوالی از نقطه‌ی آغازین، از v_i به u_i کپی می‌شود. توجه داشته باشید که متوالی بدین معنی است که اگر در حین عملیات کپی کردن به آخرین ویژگی راه‌حل (در اینجا آخرین ویژگی راه‌حل ویژگی هفتم است) رسیدیم، به ویژگی اول رفته و عملیات کپی کردن را از آنجا ادامه می‌دهیم. پس از کپی کردن ۳ عنصر از v_i به u_i، عناصر x_i را به u_i می‌نماییم تا u_i کاملاً تعریف شده باشد. در نهایت، DE/rand/1/L حلقه‌ی "For each dimension" در شکل ۱۲-۲ را با حلقه‌ی شکل ۱۲-۳ جایگزین می‌نماید.

$L \leftarrow$ عدد اتفاقی $\in [1, n]$

$s \leftarrow$ عدد اتفاقی $\in [1, n]$

$J \leftarrow \{s, \min(n, s + L - 1)\} \cup \{1, s + L - n - 1\}$

برای هر بعد $j \in [1, n]$

اگر $j \in J$

$u_{ij} \leftarrow v_{ij}$

در غیر این صورت

$u_{ij} \leftarrow x_{ij}$

پایان اگر

بُعد بعدی

شکل ۱۲–۳ حلقه‌ی DE/rand/1/L که عناصر را از x_i و v_i به بردارِ آزمایش u_i کپی می‌نماید. تابع $a \bmod b$ باقی‌مانده‌ی تقسیم a/b را بر می‌گرداند. این حلقه جایگزین حلقه‌ی "for each dimension" در شکل ۱۲–۲ می‌شود.

در نظر گرفتن میانگینِ تعداد عناصرِ بردار جهش (v_{ij}) که به ازای یک i مشخص به ویژگی‌های بردار آزمایش (u_{ij}) کپی می‌شوند خالی از لطف نیست. در الگوریتم DE/rand/1/bin از شکل ۱۲–۲، حلقه‌ی "for each dimension" شامل n دوره می‌شود. در یکی از این حلقه‌ها احتمال کپی شدن v_{ij} به u_{ij}، ۱۰٪ بوده و در باقی $n - 1$ حلقه، این احتمال برابر c می‌باشد. این بدین معنی است که تعداد عناصرِ مورد انتظار کپی شده از v_{ij} به بردار آزمایش برابرست با

$$E\left(\text{تعداد عناصر کپی شده } v_i\right) = 1 + c(n-1) \quad \text{برای DE/rand/1/bin} \qquad (۱۲–۳)$$

در الگوریتم DE/rand/1/L، L ویژگی از بردار جهش‌یافته (v_{ij}) به بردار آزمایش کپی می‌شود. از آنجا که L دارای توزیع یکنواخت بر روی $[1, n]$ می‌باشد:

$$E\left(\text{تعداد عناصر کپی شده } v_i\right) = n/2 \quad \text{برای DE/rand/1/bin} \qquad (۱۲–٤)$$

حال می‌توان این سؤال را مطرح نمود: تحت چه شرایطی تعدادِ مورد انتظارِ عناصر کپی شده از بردار جهش‌یافته به بردار آزمایش برای دو الگوریتم DE/rand/1/bin و De/rand/1/L یکسان خواهد بود؟ با مساوی قرار دادن معادلات (۱۲–۳) و (۱۲–٤) خواهیم داشت

$$c = \frac{n-2}{2(n-1)} \qquad (۱۲–۵)$$

همان‌طور که به یاد دارید، c پارامتر برش در الگوریتم De/rand/1/bin از شکل ۱۲-۲ می‌باشد و معمولاً دارای مقداری کمی کوچکتر از ۰٫۵ می‌باشد. بنابراین اگر پارامتر برش در الگوریتم DE/rand/1/bin برابر با معادله‌ی ۱۲-۵ انتخاب شود، تعداد عناصر کپی شده از بردار جهش‌یافته به بردار آزمایش با تعداد عناصر کپی شده‌ی بردار جهش‌یافته به بردار آزمایش در الگوریتم DE/rand/1/L از شکل ۱۲-۳ برابر خواهد بود.

به‌طور کلی، ما می‌توانیم L را عددی صحیح و اتفاقی در بازه‌ی $[1, L_{max}]$ در نظر بگیریم که در این صورت L_{max} ثابتی است که توسط کاربر تعیین خواهد شد. در شکل ۱۲-۳، $L_{max} = n$ می‌باشد، هرچند مقادیر کوچکتر از n برای L_{max} می‌توانند عملکرد بهتری را نتیجه دهند.

مثال ۱۲-۱

در این مثال، DE را به تابع ۲۰ بعدی آکلی که در ضمیمه‌ی ج.۲-۱ تعریف شده است اعمال می‌نماییم. برای این منظور از پارامترهای زیر استفاده می‌نماییم:

- اندازه‌ی جمعیت: ۵۰.
- اندازه‌ی گام $F = 0.4$.
- نرخ برش $c = 0.49$ که از معادله‌ی (۱۲-۵) به دست آمده است.

در این مثال، به تفاوت میان تولید بردار آزمایش با استفاده از گزینه‌های bin از شکل ۱۲-۲ و L از شکل ۱۲-۳ توجه خواهیم نمود. شکل ۱۲-٤ بهترین ذره در هر نسل را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. می‌توان دید که گزینه‌ی L در ابتدای شبیه‌سازی بسیار سریع همگرا شده اما گزینه‌ی bin عملکرد بسیار بهتری را در طولانی مدت از خود نشان می‌دهد. ما نباید انتظار هیچ گونه بهبودی از استفاده از گزینه‌ی L داشته باشیم چرا که ویژگی‌های راه‌حل در تابع آکلی دارای هیچ گونه کوپل‌شدگی و ارتباطی نمی‌باشند. این بدین معناست که تابع آکلی یک مسئله‌ی تفکیک‌پذیر است. با این حال، دقیقاً معلوم نیست که چرا گزینه‌ی bin بسیار بهتر از گزینه‌ی L می‌نماید.

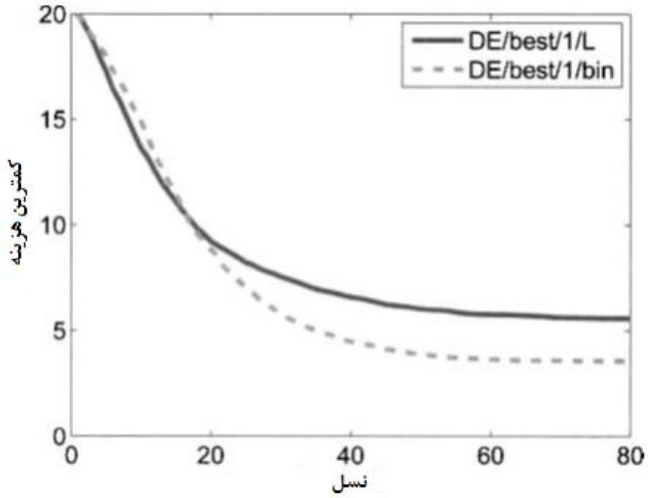
۱۲-۲-۲ بردارهای جهش‌یافته

در این بخش به برخی گزینه‌های محتمل برای ایجاد بردار جهش‌یافته نگاهی خواهیم داشت. برای مثال، به جای انتخاب بردار پایه‌ی x_{r1} به‌صورت اتفاقی، ممکن است بهتر باشد بردار پایه را برابر با بهترین ذره‌ی موجود در جمعیت در نظر بگیریم. در این صورت تمام بردارهای آزمایش u_i از جهشی از بردار پایه تشکیل خواهند شد. این شیوه، DE/best/1/bin نام دارد [استورن و پرایس، ۱۹۹٦]، [استورن، ۱۹۹٦b]. این الگوریتم

مانند همان الگوریتم شکل ۱۲–۲ خواهد بود تنها با این تفاوت که محاسباتِ بردار جهش‌یافته با معادلات زیر جایگزین خواهد شد

$$v_i \leftarrow x_b + F(x_{r2} - x_{r3}) \qquad (۱۲–۶)$$

که در آن x_b بهترین ذره در جمعیت می‌باشد. این کار باعث تقویت بهره‌وری و تضعیف کاوش می‌شود. این ایده مشابه الگوریتم تکاملی stud، که در بخش ۸–۷–۷ مورد بحث واقع شده است، می‌باشد.



شکل ۱۲–٤ مثال ۱۲–۱: عملکرد DE بر روی تابع ۲۰ بعدی آکلی در مثال ۱۲–۱. منحنی‌ها هزینه‌ی بهترین ذره در هر نسل را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. گزینه‌ی bin برای تولید تابع آزمایش بسیار بهتر از گزینه‌ی *L* عمل می‌نماید.

اگر از معادله‌ی (۱۲–٦) برای تولید بردار جهش‌یافته و از شکل ۱۲–۳ برای کپی نمودن ویژگی‌ها به بردار آزمایش استفاده نماییم، الگوریتم DE/best/1/L به دست خواهد آمد.

یکی دیگر از گزینه‌های پیش رو استفاده از دو بردار تفاضل برای تولید بردار جهش‌یافته می‌باشد [استورن و پرایس، ۱۹۹٦]، [استورن، ۱۹۹٦b]. این کار باعث افزایش کاوش می‌شود چرا که در این صورت جهت بردار تفاضل کلی به جهتِ بردار تفاضل میان دو بردار محدود نخواهد بود. به معنای دیگر، بردار تفاضل کلی از درجه‌ی آزادی بیشتری برخوردار خواهد بود. این ایده را می‌توان با انتخاب اتفاقی بردار پایه در هر حلقه در شکل ۱۲–۲ و یا با انتخاب بهترین ذره به‌عنوان بردار پایه در شکل ۱۲–٦ ترکیب نمود. این کار، دو گزینه‌ی زیر را برای تولید بردار جهش‌یافته نتیجه خواهد داد:

$$r_4 \leftarrow \text{عدد اتفاقی } \in [1,N] : \ r_4 \notin \{i, r_1, r_2, r_3\}$$
$$r_5 \leftarrow \text{عدد اتفاقی } \in [1,N] : \ r_4 \notin \{i, r_1, r_2, r_3, r_4\} \qquad (۷-۱۲)$$
$$v_i \leftarrow \begin{cases} x_{r1} + F(x_{r2} - x_{r3} + x_{r4} - x_{r5}) & DE/rand/2/? \\ x_b + F(x_{r2} - x_{r3} + x_{r4} - x_{r5}) & DE/rand/2/? \end{cases}$$

حال به توضیح علامت سؤال‌های موجود در معادله‌ی بالا می‌پردازیم. اگر از یکی از دو گزینه‌ی بالا برای تولید بردار جهش‌یافته و از شکل ۱۲-۲ برای کپی نمودن ویژگی‌ها به بردار آزمایش استفاده نماییم، یکی از دو الگوریتم DE/rand/2/bin یا DE/best/2/bin حاصل خواهد شد. از سوی دیگر از از معادله‌ی بالا برای تولید بردار جهش‌یافته و از شکل ۱۲-۳ برای تولید بردار آزمایش استفاده نماییم، یکی از دو الگوریتم DE/rand/2/L یا DE/best/2/L حاصل خواهد شد.

توجه داشته باشید که معادله‌ی (۱۲-۷) تأثیر بردار تفاضل بر روی بردار جهش‌یافته را افزایش می‌دهد. اگر در شکل ۱۲-۲ یا معادله‌ی (۱۲-٦) از $F = F_0$ استفاده شود، آنگاه برای دستیابی به یک مقایسه‌ی عادلانه باید از مقدار $F < F_0$ در معادله‌ی (۱۲-۷) استفاده گردد. رابطه‌ی دقیق میان دو مقدار F به شکل تابع هدف بستگی دارد.

DE را همچنین می‌توان با در نظر گرفتن x_i به‌عنوان بردار پایه پیاده‌سازی نمود [استورن، ۱۹۹٦a]. برای مثال می‌توان از معادله‌ی زیر استفاده نمود

$$v_i \leftarrow x_i + F\Delta x \qquad (۸-۱۲)$$

که در آن Δx بردار تفاضل است. بسته به روش تولید بردار تفاضل و همچنین بردار آزمایش، یکی از چهار الگوریتم DE/target/1/bin، DE/target/2/bin، DE/target/1/L و یا DE/target/2/L حاصل خواهد شد[۱]. بر خلاف الگوریتم DE/rand از بخش ۱۲-۱ و ۱۲-۲-۱، الگوریتم DE/target حساسیت بسیار کمتری نسبت به مقدار F خواهد داشت [پرایس، ۲۰۱۳].

یکی دیگر از روش‌های ممکن برای تولید بردار تفاضل، استفاده از بهترین ذره‌ی موجود در جمعیت (x_b) می‌باشد. این کار باعث ایجاد بردارهای جهش‌یافته‌ای خواهد شد که همگی تمایل به حرکت به سوی x_b خواهند داشت. برداری که از x_b کم خواهد شد می‌تواند یک ذره‌ی اتفاقی و ذره‌ی پایه باشد. در کل امکانات زیادی را با توجه به این ایده می‌توان در نظر گرفت. برای مثال می‌توان از معادله‌ی زیر استفاده نمود [استورن، ۱۹۹٦a]،

[۱] الگوریتم DE/target گاهاً با عنوان DE/current و یا DE/i نیز شناخته می‌شود.

$$\begin{aligned} v_i &\leftarrow x_i + F(x_b - x_i) \\ v_i &\leftarrow x_{r1} + F(x_b - x_{r3}) \\ v_i &\leftarrow x_b + F(x_{r2} - x_{r3} + x_b - x_{r5}) \\ v_i &\leftarrow x_i + F(x_b - x_i + x_{r2} - x_{r3}) \end{aligned} \qquad (۹-۱۲)$$

علاوه بر معادله‌ی بالا می‌توان از معادلات بسیار دیگری نیز استفاده نمود. اگر از آخرین معادله از مجموعه‌ی معادلات بالا برای تولید v_i استفاده شود، الگوریتم به دست آمده، الگوریتم DE/target-to-base/1/1bin نامیده خواهد شد [پرایس و همکاران، ۲۰۰۵، بخش ۳-۳-۱][1]. توجه داشته باشید که گاهی برای تولید v_i از بازترکیب، گاهی از جهش و گاهی از هر دو استفاده می‌نماییم. گزینه‌ی اول در مجموعه معادلات (۱۲-۹) یک عمل بازترکیب است چرا که شامل ترکیب نمودن x_i با یک بردار دیگر می‌شود. گزینه‌های دوم و سوم هر دو عمل جهش بوده چرا که x_i در معادلات ظاهر نمی‌شود. گزینه‌ی چهارم نیز یک عمل ترمیمی است چرا که هم شامل x_i بوده و هم شامل بردار تفاضل $(x_{r2} - x_{r3})$ می‌شود.

ما همچنین می‌توانیم با تصمیم‌گیری اتفاقی در مورد چگونگی تولید بردار جهش، روش‌های مختلف را با هم ترکیب نماییم. برای مثال، شکل ۱۲-۵ روشی را برای تولید بردار جهش نشان می‌دهد که باعث ایجاد الگوریتم DE/rand/1/either-or می‌شود [پرایس و همکاران، ۲۰۰۵، بخش ۲-۶-۵]. اگر $a < p_f$ باشد، آنگاه از الگوریتم استاندارد DE/rand/1/bin برای تولید بردار v استفاده خواهد شد. اما اگر $a \geq p_f$ باشد، یک حالت خاص از الگوریتم DE/rand/2 برای تولید بردار v مورد استفاده قرار خواهد گرفت.

تا به اینجا تعداد زیادی از جایگشت‌های DE به دست آمده است. با این حال بسیاری از این موارد دارای اهمیت ثانوی هستند. ایده‌ی کلی DE در شکل‌های ۱۲-۱ و ۱۲-۲ به تصویر کشیده شده است و تمامی تنوعات ذکر شده تاکنون تنها جزئیات هستند.

[1] به نظر می‌رسد نام‌گذاری درست برای این الگوریتم، DE/target-to-base/2/bi باشد، اما از سویی این معادله را می‌توان به‌صورت اضافه نمودن یک بردار جهش واحد به بردار پایه‌ی x_i در نظر گرفت. در این صورت نام‌گذاری DE/target-to-base/1/bin چندان بیراه نخواهد بود.

$p_f =$ احتمال جهش $\in [0,1]$

$a \leftarrow$ عدد اتفاقی $\in [0,1]$

اگر $a < p_f$ آنگاه

$v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3})$

در غیر این صورت

$v_i \leftarrow x_{r1} + K(x_{r2} - x_{r1} + x_{r3} - x_{r1})$

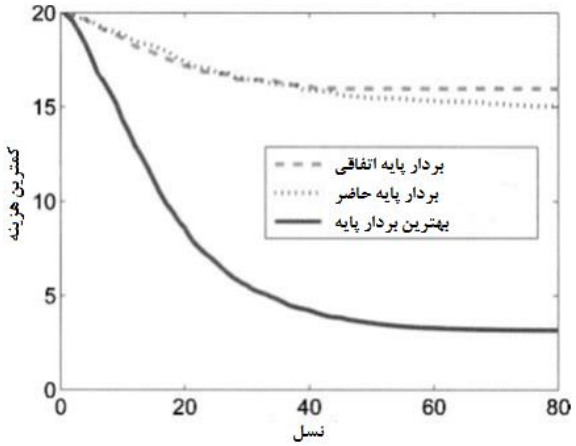
شکل ۱۲–۵ روشی برای تولید بردار جهش که باعث ایجاد الگوریتم DE/rand/1/either-or می‌شود. به‌صورت کلی، $k = (F + 1)/2$ نتایج خوبی را در مورد توابع محک به دست می‌دهد.

مثال ۱۲–۲

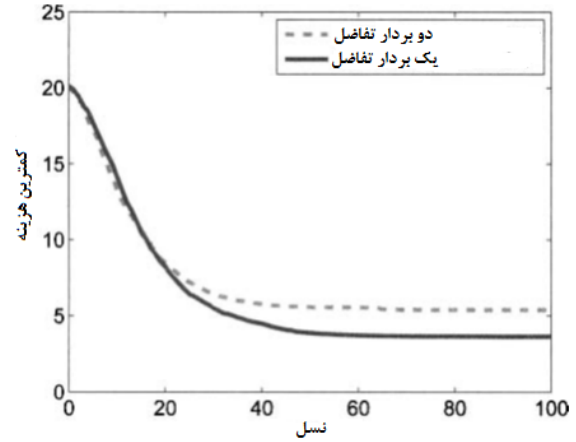
در این مثال برای بار دیگر DE را به تابع ۲۰ بعدی اَکلی که در ضمیمه‌ی ج.۲–۱ توصیف شده است اعمال می‌نماییم. از آنجا که نتایج حاصل از مثال ۱۲–۱ از بهتر بودن عملکرد گزینه‌ی bin نسبت به L حکایت داشت، در این مثال از گزینه‌ی bin استفاده می‌نماییم. در این مثال به مطالعه‌ی نحوه‌ی تغییر عملکرد بسته به بردارِ پایه‌ی مورد استفاده می‌پردازیم. ما سه گزینه در اختیار داریم. گزینه‌ی اول استفاده از بردار اتفاقی x_{r1} به‌عنوان بردار پایه بوده (شکل ۱۲–۲)، گزینه‌ی دوم استفاده از بهترین بردار به‌عنوان بردار پایه بوده (معادله‌ی (۱۲–٦)) و گزینه‌ی سوم، استفاده از عضو حال حاضر جمعیت به‌عنوان بردار پایه می‌باشد (معادله‌ی (۱۲–۸)). شکل ۱۲–٦ بهترین ذره‌ی هر نسل را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است نشان می‌دهد. می‌توان دید که گزینه‌های اول و سوم عملکرد تقریبا یکسانی دارند. این نتیجه کاملاً مورد انتظار است چرا که: (۱) گزینه‌ی سوم باعث می‌شود هر ذره از جمعیت حال حاضر دقیقا یک بار در هر نسل به‌عنوان تابع پایه استفاده شود و (۲) گزینه‌ی اول باعث می‌شود هر ذره در هر نسل به طور میانگین یک بار به‌عنوان بردار پایه استفاده شود. از سوی دیگر، شکل ۱۲–٦ نشان می‌دهد بهترین گزینه، کاملاً دو گزینه‌ی دیگر را کنار می‌زند و پیشی می‌گیرد. کاملاً واضح است که متمرکز کردن جستجو در اطراف بهترین ذره‌ی هر نسل استراتژی مفیدی است (گزینه‌ی دوم).

از آنجا که استفاده از بهترین ذره به‌عنوان بردار پایه بهترین عملکرد را به دست می‌دهد، از این گزینه برای باقی این مثال استفاده خواهیم نمود. برای قسمت آخر شبیه‌سازی در این مثال، نگاهی به نحوه‌ی تغییر عملکرد با تغییر تعداد بردارهای مورد استفاده برای تولید بردار جهش خواهیم داشت. می‌توان مانند معادله‌ی (۱۲–٦) از یک بردار تفاضل و یا مانند معادله‌ی (۱۲–۷) از دو بردار تفاضل استفاده نمود. شکل ۱۲–۷ بهترین ذره‌ی هر نسل را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است نشان می‌دهد. می‌توان دید که

استفاده از یک بردار تفاضل کمی بهتر از استفاده از دو بردار تفاضل می‌باشد، حتی اگر مقدار F برای تأثیر استفاده از دو بردار تفاضل اصلاح گردد. علت این پدیده دقیقاً مشخص نیست اما با نتایج به دست آمده در [پرایس و همکاران، ۲۰۰۵، بخش ۲-٤-۷] همخوانی دارد. می‌توانید این پدیده را به‌عنوان موضوعی برای تحقیقات آتی در نظر بگیرید.



شکل ۱۲-٦ مثال ۱۲-۲: عملکرد DE بر روی تابع ۲۰ بعدی آکلی. منحنی‌ها هزینه‌ی بهترین ذره در هر نسل را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده‌اند نشان می‌دهند. استفاده از بهترین ذره به‌عنوان بردار پایه عملکرد بسیار بهتری را به دست می‌دهد.



شکل ۱۲-۷ مثال ۱۲-۲: عملکرد DE بر روی تابع ۲۰ بعدی آکلی از مثال ۱۲-۲. منحنی‌ها هزینه‌ی بهترین ذره در هر نسل را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده‌اند نشان می‌دهند. استفاده از تنها یک بردار تفاضل برای تولید بردار جهش کمی بهتر از استفاده از دو بردار تفاضل می‌باشد.

ما همچنین می‌توانیم از دیگر گزینه‌های موجود برای الگوریتم‌های تکاملی برای پیاده‌سازی در DE استفاده نماییم. برای مثال، می‌توان یک عمل جهش استانداردتر مانند جهش گاوسی یا یکنواحت متمرکز در راه‌حل نامزد را به الگوریتم DE اضافه نماییم (بخش ۸-۹ را ببینید). با این حال، از آنجا که بردارِ جهش DE خود بسیار کاوش‌گر است، استفاده از گزینه‌ی جهش شاید تأثیر چندانی بر عملکرد DE نداشته باشد.

به یاد آورید که نخبه‌گرایی یک ویژگی معمول در الگوریتم‌های تکاملی بوده که باعث می‌شود ذرات با عملکرد بسیار خوب را از دست ندهیم و همچنین ضمانت می‌کند عملکرد بهترین ذره از یک نسل به نسل دیگر بدتر نمی‌شود (بخش ۸-٤ را ببینید). نخبه‌گرایی یک گزینه‌ی جذاب برای تمامی الگوریتم‌های تکاملی بوده و معمولاً بهبودی عظیم در عملکرد الگوریتم ایجاد می‌نماید. با این حال، در DE نیازی به پیاده‌سازی DE نمی‌باشد، چرا که همان‌طور که از حلقه‌ی "for each population index" در شکل ۱۲-۲ نیز دیده می‌شود، DE به‌طور خودکار بهترین ذره در هر نسل را ذخیره می‌نماید. اما این موضوع باعث بروز این مسئله می‌شود که شاید DE با یک استراتژی نخبه‌گرایی آرام‌تر عملکرد بهتری داشته باشد. گاهی برای دستیابی به راه‌حل خوب لازم است الگوریتم‌های تکاملی از میان نواحی نامرغوب تونل ایجاد کنند. الگوریتم‌های تکاملی بدون نخبه‌گرایی ممکن است برای برخی مسائل مشخص با توابع هزینه‌ی پرهزینه و یا پویا مناسب‌تر باشند (فصل ۲۱ را ببینید).

۱۲-۲-۳ تنظیم فاکتور مقیاس

فاکتور مقیاس F در DE تأثیر بردار تفاضل بر روی بردار جهش‌یافته را مشخص می‌نماید. تا به اینجا F ثابت فرض شد. با این حال، اتفاقی نمودن پارامترها یکی از نشان‌ها و ویژگی‌های الگوریتم‌های تکاملی می‌باشد. بنابراین، می‌توان F را نیز متغیری اتفاقی در نظر گرفت. این کار گستره‌ی وسیع‌تری از بردارهای جهش‌یافته را نتیجه داده و ممکن است باعث افزایش کاوش در الگوریتم DE شود. همچنین، اتفاقی ساختن F امکان تحلیل ویژگی‌های همگرایی DE را به ما می‌دهد [زاهاری[1]، ۲۰۰۲].

فاکتور مقیاس را به دو روش می‌توان تغییر داد. راه اول آن است که اجازه دهیم F یک اسکالر باقی مانده و مقدار آن را در هر حلقه‌ی "for each individual" در شکل ۱۲-۲ تغییر دهیم. این نوع تغییر، Dither نام دارد. راه دوم آن است که F را به یک بردار n بعدی تبدیل کرده و هر عنصر آن را در هر حلقه‌ی "for each individual" تغییر دهیم. بدین ترتیب هر عنصر از بردار جهش‌یافته‌ی v توسط یک عنصر مقیاس‌شده‌ی منحصر به فرد از بردار تفاضل اصلاح خواهد شد. این نوع تغییر، jitter نام دارد.

[1] Zaharie

در نوع تغییر Dither، تولید بردار جهش از شکل ۱۲–۲ با معادله‌ی زیر جایگزین می‌شود:

$$F \leftarrow U[F_{min}, F_{max}] \\ v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3}) \qquad (۱۰–۱۲)$$

که در این صورت فاکتور مقیاس عددی اتفاقی با توزیع یکنواخت میان F_{min} و F_{max} خواهد بود. یکی دیگر از روش‌های موجود برای Dither آن است که F از یک توزیع گاوسی گرفته شود [پرایس و همکاران، ۲۰۰۵، بخش ۲–۵–۲].

در نوع jitter، تولید بردار جهش‌یافته از شکل ۱۲–۲ با معادله‌ی زیر جایگزین خواهد شد:

$$\text{برای هر بعد } j \in [1, n] \\ F_j \leftarrow U[F_{min}, F_{max}] \\ v_{ij} \leftarrow x_{r1,j} + F_j(x_{r2,j} - x_{r3,j}) \\ \text{بعد بعدی} \qquad (۱۱–۱۲)$$

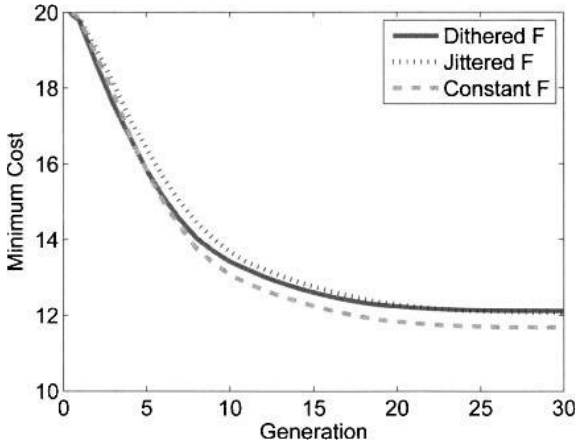
که در این صورت هر عنصر از بردار تفاضل توسط مقدار متفاوتی برای تولید بردار جهش‌یافته مقیاس خواهد شد.

در کل به نظر می‌رسد مقادیر ثابت F برای توابع ساده (برای مثال تابع کروی) بهتر عمل کرده و مقادیر اتفاقی‌شده‌ی F برای بیشتر توابع چندپیمانه‌ای مؤثر واقع می‌شود. تعییر نوع jitter برای توابع تفکیک‌پذیر بهترین عملکرد را داشته و تعییر نوع Dither نیز برای توابع با درجه‌ی تفکیک‌پذیری بسیار کم مؤثر واقع می‌شود [پرایس، ۲۰۱۳].

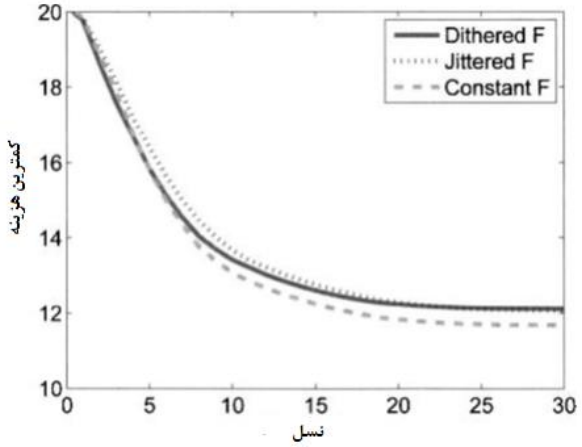
مثال ۱۲–۳

در این مثال به مطالعه‌ی تغییرات نوع Dither و jitter می‌پردازیم. در این مثال نیز مانند مثال قبل از اندازه‌ی جمعیتی برابر ۵۰ استفاده می‌نماییم. همچنین در معادلات (۱۲–۱۰) و (۱۲–۱۱) از $F_{min} = 0.2$ و $F_{max} = 0.6$ استفاده می‌کنیم. شکل ۱۲–۸ میانگین عملکرد DE بر روی تابع ۲۰ بعدی اَکلی با نرخ جهش $c = 0.9$ و سه نوع متفاوت از فاکتور مقیاس را نشان می‌دهد: (۱) F ثابت، (۲) Dither و (۳) jitter. می‌توان دید که عملکرد تغییرات نوع jitter و Dither تقریبا مانند هم می‌باشد در حالی که F ثابت بهترین عملکرد را دارد. این موضوع بدین معنی است که در این مثال اتفاقی نمودن F باعث تضعیف عملکرد DE می‌شود. از سوی دیگر، شکل ۱۲–۹ نتایج مشابه را برای تابع بهینه‌سازی محک فلچر نشان می‌دهد. در این مورد، گزینه‌ی jitter هر چند کم اما به‌صورت کاملاً واضح از F ثابت بهتر عمل می‌نماید. در این مورد گزینه‌ی Dither بدترین عملکرد را دارد.

این مثال نشان می‌دهد که تصمیم‌گیری در مورد انتخاب گزینه‌ی Dither یا jitter به نوع خاص مسئله و همچنین سایر پارامترهای موجود در DE بستگی دارد. تأثیر تغییر دادن F به عواملی چون نرخ جهش، برد و نوع توزیع مورد استفاده برای تغییر دادن F و غیره بستگی دارد.



شکل ۱۲–۸ مثال ۱۲–۳: عملکرد DE بر روی تابع ۲۰ بعدی اَکلی با نرخ جهش $c = 0.9$. منحنی‌ها هزینه‌ی بهترین ذره در هر نسل را که بر روی ۱۰۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است نشان می‌دهند. استفاده از F ثابت عملکرد نسبتاً بهتری را نسبت به گزینه‌های jitter و Dither به دست می‌دهد.



شکل ۱۲–۹ مثال ۱۲–۳: عملکرد DE بر روی تابع ۲۰ بعدی فلچر با نرخ جهش $c = 0.9$. منحنی‌ها هزینه‌ی بهترین ذره در هر نسل را که بر روی ۱۰۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است نشان می‌دهند. گزینه‌ی jitter کمی بهتر از گزینه‌ی F ثابت عمل کرده و گزینه‌ی F ثابت نیز خود کمی بهتر از گزینه‌ی Dither عمل می‌نماید.

معادلات (۱۲–۱۰) و (۱۲–۱۱) هر دو از توزیعی با میانگین صفر برای تعیین تغییرات فاکتور مقیاس ΔF از مقدار نامی آن که برابر با $(F_{min} + F_{max})/2$ استفاده می‌نمایند. سایر توزیعات مانند توزیع یکنواخت با میانگین غیر صفر و یا توزیع log-normal را نیز می‌توان برای این منظور مورد استفاده قرار داد. این توزیعات برای برخی مسائل عملکرد را بهبود بخشیده‌اند [پرایس و همکاران، ۲۰۰۵، بخش ۲–۵–۲].

۱۲–۳ بهینه‌سازی گسسته

در این بخش نشان می‌دهیم چگونه می‌توان از DE برای بهینه‌سازی بر روی دامنه‌ی گسسته استفاده نمود. در DE تنها جایی که دامنه‌ی گسسته مشکل‌زا می‌شود، تولید بردار جهش‌یافته است. از شکل ۱۲–۲ به یاد آورید که

$$v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3}) \qquad (۱۲–۱۲)$$

از آنجا که $F \in [0,1]$ است، v_i ممکن است به دامنه‌ی D تعلق نداشته باشد. DE در اصل برای مسائل با دامنه‌ی پیوسته طراحی گردید اما می‌توان با اعمال اصلاحاتی آن را برای مسائل با دامنه‌ی گسسته نیز به کار برد. برای این کار دو روش وجود دارد. این دو روش در ظاهر شبیه هم می‌باشند اما در اصل کاملاً متفاوت از یکدیگر هستند. راه اول آن است که بردار جهش‌یافته با استفاده از روش‌های موجود در DE استاندارد، مانند معادله‌ی (۱۲–۱۲) ایجاد شده و سپس به‌گونه‌ای اصلاح گردد که در دامنه‌ی مسئله D واقع شود. راه دوم آن است که فرآیند تولید بردار جهش‌یافته به‌گونه‌ای اصلاح گردد که بردار جهش‌یافته مستقیماً در D واقع شود. یکی از این روش‌ها در بخش ۱۲–۳–۲ مورد بحث واقع خواهد شد. برای مطالب بیشتر در مورد DE در دامنه‌های گسسته، [اُنووبولو[1] و داوِندِرا[2]، ۲۰۰۹] را ببینید.

۱۲–۳–۱ تکامل تفاضلی Mixed-Integer

یک راه مشخص برای اطمینان از اینکه $v_i \in D$ آن است که آن را به داخل D بنگاریم. هنگامی که DE بدین ترتیب برای استفاده بر روی دامنه‌های گسسته اصلاح می‌شود، معمولاً mixed integer DE خوانده

[1] Onwubolu
[2] Davendra

می‌شود [هوانگ[1] و وانگ، ۲۰۰۲]، [سو[2] و لی[3]، ۲۰۰۳]. برای مثال، اگر D مجموعه‌ی بردارهای صحیح n بعدی باشد، آنگاه می‌توان معادله‌ی (۱۲–۱۲) را با معادله‌ی زیر جایگزین نمود:

$$v_i \leftarrow round[x_{r1} + F(x_{r2} - x_{r3})] \qquad (۱۳–۱۲)$$

که در آن اپراتور round به‌صورت عنصر به عنصر بر روی یک بردار عمل می‌نماید. یک راه کلی‌تر برای انجام این کار به‌صورت زیر است

$$v_i \leftarrow P[x_{r1} + F(x_{r2} - x_{r3})] \qquad (۱٤–۱۲)$$

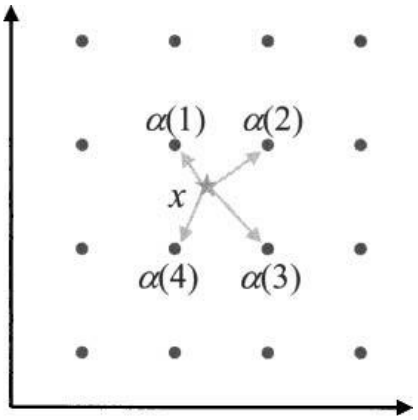
که در آن P اپراتور نگاشت است به طوری که برای تمامی xها، $P(x) \in D$ است.

در کل P می‌تواند بسیار پیچیده‌تر از معادله‌ی (۱۲–۱۳) باشد. برای مثال، دوباره فرض کنید دامنه‌ی مسئله D مجموعه‌ای از بردارهای صحیح N بعدی می‌باشد. در این صورت می‌توان P را به‌صورت زیر تعریف نمود

$$P(x) = arg\min_{\alpha} f(\alpha) : \ \alpha \in D, |x_j - \alpha_j| < 1 \quad \text{برای تمامی } j \in [1, n] \qquad (۱۵–۱۲)$$

این کار، بردار حقیقی x را به بردار صحیح α، که کمترین هزینه را نتیجه می‌دهد، می‌نگارد. همچنین هر عنصر α در فاصله‌ی ۱ واحدی از عنصر متناظر در بردار x واقع می‌شود. این ایده در شکل ۱۲–۱۰ برای دو بعد نمایش داده شده است. اشکال دیگری از تابع نگاشت را نیز می‌توان با توجه به مختصات مسئله مورد استفاده قرار داد.

[1] Huang
[2] Su
[3] Lee



شکل ۱۲-۱۰ نگاشت بردار حقیقی x به بردار گسسته‌ی α. این مثال دو بعدی نشان می‌دهد که x در دامنه‌ی مسئله‌ی بهینه‌سازی گسسته قرار ندارد. مقادیر تابع هزینه‌ی نزدیکترین نقاط به x در دامنه‌ی مسئله آزموده شده‌اند. نقطه‌ی $\alpha(i)$ که کمترین مقدار هزینه را نتیجه می‌دهد به‌عنوان نگاشت x (یا همان $P(x)$) انتخاب می‌شود.

۱۲-۳-۲ تکامل تفاضلی گسسته

یک راه دیگر برای اصلاح DE جهت استفاده در مسائل گسسته آن است که روش تولید بردار جهش‌یافته را به‌گونه‌ای تغییر داد که بردارهای جهش مستقیماً در دامنه‌ی D قرار بگیرند. هنگامی که DE بدین ترتیب مورد اصلاح واقع می‌شود، DE گسسته خوانده می‌شود [پَن[1] و همکاران، ۲۰۰۸]. در این روش، معادله‌ی (۱۲-۱۲) با معادله‌ی زیر جایگزین خواهد شد

$$v_i \leftarrow G(x_{r1}, x_{r2}, x_{r3}) \qquad (۱۲-۱۶)$$

که در آن $G(.) \in D$ خواهد بود اگر تمامی آرگومان‌های آن در D واقع شوند. این در واقع تعمیمی است از معادله‌ی (۱۲-۱۴)، بنابراین می‌توان گفت DE گسسته تعمیمی است از mixed integer DE. تابع $G(.)$ را می‌توان به‌گونه‌ای نوشت که برای تمامی توابع گسسته مناسب واقع شود و یا می‌توان آن را با توجه به مشخصات خاص مسئله تعیین نمود. برای مثال، بار دیگر فرض کنید D مجموعه‌ی تمامی بردارهای صحیح n بعدی باشد. آنگاه می‌توان از دو گزینه‌ی زیر برای تولید بردار جهش‌یافته استفاده نمود:

$$\begin{aligned} &\text{گزینه 1:} \quad v_i \leftarrow x_{r1} + round[F(x_{r2} - x_{r3})] \\ &\text{گزینه 2:} \quad v_i \leftarrow x_{r1} + sign(x_{r2} - x_{r3}) \end{aligned} \qquad (۱۲-۱۷)$$

[1] Pan

که در آن توابع round و sign به‌صورت عنصر به عنصر بر روی بردارها عمل خواهند کرد.

به یاد آورید که ایده‌ی اصلیِ تولید بردار جهش‌یافته در DE گسسته، به دست آوردن v_i از اصلاح یک بردار راه‌حل نامزد (x_{r1} در معادله‌ی بالا) با استفاده از تفاوت موجود میان دو بردار راه‌حل نامزد دیگر (x_{r2} و x_{r3} در معادله‌ی بالا) می‌باشد. هر روشی برای انجام این کار که به $v_i \in D$ منجر شود برای DE گسسته مناسب خواهد بود. روش‌های بسیاری برای تحقیقات آتی در این زمینه وجود دارد.

۱۲-۴ تکامل تفاضلی و الگوریتم ژنتیک

در این بخش نشان خواهیم داد که DE حالت خاصی از GA پیوسته می‌باشد. فرض کنید چیزی در مورد DE نمی‌دانیم ولی می‌خواهیم یک الگوریتم تکاملی با استفاده از مطالبی که در بخش دوم این کتاب آموختیم ایجاد نماییم. به‌طور خاص، تصور نمایید می‌خواهیم گونه‌ای اصلاح شده از یک GA را ایجاد نماییم. می‌خواهیم به ازای هر ذره‌ی x_i، متغیرهای مستقل را به‌صورت اتفاقی از یک ذره‌ی انتخاب شده به‌صورت اتفاقی مانند v_i، که آن را بردار جهش‌یافته خواهیم خواند، به x_i کپی کرده تا فرزند u_i را به دست آوریم. برای این منظور از نرخ برش c استفاده می‌نماییم. این نرخ، میزان احتمال جایگزین شدن یک متغیر مستقل در x_i توسط متغیر مستقل متناظر در v_i را نشان خواهد داد. اگر در بخش ۸-۸-٤، $x_a = x_i$، $x_b = v_i$ و $y = u_i$ تعریف شود، آنگاه ایده‌ی به کار رفته در این بخش بسیار مشابه ایده‌ی به کار رفته در بخش ۸-۸-٤ خواهد بود. علاوه بر این، می‌خواهیم اگر فرزند بهتر از والد بود، x_i را با u_i جایگزین نماییم، این کار نیز مشابه (1+1)-ES از بخش ٦-١ می‌باشد. با در نظر داشتن این ایده‌ها، الگوریتم GA اصلاح شده از شکل ۱۲-۱۱ را ارائه می‌نماییم.

حال فرض کنید می‌خواهیم الگوریتم‌مان را برای عملکرد بهتر میزان‌سازی نماییم. به جای تخصیص دادن یک بردار اتفاقی به ذره‌ی v_i، می‌خواهیم با ایجاد آشفتگی در یک ذره‌ی اتفاقی v_i را تولید نماییم. به‌طور خاص، می‌خواهیم یک ذره‌ی اتفاقی را مانند آنچه که در شکل ۱۲-۱ نشان داده شده است آشفته نماییم. این تغییر در نحوه‌ی تولید v_i یک تغییر مفهومی است، اما همچنان بر مبنای اعضای حاضر در جمعیت قرار دارد. همچنین توجه داشته باشید که به دلیل وجود عبارت "if rand(0,1) < c" در شکل ۱۲-۱۱، امکان $u_{ij} = x_{ij}$ برای تمامی $j \in [1, n]$ وجود دارد. این بدین معنی است که امکان آنکه فرزند u_i کلونی از والد خود یعنی x_i باشد وجود دارد. این اتفاق مطلوب نیست و ما می‌خواهیم مطمئن شویم حداقل یک متغیر مستقل در u_i از v_i کپی شده است. برای این منظور، یک شرط دیگر را به عبارت "if rand(0,1) < c" اضافه کرده و آن را به عبارت "If (rand(0,1) < c) or (j = random index ∈ [1, n])" تبدیل

می‌نماییم. در این عبارت n ابعاد مسئله است. با در نظر داشتن این ایده‌ها، تعمیمی از شکل ۱۲–۱۱ را که در شکل ۱۲–۱۲ نشان داده شده است، به دست خواهیم آورد.

جمعیتی از راه‌حل‌های نامزد را مقداردهی کن $\{x_i\}$، $i \in [1,N]$
تا زمانی که شرایط توقف برآورده نشده است
برای هر ذره x_i، $i \in [1,N]$
$r_1 \leftarrow$ عدد صحیح اتفاقی $\in [1,N] : r_1 \neq i$
$v_i \leftarrow x_{r1}$
برای هر بعد $j \in [1,n]$
اگر $rand(0,1) < c$ آنگاه
$u_{ij} \leftarrow v_{ij}$
در غیر این صورت
$u_{ij} \leftarrow x_{ij}$
پایان اگر
بعد بعدی
ذره‌ی بعدی
برای هر $i \in [1,N]$، اگر $f(u_i) < f(x_i)$ آنگاه $x_i \leftarrow u_i$
نسل بعد

شکل ۱۲–۱۱ شبه کد بالا نسخه‌ی ۱ از الگوریتم ژنتیک اصلاح شده برای مینیمم‌سازی $f(x)$ را به تصویر می‌کشد. در این شبه کد c نرخ جهش بوده و $rand(0,1)$ عددی اتفاقی در بازه‌ی $[0,1]$ می‌باشد.

حال توجه کنید که شکل ۱۲–۱۲ معادل الگوریتم پایه‌ای DE از شکل ۱۲–۲ می‌باشد و این بدین معناست که DE حالت خاصی از الگوریتم ژنتیک است. این مسئله دو سؤال را ایجاد می‌کند.

۱. آیا یک GA باید یک GA خوانده شود و یا اینکه باید یک حالت خاص از DE در نظر گرفته شود؟
۲. آیا یک DE باید یک DE خوانده شود و یا باید یک نوع از GA در نظر گرفته شود؟

برای پاسخ به سؤال اول باید در نظر داشت که عنوان GA به دلیل تاریخچه و اهمیت بنیادین آن در توسعه‌ی الگوریتم‌های تکاملی هیچ‌گاه منسوخ نخواهد شد. علاوه بر این، عنووان GA مفید است چرا که باعث ایجاد نوعی تلفیق میان خاصیت‌های بیولوژیکی و الگوریتم می‌شود (تولید مثل جنسی، پیر شدن و غیره). این موضوع نیز به نوبه‌ی خود می‌تواند به تعمیم‌هایی مفید از GA منجر شود.

برای پاسخ به سؤال دوم می‌توان گفت که جامعه‌ی الگوریتم‌های تکاملی از حدود دهه‌ی ۹۰ دریافته است که DE آنقدر متمایز هست که بتوان آن را به‌عنوان یک الگوریتم تکاملی جدا در نظر گرفت و نباید آن را به‌عنوان حالت خاصی از یک الگوریتم تکاملی دیگر دانست. با این حال، هر چند که این سؤال‌ها برای DE پاسخ داده شده‌اند، اما مفهوم گسترده‌تری برای سایر الگوریتم‌های تکاملی دارند. هر ساله الگوریتم‌های تکاملی جدیدی، که برخی از آن‌ها را در فصل ۱۷ مورد بررسی قرار خواهیم داد، معرفی می‌شوند. کدامیک از آن‌ها لایق یک کلاس جداگانه برای خود می‌باشند و کدامیک را باید به‌عنوان تعمیم یا حالت خاصی از سایر الگوریتم‌های تکاملی در نظر بگیریم؟ هر چه تعداد الگوریتم‌های تکاملی بیشتر می‌شود، جا برای الگوریتم‌های جدید تنگتر می‌شود. از سویی نیز، همان‌طور که DE علیرغم شباهت‌های آن به GA لایق یک کلاس مخصوص به خود می‌باشد، برخی از این الگوریتم‌های تکاملی جدید نیز لایق یک کلاس مخصوص به خود خواهند بود.

جمعیتی از راه‌حل‌های نامزد را مقداردهی کن، $\{x_i\}$ برای $i \in [1,N]$
تا زمانی که شرایط توقف برآورده نشده است
  برای هر ذره x_i، $i \in [1,N]$
    $r_1 \leftarrow$ عدد صحیح اتفاقی $\in [1,N] : r_1 \neq i$
    $r_2 \leftarrow$ عدد صحیح اتفاقی $\in [1,N] : r_1 \neq \{i, r_1\}$
    $r_3 \leftarrow$ عدد صحیح اتفاقی $\in [1,N] : r_1 \neq \{i, r_1, r_2\}$
    $v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3})$
    $\mathcal{J}_r \leftarrow$ عدد صحیح اتفاقی $\in [1,n]$
    برای هر بعد $j \in [1,n]$
      $r_{cj} \leftarrow$ عدد اتفاقی $\in [0,1]$
      اگر $(rand(0,1) < c)$ و یا $(j = \mathcal{J}_r)$، آنگاه
        $u_{ij} \leftarrow v_{ij}$
      در غیر این صورت
        $u_{ij} \leftarrow x_{ij}$
      پایان اگر
    بعد بعدی
  ذره بعدی
  برای هر $i \in [1,N]$، اگر $f(u_i) < f(x_i)$ آنگاه $x_i \leftarrow u_i$
نسل بعدی

شکل ۱۲-۱۲ شبه کد بالا نسخه‌ی دوم از یک الگوریتم ژنتیک اصلاح شده را برای مینیمم‌سازی تابع $f(x)$ نشان می‌دهد. در این شکل F اندازه‌ی گام، c نرخ برش و rand(0,1) عددی است اتفاقی در بازه‌ی $[0,1]$.

۱۲-۵ نتیجه‌گیری

تحقیقات حاضر در زمینه‌ی DE آینه‌ای است از تحقیقات حاضر در زمینه‌ی سایر الگوریتم‌های تکاملی: ساده‌سازی الگوریتم DE [عمران[1] و همکاران، ۲۰۰۹]، تطبیق پارامترهای کنترلی DE [کین[2] و همکاران،

[1] Omran
[2] Qin

۲۰۰۹]، ترکیب با سایر الگوریتم‌ها [نومُن[1] و ایبا، ۲۰۰۸]، بسط دادن DE به موارد خاصی از مسائل بهینه‌سازی مانند مسائل پویا [برست[2] و همکاران، ۲۰۰۹]، مسائل چندهدفه [مِزورا- منتِس[3] و همکاران، ۲۰۰۸]، [دومینگوئز[4] و پولیدو[5]، ۲۰۱۱] و مسائل مقید [لَمپینن[6]، ۲۰۰۲]، [مزورا- منتس و کوئلو کوئلو، ۲۰۰۸]. مانند سایر الگوریتم‌های تکاملی، جای زیادی برای تحلیل نظری و ریاضی DE وجود دارد، بنابراین این موضوع را می‌توان به‌عنوان زمینه‌ای مفید برای تحقیقات آتی در نظر گرفت. مقایسه‌ی رویکرد DE به تطابق شکل (بحث انتهای بخش ۲-۱ را به یاد آورید) با آنچه که در CMA-ES وجود داشت نیز جالب خواهد بود (انتهای بخش ٦-۵ را ببینید). برای مطالعه‌ی بیشتر در زمینه‌ی DE می‌توانید به کتاب‌های [پرایس و همکاران، ۲۰۰۵]، [فئوکتیستوو[7]، ۲۰۰۰٦]، [کینگ، ۲۰۰۹]، [ژانگ و ساندِرسون[8]، ۲۰۰۹] و مقاله‌های [داس و سوگانتان[9]، ۲۰۱۱]، [نِری[10] و تیرونِن[11]، ۲۰۱۰] و فصل‌هایی از کتاب [سیسورِدا[12]، ۲۰۱۰] مراجعه نمایید.

[1] Noman
[2] Brest
[3] Mezura-Mentes
[4] Dominguez
[5] Pulidio
[6] Lampinen
[7] Feoktistov
[8] Sanderson
[9] Suganthan
[10] Neri
[11] Tirronen
[12] Syswerda

مسائل

مسائل نوشتاری

۱۲–۱ در بخش ۱۲–۱ دیدیم که تعداد عناصر بردار جهش‌یافته (k) که در تولید بردار آزمایش دخالت دارد، تقریبا از توزیع دوجمله‌ای پیروی می‌کند (برای مسئله‌ی کامپیوتری مشابه با این مسئله، مسئله‌ی ۱۲–۹ را ببینید).

الف) در آزمایشی که احتمال موفقیت در آن برابر c باشد، احتمال حصول k موفقیت از n آزمایش چه قدر خواهد بود؟

ب) در یک الگوریتم DE کلاسیک، احتمال آنکه k عنصر از بردار جهش در تولید بردار آزمایش دخالت داشته باشد چه قدر خواهد بود؟

۱۲–۲ در الگوریتم DE کلاسیک از شکل ۱۲–۱ نیاز به تولید سه عدد اتفاقی صحیح می‌باشد، اما ممکن است به دلیل محدودیت مقادیر مجاز برای این اعداد، نیاز به تکرار فرایند تولید عدد اتفاقی باشد.

الف) به‌طور میانگین به چند بار تولید عدد اتفاقی جهت حصول مقادیر مجاز برای r_1، r_2 و r_3 نیاز است؟

ب) اگر $n = 20$ باشد برای حصول مقادیر مجاز برای r_1، r_2 و r_3 به چند بار تولید عدد اتفاقی به طور میانگین نیاز است؟

۱۲–۳ فرض کنید آزمایش $j = J_r$ را از شکل ۱۲–۲ حذف نماییم. در این صورت احتمال آنکه u_i کلونی از x_i باشد چه قدر است؟ احتمال $c = 0.5$ و $n = 20$ چه قدر خواهد بود؟

۱۲–٤ فرض کنید می‌خواهیم DE/rand/1/L را مطابق آنچه که در بخش ۱۲–۲–۱ توصیف شد پیاده‌سازی نماییم تنها با این تفاوت که نمی‌خواهیم به عناصر v_i هنگام کپی شدن به u_i اجازه گردش از عنصر اول به آخر را بدهیم. در این صورت می‌توانیم مقدار J در شکل ۱۲–۳ را با عبارت $J \leftarrow \{s, \min(n, s + L - 1)\}$ جایگزین نماییم. میانگین تعداد عناصر کپی‌شونده به بردار آزمایش چه قدر خواهد بود؟

۱۲–۵ چگونه می‌توانید الگوریتم DE را تغییر دهید تا نخبه‌گرا نباشد؟

۱۲–٦ یک اپراتور نگاشت اتفاقی برای mixed-integer DE ارائه نمایید؟

۱۲–۷ یک مولد بردار جهش اتفاقی برای DE گسسته ارائه نمایید؟

۱۲–۸ عبارت $if\ f(u_i) < f(x_i)\ then\ x_i \leftarrow u_i$ از شکل ۱۲–۲ را به‌گونه‌ای اصلاح نمایید که DE بیشتر مانند $(\mu + \lambda) - ES$ شود.

مسائل کامپیوتری

۱۲-۹ تعداد جهش‌های دخالت‌کننده: در مسئله‌ی ۱۲-۱ دو احتمال به دست آوردید: (۱) احتمال k موفقیت از n آزمایش در صورتی که احتمال هر بار موفقیت برابر c باشد، (۲) احتمال آنکه k عنصر از بردار جهش‌یافته در تولید بردار آزمایش دخالت داشته باشد. این دو احتمال را به‌عنوان تابعی از k برای $n = 20$ و $c = 0.5$ رسم کنید.

۱۲-۱۰ اندازه‌ی گام DE: الگوریتم کلاسیک DE از شکل ۱۲-۲ را برای مینیمم‌سازی تابع ۱۰ بعدی روزنبروک پیاده‌سازی نمایید (برایی تعریف تابع روزنبروک ضمیمه‌ی ج.٤-۱ را ببینید). از اندازه‌ی جمعیتی برابر $N = 100$، نرخ جهشی برابر با $c = 0.9$ و تعداد نسل ۵۰ استفاده نمایید. برای هر یک از اندازه‌ی گام‌های ۰,۱، ۰,۳، ۰,۵، ۰,۷ و ۰,۹ چهل بار شبیه‌سازی مونت کارلو انجام دهید و برای هر دسته شبیه‌سازی مقدار میانگین بهترین هزینه در هر نسل را محاسبه نمایید. عملکرد میانگین هر دسته از شبیه‌سازی‌ها را به‌عنوان تابعی از شماره‌ی نسل رسم کرده و در مورد نتایج به دست آمده توضیح دهید.

۱۲-۱۱ نرخ برش DE: الگوریتم DE کلاسیک از شکل ۱۲-۲ را برای مینیمم‌سازی تابع ۱۰ بعدی رستریجین پیاده‌سازی نمایید (برای تعریف تابع رستریجین ضمیمه‌ی ج.۱۱-۱ را ببینید). از اندازه‌ی جمعیتی برابر $N = 100$، اندازه‌ی گامی برابر $F = 0.4$ و تعداد نسل ۵۰ استفاده نمایید. برای هر یک نرخ‌های برش ۰,۱، ۰,۵ و ۰,۹ چهل بار شبیه‌سازی مونت کارلو انجام دهید و برای هر دسته از شبیه‌سازی‌ها مقدار میانگین بهترین هزینه در هر نسل را محاسبه نمایید. عملکرد میانگین هر دسته از شبیه‌سازی‌های مونت کارلو را به‌عنوان تابعی از شماره‌ی نسل رسم کرده و در مورد نتایج حاصله توضیح دهید.

فصل سیزدهم

الگوریتم‌های تخمین توزیع

الگوریتم‌های تخمین توزیع از روشی متفاوت برای نمونه‌گیری از فضای جستجو استفاده می‌کنند. در این روش، از جمعیت برای تخمین توزیع احتمال بر روی فضای جستجو استفاده می‌شود. این توزیع نشانگر مشخصه‌های مهم جمعیت می‌باشد.

آلدن[1] رایت [رایت و همکاران، ۲۰۰٤]

یک الگوریتم تخمین توزیع (EDA[2]) یک تابع را با استفاده از ردگیری خواص آماری جمعیت راه‌حل‌های نامزد بهینه‌سازی می‌نماید [لاراناگا[3] و لوزانو، ۲۰۰۲]. از آنجا که خواص آماری جمعیت نگه‌داری می‌شود، دیگر نیازی به نگه‌داری از خود جمعیت از یک نسل به نسل دیگر نمی‌باشد. در هر نسل، ابتدا یک جمعیت با استفاده از خواص آماری جمعیت نسل‌های قبلی ایجاد شده و سپس خواص آماری برازنده‌ترین ذرات محاسبه می‌شود. در آخر، یک جمعیت جدید با استفاده از خواص برازنده‌ترین ذرات ایجاد می‌گردد. این فرایند از یک نسل به نسل دیگر تکرار می‌شود. بنابراین، EDAها الگوریتم‌های جمعیت–محور بوده که حداقل بخشی از جمعیت را از نسلی به نسل دیگر حذف کرده و آن را با استفاده از خواص آماری ذرات با برازندگی بالا جایگزین می‌نماید. EDAها از این جهت که شامل بازترکیب نمی‌شوند با بسیاری از الگوریتم‌های تکاملی تفاوت دارند. EDAها را با نام‌های الگوریتم‌های ژنتیک مدل‌سازی احتمالی (PMBGA[4]) [پلیکان[5] و همکاران، ۲۰۰۲] و الگوریتم‌های تخمین چگالی دوره‌ای (IDEA[6]) [بوزمن[7] و تیِرِن[8]، ۲۰۰۳] نیز می‌شناسند.

مروری بر فصل

بخش ۱۳–۱ این فصل را با ارائه‌ی طرحی پایه‌ای از یک EDA عمومی و همچنین نشان دادن معنای استفاده از خواص آماری ذرات جمعیت، آغاز می‌کند. همه‌ی EDAها از خواص آماری، مانند آنچه که در بخش ۱۳–۱–۲ محاسبه شده است، برای ایجاد جمعیت بعدی ذرات استفاده می‌کنند. بخش ۱۳–۲ برخی از EDAهای محبوب برای بهینه‌سازی مسائل گسسته‌ای که تنها به خواص آماری مرتبه‌ی اول بستگی دارند، را طرح‌ریزی می‌کند. این الگوریتم‌ها شامل الگوریتم تخمین حاشیه‌ای تک‌متغیره (UMDA[9])، الگوریتم ژنتیک فشرده (cGA[10]) و یادگیری افزایشی جمعیت–محور (PBIL[11])، که تعمیمی از UMDA می‌باشد، می‌شود.

---

[1] Alden
[2] Estimation of Distribution Algorithm
[3] Larranaga
[4] Probabilistic Model-Building Genetic Algorithm
[5] Pelikan
[6] Iterated Distribution Estimation Algotrithm
[7] Boseman
[8] Thieren
[9] Univariate Marginal Distribution Algorithm
[10] Compact Genetic Algorithm
[11] Population Based Incremental Learning

بخش ۱۳-۳ برخی EDAهایی را که از خواص آماری مرتبه دوم استفاده می‌نمایند، معرفی می‌نماید. این الگوریتم‌ها شامل بیشینه‌سازی اطلاعات متقابل برای خوشه‌سازی ورودی (MIMIC[1])، ترکیب بهینه‌سازها با استفاده از درخت اطلاعات متقابل (COMIT[2]) و الگوریتم‌های تخمین حاشیه‌ای دومتغیره (BMDA[3]) می‌شود. بخش ۱۳-٤ به بحث در مورد EDAهای چندمتغیره که از خواص آماری مراتب بالاتر استفاده می‌کنند، پرداخته و طرحی از الگوریتم‌های ژنتیک فشرده‌ی تعمیم‌یافته (ECGA[4]) به دست می‌دهد.

تمام EDAهای ذکر شده در بالا برایی مسائل با دامنه‌ی دودویی طراحی شده‌اند. ما این فصل را با نشان دادن چگونگی تعمیم این EDAها به مسائل با دامنه‌ی پیوسته به پایان خواهیم برد. این ایده را با ارائه‌ی UMDA و PBIL پیوسته در بخش ۱۳-۵ نشان داده شده است.

۱۳-۱ الگوریتم‌های تخمین توزیع: مفاهیم بنیادی

این بخش طرح اساسی یک EDA عمومی را در بخش ۱۳-۱-۱ ارائه داده و در بخش ۱۳-۱-۲ معنای خواص آماری محاسبه شده از ذرات جمعیت را نشان می‌دهد.

۱۳-۱-۱ یک الگوریتم تخمین توزیع ساده

شکل ۱۳-۱ طرحی کلی از یک EDA را نشان می‌دهد. هر EDA به روش خاص خود از سه گام عمده‌ی شکل ۱۳-۱ تبعیت می‌کند. سؤال اول آن است که چگونه M ذره از میان N ذره‌ی جمعیت انتخاب می‌شود؟ سؤال دوم آن است که کدام خواص آماری از این M ذره محاسبه شده و این خواص چگونه محاسبه می‌شوند؟ سوم آنکه، چگونه از این خواص آماری برای ایجاد یک جمعیت جدید در نسل بعد استفاده می‌شود؟ جواب‌هایی که به این سؤالات داده می‌شود می‌تواند به انواع مختلف EDA منجر شود و همین موضوع است که بیشتر تمرکز ما را در ادامه‌ی این فصل به خود اختصاص می‌دهد.

اولین گام در شکل ۱۳-۱ حلقه‌ی انتخاب M ذره از میان N ذره‌ی جمعیت می‌باشد ($M < N$). این کار را می‌توان به طرق مختلف انجام داد. روش‌های انتخاب در EDA همان روش‌هایی است که در مورد سایر الگوریتم‌های تکاملی به کار می‌روند (بخش ۸-۷ را ببینید). بنابراین، فرایندهای انتخاب را دیگر در این فصل مورد بحث قرار نخواهیم داد.

[1] Mutual Information maximization for Input Clustering
[2] Combining Optimizers with Mutual Information Trees
[3] Bivariate Marginal Distribution Algorithm
[4] Extended Compact Genetic Algorithm

یک جمعیت آغازین از راه‌حل‌های نامزد را مقداردهی کن $\{x_i\}$، $i \in [1, N]$
تا زمانی که شرایط توقف برآورده نشده است
M ذره از $\{x_i\}$ را متناسب با برازندگیشان انتخاب کن
خواص آماری M ذره را محاسبه کن
از خواص آماری برای ایجاد یک جمعیت جدید $\{x_i\}$ استفاده کن، $i \in [1, N]$
نسل بعد

شکل ۱۳-۱ طرح کلی از یک الگوریتم تخمین توزیع (EDA).

۱۳-۱-۲ محاسبات آماری

این بخش به بحث در مورد محاسبات آماری از یک جمعیت از ذرات، که گام دوم در حلقه‌ی شکل ۱۳-۱ می‌باشد، می‌پردازد. این موضوع را با مثالی ساده توضیح می‌دهیم. فرض کنید یک مسئله‌ی بهینه‌سازی دودویی داریم. همچنین فرض کنید N راه‌حل نامزد داریم و میزان برازندگی آن‌ها را محاسبه نموده‌ایم و همچنین از نوعی انتخاب برازنده-محور برای انتخاب M ذره استفاده می‌کنیم. فرایند انتخاب در EDA مانند سایر الگوریتم‌های تکاملی باید به سمت ذرات با برازندگی بیشتر گرایش‌دهی شده باشد (بخش ۸-۷ را ببینید). فرض کنید $M = 10$ و فرض کنید ذرات زیر را انتخاب نموده‌ایم:

$$\begin{aligned} x_1 &= (0,1,1,1,1,0), \quad x_2 = (0,1,1,1,1,1) \\ x_2 &= (1,0,0,1,1,0), \quad x_4 = (1,1,1,0,1,0) \\ x_5 &= (0,1,0,0,0,1), \quad x_6 = (0,1,0,0,1,0) \\ x_7 &= (0,0,1,1,1,0), \quad x_2 = (1,0,1,0,1,0) \\ x_9 &= (0,1,0,0,0,0), \quad x_{10} = (0,1,1,1,1,1) \end{aligned} \tag{۱-۱۳}$$

مقدار میانگین هر یک از این ذرات را می‌توان به سادگی محاسبه نمود

$$\bar{x} = (0.3,0.7,0.6,0.5,0.8,0.3) \tag{۲-۱۳}$$

میانگین، یک خاصیت آماری مرتبه اول است. می‌توان دید احتمال آنکه اولین بیت از این زیرجمعیتِ نسبتاً برازنده ۱ باشد تنها برابر ۳۰٪ است. بنابراین، هنگامی که ما جمعیت بعدی را تولید می‌نماییم، اولین بیت هر ذره باید با احتمال ۳۰٪ یک بوده و با احتمال ۷۰٪ صفر باشد. به همین ترتیب می‌توان دید که دومین بیت دارای ۷۰٪ احتمال یک بودن و ۳۰٪ احتمال صفر بودن می‌باشد. بنابراین، هنگامی که جمعیت بعدی را تولید می‌نماییم، بیت دوم باید دارای ۷۰٪ احتمال یک بودن باشد.

با این حال، می‌توان از خواص آماری مرتبه دوم نیز استفاده نمود. توجه داشته باشید که اگر بیت اول (چپ‌ترین بیت) در معادله‌ی (۱۳-۱)، $x_i(1) = 1$ باشد، آنگاه دومین بیت $x_i(2)$ تنها دارای $1/3$ احتمال یک بودن خواهد بود. همچنین اگر در معادله‌ی (۱۳-۱)، $x_i(1) = 0$ باشد، آنگاه بیت دوم دارای $6/7$ احتمال یک بودن خواهد بود. به نظر می‌رسد نوعی ارتباط و همبستگی بین مقادیر بیت اول و دوم وجود دارد. بنابراین شاید بهتر باشد به جای آنکه احتمال یک بودن بیت دوم را برابر ۷۹٪ قرار دهیم، منتظر بمانیم تا ابتدا مقدار بیت اول مشخص شود. سپس، اگر مقدار بیت اول برابر ۱ بود، احتمال یک بودن مقدار بیت دوم را برابر $1/3$ قرار داده و اگر بیت اول صفر بود، احتمال یک بودن بیت دوم را برابر $6/7$ قرار دهیم.

توجه داشته باشید که می‌توان از خواص آماری مرتبه‌ی سوم و مراتب بالاتر نیز برای تولید جمعیت نسل بعد استفاده نمود. برای مثال، می‌توان دید که اگر بیت چهارم و پنجم به ترتیب برابر ۰ و ۱ باشند، آخرین بیت همواره صفر خواهد بود.

۱۳-۲ الگوریتم‌های تخمین توزیع مرتبه اول

این بخش سه EDA مرتبه اول از جمله الگوریتم توزیع حاشیه‌ای تک‌متغیره (UMDA) در بخش ۱۳-۲-۱، الگوریتم ژنتیک فشرده (cGA) در بخش ۱۳-۲-۲ و یادگیری افزایشی جمعیت-محور (PBIL) در بخش ۱۳-۲-۳ را ارائه می‌نماید.

۱۳-۲-۱ الگوریتم توزیع حاشیه‌ای تک‌متغیره

الگوریتم توزیع حاشیه‌ای تک متغیره (UMDA) بنیادی‌ترین نوع EDA است که در اواخر دهه‌ی ۱۹۹۰ توسط هاینز موهلنبین[1] برای مسائل دودیی معرفی شد [موهلنبین و پا[2]، ۱۹۹۶]، [موهلنبین و شلیرکمپ-ووسن[3]، ۱۹۹۷]. این الگوریتم تنها از خواص آماری مرتبه اول برای تولید جمعیت نسل بعد استفاده می‌نماید. شکل ۱۳-۲ طرح کلی UMDA برای مسائل بهینه‌سازی دودویی را نشان می‌دهد.

هرچند UMDA استاندارد شامل نخبه‌گرایی نمی‌شود، می‌توان نخبه‌گرایی را در آن، مانند هر الگوریتم تکاملی دیگر به کار برد. پارامتر نخبه‌گرایی e به معنای آن است که e ذره‌ی برتر در هر نسل را برای نسل بعد نگه می‌داریم. این کار باعث می‌شود مطمئن باشیم که بهترین ذره در هر نسل از بهترین ذره در نسل پیشین بدتر نخواهد بود و بدین ترتیب بهبود دائمی را از یک نسل به نسل دیگر تضمین می‌کند.

[1] Heinz Muhlenbein
[2] paaβ
[3] Schlierkamp-Voosen

یک جمعیت آغازین از راه‌حل‌های نامزد را مقداردهی کن $\{x_i\}$، $i \in [1, N]$
توجه داشته باش که هر x_i شامل n بیت می‌باشد: $x_i(1) \dots x_i(n)$
تا زمانی که شرایط توقف برآورده نشده است
M ذره از $\{x_i\}$ را متناسب با برازندگیشان انتخاب کن، $M < N$
M ذره‌ی انتخاب شده را به‌صورت $\{x_i\}, i \in [1, M]$ اندیس‌گذاری کن
$\Pr(x(k) = 1) \leftarrow \sum_{i=1}^{M} \delta(x_i(k) - 1)/M$ برای $k \in [1, n]$
برای $i = 1$ تا N
برای $k = 1$ تا n
$r \leftarrow U[0,1]$
اگر $r < \Pr(x(k) = 1)$
$x_i(k) \leftarrow 1$
در غیر این صورت
$x_i(k) \leftarrow 0$
پایان اگر
بیت بعدی
ذره‌ی بعدی
نسل بعد

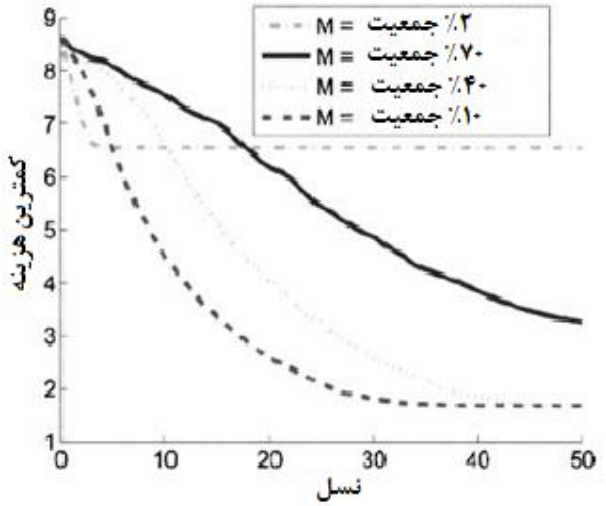
شکل ۱۳-۲ طرح بنیادی الگوریتم توزیع حاشیه‌ای تک‌متغیره (UMDA) برای بهینه‌سازی بر روی دامنه‌ی دودویی n-بیتی. $\delta(y)$ تابع دلتای کرونکر[1] است ($\delta(y) = 1$ اگر $y = 0$ و $\delta(y) = 0$ اگر $y \neq 0$). $U[0, 1]$ عددی اتفاقی با توزیع یکنواخت میان ۰ و ۱ بوده و $x_i(k)$، kاُمین بیت از iاُمین ذره می‌باشد.

مثال ۱۳-۱

در این مثال از UMDA برای مینیمم‌سازی تابع ۲۰ بعدی آکلی که در ضمیمه‌ی ج.۲-۱ تعریف شده است، استفاده می‌نماییم. ما از شش بیت در هر بعد استفاده خواهیم نمود. بدین ترتیب مسئله شامل $n = 120$ بیت خواهد بود. ما از دامنه‌ی $[-5, +5]$ برای هر بعد استفاده می‌نماییم که در این صورت در هر بعد دقتی برابر با $10/(2^6 - 1) = 0.16$ خواهیم داشت. ما همچنین از اندازه‌ی جمعیتی برابر با $N = 100$ استفاده کرده و پارامتر نخبه‌گرایی را برابر دو قرار می‌دهیم، بدین معنا که از هر نسل به نسل دیگر بهترین دو ذره را حفظ

[1] Kronecker

خواهیم نمود. این مثال را برای چهار مقدار مختلف از M حل نموده‌ایم ($M = 2, 10, 40, 70$). شکل ۱۳–۳ عملکرد UMDA را که بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. نتیجه‌ی نشان داده شده در شکل ۱۳–۳ حاکی از آن است که اگر از تعداد بسیار زیاد و یا بسیار کمِ ذرات برای محاسبه‌ی احتمالات استفاده نماییم، عملکرد مطلوبی حاصل نخواهد شد. بنابراین باید از مقدار مشخصی از ذرات برای محاسبه‌ی احتمالات استفاده نمود تا عملکرد مطلوب حاصل شود.



شکل ۱۳–۳ مثال ۱۳–۱: نتایج UMDA برای مینیمم‌سازی تابع ۲۰ بعدی اَکلی با شش بیت در هر بعد. هر منحنی هزینه‌ی بهترین ذره در هر نسل را که بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد.

می‌توان با وزن‌دهی به میزان تأثیر هر ذره، محاسبه‌ی بردار احتمال را اصلاح نمود. شکل ۱۳–۲ محاسبه‌ی بردار احتمال را به‌صورت زیر نشان می‌دهد:

$$\Pr(x(k) = 1) \leftarrow \frac{1}{M}\sum_{i=1}^{M} \delta(x_i(k) - 1), \quad k \in [1, n] \qquad (۱۳–۳)$$

در معادله‌ی بالا، هر یک از M ذره از جمعیت، تأثیر یکسانی بر محاسبه‌ی بردار احتمال دارند. اما منطقی است به ذرات بهتر وزن سنگین‌تر و به ذرات بدتر وزن سبک‌تری برای تأثیرگذاری بر بردار احتمال داده شود. این تعمیم چیزی است شبیه به بهینه‌سازی تجمع ذرات کاملاً آگاه. PSO استاندارد از یک همسایگی با اندازه‌ی مشخص برای تنظیم سرعت هر ذره استفاده می‌کند اما PSO کاملاً آگاه (بخش ۱۱–۵) از کل جمعیت برای

تنظیم سرعت هر ذره استفاده می‌نماید. این ایده منجر به جایگزینی معادله‌ی (۱۳-۳) با معادله‌ای مانند معادله‌ی زیر می‌شود:

$$\Pr(x(k)=1) \leftarrow \sum_{i=1}^{N} w_i \delta(x_i(k)-1) \Big/ \sum_{i=1}^{N} w_i, \quad k \in [1,n] \qquad (۱۳-٤)$$

که در آن w_i با میزان برازندگی x_i متناسب است.

۱۳-۲-۲ الگوریتم ژنتیک فشرده (cGA)

الگوریتم ژنتیک فشرده (cGA) توسط جورج هاریک[1]، فِرناندو لوبو[2] و دیوید گلدبرگ[3] ایجاد شده است [هاریک و همکاران، ۱۹۹۹]. همان‌طور که از نام آن پیداست، این الگوریتم روشی مینیمالیستی برای محاسبات تکاملی است. اگرچه عبارت GA در نام آن وجود دارد، اما الگوریتم cGA بیشتر یک EDA محسوب می‌شود تا یک GA. همانند UMDA، این الگوریتم نیز تنها از خواص آماری مرتبه‌ی اول برای تولید نسل بعد استفاده می‌نماید. در یک مسئله‌ی بهینه‌سازی که بر روی دامنه‌ی دودویی تعریف شده است، کار این الگوریتم با مقداردهی اولیه‌ی یک بردار احتمال n-عنصریِ که با p نشان می‌دهیم، آغاز می‌شود. مقدار اولیه‌ی تمامی عناصر این بردار برابر $1/2$ قرار داده می‌شود. سپس دو ذره‌ی x_1 و x_2 با استفاده از p ایجاد می‌شوند. بردار p مقادیر احتمال هر یک از عناصر این دو بردار را مشخص می‌نماید. سپس برازندگی هر دو ذره اندازه گرفته می‌شود. اگر یکی از ذرات از دیگری برازنده‌تر بوده و دو ذره در بیت iاُم متفاوت باشند، آنگاه عنصر iاُم بردار p متعاقباً تنظیم خواهد شد. سپس از بردار p به روزرسانی شده در نسل بعد استفاده خواهد شد. شکل ۱۳-٤ الگوریتم بنیادی cGA را نشان می‌دهد.

[1] Goerge Harik
[2] Fernando Lobo
[3] David Goldberg

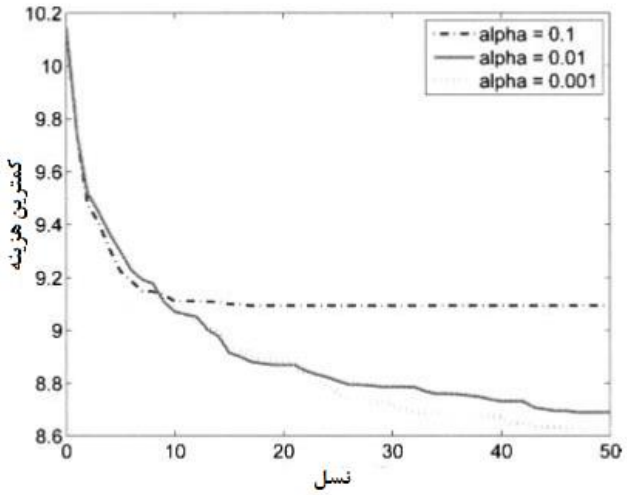
بردار n– عنصری احتمال $p = [0.5, \dots, 0.5]$ را مقداردهی کن
مقدار p_{min} و p_{max} را به ترتیب برابر مقادیر کمینه و بیشینه بریا هعر عنصر p قرار بده
مقدار α (افزایش به‌روزرسانی احتمال) را تعیین کن
تا زمانی که شرایط توقف برآورده نشده است
برای 2 تا $i = 1$ (اندازه جمعیت)
برای n تا $k = 1$ (تعداد بیت‌های هر راه‌حل نامزد)
$r \leftarrow U[0,1]$
اگر $r < \Pr(x(k) = 1)$
$x_i(k) \leftarrow 1$
در غیر این صورت
$x_i(k) \leftarrow 0$
پایان اگر
بیت بعدی
ذره‌ی بعدی
x_1 و x_2 را ارزیابی کرده و سپس به آن‌ها را به‌گونه‌ای مرتب کن که x_1 برازنده‌تر از x_2 باشد
برای n تا $k = 1$ (تعداد بیت‌های هر راه‌حل نامزد)
اگر $x_1(k) \neq x_2(k)$ آنگاه
$p(k) \leftarrow p(k) + \alpha$
در غیر این صورت
$p(k) \leftarrow p(k) - \alpha$
پایان اگر
بیت بعدی
نسل بعد

شکل ۱۳–٤ طرح کلی الگوریتم ژنتیک فشرده (cGA) برای بهینه‌سازی بر روی دامنه‌ی n–بعدی دودویی. $U[0,1]$ عددی اتفاقی با توزیع یکنواخت میان ۰ و ۱ بوده و $\alpha \in (0,,1)$ سرعت همگرایی را کنترل می‌کند. $x_i(k)$ نیز نشانگر kاُمین بیت از iاُمین ذره می‌باشد.

مانند سایر الگوریتم‌های تکاملی می‌توان از نخبه‌گرایی در cGA نیز استفاده نمود. در این صورت، بهترین ذره‌ی هر نسل با دو ذره‌ی جدید ایجاد شده در نسل بعد همراه شده و بهترین ذره از میان این سه ذره برای تنظیم بردار احتمال استفاده می‌شود.

مثال ۱۳-۲

در این مثال نیز به بهینه‌سازی تابع ۲۰ بعدی اَکلی، این بار با استفاده از cGA از شکل ۱۳-٤ می‌پردازیم. پارامترهای مسئله مانند پارامترهای مثال ۱۳-۱ می‌باشد: شش بیت در هر بعد، دامنه‌ی $[-5,+5]$ برای هر یک از ۲۰ بعد مسئله با دقتی برابر با $10/(2^6-1)=0.16$ برای هر بعد. در این مسئله از اندازه‌ی جمعیتی برابر با $N=2$ استفاده می‌نماییم (به اقتضای الگوریتم cGA). در این مسئله ما از $p_{min}=0.05$ و $p_{max}=0.95$ استفاده نموده‌ایم. ما همچنین از نخبه‌گرایی نیز استفاده نموده‌ایم، بدین معنا که از هر نسل به نسل دیگر بهترین ذره را حفظ نموده‌ایم. همچنین برای α از سه مقدار مختلف ۰,۰۰۱، ۰,۰۱ و ۰,۱ بهره برده‌ایم. شکل ۱۳-۵ عملکرد cGA را که بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. می‌توان دید که اگر α بسیار بزرگ باشد، پرش‌های زیادی در فضای جستجو بوده و همگرایی ضعیف خواهد بود. از طرف دیگر، اگر α بسیار کوچک باشد، بردار احتمال در نسل‌های ابتدایی همگرایی آهسته‌تری داشته اما در بلندمدت عملکرد بهتری حاصل می‌شود. حتی با بهترین مقدار α، همگرایی به خوبی UDMA از مثال ۱۳-۱ نخواهد بود. با این حال باید توجه کرد که cGA تنها به دو ذره‌ی جدید و یک بار مقایسه‌ی تابع برازندگی در هر نسل نیاز دارد. بنابراین، مقایسه‌ی UMDA و cGA با تعداد نسل‌های برابر عادلانه نخواهد بود. راه عادلانه‌تر برای مقایسه‌ی این دو الگوریتم استفاده از تعداد محاسبات تابع برازندگی به‌عنوان معیار است (مسائل ۱۳-۳ و ۱۳-۱۲ را ببینید).



شکل ۱۳–۵ مثال ۱۳–۲: نتایج مینیمم‌سازی تابع ۲۰ بعدی اَکلی با استفاده از الگوریتم cGA با شش بیت در هر بعد. هر منحنی هزینه‌ی بهترین ذره در هر نسل را که بر روی ۵۰ شبیه‌سازی مونت کارلو میان‌گیری شده است، نشان می‌دهد.

مثال ۱۳–۲ نتایج حاصل از به کار بردن نخبه‌گرایی همراه با cGA را نشان می‌دهد. خواننده می‌تواند نتایج این مسئله را با نتایج آزمایش خود که در آن از نخبه‌گرایی استفاده نشده است مقایسه کند تا تأثیر مخرب عدم استفاده از نخبه‌گرایی بر عملکرد الگوریتم را مشاهده نماید. p_{min} و p_{max} نیز می‌توانند تأثیری قوی بر عملکرد cGA داشته باشند. در آخر، توجه داشته باشید که می‌توان از بیش از دو ذره در هر نسل استفاده نمود. اگر بیش از دو ذره در هر نسل تولید نماییم، می‌توانیم بردار احتمال را با مقایسه‌ی بهترین و بدترین ذره اصلاح نماییم. شکل ۱۳–٦ نسخه‌ی تعمیم‌یافته‌ی cGA را نشان می‌دهد.

بردار n– عنصری احتمال $p = [0.5, \ldots, 0.5]$ را مقداردهی کن
مقدار p_{min} و p_{max} را به ترتیب برابر مقادیر کمینه و بیشینه بریا هعر عنصر p قرار بده
مقدار α (افزایش به‌روزرسانی احتمال) را تعیین کن
مقدار N (اندازه‌ی جمعیت) را تعیین کن
ذره‌ی نخبه را به‌صورت (بردار تهی) $x_e \leftarrow \emptyset$ مقداردهی کن
تا زمانی که شرایط توقف برآورده نشده است
  برای N تا $i = 1$ (اندازه جمعیت)
    برای n تا $k = 1$ (تعداد بیت‌های هر راه‌حل نامزد)

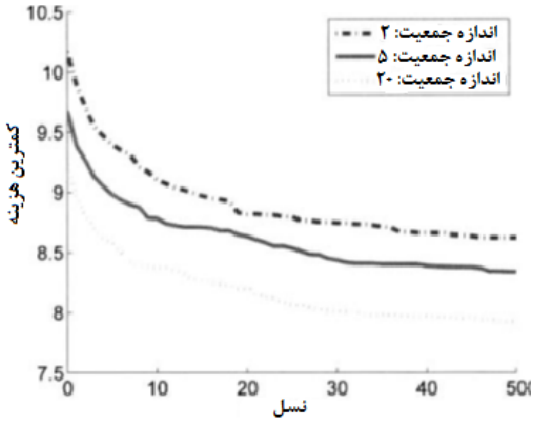
$r \leftarrow U[0,1]$
اگر $r < \Pr(x(k) = 1)$
$x_i(k) \leftarrow 1$
در غیر این صورت
$x_i(k) \leftarrow 0$
پایان اگر
بیت بعدی
ذره‌ی بعدی
بهترین از میان $x_{best} \leftarrow \{x_e, x_1, x_2, \dots, x_N\}$
بدترین از میان $x_{worst} \leftarrow \{x_e, x_1, x_2, \dots, x_N\}$
برای $k = 1$ تا n (تعداد بیت‌های هر راه‌حل نامزد)
اگر $x_{best}(k) \neq x_{worst}(k)$ آنگاه
اگر $x_{best}(k) = 1$ آنگاه
$p(k) \leftarrow p(k) + \alpha$
در غیر این صورت
$p(k) \leftarrow p(k) - \alpha$
پایان اگر
$p(k) \leftarrow \max(min(p(k), p_{max}), p_{min})$
پایان اگر
بیت بعدی
$x_e \leftarrow x_{best}$
نسل بعد

شکل ۱۳–٦ نسخه‌ی تعمیم‌یافته از الگوریتم ژنتیک فشرده (cGA) با نخبه‌گرایی برای بهینه‌سازی بر روی دامنه‌ی n بعدی دودویی. $U[0,1]$ عددی اتفاقی با توزیع یکنواخت میان ۰ و ۱ بوده و $\alpha \in (0,,1)$ سرعت همگرایی را کنترل می‌کند. $x_i(k)$ نیز نشانگر kاُمین بیت از iاُمین ذره می‌باشد.

مثال ۱۳–۳

در این مثال به دنبال مینیمم‌سازی تابع ۲۰ بعدی اَکلی ۲۰ بعدی با استفاده از الگوریتم cGA تعمیم‌یافته از شکل ۱۳–٦ خواهیم بود. پارامترهای این مثال مانند پارامترهای مثال ۱۳–۲ است با این تفاوت که در این مثال $\alpha = 0.001$ می‌باشد. در این مثال از مقادیر مختلف برای اندازه‌ی جمعیت استفاده نموده‌ایم:

20 و $N = 2,5$. شکل ۱۳–۷ عملکرد این الگوریتم را که بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. می‌توان دید با بزرگتر شدن اندازه‌ی جمعیت، عملکرد الگوریتم بهبود می‌یابد. با این حال، هزینه‌ی محاسباتی مستقیماً با اندازه‌ی جمعیت متناسب است. بنابراین، معیار عادلانه برای مقایسه تعداد نه تعداد نسل‌ها بلکه تعداد ارزیابی‌های تابع می‌باشد (مسئله‌ی ۱۳–٤ و ۱۳–۱۳ را ببینید).



شکل ۱۳–۷ مثال ۱۳–۳: نتایج *cGA* برای مینیمم‌سازی تابع ۲۰ بعدی اَکلی با شش بیت در هر بعد. هر منحنی هزینه‌ی بهترین ذره در هر نسل را که بر روی ۵۰ بار شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. عملکرد *cGA* و همچنین هزینه‌ی محاسباتی هر دو به‌طور مستقیم با اندازه‌ی جمعیت متناسب‌اند.

۱۳–۲–۳ یادگیری افزایشی جمعیت–محور

این بخش یادگیری افزایشی جمعیت–محور (*PBIL*) را که گونه‌ای *EDA* است که از خواص آماری مرتبه‌ی اول استفاده می‌نماید، ارائه می‌کند. *PBIL* در واقع تعمیمی از *UMDA* است. *PBIL* در [بالوجا[1]، ۱۹۹٤] و [بالوجا و کاروآنا[2]، ۱۹۹۵] معرفی شده است. نام‌های دیگر این الگوریتم، تپه‌نوردی با یادگیری (*HCwL*[3]) [کواسنیکا[4] و همکاران، ۱۹۹٦] و الگوریتم توزیع حاشیه‌ای تک‌متغیره‌ی افزایشی (*IUMDA*[5]) [موهلنبین و شلیرکمپ–ووسن، ۱۹۹۷] می‌باشد. در یک مسئله‌ی بهینه‌سازی n–بعدی دودویی، *PBIL* یک بردار احتمال p تشکیل داده و از آن نگه‌داری می‌کند. kاُمین عنصر این بردار احتمال ۱ بودن kاُمین بیت

[1] Baluja
[2] Karuana
[3] Hill Climbing with Learning
[4] Kvasnicka
[5] Incremental Univariate Marginal Distribution Algorithm

راه‌حل نامزد را مشخص می‌نماید. ایده‌ی *PBIL* از یادگیری مسابقه‌ای که نوعی روش ساده‌ی یادگیری در شبکه‌های عصبی مصنوعی می‌باشد گرفته شده است [فاست[1]، ۱۹۹٤].

در هر نسل از بردار احتمال p برای تولید اتفاقی یک راه‌حل نامزد استفاده می‌شود. سپس برازندگی هر راه‌حل سنجیده می‌شود و بعد از آن بردار احتمال به نحوی اصلاح می‌شود که نسل بعد بیشتر شبیه ذرات با بیشترین برازندگی بوده و کمتر شبیه ذرات با کمترین برازندگی باشند. بنابراین از این بردار احتمال جدید برای ایجاد یک جمعیت اتفاقی دیگر از راه‌حل‌های نامزد در نسل بعد استفاده می‌شود. این فرایند آنقدر ادامه می‌یابد تا شرط همگرایی که توسط کاربر تعریف شده است برآورده گردد. شکل ۱۳–۸ یک الگوریتم *PBIL* بنیادی برای بهینه‌سازی مسائل دودویی n–بعدی را نشان می‌دهد.

اندازه جمعیت $= N$

تعداد ذرات خوبی که برای تنظیم p استفاده شده‌اند $= N_{best}$

تعداد ذرات بدی که برای تنظیم p استفاده شده‌اند $= N_{worst}$

بیشترین مقدار مجاز $p = [0,1] \in p_{max}$

کمترین مقدار مجاز $p = [0,1] \in p_{min}$

نرخ یادگیری $= \eta \in [0,1]$

بردار n– عنصری احتمال $p = [0.5, \dots, 0.5]$ را مقداردهی کن

تا زمانی که شرایط توقف برآورده نشده است

از p جهت تولید N ذره‌ی اتفاقیِ $\{x_i\}$ به‌صورت زیر استفاده کن

برای $i \in [1, N]$ (برای هر ذره)

برای $k = 1$ تا n (تعداد بیت‌های هر راه‌حل نامزد)

$r \leftarrow U[0,1]$

اگر $r < \Pr(x(k) = 1)$

$x_i(k) \leftarrow 1$

در غیر این صورت

$x_i(k) \leftarrow 0$

پایان اگر

بُعد k بعدی

[1] Fausset

ذره‌ی i بعدی
ذرات را به‌گونه‌ای مرتب کن که $f(x_1) < f(x_2) < \cdots < f(x_N)$
برای $i \in [1, N_{best}]$
$p \leftarrow p + \eta(x_i - p)$
i بعدی
p را به‌صورت احتمالاتی دچار جهش کن
$p \leftarrow \max(min(p, p_{max}), p_{min})$
نسل بعد

شکل ۱۳–۸ یک الگوریتم ساده‌ی *PBIL* برای مینیمم‌سازی تابع $f(x)$ که دارای دامنه‌ی دودویی n–بعدی بوده و $x_i(k) \in \{0, 1\}$، kاُمین عنصر از iاُمین ذره می‌باشد.

شکل ۱۳–۸ چندین پارامتر میزان‌سازی را در الگوریتم *PBIL* نشان می‌دهد.

- در این الگوریتم نیز باید مانند سایر الگوریتم‌های تکاملی در مورد اندازه‌ی جمعیت N تصمیم گرفته شود.
- پارامترهای N_{best} و N_{worst} از دیگر پارامترهایی است که باید تعیین گردند. این دو پارامتر تعداد ذراتی را مشخص می‌کنند که در هر نسل از آن‌ها برای اصلاح بردار احتمال استفاده می‌شود. مقادیر بزرگ (نزدیک به $N/2$) برای این پارامترها منجر به فرایند تکاملی راکد و بسیار کند می‌شود. مقادیر کوچک (نزدیک به ۱) برای این پارامترها به فرایند یادگیری پرتکاپو منجر خواهد شد.
- نرخ یادگیری η دیگر پارامتری است که باید تعیین شود. تأثیر این پارامتر در تقابل با تأثیری است که N_{best} و N_{worst} دارند. یک η کوچک به بهینه‌سازی کند و η بزرگ به بهینه‌سازی سریع منجر می‌شود. اگر بهینه‌سازی زیادی سریع باشد، آنگاه PBIL از راه‌حل بهینه بالاتر خواهد جهید.
- انتخاب الگوریتم جهش دیگر پارامتری است که باید میزان‌سازی شود (بخش ۸–۹ را ببینید).

از شکل ۱۳–۸ می‌توان دید که جمعیت از یک نسل به نسل دیگر حفظ نمی‌شود. در عوض، اثر بردار احتمال نگه‌داریشده و بر اساس آن یک جمعیت جدید در ابتدای هر نسل تشکیل می‌شود.

از شکل ۱۳–۸ همچنین می‌توان دید که بردار احتمال به‌گونه‌ای تنظیم می‌گردد که ذره‌های نسل بعد بیشتر شبیه ذرات با برازندگی بالا بوده و کمتر شبیه ذرات با برازندگی کم باشند. این موضوع در مورد حالت دو بعدی در شکل ۱۳–۹ نشان داده شده است. در این شکل x_1 ذره‌ای است با برازندگی بالا و x_N ذره‌ای است

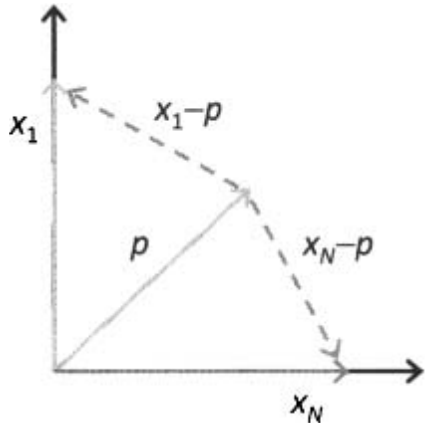
با برازندگی پایین. توجه داشته باشید که اگر ما ضریبی از $(x_1 - p)$ را به p اضافه نماییم، نتیجه یک p جدید خواهد بود که به x_1 نزدیکتر شده است:

$$p_{new} \leftarrow p + \eta(x_1 - p)$$
$$||p_{new} - x_1|| < ||p - x_1||_2 \qquad (۵-۱۳)$$

که در آن $\eta \in (0,1)$ نرخ یادگیری است. برعکس، شکل ۱۳-۹ نشان می‌دهد که اگر ضریبی از $(x_N - p)$ را از p کم کنیم، نتیجه یک p جدید خواهد بود که از x_N دور شده است:

$$p_{new} \leftarrow p + \eta(x_N - p)$$
$$||p_{new} - x_N||_2 < ||p - x_N||_2 \qquad (۶-۱۳)$$

الگوریتم PBIL معادلات (۱۳-۵) و (۱۳-۶) را برای تعدادی از ذرات نامزد (N_{best} ذره‌ی خوب و N_{worst} ذره‌ی بد) در فضای جستجویی که معمولاً دارای بعدد بالایی است، ترکیب می‌کند. این باعث خواهد شد تا بردار احتمال به ذرات خوب نزدیکتر شده و از ذرات بد دورتر شود. به همین علت، نسل‌های بعدی به ذرات خوب نزدیک و از ذرات بد دورتر خواهند شد.



شکل ۱۳-۹ تنظیم بردار احتمال برای مسئله‌ی بهینه‌سازی دوبعدی. بردار p به‌گونه‌ای تنظیم می‌گردد که به ذره‌ی خوب x_1 نزدیک شده و از ذره‌ی بد x_N دور شود.

۱۳-۳ الگوریتم‌های تخمین توزیع مرتبه‌ی دوم

برای ما ایده‌آل آن است که از کل توزیع احتمالی بهترین M ذره برای تولید نسل بعد استفاده نماییم. با این حال، این کار به لحاظ محاسباتی غیرعملی است. بنابراین از این سخت‌گیری جهت سادگی چشم‌پوشی می‌کنیم. EDAهایی همچون UMDA، cGA و PBIL که در بخش قبل معرفی گردیدند از خواص آماری مرتبه اول برای تولید نسل بعدی استفاده می‌نمایند. این کار باعث سادگی و عملی بودن الگوریتم می‌شود.

الگوریتم‌های EDA این بخش کمی سخت‌گیرتر عمل کرده و پیچیدگی بیشتری را با استفاده از خواص آماری مرتبه دوم می‌پذیرند. بخش ۱۳-۳-۱ به بحث در مورد الگوریتم بیشینه‌سازی اطلاعات متقابل برای خوشه‌سازی ورودی (MIMIC) پرداخته، بخش ۱۳-۳-۲ الگوریتم ترکیب بهینه‌سازها با استفاده از درخت اطلاعات متقابل (COMIT) را مورد بحث قرار داده و بخش ۱۳-۳-۳ الگوریتم توزیع حاشیه‌ای دومتغیره (BMDA) را ارائه خواهد نمود.

۱۳-۳-۱ بیشینه‌سازی اطلاعات متقابل برای خوشه‌سازی ورودی

این بخش به بحث در مورد الگوریتم بیشینه‌سازی اطلاعات متقابل برای خوشه‌سازی ورودی (MIMIC) که از خواص آماری مرتبه دوم استفاده کرده و توسط جرمی دوبونت[1]، چارلز ایزبل[2] و پاول وایولا[3] ایجاد شده است [دوبونت و همکاران، ۱۹۹۷] می‌پردازد. تابع چگالی احتمالِ (PDF) یک ذره‌ی اتفاقی مانند x را می‌توان به‌صورت زیر نوشت

$$\begin{aligned} &p(x) = p\big(x(1), x(2), \dots, x(n)\big) \\ &= p\big(x(1)\big|x(2), x(3), \dots, x(n)\big)p\big(x(2)\big|x(3), x(4), \dots, x(n)\big) \dots \\ &p\big(x(n-1)\big|x(n)\big)p(x(n)) \end{aligned} \qquad (۷-۱۳)$$

که در آن $x(k)$، kاُمین بیت از یک ذره‌ی نسبتا برازنده می‌باشد. برایی مثال، اگر از مشاهده‌ی بهترین M ذره متوجه شویم که اگر بیت ۵، ۸، ۹، ١٤ و ۱۵ به ترتیب ۰، ۱، ۱، ۱ و ۰ باشند بیت چهارم دارای ۷۸٪ احتمال یک بودن است، آنگاه می‌توانیم از این اطلاعات برای ایجاد نسل بعد استفاده کنیم. با این حال، برای مسائلی که بیشتر از چند بیت در آن دخیل هستند، محاسبه‌ی توزیع کامل واقع‌گرایانه نخواهد بود. به همین دلیل است که UMDA، cGA و PBIL از خواص آماری مرتبه اول استفاده می‌نمایند. در این الگوریتم‌ها فرض بر آن است که PDF جمعیت را می‌توان با خواص آماری مرتبه اول تخمین زد:

$$\hat{p}(x) = p\big(x(1)\big)p\big(x(2)\big) \dots p(x(n)) \approx p(x) \qquad (۸-۱۳)$$

الگوریتم MIMIC سعی بر پیدا کردن تقریبی بهتر از معادله‌ی (۱۳-۸) دارد:

$$\hat{p}(x) = p(x(k_1)|x(k_2))p(x(k_2)|x(k_3)) \dots p\big(x(k_{n-1})\big|x(k_n)\big)p\big(x(k_n)\big) \qquad (۹-۱۳)$$

[1] Jeremy de Bonet
[2] Charles Isbell
[3] Paul Viola

که در آن $(k_1, k_2, \dots, k_n)$ جایگشتی است از $\{1,2, \dots, n\}$. به یاد آورید که جایگشت جایه‌جایی اعداد صحیح است. برای مثال، اگر $n = 5$ باشد آنگاه (4,5,1,3,2) و (5,1,2,4,3) هر دو جایگشت‌هایی از $\{1,2,3,4,5\}$ خواهند بود. مسئله‌ای که MIMIC سعی در حل آن دارد تعیین $(k_1, k_2, \dots, k_n)$ به نحوی است که معادله‌ی (۱۳–۹) تا حد ممکن به PDF حقیقی $p(x)$ نزدیک شود. سپس MIMIC از $\hat{p}(x)$ برای ایجاد یک جمعیت جدید از راه‌حل‌های نامزد در هر نسل استفاده می‌نماید.

قبل از آنکه بتوانیم جایگشتی را پیدا کنیم که خطای تقریب $\hat{p}(x)$ را مینیمم کند، ابتدا باید خطای تقریب را تعریف نماییم. میزان شباهت دو تابع چگالی احتمال $p(x)$ و $\hat{p}(x)$ را می‌توان با استفاده از همگرایی کولبَک–لِیبلِر[1] اندازه گرفت [بیشاپ، ۲۰۰۶]:

$$\begin{aligned} D(p,\hat{p}) &= \sum_x p(x)\log_2\left(\frac{p(x)}{\hat{p}(x)}\right) \\ &= \sum_x p(x)(\log_2 p(x) - \log_2\hat{p}(x)) \\ &= \sum_x (p(x)\log_2 p(x) - p(x)\log_2\hat{p}(x)) \end{aligned} \qquad (۱۰–۱۳)$$

که در آن سری بر روی تمام نقاط x که در آن‌ها $p(x)$ و $\hat{p}(x)$ غیرصفر هستند عمل می‌کند. بنابراین، می‌خواهیم $D(p,\hat{p})$ را نسبت به $\hat{p}$ مینیمم نماییم. اولین عبارت در سمت راست معادله‌ی (۱۳–۱۰) تابعی از $\hat{p}$ نیست. بنابراین می‌توان تابع هزینه را به‌صورت زیر نوشت:

$$\begin{aligned} J(\hat{p}) &= -\sum_x p(x)\log_2\hat{p}(x) \\ &= -\sum_x p(x)\log_2[p(x(k_1)|x(k_2))p(x(k_2)|x(k_3))\dots p(x(k_{n-1})|x(k_n))p(x(k_n))] \\ &= -\sum_x [p(x)\log_2 p(x(k_1)|x(k_2)) + p(x)\log_2 p(x(k_2)|x(k_3)) + \dots \\ &\qquad + p(x)\log_2 p(x(k_{n-1})|x(k_n)) + p(x)\log_2 p(x(k_n))] \\ &= -E[\log_2 p(x(k_1)|x(k_2))] - E[\log_2 p(x(k_2)|x(k_3))] - \cdots \\ &\qquad - E[\log_2 p(x(k_{n-1})|x(k_n))] - E[\log_2 p(x(k_n))] \end{aligned} \qquad (۱۱–۱۳)$$

که در آن $\hat{p}(x)$ را با معادله‌ی (۱۳–۹) جایگزین نموده‌ایم. حال می‌توان هزینه را به‌صورت زیر نوشت

$$J(\hat{p}) = h(k_1|k_2) + h(k_2|k_3) + \cdots + h(k_{n-1}|k_n) + h(k_n) \qquad (۱۲–۱۳)$$

[1] Kullback-Liebler

که در آن توابع آنتروپی به‌صورت زیر تعریف می‌گردند [گری، ۲۰۱۱]:

$$h(k_i) = -E\left[\log_2 p\left(x(k_i)\right)\right]$$
$$h\left(k_i|k_j\right) = -E\left[\log_2 p\left(x(k_i)|x(k_j)\right)\right] \quad (۱۳-۱۳)$$

حال مسئله‌ی ما روشن‌تر شده است. ما می‌خواهیم جایگشت $(k_1, k_2, \dots, k_n)$ از $\{1,2, \dots, n\}$ را به‌گونه‌ای بیابیم که آنتروپی ترکیبی در سمت راست معادله‌ی (۱۳-۱۲) مینیمم گردد. آنتروپی یک تک‌بیت را می‌توان به‌صورت زیر نوشت

$$h(k_i) = -\sum_{\alpha} \Pr(x_i = \alpha) \log_2 \Pr(x(i) = \alpha) \quad (۱۳-۱٤)$$

آنتروپی شرطی بیت k_i را در صورتی که بیت k_j برابر β باشد، می‌توان به‌صورت زیر نوشت:

$$\begin{aligned} h\left(x(k_i)|x(k_j) = \beta\right) &= -\sum_{\alpha} \Pr(x(k_i) = \alpha|x(k_j) \\ &= \beta)\log_2 Pr\left(x(k_i) = \alpha|x(k_j) = \beta\right) \end{aligned} \quad (۱۳-۱۵)$$

در آخر، آنتروپی شرطی k_i به شرط k_j را می‌توان به‌صورت زیر نوشت

$$h\left(x(k_i)\middle|x(k_j)\right) = -\sum_{\beta} h\left(x(k_i)|x(k_j) = \beta\right) \Pr\left(x(k_j) = \beta\right) \quad (۱۳-۱٦)$$

مثال ۱۳-٤

بگذارید چند مثال ساده از محاسبات آنتروپی در نظر بگیریم. فرض کنید در یک الگووریتم تکاملی چهار ذره‌ی سه‌بیتی در اختیار داریم:

$$x_1 = (0,0,0), x_2 = (0,0,0), x_3 = (1,0,0), x_4 = (1,1,0) \quad (۱۳-۱۷)$$

آنتروپی اولین (چپ‌ترین) بیت برابرست با

$$\begin{aligned} h(1) &= -E[\log_2 p\left(x(1)\right)] \\ &= -[\Pr(x(1) = 0) \log_2 \Pr(x(1) = 0) \\ &\quad + \Pr(x(1) = 1) \log_2 \Pr(x(1) = 1)] \\ &= -[0.5 \log_2 0.5 + 0.5 \log_2 0.5] = 1 \end{aligned} \quad (۱۳-۱۸)$$

آنتروپی بیت دوم برابرست با

$$h(2) = -E[\log_2 p\left(x(2)\right)] \quad (۱۳-۱۹)$$

$$
\begin{aligned}
&= -[\Pr(x(2)=0)\log_2 \Pr(x(2)=0) \\
&\qquad + \Pr(x(2)=1)\log_2 \Pr(x(2)=1)] \\
&= -[0.75\log_2 0.75 + 0.25\log_2 0.25] = 0.8
\end{aligned}
$$

آنتروپی بیت سوم برابرست با

$$
\begin{aligned}
&h(3) = -E[\log_2 p(x(3))] \\
&= -[\Pr(x(3)=0)\log_2 \Pr(x(3)=0) \\
&\qquad + \Pr(x(3)=1)\log_2 \Pr(x(3)=1)] \\
&= -[1\log_2 1 + 0\log_2 0] = 0
\end{aligned}
\qquad (۲۰-۱۳)
$$

که در آن از قرارداد $0\log_2 0 = 0$ استفاده نموده‌ایم. می‌توان دید که آنتروپی بیت اول بیشترین مقدار ممکن را داراست. این بدین معناست که این بیت چیزی در مورد برازنده‌ترین ذره به ما نمی‌گوید. با توجه به چهار ذره‌ای که در اختیار داریم، احتمال ۰ یا ۱ بودن بیت اول برازنده‌ترین ذره با هم برابر است. از سوی دیگر، آنتروپی سومین بیت دارای کمترین مقدار ممکن است، که بدین معنی است که سومین بیت بیشترین اطلاعات را در مورد برازنده‌ترین ذره در جمعیت در اختیار ما قرار می‌دهد. بنابر چهار ذره‌ی ارائه شده در معادله‌ی (۱۳-۱۷)، آخرین بیتِ (راست‌ترین) برازنده‌ترین ذره با احتمال ۱۰۰٪، ۰ خواهد بود.

آنتروپی‌های شرطیِ بیت اول را می‌توان به‌صورت زیر نوشت

$$
\begin{aligned}
&h(x(1)|x(2)=0) \\
&= -\Pr(x(1)=0|x(2)=0)\log_2 \Pr(x(1)=0|x(2)=0) \\
&-\Pr(x(1)=1|x(2)=0)\log_2 \Pr(x(1)=1|x(2)=0) \\
&\quad = -(2/3)\log_2(2/3) - (1/3)\log_2(1/3) = 0.92 \\
&h(x(1)|x(2)=1) \\
&= -\Pr(x(1)=0|x(2)=1)\log_2 \Pr(x(1)=0|x(2)=1) \\
&-\Pr(x(1)=1|x(2)=1)\log_2 \Pr(x(1)=1|x(2)=1) \\
&= -0\log_2 0 - 1\log_2 1 = 0
\end{aligned}
\qquad (۲۱-۱۳)
$$

می‌توان دید که اگر بیت دوم برابر ۰ باشد، آنتروپی شرطی بیت اول نسبتاً بزرگ خواهد بود. این بدین معنی است که دانستن اینکه بیت دوم برابر ۰ است، اطلاعات چندانی در مورد مقدار بیت اول در اختیار ما قرار نمی‌دهد. از سوی دیگر، اگر بیت دوم ۱ باشد، آنتروپی شرطی بیت اول کمترین مقدار ممکن را خواهد داشت. این نیز بدین معنی است که اگر بدانیم بیت دوم برابر ۱ است، آنگاه می‌توانیم به‌طور ۱۰۰٪ مقدار بیت اول را تعیین کنیم. با ترکیب این نتایج می‌توان آنتروپی شرطی بیت اول را به شرط بیت دوم به‌صورت زیر نوشت:

$$
\begin{aligned}
h(x(1)|x(2)) &= h(x(1)|x(2)=0)\Pr(x(2)=0) \\
&\quad + h(x(1)|x(2)=1)\Pr(x(2)=1) \\
&= (0.92)(3/4) + (0)(1/4) = 0.69
\end{aligned}
\qquad (۲۲-۱۳)
$$

عبارت بالا برابرست با مجموع وزن‌دهی شده از دو عبارت آنتروپی در معادله‌ی (۱۳-۲۱).

با توجه به آنچه که گفته شد، می‌خواهیم جایگشتی از $\{1,2,\ldots,n\}$ را بیابیم که معادله‌ی (۱۳-۱۲) را مینیمم کند. اما تعداد جایگشت‌های ممکن $\{1,2,\ldots,n\}$ بسیار زیاد است. در کل، تعداد جایگشت‌های $\{1,2,\ldots,n\}$ برابر با $n!$ است. این عدد حتی برای مقادیر کوچک n بسیار بزرگ است. بنابراین، جستجوی brute-force برای پیدا کردن بهترین جایگشت ممکن نخواهد بود. در عوض از یک الگوریتم greedy [دوبونت و همکاران، ۱۹۹۷] استفاده خواهیم نمود که به‌طور تقریبی معادله‌ی (۱۳-۱۲) را مینیمم کرده و به سرعت تقریبی خوب از $\hat{p}(x)$ به دست می‌دهد. الگوریتم greedy در شکل ۱۳-۱۰ نشان داده شده است. اولین قدم، پیدا کردن بیت k_n است که دارای مینیمم آنتروپی (بیشترین میزان اطلاعات) است. قدم دوم، پیدا کردن بیت k_{n-1} است به‌طوری که دارای کمترین آنتروپی شرطی به شرط k_n باشد. در هر قدم، بیتی که دارای کمترین میزان آنتروپی است را به شرط بیت‌های پیدا شده‌ی قبلی می‌یابیم. باید دقت داشته باشیم از یک بیت بیش از یک بار استفاده ننماییم.

$$k_n = arg\min_j h(j)$$

برای $i = n-1, n-2, \ldots, 1$

$$k_i = arg\min_j h(j|k_{i+1}) : j \notin \{k_{i+1}, k_{i+2}, \ldots, k_n\}$$

بعدی i

شکل ۱۳-۱۰ یک الگوریتم greedy برای مینیمم‌سازی تقریبی معادله‌ی (۱۳-۱۲).

الگوریتم MIMIC زیرمجموعه‌ای از ذرات بسیار برازنده را از میان یک جمعیت انتخاب می‌نماید. این الگوریتم از الگوریتم greedy شکل ۱۳-۱۰ برای پیدا کردن راه‌حلی تقریبا بهینه برای معادله‌ی (۱۳-۱۲) استفاده می‌نماید. سپس، از این احتمالات برای ایجاد نسل بعدی راه‌حل‌های نامزد استفاده می‌نماید. شکل ۱۳-۱۱ یک الگوریتم MIMIC را برای بهینه‌سازی بر روی یک دامنه‌ی دودویی نشان می‌دهد.

یک جمعیت آغازین از راه‌حل‌های نامزد را مقداردهی کن $\{x_i\}$، $i \in [1, N]$

توجه داشته باش که هر x_i شامل n بیت می‌باشد: $x_i(1) \dots x_i(n)$

تا زمانی که شرایط توقف برآورده نشده است

M ذره از $\{x_i\}$ را متناسب با برازندگیشان انتخاب کن، $M < N$

M ذره‌ی انتخاب شده را به‌صورت $\{x_i\}, i \in [1, M]$ اندیس‌گذاری کن

$k_n = arg\min_j h(x(j))$

برای $m = n-1, n-2, \dots, 1$

$k_m \leftarrow arg\min_j h\big(x(j)\big|x(k_{m+1})\big) : j \notin \{k_n, k_{n-1}, \dots, k_{m+1}\}$

m بعدی

$\Pr(x(k_n) = 1) \leftarrow \sum_{i=1}^{M} \delta(x_i(k_n) - 1)/M$

برای $m = n-1, n-2, \dots, 1$

تعریف کن: $1_{m+1} = \{i \in [1, M] : \ x_i(k_{m+1}) = 1\}$

تعریف کن: $0_{m+1} = \{i \in [1, M] : \ x_i(k_{m+1}) = 0\}$

$\Pr(x(k_m) = 1|x(k_{m+1}) = 1) \leftarrow \sum_{i \in 1_{m+1}} \delta(x_i(k_m) - 1)/|1_{m+1}|$

$\Pr(x(k_m) = 1|x(k_{m+1}) = 0) \leftarrow \sum_{i \in 0_{m+1}} \delta(x_i(k_m) - 1)/|0_{m+1}|$

m بعدی

برای $i = 1$ تا N

$r \leftarrow U[0,1]$

اگر $r < \Pr(x(k_n) = 1)$ آنگاه $x_i(k_n) \leftarrow 1$، در غیر این صورت $x_i(k_n) \leftarrow 0$

برای $m = n-1, n-2, \dots, 1$

$r \leftarrow U[0,1]$

اگر $x_i(k_{m+1}) = 0$

اگر $r < \Pr(x(k_m) = 1|x(k_{m+1}) = 0)$

$x_i(k_m) \leftarrow 1$

در غیر این صورت

$x_i(k_m) \leftarrow 0$

پایان اگر

در غیر این صورت

اگر $r < \Pr(x(k_m) = 1|x(k_{m+1}) = 1)$

$x_i(k_m) \leftarrow 1$

در غیر این صورت
$x_i(k_m) \leftarrow 0$
پایان اگر
پایان اگر
بیت بعدی
ذره بعدی
نسل بعدی

شکل ۱۳-۱۱ طرح کلی الگوریتم MIMIC برای بهینه‌سازی بر روی دامنه‌ی دودویی. $h(y)$ آنتروپی تابع چگالی احتمال y بوده و به‌صورت تجربی از روی راه‌حل‌های نامزد تخمین زده می‌شود. $\delta(y)$ تابع دلتای کرونکر بوده و $U[0,1]$ نیز یک عدد اتفاقی با توزیع یکنواخت بین ۰ و ۱ می‌باشد.

الگوریتم MIMIC محاسبه‌ی احتمالات بسیاری را طلب می‌کند. حین پیاده‌سازی این الگوریتم لازم است اطمینان حاصل نماییم هیچ یک از این احتمالات برابر ۰ یا ۱ نخواهند بود چرا که این موضوع باعث ناتوانی الگوریتم در جستجوی کامل فضای جستجوی مسئله‌ی بهینه‌سازی خواهد شد. بنابراین، در حین پیاده‌سازی شکل ۱۳-۱۱، بعد از محاسبه‌ی هر یک از احتمالات p_i باید مقادیر احتمال را محدود نماییم:

$$p_i \leftarrow \max(p_i, \epsilon)$$
$$p_i \leftarrow \min(p_i, 1-\epsilon) \qquad (۲۳-۱۳)$$

که در آن ε یک پارامتر کوچک مثبت میزان‌سازی است و معمولاً برابر ۰٫۰۱ در نظر گرفته می‌شود. مثالی از کاربرد الگوریتم MIMIC را می‌توانید در مثال ۱۳-۷ بیابید.

۱۳-۳-۲ ترکیب بهینه‌سازها با درخت اطلاعات متقابل (COMIT)

الگوریتم COMIT در [بالوجا و دیویس، ۱۹۹۸] معرفی گردید. این الگوریتم مشابه الگوریتم MIMIC است. با این حال، در الگوریتم MIMIC ما با استفاده از مینیمم‌سازی عبارت‌های آنتروپی شرطی، جایگشت تقریباً بهینه را پیدا می‌نماییم. این در حالی است که در الگوریتم COMIT این کار را با استفاده از ماکزیمم‌سازی عبارت‌های اطلاعات متقابل انجام می‌دهیم. به عبارت دیگر، به جای یافتن جایگشتی که معادله‌ی (۱۳-۱۲) را مینیمم کند، جایگشتی را می‌یابیم که عبارت زیر را ماکزیمم می‌کند

$$J_c(\hat{p}) = I(k_1|k_2) + I(k_2|k_3) + \cdots + I(k_{n-1}|k_n) - h(k_n) \qquad (۲٤-۱۳)$$

اطلاعات متقابل میان بیت‌های k و m به‌صورت زیر تعریف می‌شود [کوور[1] و توماس[2]، ۱۹۹۱]:

$$I(k,m)=\sum \Pr(x(k)=i,x(m)=j)\log_2\left[\frac{\Pr(x(k)=i,x(m)=j)}{\Pr(x(k)=i)\Pr(x(m)=j)}\right] \qquad (۱۳-۲۵)$$

که در آن $i,j\in[0,1]$ می‌باشد.

مثال ۱۳-۵

در این مثال، که مبتنی بر مثالی در [چو[3] و لیو[4]، ۱۹٦۸] می‌باشد، نحوه‌ی محاسبه‌ی اطلاعات متقابل را نشان می‌دهیم. فرض کنید در حال اجرای یک الگوریتم تکاملی بر روی یک مسئله‌ی بهینه‌سازی چهاربیتی می‌باشیم. بنابراین، تعداد زیادی از ذرات (برای مثال ۱۰۰ ذره) در اختیار داریم. ما ۲۰ ذره را که دارای برازندگی نسبی هستند انتخاب می‌نماییم. فرض کنید این ۲۰ ذره به قرار زیر هستند:

$$\begin{array}{ll} x_1=(0,0,0,0), & x_2=(0,0,0,0)\\ x_3=(0,0,0,1), & x_4=(0,0,0,1)\\ x_5=(0,0,1,0), & x_6=(0,0,1,1)\\ x_7=(0,1,1,0), & x_8=(0,1,1,0)\\ x_9=(0,1,1,1), & x_{10}=(1,0,0,0)\\ x_{11}=(1,0,0,1), & x_{12}=(1,0,0,1)\\ x_{13}=(1,1,0,0), & x_{14}=(1,1,0,1)\\ x_{15}=(1,1,1,0), & x_{16}=(1,1,1,0)\\ x_{17}=(1,1,1,0), & x_{18}=(1,1,1,1)\\ x_{19}=(1,1,1,1), & x_{20}=(1,1,1,1) \end{array} \qquad (۱۳-۲٦)$$

بیت‌ها به ترتیب از چپ به راست شماره‌گذاری شده‌اند. برای مثال، $x_{13}(1)=x_{13}(2)=1$ و $x_{13}(3)=x_{13}(4)=0$ می‌باشد. با شمردن بیت‌ها و مراحلی که در مثال ۱۳-٤ توضیح داده شد خواهیم داشت:

$$\begin{array}{l} \Pr(x(1)=1)=0.55\rightarrow h(1)=0.993\\ \Pr(x(2)=1)=0.55\rightarrow h(2)=0.993\\ \Pr(x(3)=1)=0.55\rightarrow h(3)=0.993\\ \Pr(x(4)=1)=0.50\rightarrow h(4)=1 \end{array} \qquad (۱۳-۲۷)$$

می‌توان دید که بیت اول، دوم و سوم دارای یک مقدار از اطلاعات می‌باشند. حال مقدار اطلاعات متقابل میان بیت ۱ و سایر بیت‌ها را محاسبه می‌نماییم. معادله‌ی (۱۳-۲۵) نشان می‌دهد که قبل از آنکه بتوانیم

[1] Cover
[2] Thomas
[3] Chow
[4] Liu

اطلاعات متقابل را محاسبه نماییم، باید احتمالات بیت‌های منفرد $\Pr(x_i)$ را محاسبه کنیم. ذرات معادله‌ی (۱۳-۲۶) را در نظر بگیرید. با توجه به معادله‌ی (۱۳-۲۷) خواهیم داشت:

$$\begin{aligned}&\Pr(x(1)=0)=0.45,\quad \Pr(x(1)=1)=0.55\\&\Pr(x(2)=0)=0.45,\quad \Pr(x(2)=1)=0.55\end{aligned} \tag{۱۳-۲۸}$$

حال توجه داشته باشید که در معادله‌ی (۱۳-۲۶) شش ذره با $x(1)=0$ و $x(2)=0$، سه ذره با $x(1)=0$ و $x(2)=1$، سه ذره با $x(1)=1$ و $x(2)=0$ و هشت ذره با $x(1)=1$ و $x(2)=1$ وجود دارد. بنابراین،

$$\begin{aligned}&\Pr(x(1)=0,x(2)=0)=0.30\\&\Pr(x(1)=0,x(2)=1)=0.15\\&\Pr(x(1)=1,x(2)=0)=0.15\\&\Pr(x(1)=1,x(2)=1)=0.40\end{aligned} \tag{۱۳-۲۹}$$

حال از معادله‌ی (۱۳-۲۵) برای محاسبه‌ی اطلاعات متقابل میان بیت‌های ۱ و ۲ استفاده می‌نماییم:

$$\begin{aligned}I(1,2)&=\sum_{i,j}\Pr(x(1)=i,x(2)=j)\log_2\left[\frac{\Pr(x(1)=i,x(2)=j)}{\Pr(x(1)=i)\Pr(x(2)=j)}\right]\\&=\Pr(x(1)=0,x(2)=0)\log_2\left[\frac{\Pr(x(1)=0,x(2)=0)}{\Pr(x(1)=0)\Pr(x(2)=0)}\right]+\\&\quad\Pr(x(1)=0,x(2)=1)\log_2\left[\frac{\Pr(x(1)=0,x(2)=1)}{\Pr(x(1)=0)\Pr(x(2)=1)}\right]+\\&\quad\Pr(x(1)=1,x(2)=0)\log_2\left[\frac{\Pr(x(1)=1,x(2)=0)}{\Pr(x(1)=1)\Pr(x(2)=0)}\right]+\\&\Pr(x(1)=1,x(2)=1)\log_2\left[\frac{\Pr(x(1)=1,x(2)=1)}{\Pr(x(1)=1)\Pr(x(2)=1)}\right]\\&\quad=0.30\log_2[0.30/(0.45\times0.45)]+0.15\log_2[0.15/(0.45\times0.55)]+\\&=0.15\log_2[0.15/(0.55\times0.45)]+0.40\log_2[0.40/(0.54\times0.55)]\\&=0.1146\end{aligned} \tag{۱۳-۳۰}$$

اگر از همین روش برای محاسبه‌ی اطلاعات متقابل میان سایر بیت‌ها استفاده نماییم، نتایج زیر حاصل خواهد شد:

$$
\begin{aligned}
&I(1,2) = 0.1146\\
&I(1,3) = 0.0001\\
&I(1,4) = 0.0073\\
&I(2,3) = 0.2727\\
&I(2,4) = 0.0073\\
&I(3,4) = 0.0073
\end{aligned}
\qquad (۳۱-۱۳)
$$

اطلاعات متقابل $I(i,j)$ مقدار اطلاعات مشترک میان بیت‌های i و j را اندازه می‌گیرد. این مقدار به ما می‌گوید در صورت دانستن مقدار یک بیت، چه مقدار اطلاعات در مورد مقدار بیت دیگر خواهیم داشت. اگر مقدار بیت j از قبل معلوم باشد، ماکزیمم ساختن مقدار اطلاعات متقابل $I(i,j)$ بر روی تمام iها مانند مینیمم‌سازی آنتروپی شرطی $h(i|j)$ بر روی تمام iها خواهد بود. بنابراین می‌توان الگوریتم COMIT را به‌صورت الگوریتم MIMIC اجرا نمود تنها با این تفاوت که به جای استفاده از شکل ۱۳-۱۰ برای انتخاب جایگشت، از شکل ۱۳-۱۲ استفاده می‌نماییم. الگوریتم greedy برای ماکزیمم‌سازی معادله‌ی (۱۳-۲۵) در شکل ۱۳-۱۲ نشان داده شده است.

$$k_n = arg\min_j h(j)$$

برای $i = n-1, n-2, \dots, 1$

$$k_i = arg\max_j I(j, k_{i+1}) : j \notin \{k_{i+1}, k_{i+2}, \dots, k_n\}$$

بعدی i

شکل ۱۳-۱۲ الگوریتم greedy برای ماکزیمم‌سازی تقریبی معادله‌ی (۱۳-۲۵). این شکل را با شکل ۱۳-۱۰ مقایسه نمایید.

مثال ۱۳-۶

در این مثال، که در واقع ادامه‌ی مثال ۱۳-۵ است، الگوریتم greedy شکل ۱۳-۱۲ را نشان می‌دهیم. الگوریتم greedy ابتدا بیتی را که دارای بیشترین مقدار اطلاعات (کمترین میزان آنتروپی) است پیدا می‌نماید. در معادله‌ی (۱۳-۲۷) از مثال ۱۳-۵ مشاهده نمودیم که بیت‌های اول، دوم و سوم هر سه دارای میزان یکسانی از اطلاعات بوده و بیت چهارم دارای کمترین میزان اطلاعات می‌باشد. بنابراین، جواب قسمت $k_n = argmin_j h(j)$ در شکل ۱۳-۱۲ یا ۱ است یا ۲ و یا ۳. ما به‌طور دلخواه بیت ۱ را به‌عنوان راه‌حل برمی‌گزینیم. حال بیتی را انتخاب می‌نماییم که دارای بیشترین میزان اطلاعات مشترک با بیت ۱ است. با توجه به معادله‌ی (۱۳-۳۱) بیت ۲ دارای بیشترین میزان اطلاعات مشترک با بیت ۱ است. حال از بیت‌های باقی‌مانده بیتی را انتخاب می‌نماییم که دارای بیشترین مقدار اطلاعات مشترک با بیت ۲ است. با توجه به معادله‌ی (۱۳-۳۱)، از بیت‌های باقی‌مانده بیت سوم دارای بیشترین مقدار اطلاعات مشترک با بیت ۲ است. در این

صورت، بیت ٤ تنها بیت باقی‌مانده خواهد بود و بدین ترتیب الگوریتم greedy کامل شده و معادله‌ی (۱۳–۹) به‌صورت زیر در خواهد آمد:

$$\hat{p}(x) = p(x(1))p(x(2)|x(1))p(x(3)|x(2))p(x(4)|x(3)) \qquad (۱۳–۳۲)$$

می‌توان از معادله‌ی (۱۳–۲٦) برای محاسبه‌ی این احتمالات استفاده نمود. جدول ۱۳–۱ احتمال هر ترکیب از بیت‌ها، احتمال تخمینی حاصل شده از تقریب مرتبه اول از معادله‌ی (۱۳–۸) و احتمال تخمینی از معادله‌ی (۱۳–۳۲) را نشان می‌دهد.

جدول ۱۳–۱ نتایج مثال ۱۳–٦: احتمالات واقعی (ستون دوم) و احتمالات تخمینی (ستون‌های سوم و چهارم) از تمام ترکیبات ممکن بیت‌ها.

$x(1)x(2)x(3)x(4)$	$p(x(1),x(2),$ $x(3),x(4))$	UMDA $p(x(1))p(x(2)).$ $p(x(3))p(x(4))$	COMIT $p(x(1))p(x(2)\|x(1))$ $p(x(3)\|x(2))p(x(4)\|x(3))$
0000	0.100	0.0456	0.1037
0001	0.100	0.0456	0.1296
0010	0.050	0.0557	0.0364
0011	0.050	0.0557	0.0303
0100	0.000	0.0557	0.0121
0101	0.000	0.0557	0.0152
0110	0.100	0.0681	0.0669
0111	0.050	0.0681	0.0558
1000	0.050	0.0557	0.0519
1001	0.100	0.0557	0.0648
1010	0.000	0.0681	0.0182
1011	0.000	0.0681	0.0152
1100	0.050	0.0681	0.0323
1101	0.050	0.0681	0.0404
1110	0.150	0.0832	0.1785
1111	0.150	0.0832	0.1488

با توجه به جدول ۱۳–۱، می‌توان دید که تخمین احتمالات واقعی ترکیبات بیت‌ها (ستون دوم) در ستون چهارم با دقت بیشتری نسبت به ستون سوم صورت گرفته است. ما می‌توانیم از معادله‌ی (۱۳–۱۰) برای اندازه‌گیری میزان شباهت بین توزیعات احتمالات استفاده نماییم. بدین ترتیب می‌توان دید که اندازه‌ی نزدیکی میان احتمالات ستون‌های دوم و سوم برابر ۰,۵۳ می‌باشد. این در حالی است که میزان نزدیکی ستون‌های دوم و چهارم برابر ۰,۱٤ است.

توجه داشته باشید که ما از احتمالات جدول ۱۳–۱ در الگوریتم COMIT استفاده نمی‌کنیم. این احتمالات را در اینجا تنها برای نشان دادن تأثیرگذاری الگوریتم greedy از شکل ۱۳–۱۲، ارائه نموده‌ایم. الگوریتم

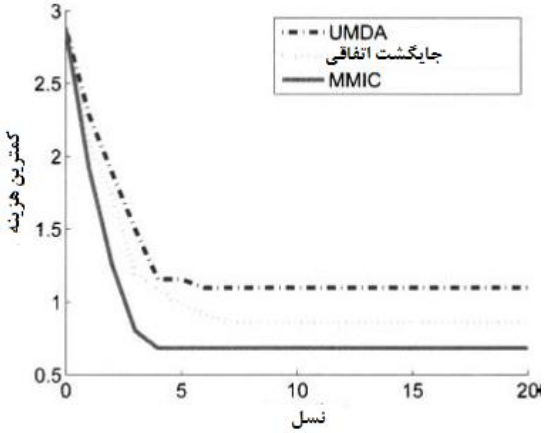
COMIT از احتمالات موجود در سمت راست معادله‌ی (۱۳-۳۲) برای ایجاد جمعیت در نسل بعد استفاده می‌نماید

مثال ۱۳-٦ چگونگی استفاده از اطلاعات متقابل (COMIT) به جای استفاده از آنتروپی شرطی (MIMIC) برای تخمین مرتبه‌ی دوم توزیع احتمال را نشان می‌دهد. اگر بتوان تخمین توزیع احتمالِ ذرات با برازندگی بالا در یک الگوریتم کاملی را به دست آورد، می‌توان از آن تخمین برای ایجاد راه‌حل‌های نامزد در هر نسل استفاده نمود. این موضوع اساس و ماهیت الگوریتم COMIT است و در مثال بعدی نشان داده شده است.

مثال ۱۳-۷

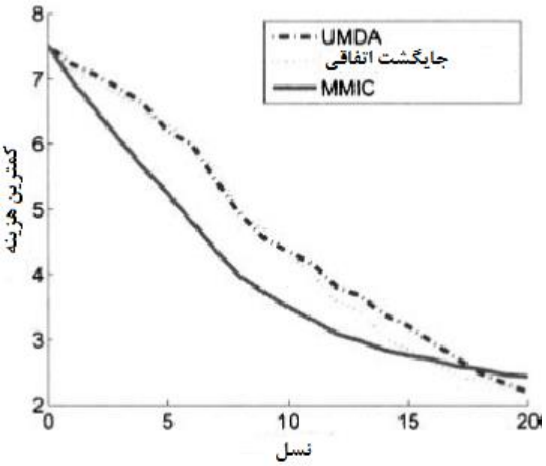
در این مثال ما از الگوریتم‌های COMIT و MIMIC از شکل‌های ۱۳-۱۰، ۱۳-۱۱ و ۱۳-۱۲ برای مینیمم‌سازی تابع اَکلی که در ضمیمه‌ی ج.۲-۱ تعریف شده است، استفاده می‌نماییم. ما از ٦ بیت در هر بعد استفاده می‌نماییم. بدین ترتیب مسئله‌ی بهینه‌سازی دارای $n = 6D$ بیت خواهد بود. D نمایانگر بعد تابع اَکلی است. ما از دامنه‌ی $[-5,+5]$ برای هر بعد استفاده می‌نماییم که در این صورت دقتی برابر با $10/(2^6 - 1) = 0.16$ برای هر بعد به دست خواهد آمد. در این مثال ما از اندازه‌ی جمعیتی برابر با $N = 100$ و $M = 40$ استفاده خواهیم نمود. همچنین پارامتر نخبه‌گرایی برای این مسئله برابر ۲ است، بدین معنی که همواره بهترین دو ذره از هر نسل را برای نسل بعد حفظ خواهیم نمود. در آخر، برای معادله‌ی (۱۳-۲۳)، $\varepsilon = 0.01$ در نظر گرفته شده است.

برای مطالعه‌ی عملکرد MIMIC آن را با UMDA (شکل ۱۳-۲) که تنها از خواص آماری مرتبه اول استفاده می‌نماید، مقایسه می‌نماییم. همچنین، ما یک الگوریتم MIMIC را به‌گونه‌ای پیاده‌سازی می‌نماییم که در آن جایگشت $(k_1, k_2, \ldots, k_n)$ به جای استفاده از الگوریتم greedy، صورت اتفاقی تعیین شود. شکل ۱۳-۱۳ عملکرد MIMIC، UMDA و MIMIC با جایگشت اتفاقی را بر روی تابع دو بعدی اَکلی، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. می‌توان دید که حتی در صورت استفاده از جایگشت اتفاقی، استفاده از خواص آماری مرتبه دوم بهتر از استفاده از خواص آماری مرتبه اول می‌باشد. با این حال، الگوریتم MIMIC بهتر از همه عمل می‌نماید چرا که از الگوریتم تقریباً بهینه‌ی greedy برای تعیین جایگشت‌های بیت‌ها استفاده می‌نماید.



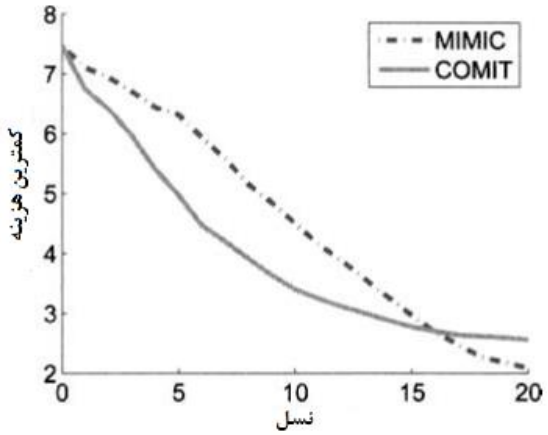
شکل ۱۳-۱۳ مثال ۱۳-۷: نتایج MIMIC و UMDA برای مینیمم‌سازی تابع دو بعدی اَکلی با شش بیت در هر بعد. نمودار هزینه‌ی بهترین ذره در هر نسل را، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد.

شکل ۱۳-۱٤ عملکرد MIMIC، UMDA و MIMIC با جایگشت اتفاقی را بر روی تابع ۱۰ بعدی اَکلی، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. می‌توان دید که در چند نسل ابتدایی MIMIC بهتر از دو الگوریتم دیگر عمل می‌نماید اما پس از گذشت چند نسل، UMDA و MIMIC با جایگشت اتفاقی خود را به MIMIC رسانده و سپس نتایج بهتری از خود نشان می‌دهند. این موضوع نشانگر آن است که هیچ تضمینی برای بهتر عمل کردن MIMIC از الگوریتم‌های مرتبه اول وجود ندارد. با این حال و بسته به مسئله، این الگوریتم ابزاری ارزشمند برای بهینه‌سازی محسوب می‌شود.



شکل ۱۳-۱٤ مثال ۱۳-۷: نتایج MIMIC و UMDA برای مینیمم‌سازی تابع ۱۰ بعدی اَکلی با شش بیت در هر بعد. نمودار هزینه‌ی بهترین ذره در هر نسل را، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد.

در آخر به مقایسه‌ی MIMIC و COMIT می‌پردازیم. هر دو الگوریتم در شکل ۱۳-۱۱ تعریف شده‌اند، تنها با این تفاوت که MIMIC از الگوریتم greedy شکل ۱۳-۱۰ و COMIT از الگوریتم greedy شکل ۱۳-۱۲ برای تصمیم‌گیری در مورد جفت بیت‌ها برای ایجاد راه‌حل‌های نامزد در هر نسل استفاده می‌نمایند. شکل ۱۳-۱۵ عملکرد MIMIC و COMIT برای مینیمم‌سازی تابع ۱۰ بعدی اَکلی را، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. می‌توان دید که COMIT در نسل‌های ابتدایی بسیار بهتر عمل می‌نماید اما با گذشت حدود ۱۵ نسل MIMIC خود را به COMIT رسانده و عملکرد بهتری از خود نشان می‌دهد.



شکل ۱۳-۱۵ مثال ۱۳-۷: نتایج MIMIC و COMIT برای مینیمم‌سازی تابع ۱۰ بعدی اَکلی با شش بیت در هر بعد. نمودار هزینه‌ی بهترین ذره در هر نسل را، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد.

۱۳-۳-۳ الگوریتم تخمین حاشیه‌ای دومتغیره (BMDA)

الگوریتم تخمین حاشیه‌ای دومتغیره (BMDA) در [پلیکان و موهلنبین، ۱۹۹۸] ایجاد شده است. BMDA نیز مانند MIMIC و COMIT از خواص آماری مرتبه دوم استفاده می‌نماید. با این حال، این الگوریتم چند تفاوت قابل ملاحظه دارد. اول آنکه این الگوریتم از آزمایش کای- دو[1] پیرسون [بوزلاف[2] و واترز[3]، ۲۰۰۸] برای ایجاد ارتباط میان بیت‌های به هم وابسته استفاده می‌نماید. دوم آنکه تخمین PDF ایجاد شده توسط این

[1] Chi-square test
[2] Boslaugh
[3] Watters

الگوریتم الزاماً از تمام بیت‌ها در یک زنجیره‌ی واحد و به هم پیوسته استفاده نمی‌کند. از معادله‌ی (۱۳-۹) به یاد آورید که MIMIC و COMIT جایگشت $(k_1, k_2, \dots, k_n)$ از $\{1,2, \dots, n\}$ را به‌گونه‌ای می‌یابند که

$$p(x) \approx p\big(x(k_1)|x(k_2)\big)p\big(x(k_2)|x(k_3)\big) \dots p\big(x(k_{n-1})|x(k_n)\big)p\big(x(k_n)\big) \qquad (۱۳-۳۳)$$

تخمین بالا را می‌توان به‌صورت زنجیره‌ای واحد از k_i مقدار مشاهده نمود:

$$k_1 \to k_2 \to \cdots \to k_{n-1} \to k_n \qquad (۱۳-۳٤)$$

از سوی دیگر، BMDA تخمینی جامع‌تر از $p(x)$ را می‌یابد:

$$p(x) \approx \prod_{x(r)\in R} p\big(x(r)\big) \prod_{x(i)\in V\backslash R} p\big(x(i)|x(m_i)\big) \qquad (۱۳-۳۵)$$

در تخمین بالا، R مجموعه‌ای از اندیس بیت‌های ریشه‌ای بوده و توسط BMDA تعیین می‌شود و V مجموعه‌ای از تمام اندیس بیت‌ها $V = \{1,2, \dots, n\}$ می‌باشد. بیت‌های $x(i)$ متعلق به مجموعه‌ی $V\backslash R$ می‌باشند. این مجموعه، مجموعه‌ی تمام اندیس‌هایی است که به R تعلق ندارند ($V\backslash R = \{i \in V : i \notin R\}$). در آخر، $m(i)$ اندیس بیتی است که توسط BMDA تعیین شده و از درجه‌ی بالای وابستگی به بیت i برخوردار است.

یک مثال می‌تواند معنی معادله‌ی (۱۳-۳۵) را روشن سازد. فرض کنید فضای جستجو دارای ۹ بیت است. معادله‌ی (۱۳-۳۳) که توسط MIMIC و COMIT استفاده می‌شود ممکن است به زنجیره‌ی زیر منجر شود

$$3 \to 9 \to 1 \to 5 \to 8 \to 2 \to 4 \to 7 \to 6 \qquad (۱۳-۳٦)$$

که خود به تخمین زیر منجر خواهد شد

$$\begin{aligned} p(x) \approx\ & p\big(x(3)|x(9)\big)p\big(x(9)|x(1)\big)p\big(x(1)|x(5)\big)p\big(x(5)|x(8)\big) \\ & \times p\big(x(8)|x(2)\big)p\big(x(2)|x(4)\big)p\big(x(4)|x(7)\big)p\big(x(7)\big) \end{aligned} \qquad (۱۳-۳۷)$$

معادله‌ی (۱۳-۳۵) که توسط BMDA مورد استفاده واقع می‌شود ممکن است به زنجیره‌های زیر منجر شود

$$\begin{aligned} & 3 \to 9 \to 1 \\ & 3 \to 5 \\ & 8 \to 2 \to 4 \to 7 \\ & 8 \to 6 \end{aligned} \qquad (۱۳-۳۸)$$

که خود به تخمین زیر منجر خواهد شد

$$\begin{aligned} & p(x) \\ & \approx p\big(x(3)\big)p\big(x(9)|x(3)\big)p\big(x(1)|x(9)\big)p\big(x(5)|x(3)\big) \\ & \times p\big(x(8)\big)p\big(x(2)|x(8)\big)p\big(x(4)|x(2)\big)p\big(x(7)|x(4)\big)p\big(x(6)|x(8)\big) \end{aligned} \qquad (۱۳-۳۹)$$

در معادلات بالا ابتدا اندیس بیت‌های ریشه (۳ و ۸) توسط BMDA تعیین گردید و سپس چند زنجیره‌ی اندیس بیت برای هر ریشه توسط BMDA تعیین گشت.

BMDA با انتخاب اتفاقی یک اندیس بیت، r، به‌عنوان اندیس ریشه کار خود را آغاز می‌کند. سپس خاصیت آماری کای- دو برای اندیس ریشه‌ی r و سایر بیت‌ها باقی‌مانده محاسبه می‌شود. خاصیت آماری کای- دو میان دو بیت r و j به‌صورت زیر محاسبه می‌گردد

$$\chi_{rj}^2 = M \sum_{\alpha,\beta} \frac{[\Pr(x(r)=\alpha, x(j)=\beta) - \Pr(x(r)=\alpha)\Pr(x(j)=\beta)]^2}{\Pr(x(r)=\alpha)\Pr(x(j)=\beta)} \qquad (۴۰-۱۳)$$

که در آن M تعداد رشته بیت‌ها بوده و جمع بر روی مقادیری از α و β صورت می‌گیرد که $\Pr(x(r)=\alpha) \neq 0$ و $\Pr(x(j)=\beta) \neq 0$. پارامتر χ_{rj}^2 میزان همبستگی میان بیت‌های r و j را اندازه می‌گیرد. مقادیر بزرگ χ_{rj}^2 حاکی از همبستگی شدید میان r و j می‌باشد. در علم آمار، $\chi_{rj}^2 < 3.84$ به‌عنوان مرزی برای استقلال بیت‌های r و j تلقی می‌شود. این مقدار از χ_{rj}^2 بیانگر آن است که به احتمال ۹۵٪ بیت‌ها از یکدیگر مستقل هستند.

پس از آنکه BMDA مقدار χ_{rj}^2 را به ازای تمامی بیت‌های باقی‌مانده (j) برای بیت ریشه‌ی r محاسبه نمود، jایی را که دارای بزرگترین مقدار χ_{rj}^2 باشد را به‌عنوان بیت بعدی زنجیره انتخاب می‌نماید. سپس، BMDA مقادیر χ_{rk}^2 و χ_{jk}^2 را به ازای تمامی بیت‌های باقی‌مانده $k \neq \{r, j\}$ محاسبه می‌نماید. هر kایی که دارای بیشترین مقادیر χ^2 باشد به‌عنوان بیت بعدی در زنجیره انتخاب خواهد شد. این فرایند تا جایی که χ^2 کمتر از یک مقدار آستانه‌ی معینی شود ادامه پیدا می‌کند. هنگامی که این اتفاق بیافتد، یک بیت ریشه‌ی دیگر به‌صورت اتفاقی انتخاب شده و فرایند برای زنجیره‌ی بعدی تکرار می‌شود. هنگامی که تمام بیت‌ها استفاده شوند، فرایند تخمین توزیع احتمال کامل شده است. الگوریتم BMDA در شکل ۱۳-۱۶ خلاصه شده است [پلیکان و موهلنبین، ۱۹۹۸].

(۱) $V \leftarrow \{1,2,\dots,n\}$
$A \leftarrow V$
(۲) عنصر اتفاقی انتخاب شده از $v \leftarrow A$
$\Pr(v)$ را به تخمین PDF اضافه کن
(۳) v را از A حذف کن
اگر $A = \emptyset$ عملیات را به پایان رسان
(٤) مقدار χ^2_{ij} را برای تمامی $i \in A$ و همه‌ی $j \in V \backslash A$ حساب کن
اگر $\max_{i,j} \chi^2_{ij} < 3.84$ برو به (۲)
(۵) $\{v, v'\} = arg \max_{i,j} \chi^2_{ij} : i \in A, j \in V \backslash A$
$\Pr(v', v)$ را به تخمین PDF اضافه کن
برو به (۳)

شکل ۱۳–۱۶ طرح اساسی یک الگوریتم توزیع حاشیه‌ای دومتغیره (BMDA) برای تولید یک تخمین PDF n–بعدی. V مجموعه‌ی ثابت تمامی اندیس بیت‌ها بوده و A مجموعه‌ای از بیت‌های در دسترس برای قرار گرفتن در زنجیره می‌باشد. مقدار 3.84 در اینجا به‌عنوان سطح اعتماد ۹۵٪ استفاده شده است.

مثال ۱۳–۸

در این مثال چند نمونه‌ی ساده از محاسبه‌ی χ^2 ارائه شده است. فرض کنید چهار ذره‌ی چهار بیتی از یک الگوریتم تکاملی در اختیار داریم:

$$x_1 = (0,0,0,1), x_2 = (0,0,0,1), x_2 = (1,0,0,0), x_2 = (1,1,0,0) \qquad (٤١–١٣)$$

احتمالات حاشیه‌ای بیت‌های ۱ و ۲ را می‌توان به‌صورت زیر یافت

$$\Pr(x(1) = 0) = 1/2, \quad \Pr(x(1) = 1) = 1/2$$
$$\Pr(x(2) = 0) = 3/4, \quad \Pr(x(2) = 1) = 1/4 \qquad (٤٢–١٣)$$

که در آن $x(1)$ به معنای بیت اول یا چپ‌ترین بیت بوده، $x(2)$ به معنای بیت بعد و به همین ترتیب می‌باشد.
احتمال مشترک بیت‌های ۱ و ۲ را می‌توان به‌صورت زیر نوشت

$$\Pr(x(1) = 0, x(2) = 0) = 1/2, \quad \Pr(x(1) = 0, x(2) = 1) = 0$$
$$\Pr(x(1) = 1, x(2) = 0) = 1/4, \quad \Pr(x(1) = 1, x(2) = 1) = 1/4 \qquad (٤٣–١٣)$$

مقدار χ^2_{12} از معادله‌ی (۱۳–٤۰) به‌صورت زیر محاسبه می‌گردد

$$\chi_{12}^2 = 4(1/24 + 1/8 + 1/24 + 1/8) = 4/3 \quad (۱۳-٤٤)$$

می‌توان دید که مقدار همبستگی میان بیت‌های ۱ و ۲ وجود دارد، اما تعداد نمونه‌های ما به قدری کم است (چهار نمونه) که نمی‌توان با اطمینان کامل گفت که این همبستگی از لحاظ آماری قابل توجه است؛ یعنی $\chi_{12}^2 < 3.84$.

حال بیت‌های ۱ و ۳ را در نظر بگیرید. در این صورت، χ_{13}^2 به‌صورت زیر محاسبه خواهد شد

$$\chi_{13}^2 = 4(0 + 0 + 0 + 0) = 0 \quad (۱۳-٤٥)$$

هیچ نوع رابطه‌ای میان بیت‌های ۱ و ۳ وجوود ندارد. این را می‌توان از معادله‌ی (۱۳-٤۱) نیز مشاهده نمود؛ بیت اول در نیمی از موارد برابر ۰ و در نیمی دیگر برابر ۱ است. این در حالی است که بیت سوم همیشه برابر ۰ است.

در آخر، بیت‌های ۱ و ٤ را در نظر بگیرید. مقدار χ_{14}^2 به‌صورت زیر محاسبه می‌شود

$$\chi_{14}^2 = 4(1 + 1 + 1 + 1) = 4 \quad (۱۳-٤٦)$$

می‌توان دید که همبستگی آماری قابل توجهی میان این دو بیت وجود دارد. با این که ما تنها چهار ذره در اختیار داریم، از آنجا که بیت ۱ و ٤ همواره مکمل هم هستند، می‌توان مطمئن بود که این دو بیت به هم وابسته‌اند.

۱۳-۴ الگوریتم‌های تخمین توزیع چندمتغیره

دیدیم که EDAهای مرتبه اول، سادگی را بر سختی ریاضی ترجیح داده‌اند. EDAهای مرتبه دوم از سوی دیگر دارای پیچیدگی بیشتری بوده اما همزمان از تأثیرگذاری بیشتری برخوردار هستند. EDAهای چندمتغیره یک گام دیگر به جلو رفته و از خواص آماری مراتب بالاتر استفاده می‌کنند. در این قسمت، یک الگوریتم از این نوع را معرفی می‌نماییم؛ الگوریتم ژنتیک فشرده‌ی تعمیم‌یافته (ECGA).

۱۳-۴-۱ الگوریتم ژنتیک فشرده‌ی تعمیم‌یافته (ECGA)

همان‌طور که از نام آن نیز بر می‌آید، ECGA تعمیم است از cGA که در بخش ۱۳-۲-۲ معرفی گردید. الگوریتم ECGA اولین بار در [هاریک، ۱۹۹۹] ارائه شد و توضیحات تکمیلی آن در [ساستری[1] و گلدبرگ،

[1] Sastry

۲۰۰۰]، [لیما و لوبو[1]، ۲۰۰٤] و [هاریک و همکاران، ۲۰۱۰] ارائه گردید. الگوریتم ECGA تخمین احتمال را به‌گونه‌ای می‌یابد که دو شرط برآورده شوند: اول آنکه تخمین ساده باشد، دوم آنکه تخمین دقیقی از توزیع احتمال ذرات با برازندگی بالا ارائه شود. توزیع احتمال تخمینی، مدل حاصلضربی حاشیه‌ای (MPM[2]) نام دارد. یک نمونه از MPM برای مسئله‌ای ۱۰ بعدی به‌صورت زیر است

$$\hat{p}(x) = p\big(x(1),x(3),x(6)\big)p\big(x(2)\big)p\big(x(4),x(5),x(7),x(10)\big)p\big(x(8),x(9)\big) \quad (٤٧–١٣)$$

متغیرهای موجود در هر تخمین حاشیه‌ای در سمت راست معادله‌ی بالا یک بلوک سازه[3] نام دارد. بنابراین MPM بالا از چهار بلوک سازه تشکیل شده است: $(x(1),x(3),x(6))$، $(x(2))$، $(x(4),x(5),x(7),x(10))$ و $(x(8),x(9))$. تعداد متغیرها در iامین بلوک (B_i) از یک MPM طول بلوک سازه L_i نام دارد. بنابراین، طول بلوک سازه‌های MPM معادله‌ی (۱۳–٤۷) به‌صورت زیر می‌باشد

$$L_1 = 3, L_2 = 1, L_3 = 4, L_4 = 2 \quad (٤٨–١٣)$$

متغیرهای موجود در بلوک‌های سازه متمایزاند. بنابراین، اگر $x(k)$ به B_i تعلق داشته باشد، آنگاه به B_j تعلق نخواهد داشت (برای هر $j \neq i$). پیچیدگی یک MPM را می‌توان به‌صورت زیر اندازه گرفت

$$C_m = (\log_2(M+1)) \sum_{i=1}^{N_b} (2^{L_i} - 1) \quad (٤٩–١٣)$$

که در آن M تعداد ذره‌ها با برازندگی بالاست که از میان جمعیت انتخاب شده‌اند، N_b تعداد بلوک‌های سازه بوده و L_i اندازه‌ی iامین بلوک سازه می‌باشد. دقت یک MPM به‌صورت زیر اندازه گرفته می‌شود

$$C_p = \sum_{i=1}^{N_b} \sum_{j=1}^{2^{L_i}} N_{ij} \log_2\big(M/N_{ij}\big) \quad (٥٠–١٣)$$

که در آن N_{ij} تعداد ذره‌هایی از جمعیت را نشان می‌دهد که شامل jامین توالی بیت در iامین بلوک سازه می‌باشند. اگر iامین بلوک سازه دارای طولی برابر L_i باشد، آنگاه این بلوک شامل 2^L توالی بیت ممکن خواهد بود. الگوریتم ECGA به دنبال یافتن MPMیی است که هزینه‌ی زیر را مینیمم سازد

[1] Lobo
[2] Marginal Product Model
[3] Building Block

$$C_c = C_m + C_p \qquad (۵۱-۱۳)$$

مثال بعد، نحوه‌ی محاسبه‌ی C_c برای یک MPM را نشان می‌دهد.

مثال ۱۳-۹

فرض کنید پنج ذره‌ی زیر را که دارای برازندگی بسیار بالا می‌باشند را در اختیار داریم $(M = 5)$:

$$x_1 = (0,0,0,0), \quad x_2 = (0,0,1,0), \quad x_3 = (0,0,1,1)$$
$$x_4 = (1,0,1,0), \quad x_5 = (1,1,0,1) \qquad (۵۲-۱۳)$$

یک MPM محتمل، مدل تک متغیره است

$$\hat{p}_1(x) = p(x(1))p(x(2))p(x(3))p(x(4)) \qquad (۵۳-۱۳)$$

در این مدل $N_b = 4$ و $L_i = 1$ می‌باشد (برای تمامی iها). پیچیدگی این مدل از معادله‌ی زیر به دست می‌آید

$$C_m = \log_2(6) \sum_{i=1}^{4} (2^{L_i} - 1) = 10.4 \qquad (۵۴-۱۳)$$

ما بلوک‌های سازه را برای $i \in [1,4]$ به‌صورت $B_i = p(x(i))$ منظم می‌نماییم. همچنین ما توالی بیت‌ها را به‌صورت دودویی منظم می‌کنیم به‌صورتی که N_{i1} نمایانگر تعدادی از ذرات باشد که درآن‌ها $x(i) = 0$ بوده و N_{i2} نمایانگر تعداد ذراتی باشد که در آن‌ها $x(i) = 1$ است. در این صورت، دقت مدل را می‌توان به‌صورت زیر نوشت

$$C_p = \sum_{i=1}^{4} \sum_{j=1}^{2} N_{ij} \log_2(M/N_{ij}) = 18.2 \qquad (۵۵-۱۳)$$

یک MPM محتمل دیگر برای جمعیت داده شده در معادله‌ی (۱۳-۵۲) مدل زیر است

$$\hat{p}_2(x) = p(x(1), x(2))p(x(3), x(4)) \qquad (۵۶-۱۳)$$

در این مدل $N_b = 3$، $L_1 = 2$ و ، $L_2 = L_3 = 1$ می‌باشد. این مدل پیچیده‌تر از مدل (۱۳-۵۳) به نظر می‌رسد. اما از آنجا که این مدل شامل احتمال مشترک $x(1)$ و $x(2)$ بوده و همچنین چون از معادله‌ی (۱۳,۵۲) انتظار می‌رود همبستگی شدیدی میان این دو بیت وجود داشته باشد، می‌توان انتظار داشت که این

مدل دقیق‌تر باشد. توجه داشته باشید که در معادله‌ی (۱۳–۵۲)، در چهار ذره از پنج ذره مقادیر $x(1)$ و $x(2)$ یکی است. پیچیدگی $\hat{p}_2(x)$ را می‌توان به‌صورت زیر نوشت

$$C_m = \log_2(6)(3+1+1) = 15.4 \qquad (۱۳–۵۷)$$

که مطابق انتظار از پیچیدگی $\hat{p}_2(x)$ نشان داده شده در معادله‌ی (۱۳–۵٤) بیشتر است. ما بلوک‌های سازه‌ی $\hat{p}_2(x)$ را به‌صورتی که در معادله‌ی (۱۳–۵٦) نشان داده شده است مرتب می‌نماییم. همچنین، ترتیب بیت‌ها را به‌صورت دودویی مرتب می‌کنیم به‌صورتی که N_{11} نشانگر تعداد ذرات با $\big(x(1),x(2)\big) = (0,0)$، N_{12} نشانگر تعداد ذرات با $\big(x(1),x(2)\big) = (0,1)$، N_{13} نشانگر تعداد ذرات با $\big(x(1),x(2)\big) = (1,0)$ و N_{14} نشانگر تعداد ذرات با $\big(x(1),x(2)\big) = (1,1)$ باشد. به همین ترتیب، N_{21} نشانگر تعداد ذرات با $x(3) = 0$ و N_{22} نشانگر تعداد ذرات با $x(3) = 1$ خواهد بود. در آخر، N_{31} نشانگر تعداد ذرات با $x(4) = 0$ و N_{32} نشانگر تعداد ذرات با $x(4) = 1$ می‌باشد. دقت مدل نیز به‌صورت زیر محاسبه می‌شود

$$C_p = \sum_{i=1}^{3}\sum_{j=1}^{2^{L_i}} N_{ij}\log_2\big(M/N_{ij}\big) = 16.6 \qquad (۱۳–۵۸)$$

مطابق انتظار، دقت $\hat{p}_2(x)$ از دقت $\hat{p}_1(x)$ بهتر است. با ترکیب این نتایج خواهیم داشت

$$\begin{aligned} &\hat{p}_1(x) \text{ برای } C_m + C_p = 10.4 + 18.2 = 28.6 \\ &\hat{p}_2(x) \text{ برای } C_m + C_p = 15.4 + 16.6 = 32.0 \end{aligned} \qquad (۱۳–۵۹)$$

با توجه به ECGA، $\hat{p}_1(x)$ به دلیل پیچیدگی کمتر مدل بهتری محسوب می‌شود.

حال که دریافتیم چگونه هزینه‌ی یک MPM را محاسبه نماییم، از الگوریتم شدیدترین شیب نزولی برای یافتن MPMیی که C_c را مینیمم می‌کند استفاده خواهیم کرد. اگر یک MPM با N_b بلوک سازه در اختیار داشته باشیم می‌توانیم با ادغام هر دو بلوک سازه‌ی ممکن، تعداد $N_b\frac{N_b-1}{2}$ بلوک سازه‌ی دیگر تولید نماییم. برای مثال، اگر چهار بلوک سازه‌ی معادله‌ی (۱۳–٤۷) را در اختیار داشته باشیم، می‌توانیم شش MPM زیر را تولید نماییم:

$$\begin{aligned} &p(x(1),x(3),x(6),x(2))p\big(x(4),x(5),x(7),x(10)\big)p\big(x(8),x(9)\big) \\ &p\big(x(1),x(3),x(6),x(4),x(5),x(7),x(10)\big)p\big(x(2)\big)p\big(x(8),x(9)\big) \\ &p\big(x(1),x(3),x(6),x(8),x(9)\big)p\big(x(2)\big)p\big(x(4),x(5),x(7),x(10)\big) \\ &p\big(x(1),x(3),x(6)\big)p\big(x(2),x(4),x(5),x(7),x(10)\big)p\big(x(8),x(9)\big) \\ &p\big(x(1),x(3),x(6)\big)p\big(x(2),x(8),x(9)\big)p\big(x(4),x(5),x(7),x(10)\big) \\ &p\big(x(1),x(3),x(6)\big)p\big(x(2)\big)p(x(4),x(5),x(7),x(10),x(8),x(9)) \end{aligned} \qquad (۱۳–٦۰)$$

ECGA با انتخاب MPM از این مجموعه، سعی در مینیمم‌سازی هزینه‌ی معادله‌ی (۱۳-۵۱) خواهد داشت. الگوریتم ECGA در شکل ۱۳-۱۷ خلاصه شده است. توجه داشته باشید که شکل ۱۳-۱۷ در هر نسل اجرا می‌شود. برای پیاده‌سازی ECGA، ابتدا باید M را که عددی کوچکتر از اندازه‌ی جمعیت N است، انتخاب نماییم. همچنین باید پارامتر P_c را که نسبت فرزندان تولید شده با استفاده از MPM نامزد می‌باشد، تعیین کنیم. ما این فرزندان را با استفاده از انتخاب اتفاقی مجموعه‌های MPM از بهترین M ذره‌ی شناسایی شده در ابتدای شکل ۱۳-۱۷ ایجاد می‌نماییم. این کار با برش $(N_b - 1)$-نقطه‌ای، که در آن هر فرزند دارای N_b والد است، معادل خواهد بود.

بهترین M ذره را از میان جمعیت حاضر انتخاب کن
$\hat{p}_0(x) \leftarrow p(x(1))p(x(2)) \dots p(x(n))$
تا زمانی که (حلقه بی‌نهایت)
  تعداد بلوک‌های سازنده $N_b \leftarrow \hat{p}_0(x)$
  از $\hat{p}_0(x)$ برای MPMهای متناوب $\hat{p}_i(x)$ برای $i \in [1, N_b\,(N_b - 1)/2]$ استفاده کن
    $\hat{p}(x) \leftarrow argmin(C_c(\hat{p}_i(x)) : i \in [1, N_b\,(N_b - 1)/2]$
    اگر $\hat{p}(x) = \hat{p}_0(x)$ از حلقه خارج شو
  دوره‌ی بعدی
  از بلوک‌های سازنده $\hat{p}_0(x)$ برای ایجاد NP_c ذره برای نسل بعد استفاده کن
  $N(1 - P_c)$ ذره‌ی اتفاقی را برای نسل بعد ایجاد کن

شکل ۱۳-۱۷ ساختار مدل حاصلضربی حاشیه‌ای (MPM) با استفاده از الگوریتم تندترین شیب نزولی در الگوریتم ژنتیک فشرده‌ی تعمیم‌یافته (ECGA). این الگوریتم در هر نسل از ECGA اجرا می‌شود.

۱۳-۴-۲ سایر الگوریتم‌های تخمین توزیع چندمتغیره

محققین EDAهای چندمتغیره‌ی بسیاری را ارائه نموده‌اند. از میان آن‌ها می‌توان به الگوریتم توزیع فاکتور گرفته شده (FDA[1]) [مولهنبین و همکاران، ۱۹۹۹]، FDA یادگیرنده [مولهنبین و همکاران، ۱۹۹۹]، الگوریتم تخمین شبکه‌ی بیزی (ENBA[2]) [لاراناگا و همکاران، ۱۹۹۹a]، الگوریتم بهینه‌سازی بیزی (BOA[3]) [پلیکان

[1] Factorized Distribution Algorithm
[2] Estimation of Bayesian networks algorithm
[3] Bayesian Optimization Algorithm

و همکاران، ۱۹۹۹]، الگوریتم توزیع فاکتوریزه شبکه‌ی مارکوف (MN-FDA[1]) [سانتانا، ۲۰۰۳] و EDA شبکه‌ی مارکوف (MN-EDA[2]) [شانتانا، ۱۹۹۸] اشاره نمود. BOA سلسله مراتبی (hBOA[3]) نیز الگوریتمی است که با استفاده از بخش کردن مسئله‌ی بهینه‌سازی به چند زیرمسئله، سعی در کاهش پیچیدگی الگوریتم BOA دارد [پلیکان، ۲۰۰۵].

۱۳–۵ الگوریتم‌های تخمین توزیع پیوسته

قسمت‌های قبل به بحث در مورد انواع EDAها که مناسب دامنه‌ی گسسته می‌باشند، پرداختند. این بخش، ایده‌ی EDA را به مسائل با دامنه‌ی پیوسته بسط می‌دهد. برای مسائل گسسته، ذرات الگوریتم تکاملی از توزیع احتمالات گسسته به وجود می‌آیند. می‌توان همان ایده را در مورد مسائل با دامنه‌ی پیوسته نیز به کار برد، تنها با این تفاوت که در مسائل پیوسته توزیع احتمالات نیز پیوسته خواهد بود و نه گسسته.

برای توصیح EDAهای پیوسته، ابتدا فرایندی را که برای EDAهای گسسته طی نمودیم را به خاطر آورید. فرض کنید مسئله‌ی گسسته‌ای در اختیار داریم که در آن احتمال ۰ بودن بیت iام از راه‌حل نامزد ($x(i)$) برابر ۰٫۷۵ است. بنابراین، احتمال ۱ بودن این بیت برابر ۰٫۲۵ خواهد بود. ما می‌توانیم $x(i)$ را با استفاده از کدی مانند کد زیر تولید نماییم:

$$r \leftarrow U[0,1]$$
$$x(i) \leftarrow \begin{cases} 0 & \text{اگر } r < 0.75 \\ 1 & \text{در غیر این صورت} \end{cases} \qquad (۶۱–۱۳)$$

که در آن r عددی اتفاقی با توزیع یکنواخت بر روی بازه‌ی $[0.1]$ است. از سوی دیگر، اگر دامنه‌ی مسئله دامنه‌ای پیوسته همچون $[0.1]$ باشد، آنگاه مکان iامِ راه‌حل نامزد دیگر یک بیت نبوده و متغیری پیوسته خواهد بود. می‌توان این متغیر را با کدی مانند کد زیر تولید نمود:

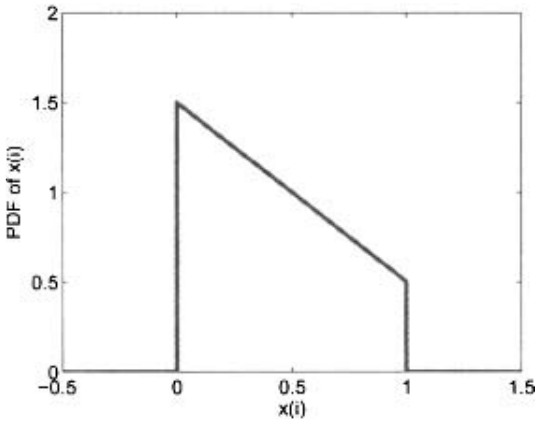
$$r \leftarrow U[0,1]$$
$$x(i) \leftarrow 3/2 - \sqrt{(9/4) - 2r} \qquad (۶۲–۱۳)$$

این کار به تابع چگالی احتمال (PDF) برای $x(i)$ در شکل ۱۳–۱۸ منجر خواهد شد. این موضوع را می‌توان به‌عنوان نقطه‌ی مقابلی برای معادله‌ی (۱۳–۶۱) در نظر گرفت چرا که احتمال $x(i)$ به‌صورت خطی با نزدیک شدن $x(i)$ به ۰ زیاد می‌شود. توجه داشته باشید که توابعی که یک PDF را به دیگری تبدیل

[1] Markov Network Factorized Distribution Algorithm
[2] Markov Network EDA
[3] Hierarchical BOA

می‌نمایند را می‌توان در متون استاندارد مرتبط به علم احتمال پیدا نمود (مسائل ۱۳–۱۱ و ۱۳–۱۵ را ببینید) [گرین‌استد و اسنل، ۱۹۹۷].

EDAهایی که مختص مسائل با دامنه‌ی پیوسته می‌باشند، همگی بر این ایده استوار هستند. با در دست داشتن یک زیرجمعیت از ذرات با برازندگی نسبتاً بالا، تخمینی از PDF پیوسته به دست آورده و سپس از این PDF برای ایجاد جمعیت بعدی از راه‌حل‌های نامزد استفاده می‌نماییم.



شکل ۱۳–۸ یک تابع چگالی احتمال نمونه (PFD) برای متغیر تعریف شده در معادله‌ی (۱۳–۶۲).

۱۳–۵–۱ الگوریتم توزیع حاشیه‌ای پیوسته‌ی تک‌متغیره

در این بخش، EDAهای مختص مسائل با دامنه‌ی پیوسته را با اصلاحی بر روی الگوریتم دودویی UMDA از بخش ۱۳–۲–۱ نشان می‌دهیم. ما از توزیع گاوسی برای ایجاد نسل بعد استفاده خواهیم نمود، به همین دلیل، این الگوریتم را $UMDA_c^G$ نیز نشان می‌دهند [گالاگِر و همکاران، ۲۰۰۷]. $UMDA_c^G$ به نوعی ساده‌ترین EDA پیوسته بوده و در شکل ۱۳–۱۹ خلاصه شده است. در این الگوریتم، ما میانگین و واریانس هر عنصر از زیرجمعیت انتخاب شده را محاسبه کرده و سپس از اعداد اتفاقی گاوسی برای ایجاد نسل بعد استفاده می‌نماییم. همچنین می‌توان شکل ۱۳–۱۹ را به‌گونه‌ای اصلاح نمود که به جای استفاده از توزیع گاوسی، از سایر توزیع‌های ممکن برای ایجاد نسل بعد استفاده شود. برای منطق استفاده از $M-1$ به جای M به معادله‌ی (۸–۱۸) رجوع کنید.

یک جمعیت آغازین از راه‌حل‌های نامزد را مقداردهی کن $\{x_i\}$، $i \in [1, N]$

توجه داشته باش که هر x_i شامل n بیت می‌باشد: $x_i(1) \dots x_i(n)$

تا زمانی که شرایط توقف برآورده نشده است

M ذره از $\{x_i\}$ را متناسب با برازندگیشان انتخاب کن، $M < N$

M ذره‌ی انتخاب شده را به‌صورت $\{x_i\}, i \in [1, M]$ اندیس‌گذاری کن

$$\mu_k \leftarrow \frac{1}{M}\sum_{j=1}^{M} x_j(k)$$

$$\sigma_k \leftarrow \left[\frac{1}{M-1}\sum_{j=1}^{M}\left(x_j(k) - \mu_k\right)^2\right]^{1/2}$$

برای N تا $i = 1$ (اندازه جمعیت)

برای n تا $k = 1$ (تعداد متغیرهای هر راه‌حل نامزد)

$$x_i(k) \leftarrow N(\mu_k, {\sigma_k}^2)$$

متغیر بعدی

ذره‌ی بعدی

نسل بعد

شکل ۱۳–۱۹ الگوریتم توزیع حاشیه‌ای تک‌متغیره‌ی گاوسی پیوسته ($UMDA_c^G$) برای بهینه‌سازی بر روی دامنه‌ی پیوسته‌ی n–بعدی. $N(\mu_k, \sigma_k^2)$ یک متغیر گاوسی اتفاقی با میانگین μ_k و واریانس σ_k^2 است. $x_i(k)$ نیز kاُمین عنصر از iاُمین ذره را نشان می‌دهد.

۱۳–۵–۲ یادگیری افزایشی جمعیت–محور پیوسته

در این بخش، EDAهای پیوسته‌ای را با ایجاد اصلاحاتی بر روی الگوریتم PBIL دودویی از بخش ۱۳–۲–۳ نشان می‌دهیم [سِبگ[1] و دوکولومبیر[2]، ۱۹۹۸]. الگوریتم PBIL با نام تپه‌نوردی اتفاقی یا یادگیری با استفاده از بردارهای توزیع نرمال (SHCLVND[3]) نیز شناخته می‌شود [رودالف و کوپن، ۱۹۹۶]، [پلیکان، ۲۰۰۵، بخش ۲–۳]. فرض کنید هر متغیر مستقل $x_i(k)$ از یک راه‌حل نامزد x_i در محدوده‌ی زیر واقع باشد:

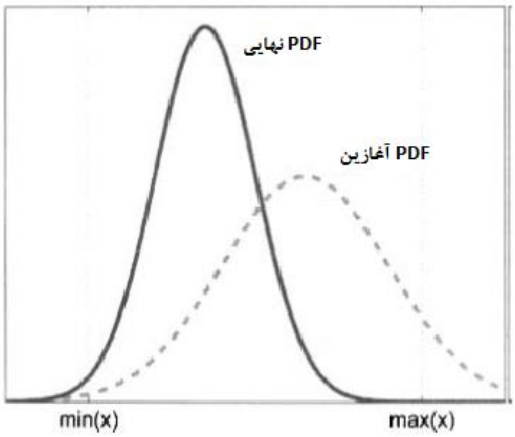
$$x_i(k) \in [x_{min}(k), x_{max}(k)] \qquad (۶۳–۱۳)$$

برای همه‌ی $i \in [1, N]$ و $k \in [1, n]$، که در آن N اندازه‌ی جمعیت بوده و n نیز ابعاد مسئله است. فرض کنید بردار n–بعدی p را در اختیار داریم به‌طوری که به ازای تمامی $k \in [1, n]$،

[1] Sebag
[2] Ducoulumbier
[3] Stochastic Hill Climbing with Learning by Vectors of Normal Distribution

$p_k \in [x_{min}(k), x_{max}(k)]$. در این صورت می‌توانیم عنصر $x_i(k)$ از هر ذره‌ی x_i را با تولید یک عدد اتفاقی گاوسی با میانگین p_k، ایجاد نماییم. با افزایش تعداد نسل‌ها انتظار خواهیم داشت p به سمت راه‌حل بهینه همگرا شود، بنابراین، معمولاً گرایش ما بر آن است که با افزایش شماره‌ی نسل، انحراف استاندارد تولیدکننده‌ی عدد اتفاقی گاوسی را کاهش دهیم. این ایده در شکل ۱۳–۲۰ به تصویر کشیده شده است.

شکل ۱۳–۲۱ یک الگوریتم نوعی PBIL برای مسائل با دامنه‌ی پیوسته را به دست می‌دهد. این الگوریتم بسیار شبیه الگوریتم PBIL دودویی از شکل ۱۳–۸ است. تنها تفاوت در این است که از بردار P برای ایجاد راه‌حل‌های نامزد در بازه‌ی پیوسته‌ی مسئله استفاده شده است. همچنین، انحراف استانداردی که از آن برای ایجاد آن راه‌حل‌های نامزد استفاده می‌شود، همانند شکل ۱۳–۲۰، هر نسل کاهش می‌یابد. این موضوع دو پارامتر میزان‌سازی دیگر، α و β، را در شکل ۱۳–۲۱ به دست می‌دهد. توجه داشته باشید که در هر بار به روزرسانی در شکل ۱۳–۲۱، $x_i(k)$ باید به $[x_{min}(k), x_{max}(k)]$ محدود شود.



شکل ۱۳–۲۰ نمایش تکامل PDF در PBIL پیوسته. تابع چگالی احتمال در ابتدای الگوریتم دارای واریانس بزرگی است که این موضوع باعث کاوش بسیار در فضای جستجو می‌شود. با این حال، در نسل‌های بعدی واریانس PDF کم شده و این موضوع به الگوریتم این اجازه را می‌دهد که به سمت راه‌حل بهینه متمرکز شود.

اندازه جمعیت $= N$

تعداد ذرات خوبی و بدی که برای تنظیم P استفاده شده‌اند $= N_{best}, N_{worst}$

نرخ یادگیری $= \eta \in (0,1)$ ا

دامنه‌ی kامین عنصر فضای جستجو $= [x_{min}(k), x_{max}(k)]$، $k \in [1, n]$

$\beta = \left(\text{برد پارامتر}\right) \div \left(\text{انحراف استاندارد آغازین}\right) \in (0,1)$

فاکتور انقباض انحراف استاندارد $= \alpha \in (0,1)$

انحراف استاندارد آغازین $= \sigma_k \leftarrow \beta\left(x_{max}(k) - x_{min}(k)\right), k \in [1, n]$

$p_k \leftarrow U[x_{min}(k), x_{max}(k)]$ برای $k \in [1, n]$

تا زمانی که شرایط توقف برآورده نشده است

از p برای تولید اتفاقی N راه‌حل نامزد به طریق زیر استفاده کن:

برای $i \in [1, N]$

برای $k \in [1, n]$

$x_i(k) \leftarrow p_k + N(0, \sigma_k)$

بُعد بعدی k

ذره‌ی بعدی

ذرات را به‌گونه‌ای مرتب کن که $f(x_1) < f(x_2) < \cdots < f(x_N)$

برای $i \in [1, N_{best}]$

$p \leftarrow p + \eta(x_i - p)$

i بعدی

برای $i \in [N - N_{best} + 1, N]$

$p \leftarrow p - \eta(x_i - p)$

i بعدی

p را به‌صورت احتمالاتی دچار جهش کن

$\sigma_k \leftarrow \alpha\sigma_k$ برای $k \in [1, n]$

نسل بعد

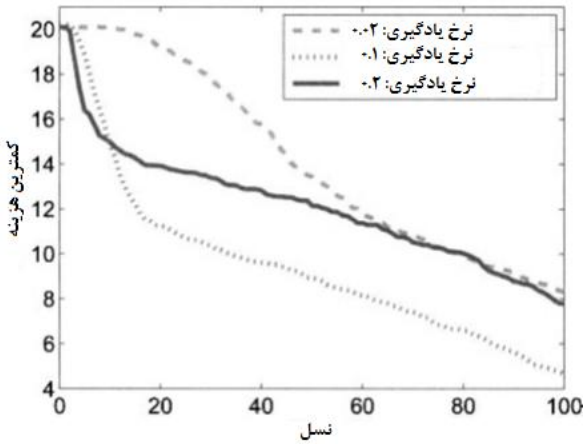
شکل۱۳-۲۱ یک الگوریتم PBIL برای بهینه‌سازی تابع $f(x)$ بر روی n بعد پیوسته. $x_i(k) \in [x_{min}(k), x_{max}(k)]$، kاُمین عنصر از راه‌حل نامزد iاُم می‌باشد. $N(0, \sigma_k)$ یک متغیر اتفاقی گاوسی با میانگین ۰ و انحراف استاندارد σ_k می‌باشد.

مثال ۱۳-۱۰

در این مثال بر آنیم که تابع ۲۰ بعدی اَکلی از ضمیمه‌ی ج.۲-۱ را مینیمم نماییم. برای این کار از الگوریتم PBIL از شکل ۱۳-۲۱ با تنظیمات زیر استفاده خواهیم نمود.

- اندازه‌ی جمعیت $N = 20$.
- کاهش خطی σ_k از ۱۰٪ برد پارامتر در نسل اول تا ۲٪ برد پارامتر در نسل انتهایی. برد پارامتر برای هر بعد $[-30, +30]$ می‌باشد، بنابراین، σ_k به‌صورت خطی از مقدار اولیه‌ی ٦ به مقدار $6/5$ کاهش می‌یابد. این تنظیم σ_k دقیقاً با شکل ۱۳-۲۱ همخوان نیست، اما همان هدف را برآورده می‌سازد.
- ما از ۵ ذره‌ی برتر و ۵ ذره‌ی بدتر ($N_{best} = N_{worst} = 5$) در هر نسل برای به روزرسانی بردار احتمال P استفاده می‌نماییم.
- از هیچ جهشی استفاده نخواهیم کرد.

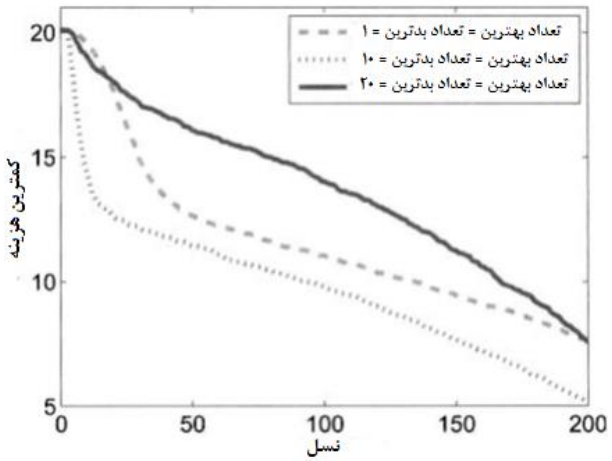
شکل ۱۳-۲۲ اثر نرخ یادگیری η را بر روی عملکرد PBIL نشان می‌دهد. اگر نرخ یادگیری خیلی کوچک باشد، آنگاه تنظیم P از شدت کافی برخوردار نبوده و همگرایی به کندی رخ می‌دهد. اگر نرخ یادگیری بسیار بزرگ باشد، آنگاه P به شدت به سمت راه‌حل‌های خوب جهش کرده و در نتیجه عملکرد اولیه الگوریتم خوب خواهد بود. با این حال، این موضوع می‌تواند باعث جهش از روی بردار احتمال بهینه شده و الگوریتم را در جهتی گمراه‌کننده در فضای جستجو هدایت کند.



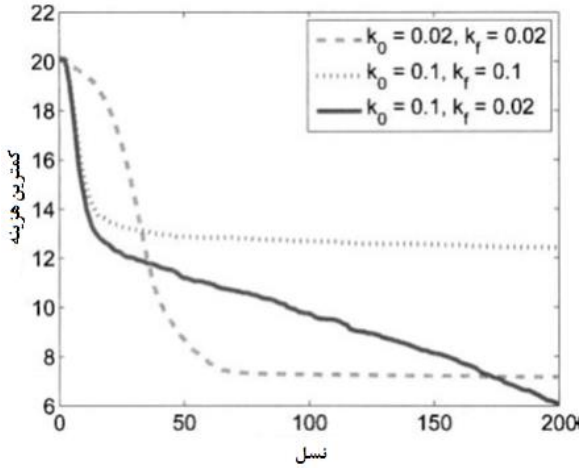
شکل ۱۳-۲۲ مثال ۱۳-۱۰: نتایج الگوریتم PBIL پیوسته برای تابع ۲۰ بعدی اَکلی. نمودار هزینه‌ی بهترین ذره در هر نسل را، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین شده است، نشان می‌دهد. باید برای دستیابی به بهترین عملکرد، از مقدار مناسبی برای η استفاده نمود.

حال به بررسی تأثیر N_{best} و N_{worst} بر روی عملکرد PBIL می‌پردازیم. برای η از مقدار ۰,۱ استفاده می‌نماییم چرا که با توجه به شکل ۱۳-۲۲ به نظر می‌رسد این مقدار بهترین مقدار برای نرخ یادگیری باشد. شکل ۱۳-۲۳ عملکرد PBIL را برای سه مقدار متفاوت از N_{best} و N_{worst} نشان می‌دهد. می‌توان دید که اگر این مقادیر بسیار کوچک باشند، PBIL تأکید زیادی بر روی چند ذره‌ی محدود گذاشته و تصویر خوبی از عملکرد ذرات مختلف در جمعیت به دست نمی‌آورد. با این حال، اگر این مقادیر بسیار بزرگ باشند، الگوریتم PBIL از ذرات بسیاری برای تنظیم نمودن بردار احتمال استفاده می‌نماید، در حالی که ممکن است برخی از این ذرات برای چنین استفاده‌ای مناسب نباشند.

در آخر، نحوه‌ی تأثیر σ_k بر روی عملکرد PBIL را مورد بررسی قرار می‌دهیم. برای این کار از $\eta = 0.1$ و $N_{best} = N_{worst} = 5$ استفاده می‌نماییم. همچنین σ_k را به‌صورت خطی از مقدار $k_0(x_{max}(k) - x_{min}(k))$ در نسل اول به مقدار $k_f(x_{max}(k) - x_{min}(k))$ در نسل آخر تغییر می‌دهیم. شکل ۱۳-۲٤ عملکرد PBIL را برای سه ترکیب متفاوت از k_0 و k_f نشان می‌دهد. می‌توان دید که اگر k_0 بسیار کوچک باشد، همگرایی در مراحل آغازین به دلیل کندی P آهسته خواهد بود. اگر k_f بسیار بزرگ باشد، PBIL به دلیل بزرگی تغییرات راه‌حل‌های نامزد به خوبی همگرا نخواهد شد. به همین ترتیب می‌توان با انجام آزمایش‌هایی تأثیر k_0 بسیار بزرگ و k_f بسیار کوچک بر روی عملکرد PBIL را مورد کاوش قرار داد.



شکل ۱۳-۲۳ مثال ۱۳-۱۰: نتایج PBIL پیوسته برای تابع ۲۰ بعدی آکلی. نمودار هزینه‌ی بهترین ذره در هر نسل را که بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. برای دستیابی به بهترین عملکرد مقادیر N_{best} و N_{worst} باید به طرز مناسبی انتخاب گردند.



شکل ۱۳-۲٤ مثال ۱۳-۱۰: نتایج PBIL پیوسته برای تابع ۲۰ بعدی آکلی. نمودار هزینه‌ی بهترین ذره در هر نسل را، که بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. k_0 و k_f انحراف استاندارد تولید راه‌حل‌های نامزد در نسل اول و آخر را کنترل می‌نمایند. برای دستیابی به بهترین عملکرد مقادیر k_0 و k_f باید به طرز مناسبی انتخاب گردند.

۱۳-۶ نتیجه‌گیری

EDAهای معرفی شده در این فصل با تخمین احتمال به مدل‌سازی فضای جستجو پرداخته و از این طریق بهینه‌ی سراسری را پیدا می‌نمایند. می‌توان از درست‌نمایی بیشینه[1] برای تخمین میانگین نمونه‌ها و کوواریانس جمعیت استفاده نمود. این روشی است که در تخمین الگوریتم‌های نرمال چندمتغیره (EMNA) به کار رفته است [لارانگا، ۲۰۰۲]. اگر از شبکه‌های گاوسی برای مدل‌سازی فضای جستجو استفاده نماییم، به الگوریتم تخمین شبکه‌های گاوسی (EGNAs) دست خواهیم یافت [لارانگا، ۲۰۰۲]، [پاول و ایبا، ۲۰۰۳].

EDAها با استفاده از ابزارهایی ریاضی همچون زنجیره‌های مارکوف [گونزالز[2] و همکاران، ۲۰۰۱]، نظریه‌ی سیستم‌های پویا [گونزالز و همکاران، ۲۰۰۱]، [ماهنیگ[3] و مولهنبین، ۲۰۰۰]، و سایر روش‌ها [گوزالز و همکاران، ۲۰۰۲] مدل شده‌اند. ما مدل‌سازی ریاضی الگوریتم‌های تکاملی را در فصل ٤ برای الگوریتم ژنتیک و در فصل ۷ برای برنامه‌نویسی ژنتیک مورد بحث قرار دادیم، اما در مورد مدل‌های ریاضی EDAها در این کتاب سخنی نخواهد رفت.

[1] Maximum Likliehood
[2] Gonzalez
[3] Mahnig

EDAها نوآوری نسبتاً جدیدی هستند، بنابراین فضای زیادی برای تحقیق و کاربردهای مختلف در این زمینه وجود دارد. جهت‌گیری‌های کنونی در زمینه‌ی EDAها بیشتر به سمت بهینه‌سازی چندهدفه [بوریرات[1] و اسریوراماس[2]، ۲۰۰۷]، بهینه‌سازی پویا [یانگ و یاوو، ۲۰۰۸a]، ترکیب EDAها با سایر الگوریتم‌ها [پنا و همکاران، ۲۰۰٤] و تطبیق روی-خط پارامترهای EDA [سانتانا و همکاران، ۲۰۰۸] معطوف است.

این فصل EDAهایی را که از خواص آماری مرتبه اول و مرتبه دوم استفاده می‌نمایند را معرفی نمود. پاراگراف اول از این نتیجه‌گیری نیز برخی EDAهایی را که از خواص آماری مرتبه بالاتر استفاده می‌نمایند را ذکر می‌کند. این ایده می‌تواند به نوعی EDA منتهی شود که در آن با نزدیک شدن به همگرایی از خواص آماری بالاتر استفاده شود. این بدین معنی است که می‌توان در مراحل ابتدایی EDA ار خواص آماری مرتبه‌ی اول استفاده کرد تا بتوان جمعیتی را به دست آورد که به اندازه‌ی معقولی به بهینه‌ی محلی نزدیک باشد و سپس در مراحل پایانی EDA از خواص آماری مراتب بالاتر برای تنظیم هر چه بهتر نتایج استفاده نمود.

یک جهت نویددهنده‌ی دیگر برای کارهای آتی ترکیب EDAها با سایر الگوریتم‌های تکاملی سنتی می‌باشد. این کار می‌تواند به ترکیب ایده‌های بازترکیب، جهش و نظریه‌ی احتمال برای تولید هر نسل از ذرات منجر شود. یک جهت تحقیقاتی مهم دیگر برای EDAها با دامنه‌ی پیوسته، کاوش روش‌های مختلف برای تخمین PDFهای پیوسته بر اساس یک مجموعه‌ی گسسته از ذرات EDA می‌باشد. تخمین PDF کاری است که باید در فیلتر ذرات[3] نیز صورت بپذیرد، بنابراین جای زیادی برای باروری متقابل میان تحقیقات EDA و تحقیقات فیلتر ذرات وجود دارد [لارانگا و لوزانو[4]، ۲۰۰۲]، [پلیکان و همکاران، ۲۰۰۲]، [کرن[5] و همکاران، ۲۰۰٤]، [لوزانو و همکاران، ۲۰۰٦]، [شایکا[6] و سانتانا، ۲۰۱۲] و [لارانگا و همکاران، ۲۰۱۲]. یک جعبه ابزار MATLAB برای بهینه‌سازی مبتنی بر EDA بر روی اینترنت و از طریق [سانتانا و اِکِگوین[7]، ۲۰۱۲] قابل دسترسی است.

[1] Bureerat
[2] Sriworamas
[3] Particle Filtering
[4] Lozano
[5] Kern
[6] Shayka
[7] Echegoyen

مسائل

تمارین نوشتاری

۱-۱۳ در معادله‌ی (۱۳-۱)، احتمال ۱ بودن بیت پنجم در صورت ۱ بودن بیت‌های سوم و چهارم چه قدر است؟ احتمال ۱ بودن بیت چهارم در صورت ۱ بودن بیت‌های پنجم و سوم چه قدر است؟ احتمال ۱ بودن بیت سوم در صورت ۱ بودن بیت‌های چهارم و پنجم چه قدر است؟

۲-۱۳ مقادیر w_i در معادله‌ی (۱۳-٤) را چه انتخاب کنیم تا معادله‌ی (۱۳-۳) حاصل شود؟

۳-۱۳ چه تعداد نسل در الگوریتم cGA از شکل ۱۳-٤ لازم است تا تعداد ارزیابی‌های تابع برازندگی با تعداد ارزیابی‌های برازندگی در یک نسل از UMDA از شکل ۱۳-۲ برابر شود؟

٤-۱۳ در شکل ۱۳-٦، چه تعداد نسل از الگوریتم cGA تعمیم‌یافته با اندازه‌ی جمعیت N_1 لازم است تا تعداد تعداد ارزیابی‌های تابع برازندگی با تعداد ارزیابی‌های برازندگی در یک نسل از همین الگوریتم با اندازه‌ی جمعیت N_2 برابر شود؟

۵-۱۳ در مثال ۱۳-٤ آنتروپی شرطی بیت ۲ را به شرط دانستن بیت ۱ به دست آورید.

٦-۱۳ با فرض داشتن یک مجموعه از ذرات دودویی در یک الگوریتم تکاملی، آنتروپی شرطی بیت k به شرط خودش را به دست آورید.

۷-۱۳ فرض کنید آنتروپی پنج بیت در یک الگوریتم تکاملی بدین قرار است: $h(1) = 0.3$، $h(2) = 0.4$، $h(3) = 0.5$، $h(2) = 0.5$ و $h(1) = 0.6$. همچنین فرض کنید مقادیر آنتروپی‌های شرطی بیت j با فرض دانستن بیت k به قرار زیر است:

	$k=1$	$k=2$	$k=3$	$k=3$	$k=5$
$j=1$	0.0	0.1	0.4	0.3	0.4
$j=2$	0.4	0.0	0.5	0.6	0.7
$j=3$	0.9	0.8	0.0	0.7	0.6
$j=4$	0.8	0.2	0.1	0.0	0.1
$j=5$	0.2	0.5	0.2	0.5	0.0

الف) از الگوریتم greedy شکل ۱۳-۱۰ برای مینیمم‌سازی معادله‌ی (۱۳-۱۲) استفاده کنید. الگوریتم برای این معادله به دست می‌آید؟

ب) با $k_5 = 2$ آغاز کنید و سپس از الگوریتم greedy برای به دست آوردن سایر مقادیر k_i استفاده نمایید. این روش چه مقادیری را برای معادله‌ی (۱۳-۱۲) به دست می‌دهد؟

۱۳–۸ نشان دهید مقدار اطلاعات متقابل میان بیت‌های m و k با مقدار اطلاعات متقابل میان بیت‌های k و m برابرست.

۱۳–۹ صحت محاسبات ارائه شده برای $I(1,3)$ در مثال ۱۳–۵ را بررسی کنید.

۱۳–۱۰ χ^2_{23} را برای مثال ۱۳–۸ محاسبه کنید.

۱۳–۱۱ با فرض داشتن یک متغیر اتفاقی $x\sim U[0,1]$ یک تابع $y(x)$ با PDF زیر پیدا کنید

$$g(y)=\begin{cases}2a & \text{اگر } 0<y<3/4\\ a & \text{اگر } 3/4<y<1\end{cases}$$

چه مقادیری از a نیاز است تا $g(y)$ یک PDF معتبر باشد؟

تمارین کامپیوتری

۱۳–۱۲ cGA در مقابل UMDA: مثال ۱۳–۲ را با اعمال محدودیت به تعداد نسل‌های cGA تکرار کنید تا مقایسه‌ی عادلانه‌ای میان نتایج UMDA از مثال ۱۳–۱ و نتایج این الگوریتم صورت گیرد (مسئله‌ی ۱۳–۳ را ببینید). این محدودیت چه قدر باید باشد؟ نتایج cGA را با نتایج UMDA مقایسه کنید.

۱۳–۱۳ اندازه‌ی جمعیت cGA: مثال ۱۳–۳ را با $N=2$ و $N=20$ تکرار کنید اما از محدودیت نسل بزرگتری برای $N=2$ استفاده کنید تا مقایسه‌ی عادلانه‌ای میان دو اندازه‌ی جمعیت صورت بگیرد (مسئله‌ی ۱۳–٤ را ببینید). باید از چه محدودیت نسلی برای $N=2$ استفاده نمایید؟ نتایج این دو اندازه‌ی جمعیت را با هم مقایسه نمایید.

۱۳–۱٤ PBIL: الگوریتم PBIL از شکل ۱۳–۸ را با استفاده از شش بیت در هر بعد، برای مینیمم‌سازی تابع ۲۰ بعدی آکلی شبیه‌سازی کنید. شبیه‌سازی را برای ۵۰ نسل اجرا کرده و از مقادیر $N_{best}=N_{worst}=5$، $P_{min}=0$ و $P_{max}=1$ استفاده نمایید. همچنین P را دچار جهش نکنید. ۲۰ شبیه‌سازی مونت کارلو انجام دهید. این کار را برای مقادیر زیر از نرخ یادگیری انجام دهید: $\eta=0.001, 0.01, 0.1$. نتایج به دست آمده را توضیح دهید.

۱۳–۱۵ تبدیل PDF: ۱۰۰۰۰۰ عدد اتفاقی $\{x_i\}$ که دارای توزیع یکنواخت بر روی $[0,1]$ می‌باشند تولید نمایید. تابعی را که در مسئله‌ی ۱۳–۱۱ به دست آوردید به $\{x_i\}$ اعمال کنید تا $\{y_i\}$ را به دست آورید. نمودار فراوانی $\{y_i\}$ را رسم کنید و از این طریق به دست آمدن تابع مطلوب PDF را تأیید کنید.

فصل چهاردهم

بهینه‌سازی
زیست‌جغرافی–محور

جانورشناسی مجمع‌الجزایرها ارزش امتحان را دارند....

چارلز داروین [کینز[1]، ۲۰۰۱]، [مک‌آرتور[2] و ویلسون، ۱۹٦۷، صفحه‌ی ۳]

جغرافیای زیستی دانش مطالعه‌ی زایش، انقراض و توزیع جغرافیایی گونه‌های زیستی می‌باشد. همان‌طور که از گفته‌ی چارلز داروین در خط اول از این فصل برمی‌آید، جغرافیای زیستی ناحیه‌ای مفید برای تحقیق و امتحان می‌باشد. یک تحقیق که اخیراً صورت گرفته است نشان می‌دهد ۳۷۸٤۷ مقاله‌ی زیست‌جغرافیایی در سال ۲۰۱۰ نوشته شده است و چندین ژورنال به این موضوع اختصاص دارند. نویسنده‌ی علمی مشهور دیوید کوامِن[3] در کتاب خود به اسم *آهنگ دودو*[4]، شرح بسیار جالبی از جغرافیای زیستی ارائه کرده است [کوامن، ۱۹۹۷].

همان‌طور که رفتار مورچه‌های زیستی به بهینه‌سازی کلونی مورچگان، علم ژنتیک به الگوریتم‌های ژنتیک و مطالعه‌ی گروه‌های جانوری به بهینه‌سازی تجمع ذرات ختم می‌شود، علم جغرافیای زیستی نیز به بهینه‌سازی زیست‌جغرافی–محور(BBO[5]) ختم خواهد شد. BBO الگوریتمی است که به تازگی به جرگه‌ی الگوریتم‌های تکاملی پیوسته است، اما ما در این کتاب به دلایل زیر یک فصل کامل را به آن اختصاص خواهیم داد.

- محبوبیت BBO به سرعت در حال افزایش است. با یک جستجو در Google Scholar می‌توان دریافت که:
  - ۱ مقاله در مورد BBO در سال ۲۰۰۸ منتشر شده است.
  - ۳۷ مقاله در مورد BBO در سال ۲۰۰۹ منتشر شده است.
  - ۸۱ مقاله در مورد BBO در سال ۲۰۱۰ منشر شده است.
  - ۱٤۵ مقاله در مورد BBO در سال ۲۰۱۱ منتشر شده است.

بنابراین انتظار می‌رود در سال ۲۰۱۲ تعداد مقالات در مورد BBO از ۲۰۰ مقاله تجاوز کند. این‌ها نشان‌دهنده‌ی افزایش فزاینده‌ی محبوبیت BBO می‌باشد.

- برخلاف معرفی اولیه‌ی آن، BBO موفقیت‌های بسیاری را در زمینه‌ی کاربرد دنیای واقعی به خود دیده است؛ کاربرد در زمینه‌هایی همچون مسائل پزشکی، بهینه‌سازی توان، طراحی آنتن، طراحی مکانیکی، روبوتیک، زمان‌بندی، مسیریابی، مسائل نظامی و بسیاری دیگر. برای اطلاعات بیشتر به سایت BBO [سایمون، ۲۰۱۲] مراجعه کنید.

---

[1] Keynes
[2] McArthur
[3] David Quammen
[4] The song of Dodo
[5] Biogeography_Based Optimization (BBO)

- برخلاف سایر الگوریتم‌های تکاملی جدید، نوشته‌های بسیاری در مورد نظریه‌ی BBO در کوتاه مدتی پس از معرفی آن انتشار یافته است که این نوشته‌ها شامل مقالاتی در مورد مدل مارکوف [سایمون و همکاران، ۲۰۱۱a]، مدل‌های سیستم پویا [سایمون، ۲۰۱۱a] و مدل‌های مکانیک آماری [ما[1] و همکاران، ۲۰۱۳] می‌شود.
- نویسنده‌ی این کتاب خود مبدع BBO است و به همین دلیل علاقه‌ی فردی به آن دارد.

مروری بر فصل

این فصل ابتدا خلاصه‌ای از جغرافیای زیستی طبیعی در بخش ۱۴–۱ به دست می‌دهد و سپس به بحث در مورد برگردان آن به یک فرایند بهینه‌سازی در بخش ۱۴–۲ می‌پردازد. بخش ۱۴–۳ نشان خواهد داد که چگونه می‌توان جغرافیای زیستی را برای دستیابی به الگوریتم BBO اقتباس نمود. در بخش ۱۴–۴ نیز برخی اصلاحات و افزونه‌های مفید برای BBO را ارائه خواهیم داد.

۱۴–۱ جغرافیای زیستی

جغرافیای زیستی ریشه در کار ناتورالیست‌های[2] قرن ۱۹ و به خصوص آلفرد والاس[3] [والاس، ۲۰۰۶] و چارلز داروین [کینز، ۲۰۰۱] دارد. والاس را معمولاً به اسم پدر جغرافیای زیستی می‌شناسند، هرچند که داروین به دلیل نظریه‌ی تکاملش شناخته‌شده‌تر است.

تا قبل از دهه‌ی ۱۹۶۰، جغرافیای زیستی بیشتر حالت توصیفی و تاریخی داشت. با این حال استثناهایی نیز مانند رساله‌ی دکتری کمی ایوگن مونرو[4] وجود داشتند [مونرو، ۱۹۴۸]. در اوایل دهه‌ی ۱۹۶۰، رابرت مک‌آرتور و ادوارد ویلسون کار بر روی مدل‌های ریاضی جغرافیای زیستی را آغاز کردند تا کتاب سال ۱۹۶۷ خود با نام *نظریه‌ی جغرافیای زیستی جزیره* را به اوج برسانند [مک‌آرتور و ویلسون، ۱۹۶۷]. آن‌ها به‌طور خاص به توزیع گونه‌ها در جزیره‌های همسایه و مدل‌های ریاضی برای انقراض و مهاجرت گونه‌ها علاقه داشتند. پس از کار مک‌آرتور و ویلسون، جغرافیای زیستی به زیرشاخه‌ای بزرگ از زیست‌شناسی مبدل شد [هانسکی[5] و گیلپین[6]، ۱۹۹۷].

[1] Ma
[2] Naturalists
[3] Alfred Wallace
[4] Eugene munroe
[5] Hanski
[6] Gilpin

مدل‌های ریاضی جغرافیای زیستی زایش (تکامل گونه‌های جدید)، مهاجرت گونه‌ها بین جزیره‌ها و انقراض گونه‌ها را توصیف می‌کند. در اینجا واژه‌ی جزیره بیشتر مفهومی توصیفی دارد تا ادبی. به بیان دیگر در اینجا جزیره به معنای هر زیستگاهی است که به لحاظ جغرافیایی از سایر زیستگاه‌ها منزوی است. در مفهوم کلاسیک، جزیره زیستگاهی است که به وسیله‌ی آب از سایر زیستگاه‌ها جدا شده است. اما جزیره‌ها می‌توانند زیستگاه‌هایی باشند که با استفاده از کویرها، رودخانه‌ها، رشته‌کوه‌ها، مهاجمین، ساخته‌های دست بشر و یا سایر موانع از هم جدا شده باشند. یک جزیره می‌تواند حاشیه‌ی رودخانه‌ای باشد که گیاهان را حمایت کرده، برکه‌ای باشد که خانه‌ی دوزیستان بوده، صخره‌هایی باشد که محل زندگی حلزون‌ها بوده و یا تنه‌ی مرده‌ی یک درخت باشد که زیستگاه حشرات است [هانسکی و گیلپین، ۱۹۹۷].

ناحیه‌های جغرافیایی که محیطی سازگار برای زندگی را فراهم می‌کنند دارای شاخص تناسب زیستگاهی (HSI[1]) بالایی می‌باشند [وش[2] و همکاران، ۱۹۸۷]. خواصی که با HSI همبستگی دارند شامل میزان بارش، تنوع گونه‌ی گیاهی، تنوع توپوگرافی، محیط خشکی و دما می‌شود. این متغیرها که قابلیت سکنی را مشخص می‌کنند با نام متغیرهای شاخص سازگاری (SIV[3]) شناخته می‌شوند. بنابر تعریف قابلیت سکنی، SIVها متغیرهای مستقل زیستگاه بوده و HSIها متغیرهای وابسته می‌باشند.

جزیره‌هایی که دارای HSI بالا بوده قابلیت پذیرایی از گونه‌های زیادی را داشته و جزیره‌هایی که دارای HSI پایین می‌باشند می‌توانند تنها تعداد کمی از گونه‌ها را در خود جای دهند. جزیره‌ها با HSI بالا به لطف پذیرایی از تعداد زیادی از گونه‌ها، دارای گونه‌هایی هستند که به زیستگاه‌های مجاور مهاجرت می‌کنند. این مهاجرت به دلیل علاقه‌ی گونه‌ها به مهاجرت اتفاق نمی‌افتد چرا که زیستگاه اصلی دارای جذابیت بالایی برای زندگی است. علت مهاجرت از این نوع جزایر، تجمع اثرات اتفاقی بر روی تعداد زیادی از گونه‌ها با جمعیت بزرگ می‌باشد. مهاجرت با پرواز، شنا و یا سوار بر باد شدن حیوانات از یک جزیره به جزیره دیگر اتفاق می‌افتد. هنگامی که یک گونه از جزیره مهاجرت می‌کند بدین معنی نیست که به‌طور کامل از آن جزیره محو می‌شوند، بلکه بدین معنی است که تنها چند نماینده از گونه‌ی مورد نظر مهاجرت کرده و سایر اعضای گونه‌ی مورد نظر در زیستگاه اصلی باقی می‌مانند. با این حال، در بسیاری از مباحث ما، مهاجرت از یک جزیره با انقراض در آن جزیره هم‌ارز است. این فرض برای ایجاد BBO با استفاده از جغرافیای زیستی لازم است.

[1] Habitat Suitability Index
[2] Wesche
[3] Suitability Index Variables

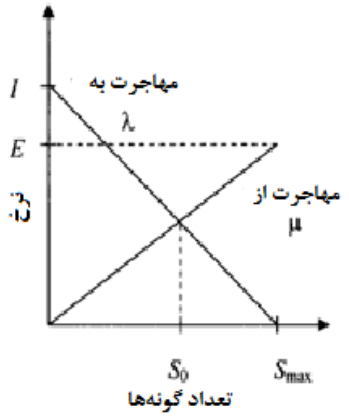
در جزایر با HSI بالا نه تنها نرخ مهاجرت از جزیره بالاست، بلکه نرخ مهاجرت به جزیره پایین است چرا که قبلا گونه‌های زیادی در این جزیره وجود دارند. گونه‌هایی که به این نوع جزایر می‌آیند، با وجود HSI بالا، از شانس کمی برای بقا برخوردار هستند چرا که رقابت سختی بر سر منابع در جریان است.

به دلیل جمعیت کم، نرخ مهاجرت به جزایر با HSI پایین، زیاد است. دوباره متذکر می‌شویم که این مهاجرت به دلیل علاقه‌ی گونه‌ها به مهاجرت به این نوع جزایر نمی‌باشد چرا که این جزایر برای زندگی مطلوب نیستند. دلیل مهاجرت، وجود فضای کافی برای گونه‌های اضافی است. این که آیا گونه‌ها می‌توانند در این جزایر به بقای خود ادامه دهند یا نه و اینکه این بقا چه قدر به طول خواهد انجامید، سوالی دیگر است. با این حال، تنوع گونه‌ها با HSI همبستگی دارد، بدین معنی که هر چه تعداد گونه‌های رسیده به جزایر با HSI پایین بیشتر شود، HSI جزیره افزایش خواهد یافت [وش و همکاران، ۱۹۸۷].

شکل ١٤-١ مدلی از فراوانی گونه‌ها در یک جزیره را نشان می‌دهد [مک‌آرتور و ویلسون، ١٩٦٧]. نرخ مهاجرت به جزیره λ و نرخ مهاجرت از جزیره μ توابعی هستند از تعداد گونه‌ها در جزیره. ما شکل مهاجرت از جزیره را به‌صورت یک خط صاف ترسیم نموده‌ایم، هرچند که این نرخ می‌توان شکل بسیار پیچیده‌تری داشته باشد. در این مورد بعداً بیشتر سخن خواهیم گفت.

نمودار مهاجرت به جزیره را در نظر بگیرید. بیشترین نرخ مهاجرت به زیستگاه ممکن I است و هنگامی محقق می‌شود که تعداد گونه‌های موجود در جزیره برابر صفر باشد. با زیاد شدن تعداد گونه‌ها، جزیره شلوغ‌تر شده و تعداد گونه‌های کمتری خواهند توانست با موفقیت مهاجرت را پشت سر گذاشته و بنابراین نرخ مهاجرت کاهش می‌یابد. بیشترین تعداد گونه‌ای که یک زیستگاه می‌تواند بپذیرد را با S_{max} نمایش داده و نقطه‌ای است که در آن نرخ مهاجرت برابر صفر است.

حال نمودار مهاجرت از جزیره را در نظر بگیرید. اگر هیچ گونه‌ای در جزیره وجود نداشته باشد، نرخ مهاجرت از جزیره صفر خواهد بود. هر چه تعداد گونه‌ها در جزیره افزایش یابد، جزیره شلوغ‌تر شده و گونه‌ای بیشتری می‌توانند جزیره را ترک کرده و بدین ترتیب نرخ مهاجرت از جزیره افزایش می‌یابد. بیشترین نرخ مهاجرت از جزیره برابر E است و هنگامی محقق می‌شود که جزیره دارای بیشترین تعداد گونه‌ی ممکن است.



شکل ١٤-١ مدل مهاجرت یک جزیره بنابر [مک‌آرتور و ویلسون، ١٩٦٧]. S_0 شماره‌ی گونه‌ی تعادل است.

یک مدل ریاضی از جغرافیای زیستی

باقی‌مانده‌ی این بخش یک مدل ریاضی از شماره‌ی گونه‌ها در جغرافیای زیستی ارائه می‌دهد. این موضوع برای درک الگوریتم BBO ضروری نیست؛ بنابراین خواننده می‌تواند در صورت تمایل از خواندن این بخش صرف نظر کند.

تعداد تعادل گونه‌ها در شکل ١٤-١ برابر S_0 است، نقطه‌ای که در آن نرخ مهاجرت از جزیره و نرخ مهاجرت به جزیره با هم برابرند. با این حال، به دلایل اتفاقی گذرا ممکن است به گذرهایی از S_0 منجر شود. گذر مثبت از S_0 می‌تواند به علت رسیدن تکه‌ی بزرگ غیرمعمولی از جزیره‌های همسایه و یا تعداد بسیار زیاد تولدها باشد. گذر منفی از S_0 می‌تواند به علت بیماری، حضور موقتی یک مهاجم جدید و یا یک فاجعه‌ی طبیعی باشد. رسیدن دوباره‌ی تعداد گونه‌ها به مقدار تعادل بعد از یک اختلال بزرگ ممکن است سال‌ها به طول انجامد [هانسکی و گیلپین، ١٩٩٧]، [هیستینگز[1] و هیگینز[2]، ١٩٩٤].

احتمال P_s را که نشان‌دهنده‌ی احتمال وجود S گونه در جزیره می‌باشد را در نظر بگیرید. این احتمال از زمان t به زمان Δt به‌صورت زیر تغییر می‌کند:

$$P_s(t+\Delta t) = P_s(t)(1-\lambda_s\Delta t-\mu_s\Delta t) + P_{s-1}(t)\lambda_{s-1}\Delta t + P_{s+1}(t)\mu_{s+1}\Delta t \qquad (١-١٤)$$

که در آن λ_s و μ_s به ترتیب نرخ مهاجرت به و مهاجرت از جزیره در صورتی است که جزیره دارای S گونه باشد. این معادله در صورتی صادق خواهد بود که فرض کنیم Δt آنقدر کوچک است که احتمال صورت

[1] Hastings
[2] Higgins

پذیرفتن یک مهاجرت میان زمان‌های t و Δt قابل صرف نظر کردن باشد. بنابراین، برای داشتن S گونه در زمان $t + \Delta t$، باید یکی از شرایط زیر برقرار باشد:

۱. تعداد گونه‌ها در زمان t برابر S بوده و هیچ مهاجرتی بین زمان‌های t و $t + \Delta t$ صورت نپذیرفته باشد.

۲. تعداد گونه‌ها در زمان t برابر $S - 1$ بوده و یک گونه به زیستگاه مهاجرت کرده باشد، و یا

۳. تعداد گونه‌ها در زمان t برابر $S + 1$ بوده و یک گونه از زیستگاه مهاجرت کرده باشد.

با گرفتن حد از معادله‌ی (۱۴-۱) با فرض $\Delta t \to 0$ خواهیم داشت

$$P_s = \begin{cases} -(\lambda_s + \mu_s)P_s + \mu_{s+1}P_{s+1} & S = 0 \\ -(\lambda_s + \mu_s)P_s + \lambda_{s-1}P_{s-1} + \mu_{s+1}P_{s+1} & 1 \le S \le S_{max} - 1 \\ -(\lambda_s + \mu_s)P_s + \lambda_{s-1}P_{s-1} & S = S_{max} \end{cases} \quad (۲-۱۴)$$

فرض کنید $n = S_{max}$ و $P = [P_0 \dots P_n]^T$. حال می‌توانیم $(n + 1)$ معادله‌ی موجود در معادله‌ی (۱۴-۲) را در یک معادله‌ی ماتریسی مرتب کنیم

$$P = AP \quad (۳-۱۴)$$

که در آن ماتریس A به‌صورت زیر است

$$A = \begin{bmatrix} -(\lambda_0 + \mu_0) & \mu_1 & 0 & \cdots & 0 \\ \lambda_0 & -(\lambda_1 + \mu_1) & \mu_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \lambda_{n-2} & -(\lambda_{n-1} + \mu_{n-1}) & \mu_n \\ 0 & \cdots & 0 & \lambda_{n-1} & -(\lambda_n + \mu_n) \end{bmatrix} \quad (۴-۱۴)$$

برای نرخ مهاجرت خطی از شکل ۱۴-۱ خواهیم داشت

$$\mu_k = \frac{Ek}{n}$$
$$\lambda_k = I(1 - \frac{k}{P}) \quad (۵-۱۴)$$

در حالت خاص $E = I$ خواهیم داشت

$$\lambda_k + \mu_k = E = I \text{ برای همه‌ی } k \in [0, n] \quad (۶-۱۴)$$

$$A = \begin{bmatrix} -1 & \frac{1}{n} & 0 & \cdots & 0 \\ \frac{n}{n} & -1 & \frac{2}{n} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \frac{2}{n} & -1 & \frac{n}{n} \\ 0 & \cdots & 0 & \frac{1}{n} & -1 \end{bmatrix}$$

$$= EA'$$

که در آن A' در همان معادله‌ی بالا تعریف شده است.

قضیه‌ی ۱۴-۱ *مقدار ویژه‌ی A' برای هر عدد طبیعی* n به قرار زیراند

$$x(A') = \left\{0, -\frac{2}{n}, -\frac{4}{n}, \dots, -n\right\} = -\frac{2k}{n}, \quad k \in [0, n] \tag{۱۴-۷}$$

همچنین، بردار ویژه‌ی متناظر با مقدار ویژه‌ی صفر برابرست با

$$v(0) = [v_0(0) \dots v_n(0)]^T$$
$$k \in [0, n] \qquad ،v_k(0) = \binom{n}{k} = \frac{n!}{k!(n-k)!} \text{ که در آن} \tag{۱۴-۸}$$

قسمت اول قضیه‌ی بالا در [سایمون، ۲۰۰۸] حدس زده شده و در [ایگلنیک[1] و سایمون، ۲۰۱۱] اثبات شده است. قسمت دوم قضیه در هر دو مرجع اثبات شده است؛ هر چند به روش‌هایی متفاوت. بعد از انتشار [ایگلنیک و سایمون، ۲۰۱۱]، ما متوجه شدیم که ایده‌ی اصلی قسمت اول قضیه‌ی بالا قبلا بدون اثبات در [کلمنت[2]، ۱۹۵۹] و [گرگوری[3] و کارنی[4]، ۱۹۶۹، مثال ۷-۱۰] بیان شده است.

در حالت مااندگار $t \to \infty$، بنابراین $P(\infty) = AP(\infty) = EA'P(\infty) = 0$ بوده و در نتیجه $P(\infty)$ بردار ویژه‌ی متناظر با مقدار ویژه‌ی صفر خواهد بود. به یاد آورید که بردارهای ویژه یکتا نیستند، بلکه تنها بنابر یک ضریب مقیاس غیر صفر تعریف می‌گردند. ار آنجا که $P(\infty)$ یک بردار احتمال است، جمع عناصر آن باید برابر ۱ باشد. این حقایق به نتایج زیر منجر می‌شوند [سایمون، ۲۰۰۸] و [ایگلنیک و سایمون، ۲۰۱۱].

قضیه‌ی ۱۴-۲ *مقدار حالت ماندگار برای احتمال تعداد هر یک از گونه‌ها برابرست با*

[1] Igelnik
[2] Clement
[3] Gregory
[4] Karney

$$P(\infty) = \frac{v(0)}{\sum_{k=0}^{n} v_{k(0)}} = 2^{-n} v(0) \qquad (۹-۱٤)$$

مثال ۱٤-۱

جزیره‌ای را در نظر بگیرید که در بیشترین حالت می‌تواند پذیرای چهار گونه باشد. بیشترین نرخ مهاجرت به و از جزیره برابر ۲ گونه در واحد زمان می‌باشد. بنابراین، $n = 4$ و $E = I = 2$. A' برای این مثال برابرست با

$$A' = \begin{bmatrix} -1 & \frac{1}{4} & 0 & 0 & 0 \\ 1 & -1 & \frac{2}{4} & 0 & 0 \\ 0 & \frac{3}{4} & -1 & \frac{3}{4} & 0 \\ 0 & 0 & \frac{2}{4} & -1 & 1 \\ 0 & 0 & 0 & \frac{1}{4} & -1 \end{bmatrix} \qquad (۱۰-۱٤)$$

قضیه‌ی ۱٤-۱ به ما می‌گوید که مقادیر ویژه عبارتند از $x = \{0, -½, -1, -\frac{3}{2}, -2\}$ و بردار ویژه‌ی متناظر با مقدار ویژه‌ی صفر برابرست با $v(0) = [1\ 4\ 6\ 4\ 1]^T$. همچنین، قضیه‌ی ۱٤-۲ به ما می‌گوید که احتمال حالت ماندگار برای هر شماره از تعداد گونه‌ها برابرست با

$$\begin{aligned} \Pr(S = 0) &= \Pr(S = 4) = \frac{1}{16} \\ \Pr(S = 1) &= \Pr(S = 3) = \frac{4}{16} \\ \Pr(S = 2) &= \frac{6}{16} \end{aligned} \qquad (۱۱-۱٤)$$

اگر شبیه‌سازی مهاجرت را برای ۵۰۰۰ گام زمانی اجرا کنیم، احتمالات زیر را برای هر شماره‌ی گونه به دست می‌آوریم:

$$\begin{aligned} &\Pr(S = 0) = 0.0714 \\ &\Pr(S = 1) = 0.2605 \\ &\Pr(S = 2) = 0.3734 \\ &\Pr(S = 3) = 0.2358 \\ &\Pr(S = 4) = 0.0544 \end{aligned} \qquad (۱۲-۱٤)$$

این مقادیر به مقادیر تحلیلی به دست آمده در معادله‌ی (۱٤-۱۱) نزدیک هستند.

۱۴-۲ جغرافیای زیستی یک فرایند بهینه‌سازی است

می‌دانیم در طبیعت فرایندهای بهینه‌سازی فراوانی وجود دارد [آلکساندر[1]، ۱۹۹۶]. در حقیقت، این فرض یک اصل بنیادیِ الگوریتم‌های تکاملی است. با این حال، آیا می‌توان جغرافیای زیستی را یک فرایند بهینه‌سازی در نظر گرفت؟ در نگاه اول به نظر می‌رسد که جغرافیای زیستی صرفا تعادل بین تعداد گونه‌ها در جزیره‌های مختلف را حفظ کرده و الزاماً بهینه نیست. این بخش جغرافیای زیستی را از منظر بهینگی مورد بررسی قرار می‌دهد.

همان‌طور که پیش از این گفته شد، جغرافیای زیستی روش طبیعت برای پخش گونه‌ها می‌باشد و اغلب به‌عنوان علمی مورد مطالعه قرار گرفته است که تعادل بین زیستگاه‌ها را حفظ می‌نماید. این تعادل را در شکل ۱۴-۱ می‌توان در نقطه‌ی S_0 مشاهده نمود، جایی که در آن منحنی‌های مهاجرت از و مهاجرت به جزیره با هم تلاقی می‌کنند. یک دلیل برای نگاه کردن به جغرافیای زیستی از دریچه‌ی تعادل آن است که که اولین بار این دیدگاه پایه‌های ریاضی محکمی برای جغرافیای زیستی فراهم نمود [مک‌آرتور و ویلسون، ۱۹۶۳]، [مک‌آرتور و ویلسون، ۱۹۶۷]. با این حال، از آن زمان به بعد این دیدگاه توسط زیست جغرافی‌شناس‌ها بسیار مورد سوال و یا تعمیم واقع شده است.

در مهندسی معمولاً پایداری و بهینگی به‌عنوان اهداف متقابل در نظر گرفته می‌شوند؛ برای مثال، پایدارسازی یک سیستم ساده آسان‌تر از پایدارسازی یک سیستم پیچیده است و این در حالی است که یک سیستم بهینه معمولاً سیستمی پیچیده است و نسبت به سیستم‌های ساده از پیچیدگی بیشتری برخوردار است [کیل[2] و باتاچاریا[3]، ۱۹۹۷]. با این حال، در جغرافیای زیستی، پایداری و بهینگی هر دو در یک روی سکه قرار دارند. بهینگی در جغرافیای زیستی شامل جوامع پیچیده و متنوعی می‌شود که با محیط خود بسیار منطبق هستند. از سوی دیگر، پایداری در جغرافیای زیستی شامل ماندگاری جمعیت می‌شود. مشاهدات میدانی نشان می‌دهد که جوامع پیچیده‌تر، سازگارتر و پایدارتر از جوامع ساده هستند [هاردینگ، ۲۰۰۶، صفحه‌ی ۸۲] و همچنین این مشاهده توسط شبیه‌سازی نیز حمایت شده است [التون[4]، ۱۹۵۸]، [مک‌آرتور، ۱۹۵۵].

با این که طبیعت مکمل بهینگی و پایداری در جغرافیای زیستی را به چالش کشیده شده است [می[5]، ۱۹۷۳] این چالش‌ها اکثرا پاسخ داده شده و ایده به‌طور کلی پذیرفته شده است [مک‌کن[6]، ۲۰۰۰]، [کوندوه[7]،

---

[1] Alexander
[2] Keel
[3] Bhattacharyya
[4] Elton
[5] May
[6] McCann
[7] Kondoh

۲۰۰۶]. بنابراین، بحث تقابل تعادل و بهینگی در جغرافیای زیستی تبدیل به یک موضوع معنایی می‌شود، چرا که در جغرافیای زیستی تعادل و بهینگی دو منظر متفاوت از یک پدیده‌ی واحد هستند.

یک مثال نمایشی از بهینگی جغرافیای زیستی کراکاتووا[1] است، یک جزیره‌ی آتشفشانی در اقیانوس هند که در ماه اوت سال ۱۸۸۳ فوران کرد [وینچستر[2]، ۲۰۰۸]. صدای فوران تا مایل‌ها شنیده شد و باعث مرگ بیش از ۳۶۰۰۰ نفر شد. بیشتر این مرگ‌ها به دلیل موج‌های کشنده‌ای بود که اثر آن‌ها تا انگلستان قابل مشاهده بود. این فوران ذرات غبار را تا ارتفاع ۳۰ مایل پرتاب نمود، غباری که تا ماه‌ها بر روی هوا باقی ماند و از همه جای جهان قابل مشاهده بود. یک زمین‌شناس و مهندس معدن به نام روجیر وربیک[3] اولین فردی بود که بعد از شش هفته از فوران از کراکاتووا دیدن کرد، اما حتی در آن زمان نیز زمین جزیره داغ‌تر از آن بود که بتوان آن را لمس کرد و هیچ نشانی از حیات یافت نمی‌شد. جزیره کاملا عقیم شده بود [ویتاکر و بوش[4]، ۱۹۹۳]. اولین حیات حیوانی در جزیره (یک عنکبوت) در می۱۸۸٤ در کاراکاتووا ۹ ماه پس از فوران پیدا شد. تا سال ۱۸۸۷، زمین‌های پوشیده از علف در جزیره ظاهر گشتند. تا سال ۱۹۰٦، حیات گیاهی و حیوانی به وفور در جزیره پیدا می‌شد. هرچند که فعالیت‌های آتشفشانی در این جزیره همچنان ادامه دارد، تا سال ۱۹۸۳ (یک سال بعد از ویرانی آن) ۸۸ گونه درخت و ۵۳ گونه درختچه در آن رشد کرده [ویتاکر و بوش، ۱۹۹۳] و تعداد گونه‌ها به‌صورت خطی با زمان در حال افزایش است. حیات به کاراکاتووا مهاجرت می‌کند و این مهاجرت باعث سازگارتر شدن جزیره و در نتیجه مطلوب‌تر شدن آن برای مهاجرت‌های بیشتر می‌شود.

جغرافیای زیستی حداقل تا یک نقطه‌ی معین یک فیدبک مثبت محسوب می‌شود. این موضوع مانند انتخاب طبیعی، که به اسم بقای برازنده‌ترین نیز شناخته می‌شود، می‌باشد. هرچه گونه‌ها برازنده‌تر باشند، شانس آن‌ها برای بقا بیشتر خواهد بود. هرچه گونه‌ها بقای بیشتری داشته باشند، پراکنده‌تر شده و بهتر قادر خواهند بود با محیطشان وفق پیدا کنند. انتخاب طبیعی مانند جغرافیای زیستی شامل فیدبک مثبت می‌باشد. با این حال، مقیاس زمانی جغرافیای زیستی بسیار کوتاه‌تر از انتخاب طبیعی است.

یک مثال خوب دیگر از جغرافیای زیستی به‌عنوان یک فرایند بهینه‌سازی جنگل‌های استوایی آمازون می‌باشد؛ یک نمونه‌ی بهینه‌سازی متقابل سیستم حیات/محیط‌زیست [هاردینگ، ۲۰۰٦]. جنگل‌های استوایی ظرفیت بالایی برای بازیافت رطوبت دارند و همین موضوع باعث خشکی کمتر و افزایش تبخیر می‌شود. این موضوع به نوبه‌ی خود باعث سردتر و خیس‌تر شدن سطوح و مناسب‌تر شدن آن‌ها برای حیات می‌شود. این

[1] Krakatoa
[2] Winchester
[3] Rogier Verbeek
[4] Bush

موضوع بیانگر آن است که دیدگاهی از جغرافیای زیستی "که بر اساس بهینه‌سازی شرایط زیست‌محیطی برای فعالیت‌های زیستی قرار دارد، مناسب‌تر از تعریف هموستازی[۱] آن می‌باشد" [کلیدون[۲]، ۲۰۰٤]. این دیدگاه از محیط زیست به‌عنوان یک سیستم بهینه‌ساز حیات در سال ۱۹۹۷ ارائه شد [ولک[۳]، ۱۹۹۷]. مثال‌های متعدد دیگری از بهینگی جغرافیای زیستی وجود دارد که از جمله می‌توان به دمای زمین [هاردینگ، ۲۰۰٦]، ترکیب جو زمین [لنتون[٤]، ۱۹۹۸] و ترکیبات معدنی اقیانوس‌ها [لاولاک[٥]، ۱۹۹۰] اشاره نمود.

تمام این‌ها بدین معنی نیست که جغرافیای زیستی برای هر گونه‌ی خاص بهینه است. برای مثال، تحقیقات بر روی جزیره‌ی مرجانی بیکینی[٦] نشان می‌دهد سطح بالای تشعشعات رادیواکتیویته ناشی شده از آزمایش‌های هسته‌ای تأثیر کمی بر روی محیط زیست طبیعی آن داشته در حالی که پستانداران به شدت از این موضوع متأثر شده‌اند [لاولاک، ۱۹۹۵، صفحه‌ی ۳۷]. مطالعات این چنینی نشان‌دهنده‌ی آن هستند که "زمین خود از خودش مراقبت می‌کند و فزونی محیط زیستی بهبود می‌یابد، اما به احتمال زیاد این ترمیم محیط زیست در جهانی عاری از مردم و انسان‌ها اتفاق خواهد افتاد" [مارگولیس[۷]، ۱۹۹٦]. در میان تمامی هشدارهای کنونی در مورد تخلیه‌ی لایه‌ی اوزون، از این مسئله غفلت کرده‌ایم که دردو میلیارد سال ابتدایی از حیات زمینی، زمین فاقد لایه‌ی اوزون بود [لاولاک، ۱۹۹۵، صفحه‌ی ۱۰۹]. حیات بدون لایه‌ی اوزون هم رشد می‌کند و شکوفا می‌شود، هر چند که این شکوفایی حول انسان متمرکز نخواهد بود. هرچند که ممکن است گرمایش جهانی و عصر یخبندان برای انسان‌ها و پستادناران فاجعه‌بار باشد، یک اتفاق جزیی در تاریخ جغرافیای زیستی سیاره‌مان به حساب خواهد آمد.

این فرض که جغرافیای زیستی یک فرایند بهینه‌سازی است انگیزه‌ی ایجاد BBO به‌عنوان یک الگوریتم بهینه‌سازی را به ما می‌دهد، موضوعی که در بخش بعد به بحث در مورد آن خواهیم پرداخت.

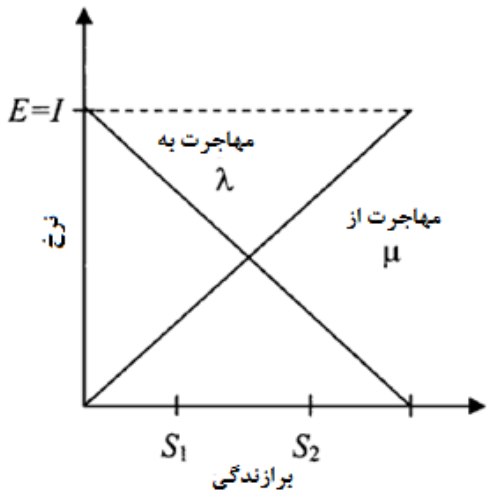
---

[۱] هموستازی گرایش به سمت تعادل پایدار نسبی میان عناصر مستقل می‌باشد. م.

[2] Kleidon
[3] Volk
[4] Lenton
[5] Lovelock
[6] Bikini Atoll
[7] Margulis

۱۴-۳ بهینه‌سازی زیست‌جغرافی-محور

جغرافیای زیستی روش طبیعت برای پخش گونه‌ها و بهینه‌سازی محیط برای حیات است و مشابه بهینه‌سازی ریاضی است. فرض کنید یک مسئله‌ی بهینه‌سازی و چند راه‌حل نامزد، که آن‌ها را ذره می‌نامیم، در اختیار داریم. ذرات خوب روی مسئله به خوبی عمل کرده و ذرات ضعیف بد عمل می‌کنند. یک ذره‌ی خوب معادل یک جزیره با HSI بالا بوده و یک ذره‌ی ضعیف معادل یک جزیره با HSI پایین می‌باشد. ذرات خوب در مقابل تغییر مقاومت می‌کنند، درست مانند جزایر با HSI بالا که دارای نرخ مهاجرت به جزیره‌ی کمتری نسبت به جزایر با HSI پایین می‌باشند. به همین ترتیب، ذرات خوب تمایل به اشتراک‌گذاری خصایص خود با ذرات ضعیف دارند، همان‌طور که جزایر با سازگاری بالا دارای نرخ مهاجرت از جزیره‌ی بالایی می‌باشند. ذرات ضعیف خصایص جدید را از ذرات خوب می‌پذیرند، همان‌طور که جزایر با سازگاری محیطی پایین از جزایر با سازگاری محیطی بالا مهاجر می‌پذیرند. اضافه شدن خصایص جدید می‌تواند کیفیت ذرات ضعیف را افزایش دهد. الگوریتم تکاملی که بر این ایده استوار باشد بهینه‌سازی زیست‌جغرافی-محور (BBO) نام دارد. برای سادگی فرض را بر آن می‌گذاریم که هر ذره‌ی BBO با استفاده از یک منحنی شماره‌ی گونه‌ی یکسان با $E = I$ نمایش داده می‌شود. شکل ۱۴-۲ نرخ‌های مهاجرت از و به جزیره را برای یک الگوریتم BBO با این فرض‌ها نشان می‌دهد. در این نمودار S_1 یک ذره‌ی ضعیف و S_2 یک ذره‌ی خوب را نشان می‌دهد. نرخ مهاجرت به جزیره برای S_1 بسیار بالاست بدین معنی که احتمال دریافت خصوصیات از سایر راه‌حل‌های نامزد برای این ذره بالاست. به همین ترتیب، نرخ مهاجرت از جزیره برای S_2 بالا خواهد بود، بدین معنی که این ذره بیشتر خصوصیات خود را با سایر ذرات به اشتراک می‌گذارد. شکل ۱۴-۲ یک مدل مهاجرت خطی نامیده می‌شود چرا که در آن μ و λ توابعی خطی از برازندگی هستند.



شکل ۱٤-۲ روابط به اشتراک‌گذاری خصوصیات در BBO. S_1 یک ذره‌ی ضعیف با احتمال کم برای به اشتراک‌گذاری خصوصیات و احتمال زیاد برای دریافت خصوصیات از سایر ذرات را نشان می‌دهد و S_2 یک ذره‌ی خوب با احتمال بالا برای به اشتراک‌گذاری خصوصیات و احتمال کم برای دریافت خصوصیات از سایر ذرات را نشان می‌دهد.

ما از نرخ‌های مهاجرت هر ذره برای به اشتراک‌گذاری اطلاعات بین ذرات به‌صورت احتمالاتی استفاده می‌نماییم. راه‌های زیادی برای پیاده‌سازی جزییات BBO وجود دارد اما ما در این فصل بر روی فرمول‌بندی اصلی BBO تمرکز خواهیم کرد [سایمون، ۲۰۰۸] که با نام BBO مهاجرت-محور جزیی نیز شناخته می‌شود [سایمون، ۲۰۱۱b]. با استفاده از نمادگذاری استانداردی که پیش از این نیز از آن استفاده کردیم، N اندازه‌ی جمعیت بوده، x_k اُمین ذره از جمعیت بوده، بُعد مسئله‌ی بهینه‌سازی برابر n بوده و $x_k(s)$ اُمین متغیر s مستقل در x_k می‌باشد به‌گونه‌ای که $k \in [1,N]$ و $s \in [1,n]$. در هر نسل و برای هر خصوصیت از راه‌حل نامزد kاُم، احتمالی مانند λ_k (احتمال مهاجرت به جزیره) وجود دارد که به‌صورت زیر جایگزین خواهد شد:

$$\lambda_k = \text{احتمال آنکه } s\text{اُمین متغیر مستقل در } x_k \text{ جایگزین شود} \qquad (۱۳-۱٤)$$

اگر یک خصوصیت راه‌حل برای جایگزینی انتخاب شود، آنگاه ما راه‌حل مهاجرت‌کننده از جزیره را با احتمالی متناسب با احتمالات مهاجرت از جزیره $\{\mu_i\}$ انتخاب خواهیم نمود. ما می‌توانیم از هر گونه روش برازندگی-محور برای این مرحله استفاده نماییم (بخش ۸-۷ را ببینید). اگر از انتخاب چرخ رولت استفاده نماییم خواهیم داشت:

$$\Pr(x_j)\left(\text{احتمال انتخاب برای مهاجرت}\right) = \frac{\mu_j}{\sum_{i=1}^{N}\mu_j} \qquad (۱٤-۱٤)$$

این کار الگوریتم شکل ۱۴–۳ را نتیجه خواهد داد. مهاجرت و جهش هر ذره در نسل کنونی قبل از جایگزینی ذرات اتفاق خواهد افتاد، بنابراین به یک جمعیت موقتی z در شکل ۱۴–۳ نیاز خواهد بود. با قرض گرفتن از عبارت‌های به کار رفته در GA [واواک[1] و فوگارتی[2]، ۱۹۹۶]، می‌گوییم شکل ۱۴–۳ یک الگوریتم BBO *نسلی* (که نقطه‌ی مقابل الگوریتم *حالت ماندگار* می‌باشد) را نمایش می‌دهد. در BBO نیز مانند سایر الگوریتم‌های تکاملی معمولاً از نخبه‌گرایی استفاده می‌شود (بخش ۸–٤ را ببینید)، هرچند که این موضوع در شکل ۱۴–۳ نشان داده نشده است.

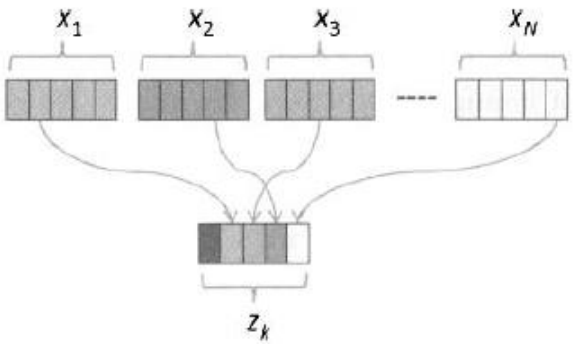
شکل ۱۴–٤ مهاجرت در BBO را به تصویر می‌کشد. این شکل نشان می‌دهد که ذره‌ی z_k خصوصیات سایر ذرات را می‌پذیرد. ما از معادله‌ی (۱۴–۱۳) برای تصمیم‌گیری در مورد جایگزینی هر یک از خصوصیات z_k استفاده می‌کنیم. در مورد شکل ۱۴–٤، تصمیمات زیر حاصل خواهد شد:

۱. برای اولین ویژگی گزینه‌ی مهاجرت انتخاب نمی‌شود، به همین دلیل است که اولین خصوصیت z_k بدون تغییر باقی می‌ماند.
۲. برای دومین ویژگی گزینه‌ی مهاجرت انتخاب می‌شود و شکل ۱۴–٤، x_1 را به‌عنوان ذره‌ی مهاجرت دهنده انتخاب می‌کند. به همین دلیل است که دومین خصوصیت z_k با دومین خاصیت از x_1 جایگزین شده است.
۳. برای سومین ویژگی گزینه‌ی مهاجرت انتخاب می‌شود و شکل ۱۴–٤، x_3 را به‌عنوان ذره‌ی مهاجرت دهنده انتخاب می‌کند. به همین دلیل است که سومین خصوصیت z_k با سومین خاصیت از x_1 جایگزین شده است.
٤. برای چهارمین ویژگی گزینه‌ی مهاجرت انتخاب می‌شود و شکل ۱۴–٤، x_2 را به‌عنوان ذره‌ی مهاجرت دهنده انتخاب می‌کند. به همین دلیل است که چهارمین خصوصیت z_k با چهارمین خاصیت از x_1 جایگزین شده است.
۵. در آخر، برای پنجمین ویژگی نیز گزینه‌ی مهاجرت انتخاب می‌شود و شکل ۱۴–٤، x_N را به‌عنوان ذره‌ی مهاجرت دهنده انتخاب می‌کند. به همین دلیل است که پنجمین خصوصیت z_k با پنجمین خاصیت از x_N جایگزین شده است.

[1] Vavak
[2] Fogarty

جمعیتی آغازین از راه‌حل‌های نامزد را مقداردهی کن: $\{x_k\}$ برای $k \in [1, N]$
تا زمانی که شرایط توقف برآورده نشده است
برای هر x_k احتمال مهاجرت (از) μ_k را متناسب با برازندگی x_k و به‌صورتی که $\mu_k \in [0,1]$ باشد، تعیین کن
برای هر x_k احتمال مهاجرت (از) را به این صورت قرار بده: $\lambda_k = \mu_k - 1$
$\{z_k\} \leftarrow \{x_k\}$
برای هر ذره‌ی z_k
برای هر ویژگی راه‌حل s
از λ_k جهت تصمیم احتمالاتی در مورد مهاجرت کردن به z_k استفاده کن (معادله (۱۴-۱۳) را ببینید)
اگر مهاجرت رخ داد آنگاه
از $\{\mu_i\}_{i=1}^{N}$ برای انتخاب احتمالاتی ذره‌ی مهاجرت کننده x_j استفاده کن (معادله (۱۴-۱۴) را ببینید))
$z_k(s) \leftarrow x_j(s)$
پایان اگر
ویژگی راه‌حل بعدی
$\{z_k\}$ را به‌صورت احتمالاتی دچار جهش کن
ذره‌ی بعدی
$\{x_k\} \leftarrow \{z_k\}$
نسل بعد

شکل ۱۴-۳ طرح کلی الگوریتم BBO با اندازه‌ی جمعیت N. این الگوریتم همچنین با نام BBO مهاجرت-محور جزیی شناخته می‌شود. $\{x_k\}$ تمام جمعیت ذرات بوده، x_k، kاُمین ذره از جمعیت بوده و $x_k(s)$، sاُمین خاصیت از x_k می‌باشد. به همین ترتیب، $\{z_k\}$ جمعیتی موقت از ذرات، z_k، kاُمین ذره از جمعیت ذرات موقت و $z_k(s)$، sاُمین خاصیت از z_k می‌باشند.



شکل ١٤-٤ نمایش مهاجرت BBO برای یک مسئله‌ی ۵ بعدی. خاصیت ۱ برای مهاجرت انتخاب نشده اما خواص ۲-۵ برای مهاجرت انتخاب شده‌اند. معادله‌ی (١٤-١٤) برای انتخاب ذرات مهاجرت دهنده استفاده شده است.

مثال ١٤-٢

این آزمایش ساده‌ی BBO از کار دیوید گلدبرگ "شبیه‌سازی GA به‌صورت دستی" گرفته شده است [گلدبرگ، ١٩٨٩a]. فرض کنید می‌خواهیم x^2 را که در آن x یک عدد صحیح ۵ بیتی است را ماکزیمم کنیم. ابتدا باید در مورد اندازه‌ی جمعیت و نرخ جهش تصمیم بگیریم. بدین منظور جمعیت اولیه را که به‌صورت اتفاقی تولید می‌شود را شامل ٤ ذره و نرخ جهش را برابر ١٪ در نظر می‌گیریم. برای هر ذره، ابتدا میزان برازندگی x^2 را حساب کرده و سپس نرخ مهاجرت را به‌صورت خطی و مانند آنچه که در شکل ١٤-٢ نشان داده شده است، تخصیص می‌دهیم. نرخ‌های مهاجرت باید بین ۰ و ۱ باشند. همچنین ما بیشترین مقدار را کمی کمتر از ۱ و کمترین مقدار را کمی بیشتر از ۰ در نظر می‌گیریم. این کار باعث می‌شود که حتی برای بهترین و بدترین ذرات کمی عدم قطعیت وجود داشته باشد. برای این مثال، به‌صورت دلخواه کمترین مقدار برای μ و λ را برابر $1/N$ و بیشترین مقدار آن‌ها را برابر $(N-1)/N$ در نظر می‌گیریم. $N=4$ اندازه‌ی جمعیت است. فرض کنید جمعیت اولیه که به‌صورت اتفاقی تولید شده است به شرح زیر است.

جدول ١٤-١ مثال ١٤-٢: جمعیت اولیه برای یک مسئله‌ی ساده‌ی BBO.

شماره رشته	x (دودویی)	x (ده‌دهی)	$f(x)=x^2$	μ	λ
۱	۰۱۱۰۱	۱۳	١٦٩	2/5	3/5
۲	۱۱۰۰۰	٢٤	٥٧٦	4/5	1/5
۳	۰۱۰۰۰	٨	٦٤	1/5	4/5
٤	۱۰۰۱۱	۱۹	٣٦١	3/5	2/5

اولین کاری که باید انجام داد، کپی کردن جمعیت x به جمعیت موقت z می‌باشد. سپس، احتمال مهاجرت به هر یک از بیت‌های ذره‌ی اول در جمعیت موقتی، z_1، را که برابر با x_1 است در نظر می‌گیریم. بدین ترتیب خواهیم دید که

$$z_1(1) = 0, \quad z_1(2) = 1, \quad z_1(3) = 1, \quad z_1(4) = 0, \quad z_1(5) = 1 \qquad (۱۵-۱٤)$$

از آنجا که z_1 سومین ذره از لحاظ میزان برازندگی است، نرخ مهاجرت به جزیره برای آن برابر $\lambda_1 = 3/5$ است، بنابراین نرخ مهاجرت به جزیره برای هر بیت از z_1 برابر ۶۰٪ است. برای تعیین مهاجرت به هر بیت از z_1 یک عدد اتفاقی $r \sim U[0,1]$ تولید می‌نماییم.

۱. فرض کنید $r = 0.7$. از آنجا که $r > \lambda_1$ است مهاجرتی به $z_1(1)$ صورت نخواهد پذیرفت و این بیت بدون تغییر باقی خواهد ماند.

۲. فرض کنید عدد اتفاقی تولید شده‌ی بعدی $r = 0.3$ است. از آنجا که $r < \lambda_1$ است مهاجرت به $z_1(2)$ صورت خواهد پذیرفت. برای انتخاب بیت مهاجرت کننده از انتخاب چرخ رولت استفاده می‌نماییم. $x_3(2)$ دارای بیشترین احتمال برای مهاجرت به $z_1(2)$ بوده، $x_1(2)$ و $x_4(2)$ از این لحاظ به ترتیب در رتبه‌ی دوم و سوم بوده، و $x_2(2)$ دارای کمترین احتمال برای مهاجرت به $z_1(2)$ خواهد بود. ما می‌توانیم از در نظر گرفتن $x_1(2)$ صرف نظر کنیم چرا که z_1 یک کپی از x_1 است، اما این کار یکی از جزییات پیاده‌سازی است و به سلیقه‌ی مهندس بستگی دارد. فرض کنید استفاده از چرخ رولت به انتخاب شدن $x_3(2)$ برای مهاجرت به منجر شود. بنابراین، $z_1(2) \leftarrow x_3(2) = 1$. هرچند که مهاجرت به $z_1(2)$ صورت پذیرفت، مقدار آن تغییری نکرد.

۳. ما این فرایند را برای $z_1(3)$، $z_1(4)$ و $z_1(5)$ ادامه خواهیم داد. فرض کنید که اعداد اتفاقی تولید شده برای این بیت‌ها به‌صورت زیر باشد:
   - $z_1(3) = 1$ (بدون مهاجرت)
   - $z_1(4) \leftarrow x_4(4) = 1$ (مهاجرت)
   - $z_1(5) = 1$ (بدون مهاجرت)

   حال فرایند مهاجرت برای z_1 تکمیل شده و $z_1 = 01111$ است.

٤. مراحل ۱-۳ را برای z_2، z_3 و z_4 تکرار می‌کنیم.

۵. سپس احتمال جهش را برای هر یک از بیت‌ها در هر یک از چهار ذره‌ی موقتی z_1، z_2، z_3 و z_4 در نظر می‌گیریم. جهش را می‌توان همان‌طور که در مورد سایر الگوریتم‌های تکاملی پیاده‌سازی نمودیم، در اینجا نیز پیاده‌سازی کنیم (بخش ۸-۹ را ببینید).

٦. حال که جمعیتی اصلاح شده از $\{z_k\}$ را در اختیار داریم، z_k را برای $k \in [1,4]$ به x_k کپی کرده و بدین ترتیب اولین نسل از BBO تکمیل می‌شود.

فرایند بالا تا زمان برآورده شدن یک معیار معین ادامه می‌یابد. برای نمونه، می‌توان برای یک تعداد نسل مشخص و یا تا زمان رسیدن به یک مقدار برازندگی مطلوب و تا زمان بدون تغییر ماندن مقدار برازندگی فرایند بالا را ادامه داد (بخش ۸–۲ را ببینید).

۴–۱۴ بسط BBO

این بخش بسط‌هایی از BBO را مورد بررسی قرار خواهد داد که می‌توانند به بهبود عملکرد آن کمک کنند. بدین منظور، شکل منحنی‌های مهاجرت، مهاجرت مخلوط و روش‌های جایگزین برای پیاده‌سازی BBO مورد بحث قرار خواهند گرفت. ما این بخش را در زیربخش ١٤–٤–٤ با بحث در مورد آن که آیا باید BBO را به جای یک الگوریتم تکاملی جدا به‌عنوان نوعی از GA در نظر گرفت یا خیر، به پایان خواهیم برد.

۱–۴–۱۴ منحنی‌های مهاجرت

تا به اینجا فرض کرده‌ایم منحنی‌های مهاجرت BBO مانند آنچه که در شکل ١٤–٢ نشان داده شده است، منحنی‌های خطی هستند. این یک فرض ساده است و متناظر با انتخاب رتبه–محور خطی است (بخش ٨–٧–٤ را ببینید). اما در جغرافیای زیستی، منحنی‌های مهاجرت غیرخطی هستند. شکل دقیق منحنی‌های مهاجرتِ جغرافیای زیستی به سختی قابل تعیین بوده و از یک جزیره به جزیره‌ی دیگر متفاوت هستند. با این حال، منحنی‌های بسیاری در طبیعت شکلی S گونه دارند. در مقاله‌ی اصلی BBO [سایمون، ۲۰۰۸] حدس زده شده است که منحنی‌های مهاجرت غیرخطی عملکردی بهتر از منحنی‌های خطی داشته باشند. این موضوع به کاوش برخی منحنی‌های مهاجرت متفاوت در [ما و همکاران، ۲۰۰۹] و [ما، ۲۰۱۰] منجر شده است. در اینجا ما به بررسی نویددهنده‌ترین منحنی‌ها می‌پردازیم: منحنی‌های مهاجرت به شکل S.

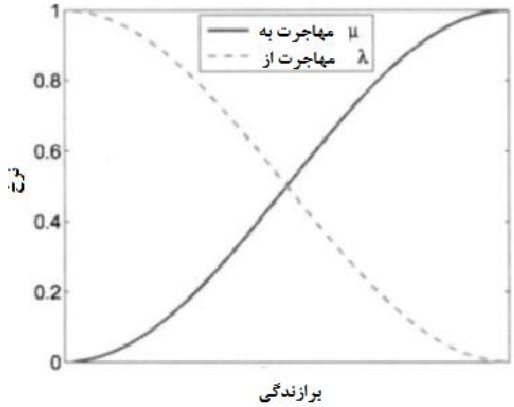
شکل ١٤–٢ نرخ‌های مهاجرت غیرنرمالیزه را به‌صورت زیر مدل می‌کند

$$\mu_k = r_k$$
$$\lambda_k = 1 - r_k \qquad (١٦–١٤)$$

که در آن r_k رتبه‌ی ذره‌ی kاُم در جمعیت بوده بدین صورت که رتبه‌ی بدترین ذره ۱ و رتبه‌ی بهترین ذره N در نظر گرفته می‌شود (N اندازه‌ی جمعیت است). مدل‌سازی نرخ‌های مهاجرت به‌صورت سینوسی، نرخ‌های جهش را به‌صورت زیر به دست خواهد داد

$$\mu_k = \frac{1}{2}\left(1 - \cos\left(\frac{\pi r_k}{N}\right)\right) \qquad (۱۴-۱۷)$$
$$\lambda_k = 1 - \mu_k$$

این معادلات به منحنی‌های S شکل زیر منجر خواهند شد.



شکل ۱۴-۵ مدل مهاجرت BBO سینوسی. این شکل را با شکل ۱۴-۲ مقایسه کنید.

مثال ۱۴-۳

اگر جغرافیای زیستی طبیعی به راستی یک فرایند بهینه‌سازی باشد، می‌توان استدلال نمود که هر چه مدل‌سازی BBO به جغرافیای زیستی طبیعی نزدیکتر باشد، عملکرد بهینه‌سازی بهتر خواهد شد. با در نظر داشتن این موضوع، BBO خطی و BBO سینوسی را برای مجموعه‌ای از مسائل محک ۲۰ بعدی شبیه‌سازی کرده و نتایج آن را در جدول ۱۴-۲ نشان می‌دهیم. در این شبیه‌سازی‌ها اندازه‌ی جمعیت برابر ۵۰، محدودیت نسل برای هر بار اجرای BBO برابر ۵۰، و نرخ جهش برابر ۱٪ در هر خصوصیت راه‌حل می‌باشد. ما جهش را با تولید یک خصوصیت راه‌حل که به‌صورت یکنواخت میان مقادیر بیشینه و کمینه‌ی دامنه توزیع شده است و با احتمال ۱٪ در ذره در نسل، پیاده‌سازی می‌نماییم. همچنین پارامتر نخبه‌گرایی را برابر ۲ قرار می‌دهیم. این بدین معنی است که بهترین دو ذره را در هر نسل برای نسل بعد نگه خواهیم داشت.

جدول ۱۴-۲ به وضوح نشان می‌دهد که برای محک‌های استاندارد، مهاجرت سینوسی بهتر از مهاجرت خطی عمل می‌نماید. به‌طور میانگین، عملکرد مهاجرت سینوسی ۴۳٪ از عملکرد مهاجرت خطی بهتر است. این نشان‌دهنده‌ی آن است که مدل‌های مهاجرتی که به طبیعت نزدیکتر هستند بهتر از مدل‌های ساده عمل کرده و این خود متضمن این است که جغرافیای زیستی به راستی یک فرایند بهینه‌سازی است.

جدول ۱۴-۲ مثال ۱۴-۳: عملکرد نسبی BBO با مدل‌های مهاجرت خطی و سینوسی. جدول مینیمم نرمالیزه شده‌ی پیدا شده توسط دو نسخه‌ی BBO را که بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. برای تعاریف توابع محک به ضمیمه‌ی ج مراجعه نمایید.

محک	مهاجرت خطی	مهاجرت سینوسی
آکلی	۱,۰۳۷۳	۱
فلچر	۱,۲۰۱۵	۱
گرینوانک	۱,۲۳۶۷	۱
مجازات ۱	۱,۴۲۴۹	۱
مجازات ۲	۴,۳۲۶۵	۱
درجه ۴	۱,۶۸۷۶	۱
رستریجین	۱,۰۶۶۵	۱
روزنبروک	۱,۰۷۵۹	۱
اشوفل ۱,۲	۱,۰۹۸۰	۱
اشوفل ۲,۲۱	۱,۰۴۶۸	۱
اشوفل ۲,۲۲	۱,۰۷۲۱	۱
اشوفل ۲,۲۶	۱,۲۴۷۱	۱
کروی	۱,۲۵۸۲	۱
پله	۱,۲۶۸۳	۱
میانگین	۱,۴۳۱۹	۱

۱۴-۴-۲ مهاجرت مخلوط

پیش از این نشان داده شده است که برش مخلوط می‌تواند باعث بهبود عملکرد GAها و سایر الگوریتم‌های تکاملی شود [مک‌تاویش[1] و رسترپو[2]، ۲۰۰۸]، [مزورا-مونتس[3] و گالومک-اوریتیز[4]، ۲۰۰۹]، [مولهنبیتو شیلرکمپ-ووسن، ۱۹۹۳] (بخش ۸-۸-۹ را ببینید). در برش مخلوط GA، به جای کپی کردن ژن

[1] McTavish
[2] Restrepo
[3] Mezura-Montes
[4] Palomeque-Oritiz

یک تک والد به ژن فرزند، ژن فرزند از ترکیب محدبِ دو ژن والد به دست می‌آید. به همین ترتیب می‌توان از یک اپراتور مهاجرت مخلوط برای BBO استفاده نمود [ما و سایمون، ۲۰۱۰]، [ما و سایمون، ۲۰۱۱b]. در الگوریتم استاندارد BBO از شکل ١٤-٣، یک خصوصیت s از ذره‌ی z_k کاملا با یک خصوصیت از ذره‌ی x_j جایگزین می‌شود:

$$z_k(s) \leftarrow x_j(s) \qquad (١٨-١٤)$$

در مهاجرت مخلوط، یک خصوصیت از ذره‌ی z_k به سادگی با یک خصوصیت از ذره‌ی x_j جایگزین نمی‌شود، بلکه خصوصیت ذره‌ی z_k برابر با ترکیب محدب خصوصیت z_k و خصوصیت x_j قرار داده می‌شود.

$$z_k(s) \leftarrow \alpha z_k(s) + (1-\alpha)x_j(s) \qquad (١٩-١٤)$$

که در آن $\alpha \in (0,1)$. اگر $\alpha = 0$ می‌باشد، BBO مخلوط به BBO استاندارد تبدیل می‌شود. بنابراین، BBO مخلوط تعمیمی است از BBO استاندارد. پارامتر مخلوط α می‌تواند اتفاقی، قاطع و یا متناسب با برازندگی نسبی z_k و x_j باشد.

مهاجرت مخلوط برای مسائل با خصوصیات راه‌حل پیوسته مناسب است. با این که می‌توان این نوع مهاجرت را برای مسائل با خصوصیات راه‌حل گسسته نیز به کار برد، اما این ایده را در اینجا بررسی نخواهیم نمود. در مورد مهاجرت مخلوط نسبت به مهاجرت استاندارد، توضیحاتی چند مورد نیاز است. اول آنکه، احتمال تخریب ذرات خوب به دلیل مهاجرت کمتر خواهد بود چرا که این ذرات نسبتی مشخص از خواص اولیه خود را در طول فرایند مهاجرت نگه می‌دارند. دوم آنکه، ذرات ضعیف همچنان حداقل قسمتی از خصوصیات راه‌حل را از ذرات خوب در طول فرایند مهاجرت می‌پذیرند.

مثال ١٤-٤

برای بررسی تأثیر مهاجرت مخلوط بر روی عملکرد BBO، BBO استاندارد و BBO مخلوط را با $\alpha = 0.5$ بر روی مجموعه‌ای از توابع محک ۲۰ بعدی اجرا می‌نماییم. در اینجا نیز از همان پارامترهای استفاده شده در مثال ١٤-٣ استفاده کرده و نتایج را در جدول ١٤-٣ ارائه کرده‌ایم.

جدول ١٤-٣ مثال ١٤-٤: عملکرد نسبی BBO استاندارد و BBO مخلوط با $\alpha = 0.5$. جدول بهینه‌ی نرمالیزه شده پیدا شده توسط دو نسخه‌ی BBO را که بر روی ۵۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. برای تعاریف توابع محک به ضمیمه‌ی ج مراجعه کنید.

محک	*BBO* استاندارد	*BBO* مخلوط
آکلی	١,٦٥٥٩	١,٠
فلچر	١,٠	٢,٣٣٨
گرینوانک	٣,٤٥٣٦	١,٠
مجازات ١	٧٠١,٤٧	١,٠
مجازات ٢	٨٨١٧,٧	١,٠
درجه ٤	٤٩,٦٦٣	١,٠
رستریجین	١,٠	١,٦٨٩٢
روزنبروک	٣,٩٠٠٩	١,٠
اشوفل ١,٢	١٢,٦٣	١,٠
اشوفل ٢,٢١	٤,٠٨٤٦	١,٠
اشوفل ٢,٢٢	١,٣٢٨٠	١,٠
اشوفل ٢,٢٦	١,٠	٤,٨٢١٣
کروی	٥,٤٣٥٩	١,٠
پله	٤,٥٠٠٧	١,٠
میانگین	٦٨٦,٣٤	١,٤٢١٣

جدول ١٤-٣ نشان می‌دهد که BBO مخلوط در ١١ مورد از ١٤ تابع محک بهتر از BBO استاندارد عمل می‌کند. بزرگی این بهبود بسیار قابل توجه است. در سه موردی که BBO استاندارد از BBO مخلوط بهتر عمل می‌نماید، ضریب بهبود میانگین برابر ٣ است. این در حالی است که در ١١ مورد دیگر که BBO مخلوط از BBO استاندارد بهتر عمل می‌نماید، میانگین ضریب بهبود حدود ٨٨١٨ است.

۱۴-۴-۳ رویکردهای دیگر به BBO

الگوریتم ارائه شده در شکل ١٤-٣ BBO مهاجرت-محور جزیی نام دارد [سایمون، ٢٠١١b]. واژه‌ی جزیی در این اسم از آن رو به کار رفته است که در هر زمان تنها یک ویژگی راه‌حل برای مهاجرت در نظر

گرفته شده است. این بدین معنی است که برای هر ویژگی راه‌حل از ذره‌ی z_k، λ_k نسبت به یک عدد اتفاقی سنجیده می‌شود تا در مورد جایگزینی یا عدم جایگزینی آن ویژگی تصمیم گرفته شود. عبارت مهاجرت-محور نیز بدین معناست که ابتدا از λ_k برای تصمیم‌گیری در مورد مهاجرت یا عدم مهاجرت استفاده شده و سپس از متغیرهای $\{\mu_i\}$ برای انتخاب راه‌حل مهاجرت دهنده استفاده می‌شود (که این عمل با استفاده از فرایندی چون چرخ رولت صورت می‌پذیرد).

با این حال، روش‌های دیگری برای پیاده‌سازی BBO نیز وجود دارد. به جای مقایسه‌ی λ_k با یک عدد اتفاقی برای هر ویژگی راه‌حل، می‌توان λ_k را تنها یک بار برای هر ذره با یک عدد اتفاقی مقایسه نمود. سپس اگر تصمیم بر مهاجرت گرفته شد، همه‌ی ویژگی‌های راه‌حل را در z_k جایگزین نمود. این نوع BBO را می‌توان BBO مهاجرت-محور کلی نامید.

همچنین، می‌توان ابتدا از μ_k برای تصمیم‌گیری در مورد ذره‌ی مهاجرت‌دهنده استفاده نمود. سپس، تنها در صورتی که تصمیم بر مهاجرت دادن گرفته شد، از متغیرهای $\{\lambda_k\}$ برای انتخاب چرخ رولت و تصمیم‌گیری در مورد مقصد ویژگی راه‌حل انتخاب شده برای مهاجرت استفاده نمود. این ایده را می‌توان BBO مهاجرت‌دهنده-محور نامید.

با ترکیب ایده‌های بالا می‌توان به چهار نوع مختلف از پیاده‌سازی‌های BBO دست یافت. اول BBO مهاجرت-محور جزیی است که پیاده‌سازی پیش‌فرض است و در شکل ۱۴-۳ نمایش داده شده است. سه نوع دیگر در شکل‌های ۱۴-۶ تا ۱۴-۸ نمایش داده شده‌اند. به علاوه، هر یک از این روش‌ها را می‌توان با مهاجرت سینوسی و یا مهاجرت مخلوط که به ترتیب بخش‌های ۱۴-۴-۱ و ۱۴-۴-۲ توضیح داده شدند، ترکیب نمود. مانند سایر الگوریتم‌های تکاملی باید جهش و نخبه‌گرایی را نیز پیاده‌سازی نماییم، هرچند که این فرایندها در شکل‌های ۱۴-۶ تا ۱۴-۸ نشان داده نشده‌اند. تحقیقات نظری و عملی این نوع BBOها در [ما و سایمون، ۲۰۱۳] گزارش شده است.

جمعیتی آغازین از راه‌حل‌های نامزد را مقداردهی کن: $\{x_k\}$ برای $k \in [1, N]$
تا زمانی که شرایط توقف برآورده نشده است
  برای هر x_k احتمال مهاجرت (از) μ_k را متناسب با برازندگی x_k و به‌صورتی که $\mu_k \in [0,1]$ باشد، تعیین کن
  برای هر x_k احتمال مهاجرت (از) را به این صورت قرار بده: $\lambda_k = \mu_k - 1$
  $\{z_k\} \leftarrow \{x_k\}$
  برای هر ذره‌ی x_k

برای هر ویژگی راه‌حل s
از μ_k جهت تصمیم احتمالاتی در مورد مهاجرت دادن x_k استفاده کن
اگر مهاجرت رخ داد آنگاه
از $\{\lambda_i\}$ برای انتخاب احتمالاتی ذره‌ی مهاجرت کننده z_j استفاده کن
$z_j(s) \leftarrow x_k(s)$
پایان اگر
ویژگی راه‌حل بعدی
$\{z_k\}$ را به‌صورت احتمالاتی دچار جهش کن
ذره‌ی بعدی
$\{x_k\} \leftarrow \{z_k\}$
نسل بعد

شکل ١٤-٦ الگوریتم بالا، طرح کلی BBO مهاجرت‌دهنده-محور جزیی را با اندازه‌ی جمعیت N نشان می‌دهد. $\{x_k\}$ کل جمعیت بوده، x_k ذره‌ی kاُم از جمعیت و $x_k(s)$، ویژگی sاُم x_k می‌باشد. به همین ترتیب، $\{z_k\}$ کل جمعیت موقت بوده، z_k ذره‌ی kاُم از جمعیت موقت و $z_k(s)$، ویژگی sاُم z_k می‌باشد.

جمعیتی آغازین از راه‌حل‌های نامزد را مقداردهی کن: $\{x_k\}$ برای $k \in [1, N]$
تا زمانی که شرایط توقف برآورده نشده است
برای هر x_k احتمال مهاجرت (از) μ_k را متناسب با برازندگی x_k و به‌صورتی که $\mu_k \in [0,1]$ باشد، تعیین کن
برای هر x_k احتمال مهاجرت (از) را به این صورت قرار بده: $\lambda_k = \mu_k - 1$
$\{z_k\} \leftarrow \{x_k\}$
برای هر ذره‌ی z_k
از λ_k جهت تصمیم احتمالاتی در مورد مهاجرت دادن z_k استفاده کن
اگر مهاجرت رخ داد آنگاه
برای هر ویژگی راه‌حل s
از $\{\mu_i\}$ برای انتخاب احتمالاتی ذره‌ی مهاجرت کننده x_j استفاده کن
$z_k(s) \leftarrow x_j(s)$
ویژگی راه‌حل بعدی

```
        پایان اگر
        ذره‌ی بعدی
    {x_k} ← {z_k}
نسل بعد
```

شکل ١٤-٧ الگوریتم بالا، طرح کلی BBO مهاجرت-محور کلی را با اندازه‌ی جمعیت N نشان می‌دهد. $\{x_k\}$ کل جمعیت بوده، x_k ذره‌ی kاُم از جمعیت و $x_k(s)$، ویژگی sاُم x_k می‌باشد. به همین ترتیب، $\{z_k\}$ کل جمعیت موقت بوده، z_k ذره‌ی kاُم از جمعیت موقت و $z_k(s)$، ویژگی sاُم z_k می‌باشد.

```
جمعیتی آغازین از راه‌حل‌های نامزد را مقداردهی کن: {x_k} برای k ∈ [1, N]
تا زمانی که شرایط توقف برآورده نشده است
    برای هر x_k احتمال مهاجرت (از) μ_k را متناسب با برازندگی x_k و به‌صورتی که μ_k ∈ [0,1] باشد، تعیین کن
    برای هر x_k احتمال مهاجرت (از) را به این صورت قرار بده: λ_k = μ_k − 1
    {z_k} ← {x_k}
    برای هر ذره‌ی x_k
        از μ_k جهت تصمیم احتمالاتی در مورد مهاجرت دادن x_k استفاده کن
        اگر مهاجرت رخ داد آنگاه
            برای هر ویژگی راه‌حل s
                از {λ_i} برای انتخاب احتمالاتی ذره‌ی مهاجرت کننده z_j استفاده کن
                z_j(s) ← x_k(s)
            ویژگی راه‌حل بعدی
        پایان اگر
        ذره‌ی بعدی
    {x_k} ← {z_k}
نسل بعد
```

شکل ١٤-٨ الگوریتم بالا، طرح کلی BBO مهاجرت‌دهنده-محور کلی را با اندازه‌ی جمعیت N نشان می‌دهد. $\{x_k\}$ کل جمعیت بوده، x_k ذره‌ی kاُم از جمعیت و $x_k(s)$، ویژگی sاُم x_k می‌باشد. به همین ترتیب، $\{z_k\}$ کل جمعیت موقت بوده، z_k ذره‌ی kاُم از جمعیت موقت و $z_k(s)$، ویژگی sاُم z_k می‌باشد.

۱۴–۴–۴ BBO و الگوریتم‌های ژنتیک

این بخش به بحث در مورد رابطه‌ی میان GAها و BBO می‌پردازد. در GAها با برش یکنواخت، هر ژن فرزند به‌صورت اتفاقی از یکی از والدینش انتخاب می‌شود (بخش ۸–۸–٤ را ببینید). در بازترکیب استخر ژن، که با نام بازترکیب چند–والده و برش پویشی نیز شناخته می‌شود، هر ژن فرزند به‌صورت اتفاقی از یکی از والدین انتخاب می‌شود (تعداد والدین بیش از دو والد است) (بخش ۸–۸–۵ را ببینید). برای پیاده‌سازی بازترکیب استخر ژن، چند انتخاب باید صورت بپذیرد. برای مثال، چند ذره باید در استخر والدین بالقوه قرار داشته باشد؟ معیار انتخاب ذرات برای استخر چیست؟ بعد از تعیین استخر، والدین چگونه باید از این استخر انتخاب شوند؟ یک راه برای پیاده‌سازی بازترکیب استخر ژن روشی است که با نام بازترکیب یکنواخت جهانی شناخته می‌شود. در این روش هر ژن فرزند به‌صورت اتفاقی از یکی از والدین انتخاب شده به‌صورتی که جمعیت والدین با کل جمعیت GA برابر است و انتخاب اتفاقی بر اساس مقادیر برازندگی صورت می‌گیرد (برای مثال، انتخاب چرخ رولت).

اگر از بازترکیب یکنواخت جهانی استفاده نماییم، و همچنین اگر از انتخاب برازندگی–محور برای هر ویژگی راه‌حل فرزندان بهره ببریم، الگوریتم شکل ١٤–٩ که الگوریتم ژنتیک با بازترکیب یکنواخت جهانی (GA/GUR[1]) نامیده می‌شود، حاصل خواهد شد. با مقایسه‌ی شکل‌های ١٤–٣ و ١٤–١٩ می‌توان دید که BBO تعمیمی از نوع خاصی از GA/GUR است. این بدین دلیل است که اگر ما در شکل ١٤–٣ به جای $\lambda_k = 1 - \mu_k$ از $\lambda_k = 1$ برای همه‌ی kها استفاده نماییم، الگوریتم BBO از شکل ١٤–٣ با الگوریتم GA/GUR از شکل ١٤–١٩ معادل خواهد بود.

جمعیتی آغازین از راه‌حل‌های نامزد را مقداردهی کن: $\{x_k\}$ برای $k \in [1, N]$
تا زمانی که شرایط توقف برآورده نشده است
  برای $k = 1$ تا N
    فرزند $_k \leftarrow [0\ 0\ \ldots 0] \in R^n$
    برای هر ویژگی راه‌حل $s = 1$ تا n
      از مقادیر برازندگی برای انتخاب ذره‌ی x_j
      فرزند $_k(s) \leftarrow x_j(s)$

[1] Genetic Algorithm with Global Uniform Recombiniation

ویژگی راه‌حل بعدی

فرزند را به‌صورت احتمالاتی دچار جهش کن

k

فرزند بعدی

$\{x_k\} \leftarrow \{$ فرزند$\}$

k

نسل بعد

شکل ١٤-١٩ طرح کلی الگوریتم ژنتیک با بازترکیب یکنواخت جهانی (GA/GUR) برای یک مسئله‌ی بهینه‌سازی n بعدی. N اندازه‌ی جمعیت، $\{x_k\}$ کل جمعیت بوده، x_k ذره‌ی kاُم از جمعیت و $x_k(s)$، ویژگی sاُم x_k می‌باشد.

بحث ارائه شده در این بخش مانند بحث صورت گرفته در بخش ١٢-٤ می‌باشد، جایی که ما نشان دادیم DE حالت خاص GA پیوسته است. در آن بخش دیدیم که با وجود شباهت‌های بسیاری که میان DE و GAها وجود داشت، آنقدر متمایز بود که توانستیم آن را به‌عنوان یک الگوریتم تکاملی جداگانه به حساب آوریم. نتیجه‌ای مشابه را در این بخش نیز می‌توان گرفت. با وجود شباهت‌های بسیاری که میان BBO و GAها وجود دارد، آنقدر متمایز هست که بتوان آن را به‌عنوان یک الگوریتم تکاملی جداگانه به حساب آورد. همچنین یک دلیل محکم دیگر برای در نظر گرفتن BBO به‌عنوان یک الگوریتم تکاملی جداگانه وجود دارد و آن این است که ریشه‌ی زیست‌جغرافیایی BBO جای زیادی را برای اصلاحات و تعمیم‌های اضافی ایجاد می‌کند. ما یکی از این تعامیم را در بخش ١٤-٤-١ معرفی کردیم و سایر این تعامیم را در نتیجه‌گیری این فصل ارائه خواهیم نمود.

١٤-٥ نتیجه‌گیری

دیدیم که چگونه جغرافیای زیستی، که علم مطالعه‌ی توزیع گونه‌های زیستی می‌باشد، می‌تواند برای دستیابی به الگوریتم بهینه‌سازی زیست‌جغرافی-محور (BBO) استفاده شود. الگوریتم BBO با استفاده از نظریه‌ی مارکوف [سایمون و همکاران، ٢٠١١a]، سیستم‌های پویا [سایمون، ٢٠١١a] و مکانیک آماری [ما و همکاران، ٢٠١٣]، مدل شده است. برخی از این مدل‌ها مشابه مدل‌هایی است که برای GAها به دست آوردیم (فصل ٤ را ببینید). مدل‌های مارکوف GA و BBO در [سایمون و همکاران، ٢٠١١b] مقایسه شده‌اند. مدل مارکوف BBO در [سایمون و همکاران، ٢٠٠٩] به BBO با نخبه‌گرایی تعمیم داده شده است. مانند بسیاری الگوریتم‌های تکاملی دیگر، BBO نیز به بسیاری مسائل دنیای واقعی اعمال شده است. یک وبسایت مختص BBO را می‌توان در [سایمون، ٢٠١٢] یافت.

یکی از کمبودهای الگوریتم BBO ارائه شده در این فصل آن است که این الگوریتم در هر زمان تنها یک متغیر وابسته را میان راه‌حل‌ها مهاجرت می‌دهد. این موضوع برای مسائل تفکیک‌پذیر مشکلی ایجاد نمی‌کند. مسائل تفکیک‌پذیر مسائلی هستند که تابع برازندگی آن‌ها به‌صورت زیر است

$$f(x) = \sum_{i=1}^{n} f_i(x(s_i)) \qquad (۲۰-۱٤)$$

که در آن $x(s_i)$، iامین متغیر مستقل از x بوده و n ابعاد مسئله است. با این حال، بسیاری از مسابل بهینه‌سازی تفکیک‌پذیر نیستند. این بدین معناست که اگر یک راه‌حل نامزد شامل گروهی از متغیرهای مستقل باشد که آن را بسیار برازنده می‌سازند، هیچ راه آسانی برای مهاجرت دادن این گروه به یک راه‌حل نامزد دیگر وجود ندارد. یک راه‌حل ممکن برای این کمبود آن است که الگوریتم BBO را به‌گونه‌ای اصلاح نماییم که به جای یک متغیر مستقل در هر زمان، یک گروه اتفاقی از این متغیرها مهاجرت نمایند. چیزی شبیه به این ایده در [عمران و همکاران، ۲۰۱۳] پیشنهاد شده است.

BBO در واقع خانواده‌ای از الگوریتم‌ها است، بنابراین می‌توان آن را یک فوق ابتکار به حساب آورد. این شامل گزینه‌های نشان داده شده در جدول ١٤-٤ می‌شود. یک گزینه‌ی احتمالی برای تحقیقات آتی، مطالعه‌ی نظام‌مند ترکیبات گزینه‌های جدول ١٤-٤ می‌باشد.

جدول ١٤-٤ گزینه‌های پیاده‌سازی BBO. BBO را می‌توان با هر ترکیبی از گزینه‌های ستون اول، دوم و سوم این جدول پیاده‌سازی نمود.

رویکردهای مهاجرت	منحنی‌های مهاجرت	ترکیبات مهاجرت
مهاجرت به- محور جزیی	خطی	هیچ $(\alpha = 0)$
مهاجرت به- محور کلی	سینوسی	$\alpha = 0.5$
مهاجرت از- محور جزیی	غیره	یک ثابت دیگر $= \alpha$
مهاجرت از- محور کلی		برازندگی $\propto \alpha$

امکانات بسیاری برای هم‌ترازسازی هر چه بیشتر BBO با جغرافیای زیستی وجود دارد که از جمله‌ی آن‌ها می‌توان به موارد زیر اشاره نمود:

تشابه زیستگاه: در جغرافیای زیستی جزیره‌ها، نرخ مهاجرت به جزیره با انزوای جزیره متناسب است [آدلر[1] و نیوارنبرگر[2]، ۱۹۹٤]. جزایری که منزوی هستند به میزان متناسب و مناسبی از مهاجرت متأثر هستند. این موضوع تأثیر فاصله نام دارد [وو[3] و وانکات[4]، ۱۹۹۵]. همچنین می‌توان گفته که میزان مهاجرت از جزیره نیز با میزان انزوای جزیره ارتباط دارد. در جغرافیای زیستی جزیره‌ها، منحصر به فرد بودن محیطیِ جزیره به انزوای جزیره بستگی دارد چرا که شرایط محیطی نسبت به فاصله به طرزی قابل پیش‌بینی تغییر می‌کند [لومولینو[5]، ۲۰۰۰a]. در BBO، انزوای راه‌حل‌های نامزد با منحصر به فرد بودن آن‌ها در ارتباط است، بدین معنی که می‌توان راه‌حل‌های مشابه را در خوشه‌هایی در فضای راه‌حل کنار هم در نظر گرفت و راه‌حل‌های غیرمشابه را راه‌حل‌های منزوی در فضای راه‌حل فرض نمود. به بیان جغرافیای زیستی، راه‌حل‌های مشابه به یک مجمع‌الجزایر تعلق دارند. این موضوع باعث افزایش مهاجرت میان راه‌حل‌های مشابه و کاهش مهاجرت میان راه‌حل‌های غیرمشابه می‌شود. این موضوع را می‌توان در BBO به‌صورت افزایش احتمال به اشتراک‌گذاری خصوصیات راه‌حل میان راه‌حل‌های مشابه پیاده‌سازی نمود. این کار مشابه برش گونه-محور (یا نیچینگ) در GAها [استنلی و میککولاینن[6]، ۲۰۰۲] و همچنین مدل جزایر متمایز [گوستافسن[7] و بورک، ۲۰۰٦] می‌باشد (بخش ۸-۲ را ببینید). این موضوع همچنین مشابه ایده‌ی همسایگی‌ها در بهینه‌سازی تجمع ذرات می‌باشد [کندی و ابرهارت، ۲۰۰۱] (فصل ۱۱ را ببینید). با این حال، انگیزه و مکانیزم این ایده و ایده‌های مشابه در GA و تجمع ذرات کاملا متفاوت است. نیچینگ در GAها بر پایه‌ی احتمال ذرات برای جفت‌گیری با ذرات مشابه قرار دارد. همسایگی‌ها در PSO نیز بر پایه‌ی احتمال جمع شدن ذرات به دور یکدیگر در فضای راه‌حل است. ایده‌ی مجمع‌الجزایرها در BBO نیز بر پایه‌ی احتمال خوشه شدن جزایر مشابه قرار دارد. یک راه کمی برای تعیین تأثیر انزوای جزیره بر روی نرخ‌های مهاجرت در [هانسکی، ۱۹۹۹] ارائه شده است.

مهاجرت ابتدایی: نظریه‌ی جغرافیای زیستی کلاسیک بیان می‌دارد که همانند آنچه که در شکل‌های ۱-۱٤ و ۵-۱٤ نشان داده شده است، با افزایش تعداد گونه‌ها، نرخ مهاجرت به جزیره کاهش می‌یابد. در BBO این موضوع با کاهش یکنواخت نرخ مهاجرت به جزیره با افزایش برازندگی ذرات، متناظر است. این بدین معنی است که هر چه یک ذره برازنده‌تر می‌شود، احتمال پذیرفتن خصوصیت از سایر ذرات برای آن

[1] Adler
[2] Nuernbeger
[3] Wu
[4] Vankat
[5] Lomolino
[6] Miikkulainen
[7] Gustafson

ذره کاهش خواهد یافت. با این حال، برخی پیشرفت‌های اخیر در جغرافیای زیستی گویای آن‌اند که برای برخی گونه‌های پیشرو (برای مثال گیاهان)، یک افزایش ابتدایی در تعداد گونه‌ها باعث یک افزایش ابتدایی در نرخ مهاجرت به جزیره می‌شود [وو و وانکات، ۱۹۹۵]. این بدین دلیل است که این مهاجرین ابتدایی باعث اصلاح جزیره و سازگاری بهتر آن می‌شوند. این بدین معنی است که تأثیر مثبت افزایش تنوع ناشی شده از مهاجرت ابتدایی بر تأثیر منفی افزایش اندازه‌ی جمعیت می‌چربد. در BBO این موضوع با یک افزایش ابتدایی در نرخ مهاجرت به جزیره به دلیل بهبود ابتدایی برازندگی ذرات بسیار ضعیف، متناظر می‌باشد. این یک مکانیزم فیدبک مثبت موقتی در BBO است. یک ذره‌ی بسیار ضعیف از سایر ذرات ویژگی‌هایی را پذیرفته و بدین ترتیب برازندگی‌اش افزایش می‌یابد و این خود باعث افزایش احتمال پذیرفتن خصوصیات بیشتر توسط این ذره می‌شود. این موضوع در شکل ۱٤-۱۰ نشان داده شده است. این ایده را می‌توان در سایر الگوریتم‌های تکاملی نیز به کار برد اما باید توجه داشت که ریشه‌ی اصلی این ایده در جغرافیای زیستی است [ما و سایمون، ۲۰۱۱a].

حداقل برازندگی مورد نیاز: ما می‌توانیم فرض کنیم که یک زیستگاه برای داشتن یک نرخ مهاجرت از جزیره‌ی غیرصفر باید دارای یک رتبه‌ی برازندگی مینیمم باشد. این مانند آن است که فرض کنیم یک جزیره باید دارای یک HSI غیرصفر باشد تا بتواند هر نوع گونه‌ای را در خود جای دهد [هانسکی و گیلپین، ۱۹۹۷].

شکل ۱٤-۱۰ مدلی که نشان می‌دهد نرخ مهاجرت به جزیره در ابتدا با افزایش برازندگی افزایش می‌یابد. این موضوع به ذرات ضعیف، اما مورد بهبود، نیروی آنی برای ادامه‌ی بهبود را می‌دهد. با ادامه‌ی بهبود برازندگی ذرات بعد از افزایش ابتدایی در نرخ مهاجرت به جزیره، این نرخ دچار افول شده تا احتمال دریافت خصوصیات توسط ذرات با برازندگی کمتر افزایش یابد.

شرایط سنی: ارزش فرآوری یک ذره (که به معنی تعداد فرزندان مورد انتظار در واحد زمان می‌باشد) یک تابع مثلثی از سن آن می‌باشد. ارزش فرآوری در سنین جوانی به دلیل عدم بلوغ پایین، در سنین بچه‌دار شدن بالا و در سنین پیری باز به دلیل از دست دادن باروری، پایین است. این موضوع در مورد گونه‌ها نیز صادق است. یک گونه‌ی جوان به اندازه‌ی ناچیزی با محیط خود منطبق است و به همین دلیل دارای شانس کمی برای متمایز شدن خواهد بود، یک گونه‌ی میانسال هم به اندازه‌ی کافی بالغ بوده و هم به اندازه‌ی کافی پویاست تا بتواند متمایز شود، و یک گونه‌ی پیر راکدتر و ایستاتر از آن است که بتوان متمایز شود. این موضوع به معرفی شرایط سن در BBO منجر می‌شود.

پویایی گونه‌ها: در نظریه‌ی کلاسیک جغرافیای زیستی فرض بر آن است که توانایی تمام گونه‌ها در مهاجرت به یک اندازه است. در واقعیت، برخی گونه‌ها از برخی دیگر پویاترند و برخی گونه‌ها راحت‌تر از سایرین متفرق می‌شوند. در جغرافیای زیستی تلاش‌هایی برای در نظر گرفتن خصوصیات مختص به هر گونه در حال انجام است [لومولینو، ۲۰۰۰b]. در حال حاضر در BBO فرض بر آن است که همه‌ی گونه‌ها به یک اندازه پویا هستند. اگر پویایی گونه‌ها با میزان مشارکتشان در برازندگی راه‌حل‌ها متناسب باشد، BBO در چارچوب خود منسجم‌تر عمل خواهد کرد. این بدین معنی است که می‌توان با استفاده از مدل‌های آماری، همبستگی میان هر راه‌حل نامزد و برازندگیش را به دست آورد. پس می‌توان پویایی را به‌عنوان خاصیت یا مجموعه‌ای از خصوصیت‌های راه‌حل که به‌صورت مثبتی با برازندگی همبسته هستند، تعریف نمود. پویایی گونه‌ها در BBO را می‌توان با تخصیص دادن مقادیر پویایی با استفاده از یک توزیع گاوسی، به آنچه که در جغرافیای زیستی طبیعی وجود دارد نزدیک نمود. این کار باعث بهبود میانگین برازندگی جمعیت خواهد شد.

رابطه‌ی مهاجم/طعمه: در زیست‌شناسی، برخی گونه‌ها با یکدیگر رابطه‌ی دشمنی دارند. این رابطه الزاماً به گونه‌های طعمه آسیب نمی‌رساند. برای مثال، گونه‌ی طعمه ممکن است با کاهش بهره‌وری از منابع‌شان به مهاجم پاسخ داده و بدین ترتیب در طولانی مدت به خود سود برسانند [هانسکی و گیلپین، ۱۹۹۷]. با این حال، محتمل‌ترین سناریو آن است که گونه‌ی مهاجم آنقدر تعداد طعمه‌ها را کاهش می‌دهد که در نهایت یک یا هر دو جمعیت با خطر انقراض مواجه می‌شوند. رابطه‌ی مهاجم/طعمه را می‌توان با امتحان نمودن ذرات و مشخص نمودن اینکه کدام جفت خصوصیات راه‌حل دارای کمترین احتمال برای هم‌زیستی می‌باشند، در یک جمعیت BBO استنباط نمود. سپس می‌توان این خصوصیات راه‌حل را به‌عنوان زوج مهاجم/طعمه مدل‌سازی نمود. ترکیب این اطلاعات با سهم برازندگی هر گونه، به تعریف راه‌حل مهاجم به‌عنوان رقیبی که با برازندگی به‌صورت مثبت و راه‌حل طعمه به‌عنوان رقیبی که با برازندگی به‌صورت منفی همبسته است،

منجر خواهد شد. رابطه‌ی مهاجم/طعمه ممکن است به یک جمعیت با تعادل غیرصفر و یا انقراض یک یا هر دو گونه منجر شود [گوتلی[1]، ۲۰۰۸]، [هانسکی و گیلپین، ۱۹۹۷]. از این اطلاعات می‌توان برای افزایش احتمال حضور ویژگی‌های مهاجم و یا کاهش احتمال حضور خصوصیات طعمه در جمعیت ذرات، استفاده نمود. بیشتر مدل‌های مهاجم/طعمه در زیست‌شناسی برای سیستم‌های دو-گونه‌ای تعریف شده‌اند. این مدل‌ها را می‌توان در BBO به کار برد، اما یک تعریف کامل‌تر را می‌توان با بسط دادن مدل‌های مهاجم/طعمه‌ی حاضر به سیستم‌های چند-گونه‌ای به دست آورد.

رقابت بر سر منابع: برخلاف رابطه‌ی مهاجم/طعمه که در بالا تعریف شد، می‌توان دید که گونه‌های مشابه بر سر منابع با یکدیگر به رقابت می‌پردازند. بنابراین، احتمال حضور گونه‌های مشابه در کنار هم و در یک جزیره، به خصوص اگر دارای جمعیت‌های بزرگی باشند، بسیار کم است [تیلمن[2] و همکاران، ۱۹۹٤]. در BBO این بدین معنی است که احتمال مهاجرت خصوصیات راه‌حل به جزیره‌هایی که قبلا دارای جمعیت‌های بزرگِ مشابه هم می‌باشند، کم است. معنی دیگراین اتفاق این است که نرخ مهاجرت از جزیره متأثر نشده، اما احتمال بقا کاهش می‌یابد. رقابت بر سر منابع در BBO همچنین بدین معنی است که اگر دو ویژگی راه‌حل دارای احتمال یکسان برای انقراض باشند، به احتمال زیاد ویژگی که بیشتر شبیه سایر ویژگی‌های موجود در راه‌حل باشد منقرض خواهد شد. این نوع رابطه نسبت به رابطه‌ی مهاجم/طعمه متفاوت است. با این حال، هر دو مدل محتمل و پذیرفتنی هستند و از نگاه زیست‌شناسی، رقابت بر سر منابع تأثیر بیشتری نسبت به رابطه‌ی مهاجم/طعمه در ترکیب جوامع دارد.

همبستگی زمانی: در جغرافیای زیستی جزیره‌ای، اگر یک گونه در یک جهت جغرافیای خاص به یک جزیره مهاجرت کند، به احتمال زیاد در آینده نیز در همان جهت به جزیره‌ی بعدی مهاجرت خواهد نمود. این بدین دلیل است که مهاجرت از بادها و جریان‌های غالب تأثیر می‌پذیرد و این بادها و جریان‌ها دارای همبستگی زمانی مثبت هستند. این موضوع در نظریه‌ی انتشار زیستی، معادلات تلگراف و معادله‌ی انتشار تعریف شده است [آکوبو[3] و لوین، ۲۰۰۱]. اگر یک گونه از جزیره‌ی A به جزیره‌ی B مهاجرت کند، به احتمال زیاد در آینده در همین جهت به مهاجرت خود ادامه خواهد داد. در BBO این بدین معنی است که اگر یک ویژگی راه‌حل از یک ذره به ذره‌ی دیگر مهاجرت نماید، به احتمال زیاد در نسل تکاملی بعدی به مهاجرت خود در همین جهت ادامه خواهد داد. مفهوم جهت در BBO را می‌توان از لحاظ مکان راه‌حل تعریف نمود. مکان در این تعریف خود به‌عنوان نقطه‌ای در فضای ویژگی راه‌حل تعریف می‌شود.

[1] Gotelli
[2] Tilman
[3] Okubo

جنبه‌های دیگر جغرافیای زیستی می‌توانند الهام‌بخش تنوعات دیگری در BBO شوند. ادبیات مربوط به جغرافیای زیستی به قدری غنی است که امکانات زیادی برای این منظور وجود دارد.

مسائل

تمارین نوشتاری

۱۴-۱ ما در نوشتن معادله‌ی (۱۴-۱) فرض کردیم که Δt به قدری کوچک است که احتمال به وقوع پیوستن یک مهاجرت بین t و $t + \Delta t$ بسیار ناچیز است. این معادله را با فرض این که بیش از دو مهاجرت بین زمان‌های t و $t + \Delta t$ به وقوع نمی‌پیوندد، بازنویسی کنید.

۱۴-۲ ما فشار انتخاب را برای الگوریتم‌های تکاملی در معادله‌ی (۸-۱۳) تعریف نمودیم. با چه تغییری در منحنی‌های مهاجرت شکل ۱۴-۲ می‌توان فشار انتخاب را افزایش داد؟

۱۴-۳ الگوریتم BBO شکل ۱۴-۳ را چگونه تغییر دهیم تا به جای نسلی، حالت ماندگار باشد؟

۱۴-٤ ما معمولاً مقادیر μ_k و λ_k را به ترتیب برابر $r_k/N+1$! و $N+1-r_k/N+1$ در نظر می‌گیریم. در این مقادیر r_k رتبه‌ی ذره‌ی kاُم در جمعیت بوده به‌طوری که بهترین ذره دارای رتبه‌ی N است و بدترین ذره دارای رتبه‌ی ۱ است. این بدین معناست که $\lambda_k \in [\frac{1}{N+1}, \frac{N}{N+1}]$. نتیجه‌ی عملی برای مطمئن شدن از اینکه برای بهترین ذره $\lambda_k > 0$ است، چیست؟

۱۴-۵ این مسئله مدل مهاجرت اولیه از شکل ۱۴-۱۰ را مورد بررسی قرار می‌دهد. فرض کنید β مقدار برازندگی نرمالیزه شده در مقدار بیشینه‌ی λ را نشان می‌دهد.

الف) یک معادله برای نرخ مهاجرت از شکل ۱۴-۱۰ پیدا کنید.

ب) مقدار تعادل تعداد گونه‌ها را برای شکل ۱۴-۱۰ بیابید.

۱۴-٦ فرض کنید از نرخ مهاجرت خطی با اندازه‌ی جمعیت N استفاده می‌کنیم تا برای بهترین ذره $\lambda_1 = 1/(N+1)$ بوده و برای بدترین ذره $\lambda_N = N/(N+1)$ باشد.

الف) احتمال اینکه در الگوریتم مهاجرت-محور جزیی از شکل ۱۴-۳، حداقل یک ویژگی راه‌حل به بهترین ذره x_b مهاجرت کند چه قدر است؟

ب) احتمال اینکه در الگوریتم مهاجرت‌دهنده-محور جزیی از شکل ۱۴-۳، حداقل یک ویژگی راه‌حل به بهترین ذره x_b مهاجرت کند چه قدر است؟

۱۴-۷ در ضمیمه‌ی ج.۱ مثال‌هایی برای توابع تفکیک‌پذیر و غیرتفکیک‌پذیر بیابید.

تمارین کامپیوتری

۱۴–۸ مثال ۱۴–۱ را با $n = 5$ تکرار نمایید.

۱۴–۹ الگوریتم استاندارد BBO از شکل ۱۴–۳ بیان می‌دارد که برای هر ذره x_k، نرخ مهاجرت از جزیره μ_k باید با مقدار برازندگی x_k متناسب بوده و $\mu_k \in [0,1]$ باشد. ما از احتمالات مهاجرت برای انتخاب ذره‌ی مهاجرت دهنده استفاده کرده و می‌توانیم برای این کار از گزینه‌های زیر استفاده نماییم.

- انتخاب رتبه–محور مانند آنچه که در بخش ۸–۷–۴ مورد بحث قرار گرفت. این گزینه، مانند آنچه که در مثال ۱۴–۲ نشان داده شد، گزینه‌ی BBO استاندارد است.
- رتبه‌بندی مربعی، که این گزینه نیز در بخش ۸–۷–۴ مورد بحث قرار گرفت.
- انتخاب مسابقه‌ای، که در بخش ۸–۷–۴ مورد بحث قرار گرفت.
- انتخاب stud، که در بخش ۸–۷–۷ مورد بحث قرار گرفت.

الگوریتم BBO را برای مینیمم‌سازی تابع ۱۰ بعدی آکلی با $N = 50$، محدودیت نسل ۵۰، نرخ جهش ۱٪ و پارامتر نخبه‌گرایی ۲ پیاده‌سازی نمایید و آن را برای ۲۰ شبیه‌سازی مونت کارلو اجرا نمایید. در هر نسل از شبیه‌سازی مونت کارلو کمترین هزینه را ثبت نمایید. کمترین هزینه را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است به‌عنوان تابعی از شماره‌ی نسل رسم نمایید. این کار را برای هر یک از گزینه‌های فرایند انتخاب که در بالا ذکر شده‌اند انجام دهید و نمودارهای به دست آمده برای هر گزینه را با یکدیگر مقایسه نمایید. نتایج را توضیح دهید.

۱۴–۱۰ جواب‌هایی را که برای مسئله‌ی ۱۴–۶ به دست آوردید به‌عنوان تابعی از اندازه‌ی جمعیت N برای $N \in [10,50]$ رسم کنید و ابعاد مسئله را برابر ۱۰ در نظر بگیرید. نتایج را توضیح دهید.

۱۴–۱۱ یک راه برای بهبود عملکرد BBO انتخاب نسل بعد از میان ذرات قدیمی و جدید است [دیو[1] و همکاران، ۲۰۰۹]. این بدین معناست که عبارت $\{x_k\} \leftarrow \{z_k\}$ در انتهای شکل ۱۴–۳ را می‌توان با عبارتی مانند عبارت زیر جایگزین نمود:

$$\{x_k\} \leftarrow \{x_k\} \cup \{z_k\}$$ بهترین N ذره از

این موضوع از استراتژی تکاملی نشات گرفته و به همین دلیل می‌توان آن را BBO-ES نامید. الگوریتم BBO را برای مینیمم‌سازی تابع ۱۰ بعدی آکلی با $N = 50$، محدودیت نسل ۵۰، نرخ جهش ۱٪ و پارامتر نخبه‌گرایی ۲ پیاده‌سازی نمایید و آن را برای ۲۰ شبیه‌سازی مونت کارلو اجرا نمایید. در هر نسل از شبیه‌سازی مونت کارلو کمترین هزینه را ثبت نمایید. کمترین هزینه را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است به‌عنوان تابعی از شماره‌ی نسل رسم نمایید. نتایج به دست آمده برای BBO استاندارد را با نتایج به دست آمده برای BBO-ES مقایسه کرده و آن را توضیح دهید.

[1] Du

فصل پانزدهم

الگوریتم‌های فرهنگی

فرهنگ، ادراک را بهینه‌سازی می‌کند.

جیمز کندی [کندی، ۱۹۹۸]

نکته‌ی جمله‌ی بالا در آن است که ادراک (که فرایند فکر کردن است)، چیزی بیش از فعالیت مغزی و رفتار عصبی است. نحوه‌ی تفکر ما از فرهنگ ما تأثیر می‌پذیرد. به علاوه، این تأثیر بسیار سودمندانه است (البته بنا بر گفته‌ی بالا نه تنها سودمندانه، بلکه بهینه است). بدون فرهنگ، مهارت‌های ادراکی ما دچار نقصان می‌بود.

یک مثال از این موضوع، کودکان یاغی[1] می‌باشند[2] [نیوتون، ۲۰۰٤]. برخی از این کودکان در محیطی وحشی بزرگ شده و برخی دیگر در انزوای ناشی از مراقبین ظالم و خشن بزرگ می‌شوند. کودکان این چنینی هیچگاه جامعه را درک نکرده، هیچگاه صحبت کردن را نخواهند آموخت، هیچگاه نخواهند توانست با دیگران ارتباط برقرار کنند و هیچگاه نخواهند توانست رفتاری مقبول به لحاظ اجتماعی از خود بروز دهند. عدم توانایی آن‌ها در نشان دادن رفتار مناسب در محیط متمدن یک امر ژنتیکی نیست و به کمبود هوش ذاتی ربطی ندارد بلکه نبود تأثیر فرهنگی و اجتماعی در دوران تربیت آن‌هاست که هوش آن‌ها را دچار نقصان می‌کند. موضوع کودکان یاغی، بحث جدی برای پروش در جدال میان *پرورش و طبیعت* به راه می‌اندازد.

پیش از این دانشمندان باور داشتند که فرهنگ انسانی از یک سطح بالا آغاز گردید و سپس به مرور دچار انحطاط گردید. سپس این انحطاط باعث پخش شدن فرهنگ غیرمتمدنانه در سراسر جهان شد. این عقیده بر پایه‌ی باورهای مذهبی در مورد پیدایش و داستان‌های کتاب مقدس در مورد برج بابل در کتاب پیدایش می‌باشد. با این حال، این دیدگاه از انحطاط فرهنگی، مفاهیمی قابل آزمایش دارد. برای نمونه، این نوع انحطاط باید شواهد باستانی از خود به جای بگذارد، به‌طوری که با کندوکاو بیشتر و عمیق‌تر در مکان‌های باستانی (و رفتن به زمان‌های گذشته‌تر)، میزان پیچیدگی فرهنگی باید بیشتر شود. با این که داستان‌های خاصه‌ی مذهبی را نمی‌توان به‌صورت علمی آزمود، آزمایش انحطاط به‌عنوان یک اصل کلی قابل انجام است.

یک انسان‌شناس قرن ۱۹ به نام ادوارد تیلور[3] نشان داده است که فرهنگ‌های پیچیده از فرهنگ‌های ابتدایی نشئت گرفته‌اند (چیزی که دقیقاً عکس فرایض ارائه شده در بالا می‌باشند). [تیلور، ۲۰۱۱]. وی نشان داد که فرهنگ نیز مانند ارگانیسم‌های زنده از شکل ساده به شکل پیچیده رشد می‌کند. تیلور اولین کسی است که از واژه‌ی فرهنگ در شکل و شمایل اجتماعی مدرن استفاده کرد و آن را بدین صورت تعریف نمود: "آن کلیت

---

[1] Feral children

[2] کودکان یاغی کودکانی هستند که به دور از جوامع انسانی و با کمترین میزان ارتباط با انسان‌ها هستند و به همین دلیل از زبان انسانی، مراقبت‌های انسانی و رفتار انسانی ناآشنا هستند. م.

[3] Edward Taylor

پیچیده که شامل دانایی، عقیده، هنر، اخلاق، قانون، پوشش و هر گونه توانایی و عادت دیگری که انسان به‌عنوان عضوی از جامعه کسب نموده است، می‌شود" [تیلور، ۲۰۰۹].

فرهنگ یک جامعه نهادی پیچیده است که با محیط، افراد و دیگر فرهنگ‌ها تعامل می‌کند. افراد می‌توانند به‌صورت مستقل عمل نمایند اما همچنین با یکدیگر هم به‌صورت مستقیم و هم به‌صورت غیرمستقیم به تعامل می‌پردازند. ذرات از طریق فرهنگ به‌صورت مستقیم و غیرمستقیم بر روی یکدیگر اثر می‌گذارند. اکثر افراد از طرف فرهنگی که در آن زندگی می‌کنند دچار محدودیت‌هایی هستند. برخی افراد بر خلاف جهت جامعه حرکت می‌کنند اما اکثر آن‌ها با آن همرنگ می‌شوند.

مروری برفصل

این فصل به بحث در مورد برخی روش‌ها برای مدل کردن فرهنگ در الگوریتم‌های تکاملی برای بهبود عملکرد آن‌ها می‌پردازد. بخش ۱۵-۱ یک بخش اولیه است که استراتژی‌هایی بهینه برای روابط انسان‌ها در سطحی بالا را معرفی می‌نماید و زمینه‌ای برای باقی فصل فراهم می‌آورد. بخش ۱۵-۲ به بحث در مورد مدلی خاص از فرهنگ به نام فضای عقیده[1] پرداخته و تأثیرات آن بر تکامل راه‌حل‌های نامزد در الگوریتم‌های تکاملی را بررسی می‌نماید. بخش ۱۵-۳ از فضای عقیده برای ایجاد یک برنامه‌ی تکاملی (EP[2]) فرهنگی استفاده می‌نماید و نشان می‌دهد که این نوع EP عملکرد بهتری نسبت به یک EP عادی دارد. بخش ۱۵-٤ نگاهی متفاوت و رابطه محور نسبت به مقوله‌ی فرهنگ را ارائه کرده و به بحث در مورد مدل فرهنگی انطباقی[3] (ACM) می‌پردازد. این بخش همچنین نشان می‌دهد که ACM می‌تواند مسئله‌ی فروشنده‌ی دوره‌گرد را حل نماید.

۱۵-۱ همکاری و رقابت

این بخش فرهنگ را از دید یک رابطه‌ی بین فردی مورد بررسی قرار می‌دهد. جامعه‌ی مدرن شامل بسیاری ارتباطات بین فردی بوده و این ارتباطات به سرعت در حال افزایش‌اند. جامعه همچنین شامل همکاری‌ها و رقابت‌های بسیاری است. گاهی ما با هدف همکاری با دیگران ارتباط برقرار می‌کنیم و گاهی نیز با هدف رقابت این کار را انجام می‌دهیم. هنگامی که یک مقاله‌ی فنی یا تحقیقی می‌نویسیم، در حال ایجاد ارتباط برای نشان دادن اشکال ایده‌های رقیب و نشان دادن فواید ایده‌ی خود هستیم. گاهی ما کمی بزرگ‌نمایی می‌نماییم تا خود را بهتر و یا دیگری را بدتر نشان دهیم. گاهی این بزرگ‌نمایی‌ها عمدی هستند و گاهی

[1] Belief Space
[2] Evolutionary Program
[3] Adaptive Culture Model

سهوی و گاهی نیز حتی تشخیص دادن اهداف خودمان برایمان سخت است. در ارتباطات خود با دیگران ما گاهی حقیقت را می‌گوییم چرا که امید داریم طرف مقابل نیز به ما حقیقت را بگوید اما بیشتر مواقع به دلیل عواقب و یا فوایدی که عایدمان می‌شود دروغ می‌گوییم. جواب‌های معمولی را که به سؤال‌هایی همچون سؤال‌ها زیر داده می‌شوند را در نظر بگیرید:

۱. حالتان چطور است؟
۲. آیا غذایی را که برایتان آماده کردم پسندیدید؟
۳. معمولاً چه قدر دروغ می‌گویید؟

ما استدلال می‌کنیم که کسی که این سؤال‌ها را می‌پرسد نمی‌خواهد جواب سؤال را بداند. کسی که این نوع سؤال‌ها را می‌پرسد تنها به دنبال ایجاد یک مکالمه و یا به دنبال جوابی خاص است، پس به همین دلیل ما جوابی را که طرف در انتظار شنیدنش است، به وی می‌دهیم، هر چند که ممکن است جواب ما در کنه یک دروغ باشد. جالب آنجاست که همه فکر می‌کنند کمتر از دیگران دروغ می‌گویند [دیپاولو[1] و همکاران، ۱۹۹۶]. به علاوه، همه فکر می‌کنند توجیه بهتری نسبت به بقیه برای دروغشان دارند.

ما می‌توانیم ارتباطات بین فردی را برای مطالعه‌ی استراتژی‌های ارتباط و یا برای یافتن راه‌حل‌هایی برای مسائل بهینه‌سازی شبیه‌سازی نماییم. مسئله‌ی معمای زندانی (بخش ۵-٤ را ببینید) نمونه‌ای از ارتباط میان نماینده‌ها است. این مسئله یک مثال بسیار جالب است چرا که انواع زیادی دارد و در آن بهترین استراتژی به استراتژی بازیکن مقابل بستگی دارد و بهترین استراتژی همیشه کاملاً مشخص نیست.

ال‌فارول[2]

یک مثال جالب دیگر از ارتباطات بین فردی مسئله‌ی ال‌فارول است [کندی و ابرهارت، ۲۰۰۱، فصل ۵]. این مسئله در مورد مردی است به نام برایان آرتور[3] که می‌خواهد به یک کافه با اسم ال‌فارول در محله‌ی پایین شهر سانتافه برود. وی مخصوصاً دوست دارد در روزهای پنجشنبه که ال‌فارول موسیقی ایرلندی پخش می‌کند به این کافه برود. با این حال، اگر کافه شلوغ باشد وی ترجیح می‌دهد در خانه بماند. به بیان دیگر، وی ترجیح می‌دهد تنها در صورتی به ال‌فارول برود که کمتر از ٦۰ نفر در آنجا باشند و در صورتی که ٦۰ یا بیشتر از ٦۰ نفر آنجا باشند وی ترجیح می‌دهد در خانه بماند. شرایط دوستان برایان نیز به همین ترتیب

[1] Dipaulo
[2] El Farol
[3] Brian Arthur

است. آن‌ها نیز دوست دارند تنها در صورتی به الفارول بروند که کمتر از ٦٠ نفر در آنجا باشند و در غیر این صورت ترجیح آن‌ها بر آن است که در خانه بمانند.

برایان و دوستانش می‌دانند که در ١٤ هفته‌ی گذشته، تعداد افراد در الفارول به شرح زیر بوده است

$$44, 78, 56, 15, 23, 67, 84, 34, 45, 76, 40, 56, 22, 35 \qquad (۱-۱۵)$$

آیا او می‌تواند این پنجشنبه به الفارول برود؟ به بیان دیگر، با توجه به اطلاعات ١٤ هفته‌ی گذشته، آیا این هفته کمتر از ٦٠ نفر در الفارول خواهند بود؟ برایان می‌تواند از روش‌های متنوع الگوشناختی و آزمایش‌های رگرسیون برای پیش‌بینی تعداد افراد در الفارول برای پنجشنبه‌ی هفته‌ی جاری استفاده کند. اگر وی بتواند یک پیش‌بینی‌کننده‌ی خوب پیدا کند مشکل وی حل خواهد شد. با این حال، اگر وی در مورد پیش‌بینی‌کننده‌اش به همه‌ی دوستانش بگوید، این پیش‌بینی‌کننده دیگر کار نخواهد کرد. اگر همه‌ی دوستان وی بفهمند که پیش‌بینی‌کننده‌ی وی تعداد کمتر از ٦٠ نفر را برای الفارول پیش‌بینی کرده است، همگی به الفارول رفته و به همین دلیل بیشتر از ٦٠ نفر در الفارول خواهند بود. اگر پیش‌بینی‌کننده تعداد بیشتر از ٦٠ نفر را پیش‌بینی کند، آنگاه هیچ یک از دوستان وی به الفارول نرفته و بدین ترتیب تعداد کمتر از ٦٠ نفر در الفارول خواهند بود. این یک تناقض از یک الگوریتم خوب پیش‌بینی در هنگامی است که عنصر انسانی در نظر گرفته شده است. بدین ترتیب یک پیش‌بینی‌کننده‌ی خوب به یک پیش‌بینی‌کننده‌ی ضعیف تبدیل می‌شود.

حال فرض کنید برایان و دوستانش در مورد رفتن یا نرفتن به الفارول در روز پنجشنبه با یکدیگر صحبت کنند. اگر برایان به دوستانش بگوید که این پنجشنبه به الفارول خواهد رفت، آنگاه دوستان برایان به احتمال زیاد در خانه مانده و بدین ترتیب برایان با جمعیت کمی در الفارول مواجه خواهد شد که این مطلوب است. حال اگر برایان تصمیم بگیرد در خانه بماند و این تصمیم را به دوستانش بگوید، همه‌ی دوستان وی به الفارول رفته و در آنجا با جمعیتی زیاد مواجه خواهند شد.

با این حال، ممکن است برایان و دوستانش کاملاً با هم روراست و صادق نباشند. آن‌ها ممکن است به یکدیگر بگویند که به الفارول خواهند رفت با این امید که دیگران در خانه خواهند ماند. بعد از آنکه برایان به دوستانش بگوید به کافه خواهد رفت، ممکن است وی تصمیم بگیرد در خانه بماند چرا که فهمیده است بقیه نیز می‌خواهند به کافه بروند. به علاوه، برایان ممکن است به دوستانش بگوید او و ١٠ تن از دوستانش می‌خواهند به کافه بروند. این کار یک بزرگ‌نمایی است و برایان از این طریق سعی دارد دوستانش را تشویق به ماندن در خانه کند. صدالبته که ممکن است دوستان وی نیز همین نوع استراتژی را در پیش بگیرند و برای منافع خود به دیگران دروغ بگویند.

بهترین و بهینه‌ترین استراتژی ارتباط برای برایان چیست؟ آیا او باید همواره راست بگوید؟ اگر او همواره دروغ بگوید، بالاخره دوستان برایان متوجه این استراتژی وی (دروغ گفتن) شده و پس از آن حرف‌های وی را نادیده خواهند گرفت. با این حال، اگر هدف غایی او رفتن به الفارول در شب‌های خلوت و نرفتن به این کافه در شب‌های شلوغ باشد، به نظر می‌رسد که برایان و دوستانش مجبورند گاه‌گاه به یکدیگر دروغ بگویند.

مسئله‌ی الفارول جالب است چرا که شامل حقیقت، دروغ، اعتماد، ارتباط و احتمالاً هدف‌های متعارض می‌شود. اگر هدف برایان دوست داشته شدن توسط دوستانش باشد، احتمالاً همیشه حقیقت را خواهد گفت. اما اگر هدفش رفتن به الفارول در شب‌های خلوت و نرفتن به این کافه در شب‌های شلوغ باشد، وی مجبور است گاه‌گاهی دروغ بگوید.

مثال‌های دیگر

یک دلیل دیگر برای جالب بودن مسئله‌ی الفارول آن است که، همان‌طور که پیش از این نیز گفته شد، اگر همه از یک پیش‌بینی‌کننده‌ی خوب استفاده نمایند، آن پیش‌بینی‌کننده تبدیل به یک پیش‌بینی‌کننده‌ی ضعیف خواهد شد. این خاصیت در بسیاری از موقعیت‌های دنیای واقعی نیز خود را نشان می‌دهد. برای مثال، رابطه‌ی عاشقانه‌ی میان یک پسر و دختر را در نظر بگیرید. اگر پسر علاقه‌ی زیادی از حد به دختر نشان دهد باعث خواهد شد وی درمانده به نظر خواهد آمد و همین موضوع باعث خواهد شد که از جذابیت وی در نظر دختر کاسته شود. با این حال، اگر پسر به اندازه‌ی کافی علاقه نشان ندهد ممکن است علاقه‌مند به نظر نیاید و این موضوع برای رابطه با دختر رویاهایش مساعد نخواهد بود. همچنین، دختر باید چگونه باید این عدم داشتن علاقه‌ی ظاهری پسر را برداشت کند؟ آیا باید این عدم علاقه صادقانه در نظر گرفته شود و در این صورت دختر توجه خود را به سایر پسرها معطوف کند و یا اینکه این عدم علاقه‌ی ظاهری باید به‌عنوان یک حس عاشقانه‌ی سرکوب شده برداشت شود و به آن پاسخ داده شود؟ رابطه‌ی احساسی دوطرفه یک داد و ستد پیچیده بین دو فرد است؛ دو فردی که در راستای اهداف خود و همچنین در راستای فرهنگ خود رفتار می‌نمایند.[1]

مثال‌های دیگر ریشه در اقتصاد و برنامه‌های تحقیقاتی ما دارد. در پروپوزال‌های تحقیقاتی خود باید بر روی چه زمینه‌هایی تمرکز نماییم؟ آیا باید پروپوزال‌ها را به سمت زمینه‌هایی که بودجه‌ی بیشتری به آن‌ها تعلق می‌گیرد سوق دهیم؟ مقدار بودجه‌ی بالا شانس ما را افزایش می‌دهد اما بسیاری دیگر نیز در حال نوشتن

[1] در اینجا مثالی از بازی بیسبال توسط مؤلف آورده شده است که ما در نسخه‌ی ترجمه شده از ذکر این مثال پرهیز کرده‌ایم چرا که بازی بیسبال بازی چندان آشنایی برای خوانندگان ایرانی نیست. م.

پروپوزال در این زمینه‌ها هستند و این موضوع می‌تواند شانس ما را کاهش دهد. شاید بهتر باشد در زمینه‌هایی پروپوزال بنویسیم که بودجه‌ی کمتری دارند و به همین دلیل رقابت کمتری وجود دارد. اگر پروپوزال ما تنها پروپوزال در یک زمینه‌ی مشخص باشد، آنگاه احتمال تعلق بودجه به ما بسیار بالا خواهد بود. اما اگر رقیبان ما نیز از استراتژی مشابهی استفاده نمایند، آنگاه این استراتژی بی‌فایده خواهد بود. تصمیم‌گیری در مورد ناحیه‌ی تمرکز پروپوزال کاری است پیچیده و چندبعدی اما استراتژی بهینه احتمالاً این است که تلاش خود را هم به زمینه‌های پر ریسک (بودجه‌های زیاد) و هم به زمینه‌های کم ریسک (بودجه‌های کم) اختصاص دهیم [سایمون، ۲۰۰۵]. یک چنین استراتژی مخلوطی را می‌توان در زمینه‌ی سرمایه‌گذاری، بازاریابی و دیگر کاربردها نیز به کار برد.

مسائلی که شامل روابط بین فردی، ارتباطات، تعاون، فریب و اهداف چندگانه می‌شوند، اشتراکات زیادی با فرهنگ انسانی دارند. انسان‌ها راه‌هایی تقریباً بهینه برای ارتباط‌گیری، ساختن جامعه و گسترش فرهنگ آموخته‌اند. ما از تمام ویژگی‌های بهینه‌سازی فرهنگ انسانی اطلاع نداریم، اما در نظر گرفتن چنین ویژگی‌هایی قطعا مطالعه‌ای جالب و مفید خواهد بود. تقلید و شبیه‌سازی رفتار بهینه‌ساز فرهنگ انسانی یک زمینه‌ی جالب و مفید دیگر است و به همین علت ما در باقی این فصل توجه خود را به این موضوع معطوف می‌نماییم.

۱۵–۲ فضای عقیده در الگوریتم‌های فرهنگی

یک الگوریتم فرهنگی (CA) نیز مانند سایر الگوریتم‌های تکاملی، راه‌حل‌های نامزد برای حل یک مسئله‌ی بهینه‌سازی را مانند ذرات در نظر می‌گیرد. با این حال، یک CA قواعدی که تکامل ذرات را به پیش می‌برد، را مانند فرهنگ آن ذرات در نظر می‌گیرد. الگوریتم CA تأثیرات مابین ذرات و فرهنگشان را برای دستیابی به یک الگوریتم بهینه‌سازی مدل می‌کند. هنجارهای فرهنگی جمعیت مجازی یک CA را فضای عقیده می‌نامند. مانند همه‌ی الگوریتم‌های تکاملی که تا به حال در این کتاب در مورد آن‌ها بحث نمودیم، در هر نسل از یک CA ذرات دچار جهش شده و عمل بازترکیب انجام می‌دهند. اما در CA جهش و بازترکیب توسط فضای عقیده تحت تأثیر واقع می‌شوند. فضای عقیده توسط برنامه‌نویس طراحی شده و می‌توان برای اعمال محدودیت، نگه داشتن ویژگی‌های مطلوب و یا پرهیز از ویژگی‌های نامطلوب از آن استفاده نمود.

مثال ۱۵–۱

در این مثال ما به‌طور کلی به چگونگی پیاده‌سازی فضای عقیده در یک الگوریتم تکاملی می‌پردازیم. مسئله‌ی مورچه‌ی مصنوعی از بخش ۵–۵ را به یاد آورید. یک مورچه‌ی مصنوعی در یک شبکه که برخی از سلول‌های آن خالی بوده و برخی سلول‌های آن با غذا پر شده است قرار داده می‌شود. تنها توانایی حسی

مورچه آن است که وجود یا عدم وجود غذا در سلول روبه‌رویی خود را حس کند. در هر سلول مورچه می‌تواند یکی از سه حرکت ممکن را انجام دهد: یا می‌تواند به سمت جلو برود و اگر غذایی در آن سلول بود را بخورد، یا می‌تواند در سلول خود باقی مانده و به راست بچرخد و یا می‌تواند در سلول خود باقی مانده و به سمت چپ بچرخد. ما می‌خواهیم یک ماشین حالت متناهی (FSM) را رشد داده تا از این طریق به مورچه برای مسیریابی در میان شبکه و خوردن بیشترین غذای ممکن کمک نماییم. می‌توان به‌صورت بصری و مستقیم دریافت که هنگامی که مورچه در خانه‌ی روبه‌روی خود غذا حس می‌کند، احتمالاً بهتر است به خانه‌ی جلویی رفته و غذا را بخورد. با این حال، شاید ما نخواهیم این قاعده را به قاعده‌ای سفت و سخت تبدیل نماییم. می‌دانیم که گاه تکامل برای یافتن مطلوب‌ترین و بهترین نتیجه باید راه‌حل‌های زیربهینه را بکاود. بنابراین، در این مسئله می‌خواهیم هر FSM در جمعیت الگوریتم تکاملی را در صورت حسِ بودن غذا در سلول روبه‌رویی به حرکت رو به جلو تشویق نماییم اما نمی‌خواهیم این رفتار را به یک الزام سخت تبدیل نماییم. این نوع رفتار، رفتاری است که می‌توان آن را در فضای عقیده برای تشویق و نه الزام یک ویژگی در جمعیت یک الگوریتم تکاملی، کدگذاری نمود. میزان این تشویق برای حرکت رو به جلوی ذرات هنگام حس وجود غذا در خانه‌ی روبه‌رویی، را می‌توان نشانگر قدرت فرهنگ محسوب نمود. در جوامع انسانی، برخی فرهنگ‌ها از برخی دیگر قویتر بوده و فشار بیشتری برای پیروی وارد می‌آورد. اگر این میزان تشویق متعادل باشد، بدین معنی است که به چندین ذره اجازه داده می‌شود که حتی در صورت حس غذا در خانه‌ی روبه‌رویی، از گزینه‌های حرکتی دیگری به غیر از حرکت رو به جلو استفاده نمایند. اگر میزان تشویق قوی باشد، بدین معنی است که به تعداد بسیار کمی از ذرات اجازه‌ی کاوش سایر روش‌ها را می‌دهیم. با رشد جمعیت، ممکن است بخواهیم سطح تشویق را با توجه به برازندگی ذراتی که از فرهنگ غالب پیروی نموده‌اند نسبت به ذراتی که سرپیچی نموده‌اند، اصلاح نماییم.

مثال ۱۵-۱ نشان می‌دهد که CAها می‌توانند دو موضوع وراثتی را پیاده‌سازی نمایند: ویژگی‌های راه‌حل از والدین به فرزندان به ارث می‌رسد و فضای عقیده از یک نسل به نسل بعد به ارث می‌رسد. تکامل همچنان در سطح ذرات به وقوع می‌پیوندد اما خود فرایند تکامل تحت تأثیر فضای عقیده قرار دارد.

فضای عقیده‌ی یک CA بسته به اینکه چگونه پیاده‌سازی شود می‌تواند ایستا و یا پویا باشد. اگر فضای عقیده ایستا باشد، با زمان تغییر نخواهد کرد. اگر فضای عقیده پویا باشد با زمان تغییر خواهد کرد، بدین معنی که فرهنگ خود می‌تواند تکامل یابد. یک CA با فضای عقیده‌ی پویا نه تنها جمعیتی از ذرات را از یک نسل به نسل دیگر رشد می‌دهد، بلکه یک فضای عقیده را نیز از یک نسل به نسل دیگر رشد می‌دهد. به بیان دیگر، در یک CA با فضای عقیده‌ی پویا نه تنها فضای عقیده بر روی تکامل جمعیت تأثیر داشته، بلکه جمعیت نیز

بر روی تکامل فضای عقیده تأثیرگذار است. فضاهای عقیده‌ی پویا از آنچه که در جوامع انسانی قابل مشاهده است الهام گرفته شده است. می‌توان دید که فرهنگ بسیار سریعتر از زیست‌شناسی در حال رشد است. به همین دلیل است که انتظار می‌رود بتوان در الگوریتم‌های تکاملی با فضای عقیده‌ی پویا، راه‌حل‌های بهینه را سریعتر از الگوریتم‌های تکاملی بدون فضای عقیده‌ی پویا یافت.

همان‌طور که نظریات بسیاری در مورد فرهنگ انسانی وجود دارد [ولش[1] و اندیکَت[2]، ۲۰۰۵]، انواع CAهای بسیاری نیز وجود دارد. شکل ۱۵–۱ طرح کلی یک CA را نشان می‌دهد. همانند بسیاری الگوریتم‌های تکاملی دیگر، ابتدا یک جمعیت ابتدایی از راه‌حل‌های نامزد مقداردهی اولیه شده، اما در CA علاوه بر جمعیت آغازین، یک فضای عقیده‌ی آغازین B نیز ایجاد می‌شود. فضای عقیده بر روی تکامل جمعیت تأثیر گذاشته و می‌توان گفت که فرایند تکاملی را راهنمایی می‌کند. الگوریتم شکل ۱۵–۱ مانند هر الگوریتم تکاملی دیگر با محاسبه‌ی هزینه‌ی هر ذره به پیش می‌رود. اما سپس از ذرات برای اصلاح B استفاده می‌کند. راه‌های زیادی برای پیاده‌سازی چگونگی اصلاح B وجود دارد. برای مثال، اگر جمعیت حاکی از آن باشد که ذرات خوب دارای یک ویژگی خاص هستند، آنگاه می‌توان B را به‌گونه‌ای اصلاح نمود که راه‌حل‌های نامزد آتی را به سمت آن ویژگی سوق دهد. صد البته که ذرات آتی در هر صورت به دلیل این حقیقت که ذرات برازنده‌تر از احتمال بیشتری برای بازترکیب برخوردارند، به سمت آن ویژگی سوق داده می‌شوند. اما B می‌تواند ذرات آتی را با روش‌هایی پیچیده‌تر از بازترکیب ساده به آن سمت سوق دهد. برای مثال، می‌توانیم ترکیب خاصی از ویژگی‌ها یا رفتارها را در B جای داد (ایده‌ی مثال ۱۵–۱ را به یاد آورید).

```
جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن {x_i}، i ∈ [1, N]
فضای عقیده B را مقداردهی اولیه کن
تا زمانی که شرایط توقف برآورده نشده است
    هزینه‌ی f(x_i) را برای هر ذره‌ی موجود در جمعیت حساب کن، i ∈ [1, N]
    از جمعیت {x_i} جهت به‌روزرسانی B استفاده کن
    B را در بازترکیب و جهش جمعیت {x_i} دخیل کن
نسل بعد
```

شکل ۱۵–۱ طرح کلی یک الگوریتم فرهنگی پایه. بر پایه‌ی [رینولدز[3]، ۱۹۹٤] و [انگلبرکت، ۲۰۰۳، فصل ۱٤].

[1] Welsch
[2] Endicott
[3] Reynolds

پس از آنکه B در الگوریتم شکل ۱۵-۱ به‌روزرسانی شد، عمل بازترکیب و جهش برای جمعیت $\{x_i\}$ انجام خواهد گرفت. این مرحله را می‌توان در مورد تمامی الگوریتم‌های تکاملی که در این کتاب در مورد آن‌ها بحث نموده‌ایم، پیاده‌سازی نمود. بنابراین، یک CA را نباید به‌عنوان یک الگوریتم تکاملی جدا دانست بلکه بهتر است آن را راهی برای تقویت و تکمیل سایر الگوریتم‌های تکاملی به حساب آورد. ویژگی شاخص CAها آن است که در آن‌ها بازترکیب و جهش تحت تأثیر فضای عقیده B بوده و به همین علت ذرات نسل بعد در راستای تطابق با B خواهند بود.

جزییات زیادی در مورد الگوریتم شکل ۱۵-۱ وجود دارد. برای مثال:

- چه نوع اطلاعاتی در فضای عقیده B کد می‌شود؟
- B چگونه به‌روزرسانی می‌شود؟
- از چه نوع بازترکیب و جهشی باید استفاده نمود؟ به بیان دیگر، از کدام الگوریتم تکاملی باید به‌عنوان پایه‌ی CA استفاده نمود؟
- چگونه از B برای تأثیر بر روی جهش و بازترکیب استفاده نماییم؟

این سؤال‌ها فرصت‌های بسیاری را برای محققین فراهم می‌آورد تا جواب‌هایی مؤثر برای مسئله‌ای خاص و یا برای کلاس خاصی از مسائل بیابند. برای نمونه، اولین سؤال از لیست بالا را در نظر بگیرید. با در نظر گرفتن این که فضای عقیده جنبه‌های زیر از یک مسئله‌ی بهینه‌سازی را نشان می‌دهد، می‌توان به جزیی از این سؤال پاسخ داد.

- فضای عقیده قیدهای موجود برای راه‌حل یک مسئله‌ی بهینه‌سازی را نمایندگی می‌کند. این قیدها می‌توانند قیدهای سخت و یا قیدهای نرم باشند [کوئلو کوئلو[1] و بسرا[2]، ۲۰۰۲]، [بسرا و کوئلو کوئلو، ۲۰۰٤].
- عقیده می‌تواند نماینده‌ی دانش مربوط به زمینه‌ی خاص بهینه‌سازی برای جهت دادن تحقیق در جهت مطلوب بر پایه‌ی تخصص انسانی باشد [اسوردلیک[3] و رینولدز، ۱۹۹۳]، [اعلمی[4] و ال ایمرانی[5]، ۲۰۰۸].
- فضای عقیده می‌تواند شامل اهمیت تنوع برای کمک به حفظ آن در تحقیق باشد.

[1] Coello Coello
[2] Becerra
[3] Sverdlik
[4] Alami
[5] El Imrani

- فضای عقیده می‌تواند شامل اهمیت همکاری و تعاون برای بهبود عملکرد سیستم‌های هم-تکاملی باشد. هم-تکامل شامل ایجاد و گسترش سیستم‌های تکاملی متمایز اما متعامل در یک محیط مشترک می‌شود [دورام[1]، ۱۹۹۲]. ما در این کتاب هم-تکامل را مورد بحث قرار نمی‌دهیم اما متذکر می‌شویم که هنگامی که ارزیابی برازندگی متغیر با زمان باشد و یا ارزیابی برازندگی جمعیت به جمعیت دیگری بستگی داشته باشد به‌طوری که جمعیت دوم متغیر با زمان باشد، می‌توان از هم-تکامل برای پیدا نمودن راه‌حل‌های بهینه استفاده نمود (بخش ۲۱-۲ را ببینید) [یانگ و همکاران، ۲۰۰۸].
- فضای عقیده می‌تواند شامل اهمیت خلاقیت باشد و بدین ترتیب جستجوی الگوریتم تکاملی را به سمت راه‌حل‌های نامزد بدیع و یا جستجوی فضاهای بکر فضای جستجو سوق دهد. این ایده‌ها در یادگیری مقابله-محور (فصل ۱۶ را ببینید) و جستجو برای بدعت [لهمن[2] و استنلی[3]، ۲۰۱۱] به کار گرفته شده‌اند اما هنوز در فضای عقیده‌ی CA ترکیب نشده است.

۱۵-۳ برنامه‌نویسی تکاملی فرهنگی

این بخش نشان می‌دهد که چگونه یک فضای عقیده‌ی ساده عملکرد یک برنامه‌ی تکاملی (EP) را بهبود می‌بخشد. الگوریتم پایه‌ی EP در شکل ۵-۱ نشان داده شده است. در این بخش ما یک فضای عقیده را در EP پیاده‌سازی می‌نماییم. فضای عقیده مکان راه‌حل‌های نامزد با بهترین عملکرد را در فضای جستجو را نشان می‌دهد. EP تأثیر گرفته از CA (CAEP)، که در این بخش ارائه شده است، مشابه CAEP بحث شده در [انگلبرکت، ۲۰۰۳، فصل ۱٤] می‌باشد. فضای عقیده B با $2n$ پارامتر کد شده است به‌طوری که n ابعاد مسئله‌ی بهینه‌سازی است. بازه‌ی $[B_{min}(k), B_{max}(k)]$ بازه‌ای را نشان می‌دهد که از نظر فضای عقیده kاُمین بعد راه‌حل‌های خوب در آن واقع می‌شوند. فضای عقیده فرایند جهش در EP را تحت تأثیر قرار می‌دهد. اگر در جهش EP که در شکل ۵-۱ نشان داده شده است $\beta = 0$ باشد، آنگاه برای $i \in [1, N]$ و $k \in [1, n]$ خواهیم داشت

$$x_i'(k) \leftarrow x_i(k) + r_i(k)\sqrt{\gamma} \qquad (۲-۱۵)$$

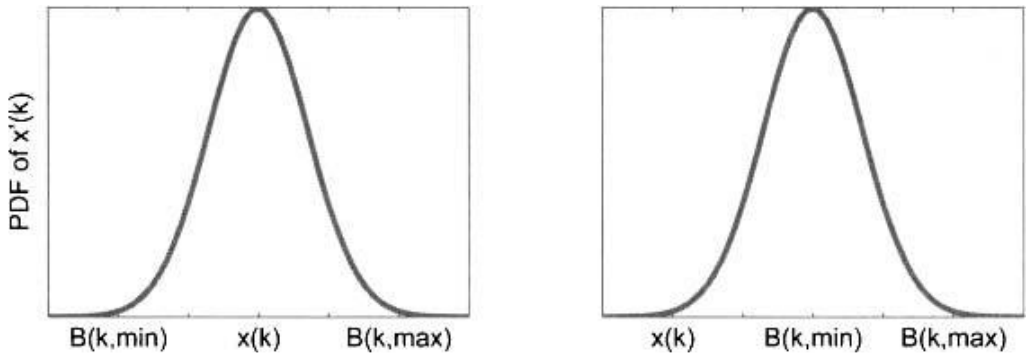
که در آن N اندازه‌ی جمعیت بوده و n ابعاد مسئله است. همچنین در معادله‌ی بالا $r_i(k) \sim N(0,1)$ و γ واریانس جهش است. در CAEP معادله‌ی (۱۵-۲) با معادله‌ی زیر جایگزین می‌شود

[1] Durham
[2] Lehman
[3] Stanley

$$\Delta_i(k) \leftarrow \begin{cases} B_{min}(k) - x_i(k) & \text{اگر } x_i(k) < B_{min}(k) \\ 0 & \text{اگر } B_{min}(k) \le x_i(k) \le B_{max}(k) \\ B_{max}(k) - x_i(k) & \text{اگر } B_{max}(k) < x_i(k) \end{cases} \qquad (۳-۱۵)$$

$$x_i'(k) \leftarrow x_i(k) + r_i(k)\sqrt{\gamma} + \Delta_i(k)$$

می‌توان دید که اگر kاُمین بعد ذره‌ی x_i در فضای عقیده واقع شده باشد، آنگاه نسخه‌ی جهش‌یافته‌ی آن $x_i'(k)$ متغیری اتفاقی با میانگین $x_i(k)$ خواهد بود. با این حال، اگر $x_i(k)$ خارج فضای عقیده باشد، آنگاه نسخه‌ی جهش‌یافته‌ی آن متغیری اتفاقی با میانگین $B_{min}(k)$ یا $B_{max}(k)$ خواهد بود (هر کدام که به $x_i(k)$ نزدیکتر باشد). شکل ۱۵-۲ این ایده را به تصویر می‌کشد. این شکل نشان می‌دهد که وقتی $x_i(k)$ درون فضای عقیده است (قسمت سمت چپ شکل)، به شیوه‌ی استاندارد دچار جهش می‌شود. با این حال، وقتی $x_i(k)$ خارج فضای عقیده است (قسمت سمت راست شکل)، نسخه‌ی جهش‌یافته‌ی آن در یکی از لبه‌های فضای عقیده متمرکز شده است. نسخه‌ی جهش‌یافته ممکن است در خارج فضای عقیده واقع شود. در واقع احتمال قرار گرفتن نسخه‌ی جهش‌یافته در خارج از بازه‌ی $[B_{min}(k), B_{max}(k)]$ برابر ۵۰٪ است. با این حال، به احتمال ۵۰٪ نیز ممکن است درون فضای عقیده واقع شود، احتمالی که بسیار بیشتر از احتمال واقع شدن در درون فضای عقیده در صورت جابه‌جا نشدن میانگین جهش است.



شکل ۱۵-۲ جهش در برنامه‌ی تکاملی فرهنگی. در شکل سمت چپ، $x(k)$ درون فضای عقیده است، بنابراین تابع چگالی احتمال (PDF) نسخه‌ی جهش‌یافته‌ی آن دارای میانگینی برابر $x(k)$ است. در شکل سمت راست، $x(k)$ خارج فضای عقیده است، بنابراین نسخه‌ی جهش‌یافته دارای میانگینی برابر با نزدیکترین لبه از فضای عقیده است.

حال به بحث در مورد چگونگی به‌روزرسانی فضای عقیده در CAEP می‌پردازیم. چندین راه برای این کار وجود دارد. برای نمونه، می‌توان از بهترین M ذره برای به‌روزرسانی فضای عقیده استفاده نمود. ابتدا برای هر $k \in [1, n]$ مقدار مینیمم و ماکزیمم هر بعد از M ذره را می‌یابیم:

$$x_{k,min}(k) \leftarrow \min\{x_j(k) : j \in [1, M]\}$$
$$x_{k,max}(k) \leftarrow \max\{x_j(k) : j \in [1, M]\} \quad (۱۵-٤)$$

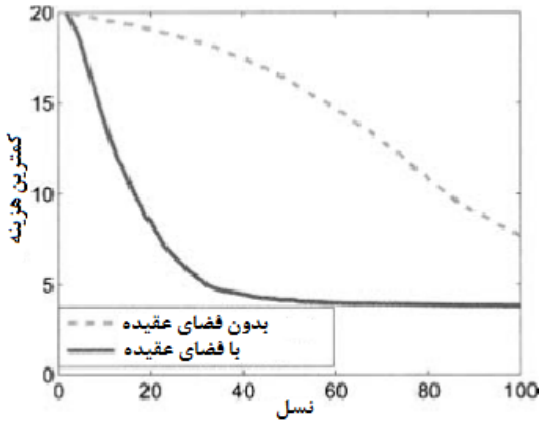
که در آن ذرات به ترتیب از بهترین تا بدترین اندیس‌گذاری شده‌اند به‌طوری که $\{x_j(k): j \in [1, M]\}$ شامل بهترین M ذره از جمعیت می‌شود. حال برای $k \in [1, n]$ از مقادیر مینیمم و ماکزیمم دامنه برای تأثیرگذاری بر روی فضای عقیده از یک نسل به نسل دیگر، استفاده می‌نماییم:

$$B_{min}(k) \leftarrow \alpha B_{min}(k) + (1 - \alpha)x_{k,min}$$
$$B_{max}(k) \leftarrow \alpha B_{max}(k) + (1 - \alpha)x_{k,max} \quad (۱۵-۵)$$

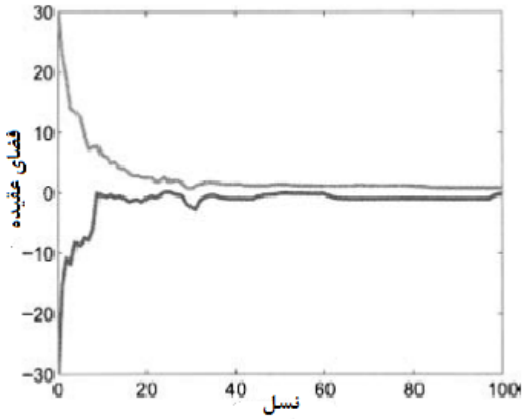
پارامتر $\alpha \in [0,1]$ مقدار اینرسی فضای عقیده است و میزان رکود فضای عقیده از یک نسل به نسل دیگر را تعیین می‌کند. معادله‌ی (۱۵-۵) نشان می‌دهد که اگر $\alpha = 1$ باشد، فضای عقیده هیچگاه تغییر نمی‌کند. همچنین $\alpha = 0$ نیز بدین معناست که فضای عقیده کاملاً توسط جمعیت حاضر تعیین شده و به هیچ عنوان تحت تأثیر فضای عقیده‌ی نسل‌های گذشته نیست.

مثال ۱۵-۲

این مثال نشان می‌دهد چگونه دخالت دادن فرهنگ می‌تواند به بهبود عملکرد EP کمک کند. در این مثال ما از $N = 50$، $\beta = 0$ و $\gamma = 1$ برای پارامترهای شکل ۵-۱ استفاده می‌نماییم. ما از EP برای مینیمم‌سازی تابع ۲۰ بعدی اکلی که در ضمیمه‌ی ج.۲-۱ تعریف شده است استفاده می‌نماییم. همچنین هر بعد از هر ذره به‌صورت اتفاقی در بازه‌ی $[-30, +30]$ مقداردهی اولیه می‌شود. برای CAEP، از $M = 5$ در معادله‌ی (۱۵-٤) و از $\alpha = 0.5$ در معادله‌ی (۱۵-۵) استفاده می‌نماییم. شکل ۱۵-۳ نتایج EP استاندارد و EP فرهنگی را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. می‌توان دید که EP فرهنگی بهتر از EP استاندارد عمل می‌کند. شکل ۱۵-٤ نحوه‌ی تغییر فضای عقیده‌ی بعد اول از یک نسل به نسل دیگر را نشان می‌دهد. می‌توان دید که فضای عقیده سریعا به یک دامنه‌ی کوچک که شامل راه‌حل بهینه (۰) می‌شود، همگرا می‌شود. توجه کنید که هیچ تضمینی برای مشمول شدن راه‌حل بهینه در فضای عقیده وجود ندارد. در حقیقت، شکل ۱۵-٤ نشان می‌دهد که حد پایین فضای عقیده گاهاً کمی از ۰ تجاوز می‌کند. اما در کل، فضای عقیده نشانی خوبی از محل قرار گرفتن راه‌حل‌های نامزد خوب در فضای جستجو به دست می‌دهد. به کار بردن مقدار کوچکتری از α در معادله‌ی (۱۵-۵) باعث همگرایی سریعتر و به کار بردن مقدار بزرگتر از α باعث همگرایی کندتر می‌شود.



شکل ۱۵-۳ مثال ۱۵-۲: برنامه‌نویسی تکاملی با فضای عقیده و بدون فضای عقیده. این شکل هزینه‌ی بهترین ذره در جمعیت در هر نسل را که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. CAEP بهتر از EP استاندارد عمل می‌کند.



شکل ۱۵-٤ مثال ۱۵-۲: فضای عقیده‌ی اولین بعد از CAEP برای تابع ۲۰ بعدی اکلی. فضای عقیده به سرعت به سمت یک ناحیه‌ی کوچک حول ۰، که راه‌حل بهینه است، همگرا می‌شود.

۱۵-۴ مدل فرهنگی اقتباسی

این فصل به بحث در مورد یک الگوریتم فرهنگی، که جایگزینی برای روش فضای عقیده‌ی بحث شده در قسمت قبل است، می‌پردازد. این الگوریتم، مدل فرهنگی اقتباسی (ACM[1]) نام دارد [اکسلرود[2]، ۱۹۹۷]،

[1] Adaptive Cultural Model
[2] Axelrod

[کندی، ۱۹۹۸]، [کندی و ابرهارت، ۲۰۰۱، فصل ٦]. ACM بر پایه‌ی تعامل ذرات در جوامع انسانی است. برای نمونه:

- ذرات بیشتر از کسانی که به آن‌ها نزدیک هستند، چه جغرافیایی چه نسبی، تأثیر می‌پذیرند تا آن‌هایی که دور هستند [لاتان[1] و همکاران، ۱۹۹٤]. این موضوع یادآور همسایگی در الگوریتم تجمع ذرات از فصل ۱۱ است.
- ذرات بیشتر از کسانی که شبیه آن‌ها هستند تأثیر می‌پذیرند تا کسانی که نسبت به آن‌ها متفاوت هستند [اکسلرود، ۱۹۹۷]، [کندی، ۱۹۹۸].
- ذرات بیشتر از کسانی که موفق هستند تأثیر می‌پذیرند [نوئل[2] و جنت[3]، ۲۰۰۵]. به بیان دقیق‌تر می‌توان گفت که ذرات بیشتر از کسانی تأثیر می‌گیرند که شبیه خود ایده‌آل آن‌ها هستند تا کسانی که شبیه خود واقعی آن‌ها هستند [وتزل[4] و اینسکو[5]، ۱۹۸۲]، [کندی، ۱۹۹۸].

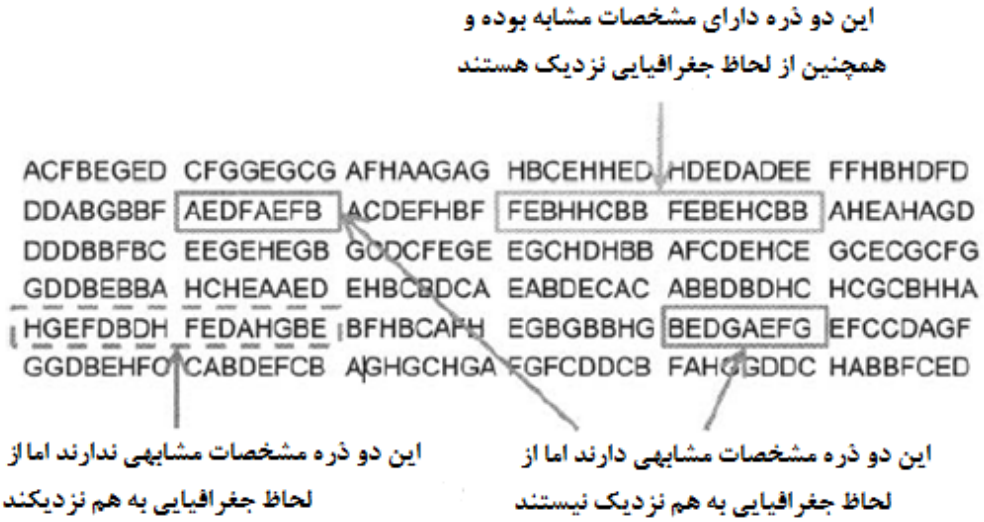
ACM را می‌توان با گستراندن شبکه‌ای از راه‌حل‌های نامزد بر روی مسئله‌ی بهینه‌سازی، شبیه‌سازی نمود. راه‌حل‌های نامزد مانند ذرات در جمعیت در نظر گرفته می‌شوند. نزدیکی دو ذره را می‌توان به چندین روش اندازه گرفت. راه اول، اندازه‌گیری مجاورت ذرات بر مبنای نزدیکی جغرافیایی است چرا که ذرات بر روی یک شبکه قرار گرفته‌اند. روش دوم اندازه‌گیری نزدیکی رفتاری ذرات به یکدیگر است، بدین معنا که چه قدر خواص راه‌حل آن‌ها به یکدیگر شبیه‌اند.

شکل ۱۵-۵ مثالی از یک شبکه از ذرات الگوریتم تکاملی را نشان می‌دهد. در این شکل هر ذره با یک رشته‌ی ۸ کاراکتری کد شده است. با رشد جمعیت، ذرات مکان خود را در شبکه حفظ می‌کنند اما نمایش آن‌ها از یک نسل به نسل دیگر عوض می‌شود. ذراتی که از لحاظ جغرافیایی یا رفتاری به یکدیگر نزدیک هستند، با احتمال بیشتری به تبادل اطلاعات با یکدیگر می‌پردازند و به همین دلیل از لحاظ رفتاری بیشتر به یکدیگر شبیه می‌شوند. همچنین وقتی دو ذره با یکدیگر به تبادل اطلاعات می‌پردازند، احتمال آنکه ذره‌ی برازنده‌تر اطلاعات خود را با ذره‌ی دیگر به اشتراک بگذارد بیشتر از احتمال عکس این حالت است.

ما می‌توانیم از تمامی این ایده‌ها برای به دست آوردن یک الگوریتم ACM استفاده نماییم. شکل ۱۵-٦ یک الگوریتم پایه‌ی ACM را نشان می‌دهد. جمعیت، مقداردهی اولیه شده و به هر راه‌حل نامزد یک مکان خاص در یک شبکه اختصاص داده می‌شود. در هر نسل دوره از الگوریتم ACM، یک ذره و یکی از

[1] Latane
[2] Noel
[3] Jannett
[4] Wetzel
[5] Insko

همسایگانش به‌صورت اتفاقی انتخاب می‌شود. همچنین ما به‌صورت اتفاقی تصمیم می‌گیریم که آیا اطلاعاتی بین این دو ذره به اشتراک گذاشته شود یا خیر. هر چه شباهت بین دو ذره بیشتر باشد، این احتمال بیشتر خواهد بود. اگر تصمیم بر به اشتراک‌گذاری اطلاعات گرفته شود، یکی از خواص راه‌حل ذره‌ی با برازندگی کمتر به‌صورت اتفاقی با یکی از ویژگی‌های راه‌حل ذره‌ی برازنده‌تر جایگزین خواهد شد.



شکل ۱۵-۵ مثالی از یک شبکه از ذرات ACM. برخی ذرات شبیه هم هستند اما به لحاظ جغرافیایی از یکدیگر دورند. احتمال به اشتراک‌گذاری اطلاعات بین این ذرات کم است. سایر ذرات، مانند دو ذره‌ی نشان داده شده در قسمت پایین و چپ شبکه، به لحاظ جغرافیایی نزدیک بوده اما شبیه هم نیستند. احتمال به اشتراک‌گذاری اطلاعات میان این دو ذره نیز کم است. با این حال، برخی ذرات مانند دو ذره‌ی نشان داده شده در در قسمت بالا و راست شبکه هم به لحاظ جغرافیایی نزدیک بوده و هم شبیه هم هستند. احتمال به اشتراک‌گذاری اطلاعات بین این نوع ذرات زیاد است.

جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن $\{x_i\}$، $i \in [1, N]$
به هر ذره محلی اتفاقی در شبکه اختصاص بده
تا زمانی که شرایط توقف برآورده نشده است
یک ذره مانند x_i را به‌صورت اتفاقی از جمعیت انتخاب کن
یک همسایه از x_i مانند x_k را به‌صورت اتفاقی انتخاب کن
تشابه رفتاری میان x_i و x_k را محاسبه کن: $b_{i,k} \in [0,1]$
$r \leftarrow U[0,1]$
اگر $r < b_{i,k}$
یک اندیس ویژگی راه‌حل را به‌صورت اتفاقی انتخاب کن $s \in [1, n]$
کامنت: به اشتراک‌گذاری اطلاعات را آغاز کن
اگر x_i از x_k برازنده‌تر بود
$x_k(s) \leftarrow x_i(s)$
در غیر این صورت
$x_i(s) \leftarrow x_k(s)$
پایان اگر
کامنت: پایان به اشتراک‌گذاری اطلاعات
پایان اگر
تعامل بعد

شکل ۱۵-۶ طرح کلی از یک مدل فرهنگی اقتباسی (ACM). N اندازه‌ی جمعیت، n ابعاد مسئله و $U[0,1]$ یک عدد اتفاقی با توزیع یکنواخت بین ۰ و ۱ می‌باشد.

از شکل ۱۵-۶ می‌توان دید که جهت انتقال اطلاعات همواره از ذره‌ی برازنده‌تر به ذره‌ی دیگر است. برای اتفاقی‌تر کردن این فرایند، می‌توان از یک تصمیم‌گیری احتمالاًتی برای انتخاب اینکه چه کسی اطلاعات خود را با دیگری به اشتراک بگذارد استفاده نماییم. برای این کار از پارامتر میزان‌سازی $p_1 \in [0.5,1]$ که نشان‌دهنده‌ی احتمال به اشتراک‌گذاری اطلاعات از ذره‌ی بهتر به بدتر است استفاده می‌نماییم. به p_1 فشار انتخاب می‌گویند. ما همواره می‌خواهیم که $p_1 \geq 0.5$ باشد چرا که منطقی نیست که جهت جاری شدن اطلاعات را به سمت "از بد به خوب" گرایش دهیم. سپس، قسمتی از کد در شکل

۱۵–۶ را که میان دو خط "کامنت: به اشتراک‌گذاری اطلاعات را آغاز کن" و "کامنت: به اشتراک‌گذاری اطلاعات را تمام کن" قرار گرفته است، با منطق نشان داده شده در شکل ۱۵–۷ جایگزین می‌نماییم.

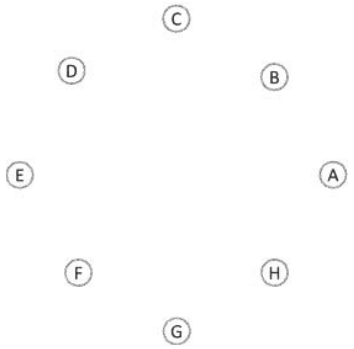
```
ρ ← U[0,1]
اگر ρ < p_1 آنگاه
    اگر x_i از x_k برازنده‌تر بود
        x_k(s) ← x_i(s)
    در غیر این صورت
        x_i(s) ← x_k(s)
    پایان اگر
در غیر این صورت
    اگر x_i از x_k برازنده‌تر بود
        x_i(s) ← x_k(s)
    در غیر این صورت
        x_k(s) ← x_i(s)
    پایان اگر
پایان اگر
```

شکل ۱۵–۷ مدل فرهنگی اقتباسی با به اشتراک‌گذاری اطلاعات اتفاقی. $p_1 \in [0.5, 1]$ احتمال به اشتراک‌گذاری اطلاعات از ذره‌ی خوب به بد را مشخص می‌کند. این تکه شبه کد جایگزین کد واقع شده میان دو عبارت "کامنت: به اشتراک‌گذاری اطلاعات را آغاز کن" و "کامنت: به اشتراک‌گذاری اطلاعات را تمام کن در شکل ۱۵–۶ می‌شود. اگر $p_1 = 1$ باشد، آنگاه این کد مانند کد شکل ۱۵–۶ می‌شود.

مثال ۱۵–۳

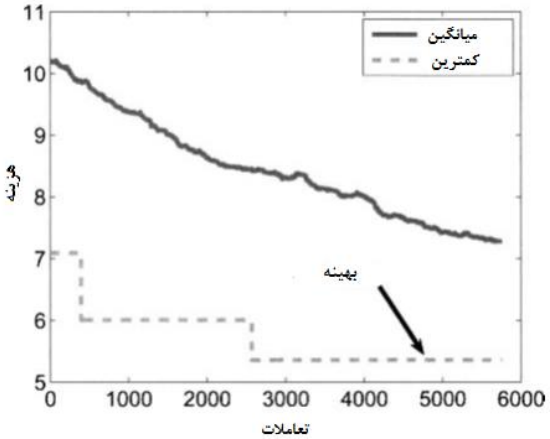
در این مثال، که از [کندی و ابرهارت، ۲۰۰۱، فصل ۶] الهام گرفته است، به حل مسئله‌ی فروشنده‌ی دوره‌گرد (TSP) با ACM می‌پردازیم. فرض کنید می‌خواهیم به هشت محل مختلف سفر کنیم به‌طوری که مجموع مسافت طی شده مینیمم باشد. ما فرض می‌کنیم که این محل‌ها بر روی یک دایره به‌صورت شکل ۱۵–۸ واقع شده‌اند. همچنین فرض بر آن است که سفر خود را از محل A آغاز خواهیم کرد. به سادگی می‌توان دید که دو راه‌حل برای این TSP وجود دارد: A-B-C-D-E-F-G-H یا A-H-G-F-E-D-C-B. ابتدا یک شبکه‌ی 8×18 از راه‌حل‌های نامزد را به‌صورت اتفاقی ایجاد می‌نماییم. همچنین فرض بر آن است که شبکه یک شبکه‌ی حلقوی است، بدین معنا که ذرات واقع شده در انتهای سمت راست شبکه با ذرات واقع

شده در انتهای چپ همسایه هستند و ذرات واقع شده در انتهای بالایی شبکه نیز با ذرات واقع شده در انتهای پایینی همسایه‌اند. ما از منطق شکل‌های ۱۵–۶ و ۱۵–۷ برای هماهنگ‌سازی به اشتراک‌گذاری اطلاعات میان راه‌حل‌های نامزد استفاده می‌نماییم. ما عبارت یک همسایه‌ی اتفاقی x_k را برای x_i انتخاب کن" از شکل ۱۵–۶ را با انتخاب اتفاقی یکی از چهار ذره‌ی نزدیک به x_i، که ذرات بالا، پایین، چپ و راست x_i هستند، پیاده‌سازی می‌نماییم. ما از $p_1 = 0.9$ در شکل ۱۵–۷ استفاده می‌نماییم.



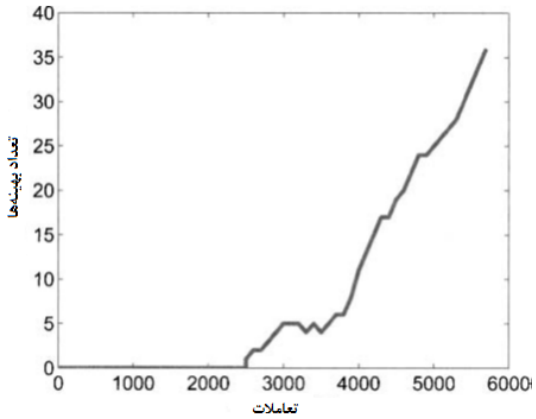
شکل ۱۵–۸ محل‌های TSP از مثال ۱۵–۳. هدف عبور از هر هشت محل همراه با مینیمم‌سازی مجموع مسافت طی شده است. اگر از محل A آغاز نماییم دو راه‌حل بهینه خواهیم داشت: A-B-C-D-E-F-G-H و A-H-G-F-E-D-C-B.

شکل ۱۵–۹ همگرایی یک شبیه‌سازی معمول از ACM را نشان می‌دهد. اولین راه‌حل بهینه بعد از ۲۵۰۰ دوره از شکل ۱۵–۶ پیدا شده است. میانگین هزینه‌ی جمعیت به‌صورت پیوسته با افزایش شماره‌ی دوره کاهش یافته است.



شکل ۱۵–۹ همگرایی ACM برای TSP از مثال ۱۵–۳. اگر هشت شهر شکل ۱۵–۸ بر روی یک دایره واقع شده باشند، هزینه‌ی مینیمم کلی برابر ۵,۳۵۷۶ خواهد بود.

با ادامه‌ی تعامل میان ذرات، به تدریج ذرات خوب در جمعیت پخش شده و ذرات بد کم‌کم محو می‌گردند. شکل ۱۵–۱۰ پخش شدن ذرات خوب را نشان می‌دهد. اولین ذره‌ی بهینه بعد از ۲۵۰۰ دوره پیدا شده است. بعد از این دوره پخش راه‌حل‌های بهینه تقریباً به‌صورت خطی افزایش یافته است.



شکل ۱۵–۱۰ مثال ۱۵–۳: پخش راه‌حل‌های بهینه در شبکه‌ی جمعیت برای TSP. حدود ۲۵۰۰ دوره طول کشیده است تا راه‌حل بهینه در جمعیت ظاهر شود و پس از این دوره پخش راه‌حل‌های بهینه تقریباً به‌صورت خطی افزایش یافته است.

شکل ۱۵–۱۱ شبکه‌ی جمعیت 8×18 را بعد از ۵۷۶۰ تعامل، که به معنای میانگین ۸۰ تعامل در هر ذره است، نشان می‌دهد. می‌توان دید که ۳۱ راه‌حل بهینه A-B-C-D-E-F-G-H در دو انتهای چپ و راست شبکه خوشه شده‌اند (به یاد آورید که شبکه حلقوی است و بنابراین لبه‌های چپ و راست شبکه با هم همسایه هستند). همچنین می‌توان دید که خوشه‌ی کوچکتری از پنج راه‌حل بهینه A-H-G-F-E-D-C-B نیز نزدیک لبه‌ی پایینی شبکه قرار گرفته‌اند. نگاهی دقیق‌تر به شکل ۱۵–۱۱ خوشه‌های دیگری را آشکار می‌سازد که راه‌حل‌های نیمه بهینه برای TSP هستند. این موضوع مانند چگونگی پخش شدن رفتار و اطلاعات در یک فرهنگ، چگونگی گروه گروه شدن ذرات شبیه به هم و همچنین چگونگی شبیه‌سازی رفتار فرهنگی برای حل مسائل بهینه‌سازی است.

شکل ۱۵–۱۱ شبکه‌ی جمعیت از مثال ۱۵–۳ بعد از ۵۷۶۰ تعامل. دو خوشه‌ی بهینه (یکی با ۳۱ ذره و دیگری با ۵ ذره) در شبکه مشخص شده‌اند. در ACM ذرات شبیه به هم با هم تشکیل گروه داده و ذرات با برازندگی بالا در جمعیت پخش می‌شوند.

مثال ۱۵–۳ نشان می‌دهد چگونه ACM می‌تواند جواب‌های متفاوتی برای یک مسئله‌ی بهینه‌سازی ترکیبی پیدا کند. این مثال را می‌توان برای حل مسائل بهینه‌سازی پیوسته نیز بسط داد. تعامیم بسیاری را می‌توان برای الگوریتم‌های ACM شکل‌های ۱۵–۶ و ۱۵–۷ در نظر گرفت.

۱. ما اجازه دادیم هر ذره از یکی از چهار همسایه‌اش تأثیر بپذیرد. می‌توان اجازه داد که همسایگان دورتر نیز بر ذره تأثیر بگذارند. احتمال و یا میزان تعامل را می‌توان تابعی با نسبت عکس نسبت به فاصله در نظر گرفت.

۲. ما معمولاً اطلاعات ذرات برازنده‌تر را با ذرات با برازندگی کمتر به اشتراک می‌گذاریم چرا که می‌خواهیم خواص راه‌حل سودمند را در میان جمعیت پخش نماییم. با این حال، گاه در جوامع مختلف شاهد تأثیر افراد ناموفق بر روی دیگر ذرات هستیم. ما معمولاً از رفتارهایی که در افراد ناموفق مشاهده می‌کنیم پرهیز می‌کنیم. در قسمت ۱۱–۶ نشان دادیم که چگونه می‌توان این ایده را برای بهینه‌سازی تجمع ذرات استفاده نمود، اما این ایده هنوز برای CA به کار نرفته است بنابراین این موضوع را می‌توان به‌عنوان زمینه‌ای مناسب برای تحقیقات آتی در نظر گرفت.

۳. در شکل ۱۵-٦ ما یک ذره‌ی x_i را به‌صورت اتفاقی برای تعامل با ذره‌ای دیگر انتخاب نمودیم. با این حال، منطقی است اگر یک ذره با برازندگی کم را برای این کار انتخاب نماییم چرا که این ذرات نیاز بیشتری به بهبود دارند. این ایده مشابه ایده‌ی احتمالات مهاجرت در BBO است (فصل ١٤ را ببینید).

٤. در شکل ۱۵-٦ یک ذره‌ی x_k را که همسایه‌ی x_i بود به‌صورت اتفاقی برای تعامل با x_i انتخاب نمودیم. یا این حال، شاید منطقی‌تر باشد که گروهی از همسایگان را انتخاب کرده و سپس بنا بر مقادیر نسبی برازندگی در مورد استراتژی به اشتراک‌گذاری اطلاعات تصمیم بگیریم. این ایده مشابه ایده‌ی احتمالات مهاجرت از جزیره در BBO است (فصل ١٤ را ببینید). این تعمیم و تعمیم پیشین هر دو می‌توانند الهام‌بخش یک الگوریتم ترکیبی BBO فرهنگی باشند.

۵. ما می‌توانیم ایده‌ی فضای عقیده را با ACM ترکیب نماییم. افراد در جوامع انسانی هم تحت تأثیر همسایگان خود هستند و هم تحت تأثیر فرهنگ خود. این ایده مدل تعمیم‌یافته‌ی دیگری (GOM[1]) نام دارد و مشابه تأثیر رسانه در جوامع انسانی می‌باشد [شیبانی[2] و همکاران، ۲۰۰۱]. در شکل ۱۵-٦ ما یکی از همسایگان را به‌صورت اتفاقی برای تعامل با x_i انتخاب می‌نماییم. در GOM، ما یک همسایه‌ی تعمیم‌یافته ایجاد می‌نماییم. این همسایه‌ی تعمیم‌یافته نماینده‌ی اجتماع کل جمعیت است. این همسایه به‌صورت واقعی در جمعیت حضور ندارد بلکه یک شبه‌ذره است که با گرفتن میانگین از کل جمعیت به وجود می‌آید. سپس ذره‌ی x_i می‌تواند با یکی از چهار همسایه‌ی خود و یا با همسایه‌ی تعمیم‌یافته تعامل نماید. این ایده یادآور تجمع ذرات کاملاً آگاه، که شامل به اشتراک‌گذاری اطلاعات سراسری می‌باشد، است (بخش ۱۱-۵ را ببینید). همسایه‌ی تعمیم‌یافته را همچنین می‌توان با گرفتن میانگین وزنی از برازندگی جمعیت به دست آورد. هر چند این تعمیم هنوز مورد مطالعه واقع نشده است.

۱۵-۵ نتیجه‌گیری

الگوریتم‌های فرهنگی شاخه‌ای بسیار جالب از از محاسبات تکاملی هستند. این الگوریتم‌ها از سایر الگوریتم‌های تکاملی متفاوت‌اند چرا که به جای زیست‌شناسی از علوم اجتماعی نشئت گرفته‌اند. این موضوع باعث می‌شود تا زمینه‌ای وسیع از تحقیقات و مطاعات علوم اجتماعی برای اعمال آن‌ها به سیستم‌های محاسباتی خودسازمانده و الگوریتم‌های بهینه‌سازی، به وجود آید. از آنجا که الگوریتم‌های فرهنگی اولین بار در دهه‌ی ۱۹۸۰ مورد مطالعه قرار گرفتند، به نظر می‌رسد تا به حال بیشتر تمرکز بر روی کاربردها و اصلاحات

[1] Generalized Other Model
[2] Shibani

ساده‌ی ایده‌های اصلی CA قرار داشته است. با این حال، قسمت اعظمی از تحقیقات علوم اجتماعی به جنبه‌های مختلف فرهنگ مانند موسیقی، اقتصاد، زبان، فناوری، روابط خانوادگی، سرگرمی، تحصیلات، ورزش، دارو، مذهب، هنر، ادبیات، سیاست، جنگ و غیره اختصاص یافته است. هر محققی که به یکی از جنبه‌های فرهنگ علاقه داشته باشد، مخزنی بی‌انتها از ایده‌ها برای اعمال به تحقیقات CA در مقابل خود خواهد یافت. برخی زمینه‌های جالب و در عین حال مهم برای تحقیقات آتی در این زمینه به قرار زیراند.

- مدل‌سازی ریاضی الگوریتم‌های فرهنگی زمینه‌ای مناسب و جا افتاده برای کارهای آتی به نظر می‌رسد. مدل‌های ریاضی سایر الگوریتم‌های تکاملی را می‌توان به راحتی در مقالات پیدا نمود اما در مورد الگوریتم‌های فرهنگی، کمبود مدل‌سازی ریاضی کاملاً حس می‌شود.
- فرهنگ‌ها دارای مجموعه‌های مختلفی از عقیده هستند. برخی از این عقاید متعلق به اکثر ذرات بوده و برخی دیگر تنها توسط اقلیت ذرات رعایت می‌شوند [لاتان و همکاران، ۱۹۹٤]. برخی اوقات، فضاهای عقیده در پدیده‌ای به اسم جنگ فرهنگی با یکدیگر تلاقی می‌کنند [تامسون، ۲۰۱۰]. این پدیده را می‌توان به خصوص در زمینه‌های بحث‌برانگیزی مانند مذهب، ورزش و سیاست مشاهده نمود.
- جوامع شامل فرهنگ‌های متفاوتی هستند. برخی فرهنگ‌ها در دل فرهنگ دیگری قرار گرفته و زیرفرهنگ نامیده می‌شوند. افراد واقع در یک زیرفرهنگ به طرز کاملاً نزدیکی با یکدیگر تعامل کرده و افرادی که در زیرفرهنگ‌های متفاوت قرار دارند تعامل ضعیفی با هم دارند. برخی سیستم‌های ارزشی در یک زیرفرهنگ مورد تأکید واقع شده‌اند و برخی دیگر در زیرفرهنگ‌های دیگر بسیار حائز اهمیت هستند. این ایده دارای کاربردی در بهینه‌سازی چندهدفه می‌باشد [کوئلو کوئلو و بکرا، ۲۰۰۳]، [اعلمی و همکاران، ۲۰۰۷].

چگونه می‌توان این عوامل را در یک CA مدل نمود؟ رابطه‌ی میان این عوامل چگونه است؟ دیگر جنبه‌های فرهنگ که در یادگیری انسانی حائز اهمیت‌اند چه هستند؟ تأثیرات فرهنگ بر افراد مختلف چگونه است؟ همه‌ی این‌ها سؤالاتی برای تحقیقات آتی هستند. برای مطالعات بیشتر در زمینه‌ی CAها می‌توانید به [رینولدز، ۱۹۹٤]، [رینولدز و چانگ[1]، ۱۹۹۷]، [رینولدز، ۱۹۹۹] و [رینولدز و همکاران، ۲۰۱۱] مراجعه نمایید.

[1] Chung

مسائل

تمارین نوشتاری

۱۵-۱ چند روش برای پیش‌بینی عدد بعدی در دنباله‌ی معادله‌ی (۱۵-۱) پیشنهاد کنید.این روش‌ها چه مقادیری را پیش‌بینی می‌کنند؟

۱۵-۲ معادله‌ی (۱۵-۳) و شکل ۱۵-۲ را در نظر بگیرید.

الف) اگر $x_i(k)$ درون فضای عقیده باشد، احتمال آنکه $x_i'(k)$ نیز درون فضای عقیده باشد چه قدر خواهد بود؟

ب) اگر $x_i(k)$ خارج فضای عقیده باشد، احتمال آنکه $x_i'(k)$ نیز درون فضای عقیده باشد چه قدر خواهد بود؟

۱۵-۳ معادله‌ی (۱۵-۳) و شکل ۱۵-۲ را در نظر بگیرید.

الف) اگر $x_i(k) \in B$ باشد، چه استراتژی برای استفاده از فضای عقیده تهاجمی‌تر خواهد بود؟ در اینجا از واژه‌ی تهاجمی‌تر برای اشاره به احتمال بیشتر برای $x_i'(k) \in B$ استفاده شده است.

ب) اگر $x_i(k) \notin B$ باشد، چه استراتژی برای استفاده از فضای عقیده تهاجمی‌تر خواهد بود؟

۱۵-٤ در هر نسل از الگوریتم ACM از شکل ۱۵-٦، چه تعداد ارزیابی تابع برازندگی مورد نیاز است؟ اگر بخواهیم مقایسه‌ای عادلانه میان ACM و سایر الگوریتم‌های تکاملی داشته باشیم، این تعداد ارزیابی تابع برازندگی به چه معنی است؟

۱۵-۵ الگوریتم ACM از شکل ۱۵-٦ در هر تعامل تنها یک ویژگی راه‌حل را به اشتراک می‌گذارد. این موضوع به چه نکته‌ای در مورد عملکرد این الگوریتم بر روی مسائل تفکیک‌پذیر اشاره دارد؟ (به یاد آورید که بحثی مشابه این در ابتدای بخش ۱٤-۵ وجود داشت). چه اصلاحی بر روی الگوریتم ACM انجام دهید تا عملکرد آن در مورد مسائل تفکیک‌ناپذیر بهبود یابد؟

۱۵-٦ شکل ۱۵-۹ نشان می‌دهد که ACM راه‌حل بهینه برای مسئله‌ی فروشنده‌ی دوره‌گرد با ۸ شهر را در ۲۵۰۰ دوره پیدا می‌کند. کیفیت عملکرد این الگوریتم در این مسئله را تحلیل کنید.

مسائل کامپیوتری

۱۵–۷ مثال ۱۵–۲ را با 1 و $\alpha = 0,0.25,0.5,0.75$ تکرار نمایید. نتایج را مشابه شکل ۱۵–۳ برای هر یک از مقادیر α رسم نمایید. نتایج خود را توضیح دهید.

۱۵–۸ مثال ۱۵–۲ را با 25 و $M = 0,2,5,10$ تکرار کنید. نتایج را مشابه شکل ۱۵–۳ برای هر یک از مقادیر M رسم نمایید. نتایج خود را توضیح دهید.

۱۵–۹ مثال ۱۵–۳ را با 1 و $p_1 = 0.5,0.7,0.9$ تکرار کنید. تعداد تعاملات را برای هر شبیه‌سازی به ۲۰۰۰ محدود کنید. نتایج را مشابه شکل ۱۵–۹ برای هر یک از مقادیر p_1 رسم نمایید. با این حال، از آنجا که شکل ۱۵–۹ نتایج یک ACM معمولی را نشان می‌دهد، شما باید برای هر مقدار از p_1، ۲۰ شبیه‌سازی مونت کارلو انجام داده و بهترین هزینه در هر نسل را برای هر یک از مقادیر p_1 ذخیره کرده و سپس رسم کنید.

فصل شانزدهم

یادگیری مقابله محور

انقلاب‌های اجتماعی .... تغییراتی سریع در جامعه‌ی انسانی‌اند. به زبان ساده این انقلاب‌ها به وقوع می‌پیوندند تا شرایط معکوس را برقرار سازند.

حمید تیزهوش[1] [تیزهوش، ۲۰۰۵]

انقلاب یک فرایند کند است. تغییر زمان می‌طلبد. با این حال، برخی از انواع تغییر سریع اتفاق می‌افتند. یکی از انواع تغییر سریع که تقریبا همه‌ی الگوریتم‌های تکاملی از آن استفاده می‌کنند، جهش است. اما نوعی دیگر از تغییر سریع وجود دارد که در جامعه‌ی انسانی به وقوع می‌پیوند و ما هنوز آن را بررسی نکرده‌ایم: انقلاب اجتماعی. یک انقلاب اجتماعی، یک الگوی جابه‌جایی از هنجار کنونی به هنجار مقابل است. گاهاً انقلاب‌های اجتماعی دارای تأثیرات مهم و بلندمدت هستند، مانند هنگامی که آمریکا در جنگ‌های انقلابی در برابر انگلیس به مقابله پرداخت و از کلونی به ایالت تغییر ماهیت داد. سایر انقلاب‌ها دارای تأثیرات چشمگیر کمتری هستند، مانند معرفی مواد مصنوعی ترکیبی در صنعت لباس و یا معرفی امواج مایکروویو برای پخت‌وپز. با این حال، همه‌ی انقلاب‌ها به معنی تغییرات قابل توجه در نحوه‌ی زندگی هستند.

یادگیری مقابله محور (OBL[2]) برای افزایش نرخ یادگیری در الگوریتم‌های تکاملی به وجود آمد. از آنجا که تکامل یک فرایند آهسته و انقلاب یک فرایند سریع است، شبیه‌سازی انقلاب در الگوریتم‌های تکاملی می‌توانند همگرایی آن‌ها را تسریع کند. OBL در ابتدا به‌عنوان روشی برای بهبود یادگیری تقویتی، الگوریتم‌های ژنتیک و شبکه‌های عصبی معرفی گردید [تیزهوش، ۲۰۰۵]. OBL همچنین در بسیاری دیگر از الگوریتم‌های بهینه‌سازی پیاده‌سازی شده است که از جمله‌ی آن‌ها می‌توان به بهینه‌سازی زیست‌جغرافی-محور (BBO) [ارگزر[3] و همکاران، ۲۰۰۹]، بهینه‌سازی تجمع ذرات [عمران، ۲۰۰۸]، [رشید[4] و بیگ[5]، ۲۰۱۰]، تکامل تفاضلی [راهنمایان[6] و همکاران، ۲۰۰۸]، بهینه‌سازی کلونی مورچگان [مالیسیا[7]، ۲۰۰۸] و ذوب فلزات [ونترسکا[8] و تیزهوش، ۲۰۰۷] اشاره نمود.

مروری بر فصل

بخش ۱۶-۱ تعاریفی از مقابله به‌عنوان عبارتی مرتبط با مسائل عددی ارائه می‌دهد. بخش ۱۶-۲ طرح کلی چگونگی وارد نمودن OBL به یک الگوریتم تکاملی و چگونگی بهبود عملکرد BBO توسط آن را نشان

---

[1] Hamid Tizhoosh
[2] Opposition Based Learning
[3] Ergezer
[4] Rashid
[5] Baig
[6] Rahnamayan
[7] Malisia
[8] Ventresca

می‌دهد. بخش ۱۶-۳ به مطالعه‌ی ریاضی احتمال بهبود الگوریتم تکاملی با استفاده از انواع مختلف مقابله می‌پردازد. بخش ۱۶-٤ به معرفی نرخ پرش، که یک مفهوم مورد استفاده در OBL است، می‌پردازد. هرچند که OBL در ابتدا برای مسائل با دامنه‌ی پیوسته تعریف گردید، فصل ۱۶-۵ به بحث در مورد چگونگی تعمیم آن به مسائل ترکیبی، به خصوص مسئله‌ی فروشنده‌ی دوره‌گرد، می‌پردازد. بخش ۱۶-٦ برخی مفاهیم یادگیری دوگانه، که بر OBL مقدم بوده، را مرور کرده و رابطه‌ی میان این دو روش را نشان می‌دهد.

۱۶-۱ مفاهیم و تعاریف مقابله

این بخش به بحث در مورد تعاریف و مفاهیم مرتبط با مقابل و مخالف یک عدد اسکالر یا بردار می‌پردازد. کار را با اعداد اسکالر آغاز می‌کنیم. فرض کنید x بر روی دامنه‌ی $[a,b]$ تعریف شده است و مرکز این دامنه c است:

$$x \in [a,b] \text{ که در آن } a < b \\ c = (a+b)/2 \qquad (۱-۱٦)$$

۱۶-۱-۱ مخالف بازتابی و مخالف madulo

روش‌های متنوعی برای تعریف مخالف یک اسکالر مانند x وجود دارد [تیزهوش و همکاران، ۲۰۰۸]. برای مثال، مخالف بازتابی x به‌صورت زیر تعریف می‌شود

$$x_{o1} = a + b - x \qquad (۲-۱٦)$$

این بدین معناست که فاصله‌ی x_{o1} از مرکز دامنه به اندازه‌ی فاصله‌ی x از مرکز دامنه است:

$$c - x = x_{o1} - c \qquad (۳-۱٦)$$

مخالف modulo اسکالر x نیز به‌صورت زیر تعریف می‌شود

$$x_{o1} = (x - a + c) \; mod \; (b - a) \qquad (٤-۱٦)$$

در این روش، دامنه‌ی $[a,b]$ به‌عنوان یک دایره در نظر گرفته شده و مخالف x نیز عددی است که در سوی مخالف دایره قرار گرفته است. شکل ۱۶-۱ مخالف بازتابی و مخالف modulo را نشان می‌دهد.

تعاریف مخالف بازتابی و مخالف modulo را می‌توان به‌صورت ساده و سرراست به بردارها نیز بسط داد. فرض کنید x یک بردار n بعدی است. که بر روی یک دامنه‌ی مستطیلی تعریف شده است؛ بدین معنی که x_i بر روی دامنه‌ی $[a_i, b_i]$ تعریف شده و c_i نیز مرکز این دامنه است:

$$x = [x_1 \dots x_n]$$
$$i \in [1,n] \text{ برای } x_i \in [a_i, b_i] \text{ و } a_i < b_i \text{ که در آن} \qquad (۵-۱۶)$$
$$i \in [1,n] \text{ برای } c_i = (a_i + b_i)/2$$

در این صورت، مخالف بازتابی بردار x به‌صورت زیر تعریف می‌شود

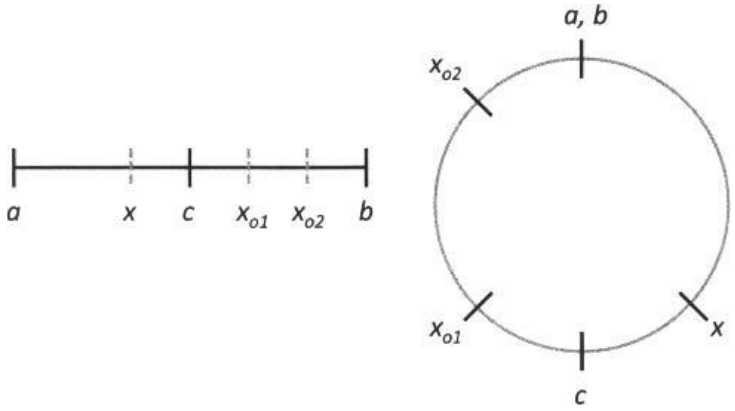
$$x_{o1} = [x_{o1,1} \dots, x_{o1,n}]$$
$$i \in [1,n] \text{ برای } x_{o1,i} = a_i + b_i - x_i \text{ که در آن} \qquad (۶-۱۶)$$

مخالف modulo بردار x نیز به‌صورت زیر تعریف می‌گردد

$$x_{o1} = [x_{o2,1} \dots, x_{o2,n}]$$
$$x_{o2,i} = (x_i - a_i + c_i)\ mod\ (b_i - a_i) \text{ که در آن} \qquad (۷-۱۶)$$

این تعاریف تنها در مورد دامنه‌های مستطیلی قابل اعمال هستند. تعمیم این تعاریف به دامنه‌های غیر مستطیلی به‌عنوان زمینه‌ای برای کارهای آتی رها می‌شود اما به احتمال زیاد، کار دشواری نخواهد بود.

ما از مخالف modulo در این فصل استفاده نخواهیم نمود. در باقی این فصل ما از عبارت "مخالف x" برای اشاره به مخالف بازتابی x و از نماد x_o برای اشاره به x_{o1} استفاده خواهیم نمود.



شکل ۱۶-۱ نمایش مخالف بازتابی x_{o1} و مخالف modulo، x_{o2} از اسکالر x. شکل سمت چپ دامنه‌ی x را به‌صورت خطی نشان داده و شکل سمت راست آن را به‌صورت یک دایره نشان می‌دهد. اسکالر x بر روی دامنه‌ی $[a,b]$ تعریف شده و c مرکز دامنه می‌باشد. فاصله‌ی مخالف بازتابی از c، به اندازه‌ی فاصله‌ی x از c است و مخالف modulo، x_{o2} در سمت مقابل دایره‌ای که دامنه‌ی x را تعریف می‌کند، واقع شده است.

۱۶-۱-۲ مخالف جزیی

اگر یک بردار مانند x در اختیار داشته باشیم، می‌توانیم مخالف جزیی x را، که با x_p نمایش داده می‌شود، با مخالف نمودن برخی از ابعاد آن و بدون تغییر گذاشتن سایر عناصر آن، تعریف نماییم. برای مثال:

$$x = [x_1 \dots x_n]$$
$$x_p = [x_{p1} \dots x_{pn}] \text{ مخالف جزیی}$$
$$x_{pi} = \begin{cases} x_{oi} & \text{برای } i \in S \\ x_i & \text{برای } i \in \overline{S} \end{cases} \text{ که در آن} \qquad (۸-۱۶)$$

که در آن S یک زیرمجموعه از $\{1,2,\dots,n\}$ بوده و $\overline{S}$ نیز متمم S است، بدین معنی که $S \cup \overline{S} = \{1,2,\dots,n\}$ و برای تمامی $j \in \{1,\dots,|\overline{S}|\}$، $S_j \notin \overline{S}$. در این صورت درجه‌ی مخالفت x_p به‌صورت زیر تعریف می‌شود

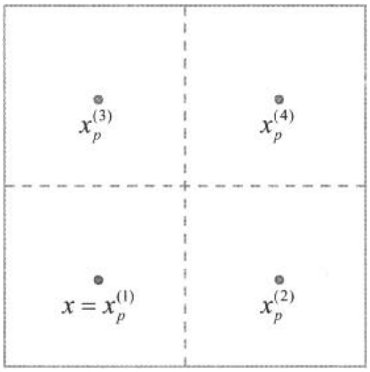
$$\tau(x_p) = |S|/n \qquad (۹-۱۶)$$

مثال ۱۶-۱

فرض کنید که $x = [0.5\ 0.5]$ به‌طوری که هر دو عنصر x بر روی دامنه‌ی $[0,2]$ تعریف شده باشند. ما می‌توانیم چهار مخاف جزیی برای x تعریف نماییم:

$$x_p^{(1)} = [0.5\ 0.5] \rightarrow \tau(x_p^{(1)}) = 0$$
$$x_p^{(2)} = [1.5\ 0.5] \rightarrow \tau(x_p^{(1)}) = 1/2$$
$$x_p^{(3)} = [0.5\ 1.5] \rightarrow \tau(x_p^{(1)}) = 1/2$$
$$x_p^{(4)} = [1.5\ 1.5] \rightarrow \tau(x_p^{(1)}) = 1 \qquad (۱۰-۱۶)$$

شکل ۱۶-۲ چهار مخالف جزیی x را نشان می‌دهد.



شکل ۱۶-۲ مثال ۱۶-۱: درجه‌ی مخالفت مخالفین جزیی بردار دو بعدی x. بردار $x_p^{(1)}$ با بردار x یکی است، بنابراین درجه‌ی مخالفت آن ۰ است. بردارهای $x_p^{(2)}$ و $x_p^{(3)}$ دارای یک عنصر مخالف عنصر متناظر در x و یک عنصر یکسان با عنصر متناظر در x می‌باشند، بنابراین درجه‌ی مخالفت این دو بردار برابر ۰,۵ است. تمام عناصر بردار $x_p^{(4)}$ مخالف عناصر متناظر در x می‌باشند، بنابراین درجه‌ی مخالفت آن برابر ۱ است.

۱۶-۱-۳ مخالفان نوع ۱ و مخالفان نوع ۲

تا به اینجا، مخالفت را تحت دامنه‌ی یک تابع تعریف نموده‌ایم؛ این مخالفت نوع ۱ نام دارد. ما همچنین می‌توانیم مخالفت را تحت برد یک تابع تعریف نماییم. که این نوع مخالفت، مخالفت نوع ۲ خواهد بود [تیزهوش و همکاران، ۲۰۰۸]. این کار را با یک تابع اسکالر $y(.)$ از یک اسکالر x که بر روی دامنه‌ی $[a,b]$ تعریف شده است، آغاز می‌نماییم. برد $[y_{min}, y_{max}]$ به‌صورت زیر تعریف می‌شود

$$y_{min} = \min y(x) : x \in [a,b]$$
$$y_{max} = \max y(x) : x \in [a,b] \qquad (۱۱-۱۶)$$

مرکز برد نیز به‌صورت زیر تعریف می‌گردد

$$y_c = (y_{max} - y_{min})/2 \qquad (۱۲-۱۶)$$

مخالف بازتابی نوع ۲ اسکالر x به‌صورت زیر تعریف می‌شود

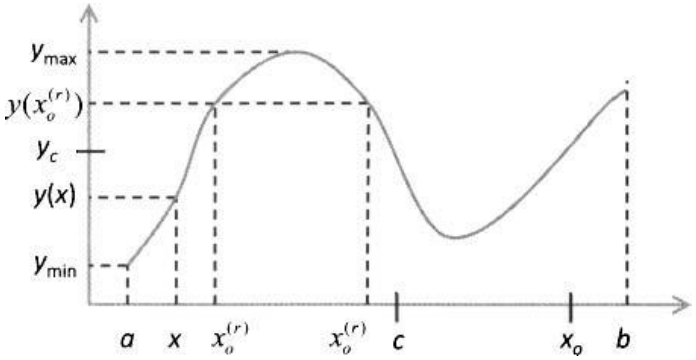
$$x_o^{(r)} = x' : y(x') = y_{max} + y_{min} - y(x) \qquad (۱۳-۱۶)$$

این بدین معنی است که فاصله‌ی $y(x_o^{(r)})$ از y_c با فاصله‌ی $y(x)$ از y_c برابر است:

$$y_c - y\left(x_o^{(r)}\right) = y_c - y(x) \qquad (۱۴-۱۶)$$

این تعریف می‌تواند مقادیر متفاوتی از $x_o^{(r)}$ را به دست بدهد، مگر آنکه $y(.)$ یک نگاشت یک-به-یک باشد. شکل ۱۶-۳ تفاوت میان مخالف‌های نوع ۱ و ۲ را نشان می‌دهد. توجه داشته باشید که می‌توان تعریف مخالفت نوع ۲ را برای به دست آوردن مخالف نوع ۲ یک بردار، مخالف modulo نوع ۲ از یک بردار و همچنین درجه‌ی مخالفت نوع ۲، تعمیم داد.

در باقی این فصل ما بحث خود را به مخالفت نوع ۱ محدود می‌نماییم و مطالعه‌ی مخالفت نوع ۲ را به تحقیقات آتی واگذار می‌نماییم.



شکل ۱۶-۳ اسکالر x بر روی دامنه‌ی $[a,b]$ و همچنین تابع $y(x)$ را در نظر بگیرید. مخالف نوع ۱ اسکالر x با x_o نمایش داده شده و با بازتاب دادن x نسبت به c که مرکز دامنه است، به دست می‌آید. مخالف نوع ۲ اسکالر x با بازتاب دادن $y(x)$ نسبت به y_c، که مرکز برد است، برای به دست آوردن $y(x_o^{(r)})$ و سپس محاسبه‌ی معکوس $y(x_o^{(r)})$ برای به دست آوردن $x_o^{(r)}$ به دست می‌آید. این کار در این مثال به دو مقدار محتمل برای $x_o^{(r)}$ منجر می‌شود.

۱۶-۱-۴ مخالفان کوشی[1] و فرامخالفان[2]

حال به معرفی سه رویکرد دیگر از مخالفت می‌پردازیم. مانند قبل، اسکالر x در دامنه‌ی $[a,b]$ با مرکز c را در نظر می‌گیریم.

مخالف کوشی x به‌صورت زیر تعریف می‌گردد [تیزهوش و همکاران، ۲۰۰۸]:

$$x_{qo} = rand(c, x_o) \qquad (۱۵-۱۶)$$

که در آن x_o مخالف بازتابی استاندارد است که در معادله‌ی (۱۶-۲) تعریف شده است. بدین ترتیب، x_{qo} عددی است اتفاقی با توزیع یکنواخت بر روی بازه‌ی $[c, x_o]$. توجه داشته باشید که تابع $rand$ به‌گونه‌ای

[1] Quasi Opposites
[2] Super Opposites

تعریف شده است که خروجی آن مستقل از ترتیب آرگومان‌های آن است. بنابراین نماد $rand(c, x_o)$ و $rand(x_o, c)$ معادل هستند.

فرامخالفان x به‌صورت زیر تعریف می‌شوند [تیزهوش و همکاران، ۲۰۰۸]:

$$x_{so} = \begin{cases} rand\ (x_o, b) & \text{اگر } x < c \\ rand(a, x_o) & \text{اگر } x > c \end{cases} \qquad (۱۶-۱۶)$$

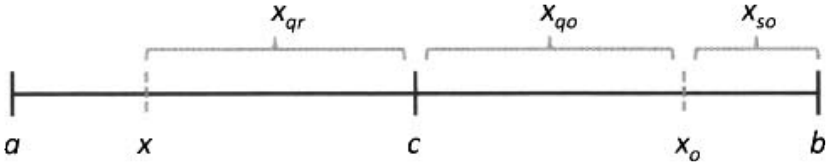
که در آن x_{so} عددی اتفاقی است که بین x_o و کرانی از دامنه که بیشترین فاصله را از x دارد، به‌طور یکنواخت توزیع شده است. این تعریف کامل نیست چرا که x_{so} را برای حالتی که $x = c$ است تعریف نمی‌کند، اما این حالت خاص را با تغییر دلخواه یکی از نامعادلات موجود در معادله (۱۶-۶) به‌طوری که هم شامل معادله و نامعادله شود، برطرف نمود.

مخالف کوشی بازتابی x به‌صورت زیر تعریف می‌شود [ارگزار[1] و همکاران، ۲۰۰۹]:

$$x_{qr} = rand(x, c) \qquad (۱۶-۱۷)$$

که در آن x_{qr} عددی اتفاقی با توزیع یکنواخت بین x و c خواهد بود. توجه داشته باشید که کلمه "بازتابی" در عبارت "کوشی بازتابی" ربطی به کلمه بازتابی در عبارت "مخالف بازتابی" ندارد (معادله (۱۶-۲) را ببینید).

شکل ۱۶-۴، چهار نوع متفاوت مخالفت (تقابل) را نشان می‌دهد. با دنبال نمودن عملیات صورت گرفته در این بخش، می‌توان این تعاریف را به بردارها، مخالف‌های modulo و مخالفان نوع ۲ بسط داد.



شکل ۱۶-۴ فرض کنید یک اسکالر مانند $x \in [a, b]$ در اختیار داریم. مخالف x را با x_o نشان داده و آن را با بازتاب دادن x حول c، مرکز دامنه، به دست می‌آوریم. مخالف کوشی x که با x_{qo} نمایش داده می‌شود نیز با تولید عددی اتفاقی میان c و x_o به دست خواهد آمد. فرامخالف x نیز که با x_{so} نمایش داده می‌شود، با تولید عددی اتفاقی میان x_o و کرانی از دامنه که بیشترین فاصله را با x دارد به دست خواهد آمد. در آخر، مخالف کوشی بازتابی x نیز که با x_{qr} نمایش داده می‌شود، با ایجاد عددی اتفاقی میان c و x به دست خواهد آمد.

[1] Ergezar

ایجاد ارتباط میان این تعاریف و منطق فازی جالب خواهد بود. شکل ١٦-٤ نشانگر عددی بودن مخالف x، x_o می‌باشد. با این حال، x_{qr}، x_{qo} و x_{so} می‌توانند به‌صورت کمیت‌هایی فازی تعریف گردند. ارتباط میان این تعاریف و منطق فازی در این کتاب ارائه نشده است اما می‌تواند به‌عنوان موضوعی بالغ برای تحقیقات آتی در نظر گرفته شود.

۱۶-۲ الگوریتم‌های تکاملی مقابله-محور

این بخش به ارائه یک الگوریتم OBL نوعی پرداخته و نشان می‌دهد چگونه این الگوریتم می‌تواند یک الگوریتم تکاملی را تقویت نماید. به‌طور کلی، یک راه ساده برای استفاده از OBL با هر نوع الگوریتم تکاملی شامل مراحل زیر می‌شود.

۱. هنگامی که N ذره‌ی اولیه جمعیت الگوریتم تکاملی ایجاد می‌شود، N ذره مخالف نیز تولید می‌گردند، به‌طوری که هر ذره مخالف متناظر با یکی از ذرات اصلی در جمعیت الگوریتم تکاملی است. بدین ترتیب $2N$ راه‌حل نامزد وجود خواهد داشت و ما N ذره برتر را از میان این $2N$ ذره برای جمعیت شروع کننده الگوریتم تکاملی نگه می‌داریم. این ایده کلی در بخش ۸-۱ مورد بحث واقع شده است.

۲. ما یک پیاده‌سازی استاندارد از الگوریتم تکاملی را اجرا می‌نماییم. همان‌طور که پیش از این در این کتاب مشاهده نموده‌ایم، این عمل شامل یک حلقه از ارزیابی‌های تابع هزینه، بازترکیب‌ها و جهش‌ها می‌شود. بنا بر تعریف، هر یک حلقه در یک نسل اجرا می‌شود.

۳. هر چند نسل یکبار، مخالف هر یک از N ذره موجود در جمعیت را محاسبه می‌نماییم. از این $2N$ راه‌حل نامزد (N ذره تکاملی استاندارد و N ذره مخالف) بهترین N ذره را برای نسل بعدی الگوریتم تکاملی نگه می‌داریم. در هر نسل، این مرحله را با احتمال $J_r \in [0,1]$ که نرخ پرش نام دارد، انجام می‌دهیم.

ما همچنین باید برخی تصمیمات را در مورد الگوریتم تکاملی مقابله محور خود اتخاذ نماییم.

۱. از چه الگوریتم تکاملی باید استفاده نمود؟ جواب به این سوال شامل تمام پارامترهای میزان‌سازی نیز می‌شود.

۲. از چه نوع مقابله‌ای باید استفاده نماییم؟

۳. مقدار J_r چه قدر باید باشد؟

نرخ پرش یک پارامتر میزان‌سازی است. ما هیچ خط مشی برای J_r نداریم اما نمی‌خواهیم آن را بیش از اندازه بزرگ انتخاب نماییم. دلیل اینکه به‌صورت متناوب یک جمعیت مخالف به وجود می‌آوریم آن است

که می‌خواهیم مناطق کشف نشده فضای جستجو را مورد کاوش واقع دهیم. با این حال نمی‌خواهیم در هر نسل یک جمعیت مخالف به وجود آوریم چرا که در این صورت دائماً در حال جلو و عقب پریدن در فضای جستجو خواهیم بود. این کار باعث هرز رفتن توان با محاسبات زیاد تابع هزینه خواهد شد. نتایج به دست آمده از تکامل دیفرانسیلی مقابله–محور نشانگر آن است که $J_r \approx 0.3$ می‌تواند تعادل خوبی ایجاد کند [رهنمایان و همکاران، ۲۰۰۸].

توجه داشته باشید که یک الگوریتم غیر مقابله–محور که دارای N ذره بوده و برای G نسل اجرا می‌شود شامل GN محاسبه تابع خواهد شد. یک الگوریتم تکاملی مقابله–محور که دارای N' ذره و نرخ پرشی برابر J_r بوده و همچنین برای G' نسل اجرا می‌گردد، به تعداد $G'N'(1+J_r)$ محاسبه تابع به‌صورت میانگین نیاز خواهد داشت. برای آنکه بتوانیم مقایسه‌ای عادلانه میان الگوریتم تکاملی غیر مقابله–محور و الگوریتم تکاملی مقابله–محور انجام دهیم، باید G'، N' و J_r را به‌گونه‌ای انتخاب نماییم که:

$$GN = G'N'(1+J_r) \qquad (۱۶–۱۸)$$

این کار را می‌توان به دو صورت انجام داد؛ یا $N' = N$ قرار دهیم و سپس حد نسل را به اندازه‌ای کم کنیم که $G' = G/(1+J_r)$ حاصل شود و یا $G' = G$ قرار دهیم و اندازه جمعیت را به اندازه‌ای کم نماییم که $N' = N/(1+J_r)$ حاصل شود. روش دیگر آن است که G' و N' را به‌صورت همزمان کاهش دهیم تا معادله (۱۶–۱۸) برآورده شود.

بهینه‌سازی مقابله‌ای زیست‌جغرافی–محور

حال نشان می‌دهیم که چگونه می‌توان از طرح کلی OBL، که در بالا نشان داده شده است، در بهینه‌سازی زیست‌جغرافی–محور (BBO) استفاده نمود. ما الگوریتم BBO استاندارد از شکل ۱۴–۳ را با OBL ترکیب می‌نماییم تا الگوریتم مقابله‌ای BBO (OBBO[1]) به دست آید [ارگزار و همکاران، ۲۰۰۹]. شکل ۱۶–۵ طرح کلی الگوریتم OBBO را نشان می‌دهد. توجه داشته باشید که الگوریتم شکل ۱۶–۵ همان الگوریتم شکل ۱۴–۳ است و فقط خط‌های بین "کامنت: منطق مقابله را آغاز کن" و "کامنت: پایان منطق مقابله" را اضافی دارد.

[1] Oppositional BBO

جمعیتی ابتدایی از راه‌حل‌های نامزد را مقداردهی کن $\{x_k\}$ برای $k \in [1, N]$
تا زمانی که شرایط توقف برآورده نشده است
برای هر μ_k احتمال مهاجرت (از) یا μ_k را متناسب با برازندگی x_k قرار بده به‌طوری که $\mu_k \in [0,1]$
برای هر ذره‌ی x_k احتمال مهاجرت (به) را به‌صورت $\lambda_k = 1 - \mu_k$ قرار بده
$\{z_k\} \leftarrow \{x_k\}$
برای هر ذره‌ی z_k
برای هر ویژگی راه‌حل s
از λ_k برای تصمیم احتمالاتی در مورد مهاجرت نمودن z_k استفاده کن
اگر مهاجرت رخ داد آنگاه
از $\{\mu_i\}_{i=1}^{N}$ برای انتخاب احتمالاتی ذره‌ی مهاجرت کننده‌ی x_j استفاده کن
$z_k(s) \leftarrow x_j(s)$
پایان اگر
ویژگی راه‌حل بعدی
$\{z_k\}$ را به‌صورت احتمالاتی دچار جهش کن
ذره‌ی بعد
کامنت: منطق تقابل را آغاز کن
$r \leftarrow U[0,1]$
اگر $r < J_r$ آنگاه
از $\{z_k\}$ برای ایجاد جمعیت مخالف $\{\bar{z}_k\}$ استفاده کن
بهترین N ذره از $\{z_k\} \cup \{\bar{z}_k\} \leftarrow \{z_k\}$
پایان اگر
کامنت: پایان منطق تقابل
$\{x_k\} \leftarrow \{z_k\}$
نسل بعد

شکل ۱۶-۵ الگوریتم مقابله‌ای زیست‌جغرافی-محور (OBBO) با اندازه جمعیت N. مجموعه $\{x_k\}$ مجموعه تمام ذره‌های موجود در جمعیت بوده به‌صورتی که x_k، kاُمین ذره بوده و $x_k(s)$ نیز sاُمین ویژگی x_k می‌باشد. به همین ترتیب، $\{z_k\}$ جمعیتی موقتی از ذرات بوده به‌صورتی که z_k، kاُمین ذره‌ی جمعیت موقتی بوده و $z_k(s)$، sاُمین ویژگی z_k می‌باشد.

مثال ۱۶-۲

در این مثال یک تابع ۲۰ بعدی Griewank را بهینه‌سازی می‌نماییم. این تابع در ضمیمه ج.۶-۱ تعریف شده و در اینجا نیز جهت سادگی آورده شده است:

$$f(x) = 1 + \sum_{i=1}^{n} x_i^2/4000 - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) \qquad (۱۶-۱۹)$$

که در آن $x_i \in [-600, +600]$ است. مقدار مینیمم‌ساز x_i برای تمامی $i \in [1, n]$ برابر ۰ است. ما از BBO با اندازه جمعیت $N = 50$ و حد محاسبات تابعی برابر با ۲۵۰۰ استفاده می‌نماییم. اگر هر ذره‌ی BBO یک بار در هر نسل ارزیابی شود، ۵۰ نسل خواهیم داشت. ما از نرخ جهشی برابر با ۱٪ در بعد در ذره استفاده خواهیم نمود و پارامتر نخبه‌گرایی را برابر ۲ قرار خواهیم داد. ما همچنین OBL را مانند آنچه که در شکل ۱۶-۵ نشان داده شده است به الگوریتم BBO اضافه می‌نماییم. برای این کار J_r را برابر ۰,۲ در نظر گرفته و بدین ترتیب تعداد نسل‌ها به حدود ۴۱ یا ۴۲ کاهش می‌یابد. مقدار این کاهش به عدد اتفاقی که تولید نسل مخالف را بر عهده دارد، بستگی دارد. پس از ۲۰ شبیه‌سازی مونت کارلو میانگین کمترین هزینه‌های یافته شده توسط BBO و OBBO به قرار زیر است:

BBO: ۸,۸۵

OBBO بازتابی : ۹,۹=۶۹

OBBO کوشی: ۰,۰۵

فرا OBBO: ۱۱,۸۲

OBBO کوشی بازتابی: ۰,۰۳

معانی عبارت‌های بالا را می‌توان در شکل ۱۶-۴ مشاهده نمود. OBBO بازتابی به x_o، OBBO کوشی به x_{qo}، فرا OBBO به x_{so} و OBBO کوشی بازتابی به x_{qr} اشاره دارد. می‌توان دید که OBBO بازتابی و فرا OBBO هر دو عملکرد بدتری نسبت به BBO دارند. با این حال، OBBO کوشی و OBBO کوشی بازتابی به‌صورت خارق‌العاده‌ای بهتر از BBO عمل می‌کنند.

همان‌طور که از قضیه no-free-lunch می‌دانیم (ضمیمه ب را ببینید)، عملکرد عجیب OBBO کوشی و OBBO کوشی بازتابی در مثال ۱۶-۲ جادو نیست. دلیل این عملکرد برتر آن است که راه‌حل مسئله Griewank دقیقاً در مرکز دامنه آن واقع شده است. شکل ۱۶-۴ نشان می‌دهد که هر دو الگوریتم OBBO کوشی و OBBO کوشی بازتابی تمایل دارند ذرات را به سمت مرکز دامنه جستجو سوق دهند. OBBO بازتابی ذرات را در همان فاصله که از مرکز دارند نگه می‌دارد (اما در سمت مخالف دامنه جستجو) و به

همین دلیل عملکردی بدتر از BBO دارد. به بیان دیگر OBBO بازتابی در مسئله‌ی گرینوانک ذرات را نه بهتر کرده و نه بدتر و صرفاً به ارزیابی توابع می‌پردازد. فرا OBBO از این هم بدتر است. شکل ١٦-٤ نشان می‌دهد که فرا OBBO همواره می‌خواهد ذرات را از مرکز دامنه دور کند و این باعث بد بودن عملکرد آن در مسئله‌ی گرینوانک است. بنابراین نتایج مثال ١٦-٢ دقیقاً همان‌هایی هستند که با توجه به درکمان از OBL انتظار داشتیم در مسئله‌ی گرینوانک اتفاق بیافتد.

صدالبته که اگر می‌دانستیم راه‌حل در نزدیکی مرکز دامنه قرار گرفته است می‌توانستیم به جای OBL از هر روش دیگری برای گرایش دادن ذرات BBO به سمت مرکز دامنه استفاده نماییم. از این لحاظ، استفاده از OBL در مسئله‌ی گرینوانک یک جور تقلب به حساب می‌آید چرا که وابسته به این حقیقت است که راه‌حل مسئله‌ی گرینوانک در نزدیکی مرکز دامنه واقع شده است. به بیان دیگر، عملکرد آن به اطلاعات خاص مسئله وابسته است. این موضوع به‌صورت تنگاتنگی با قضیه no-free-lunch که در ضمیمه ب آورده شده است، در ارتباط است. یک مسئله که راه‌حل آن بتواند در هر کجای دامنه واقع شود می‌تواند معیار بهتری برای آزمایش OBL باشد. این موضوع اساس مثال بعدی است.

مثال ١٦-٣

در این مثال نیز به بهینه‌سازی تابع ٢٠ بعدی گرینوانک می‌پردازیم (ضمیمه ج.٦-١ را ببینید). در اینجا نیز از همان پارامترهایی که در مثال ١٦-٢ استفاده کردیم، بهره خواهیم برد. با این حال، این بار راه‌حل مسئله‌ی گرینوانک را به‌صورت اتفاقی شیفت می‌دهیم:

$$f(x) = 1 + \sum_{i=1}^{n} (x_i - r_i)^2/4000 - \prod_{i=1}^{n} \cos\left((x_i - r_i)/\sqrt{i}\right) \qquad (٢٠-١٦)$$

که در آن r_i عددی اتفاقی با توزیع یکنواخت بر روی دامنه جستجو $[-600, +600]$ می‌باشد. مقدار مینیمم‌کننده‌ی $f(x)$ برای $i \in [1, n]$ برابر $x_i^* = r_i$ می‌باشد. پس از ٢٠ شبیه‌سازی مونت کارلو که در آن برای هر بار شبیه‌سازی از مجموعه‌ای متفاوت از مقادیر $\{r_i\}$ استفاده می‌نماییم، میانگین کمترین مقادیر پیدا شده توسط BBO و OBBOها به قرار زیر است:

BBO: ١٠,٤

OBBO بازتابی: ١٤,١

OBBO کوشی: ١٣,٨

فرا OBBO: ١٣,٤

OBBO کوشی بازتابی: ۱۳,۹

می‌بینید که تمامی BBOهای مقابله–محور به طرز قابل توجهی بدتر از BBO استاندارد عمل می‌کنند. این به آن دلیل است که راه‌حل گرینوانک به‌صورت یکنواخت بر روی فضای جستجو توزیع شده است و به همین دلیل یک نقطه مخالف (مقابل) از احتمال بیشتری نسبت به نقطه استاندارد برای نزدیک بودن به راه‌حل بهینه برخودار نخواهد بود. در حقیقت، با افزایش تعداد نسل‌های BBO، نقطه مخالف از احتمال کمتری برای نزدیک بودن به راه‌حل بهینه برخوردار خواهند بود. این بدین دلیل است که با افزایش شماره‌ی نسل، ذرات BBO با استفاده از مکانیزم به اشتراک‌گذاری اطلاعاتشان به راه‌حل بهینه نزدیکتر می‌شوند. بنابراین، احتمال دور شدن ذرات مخالف از راه‌حل بهینه توسط تابع تقابل بیشتر می‌شود. استفاده از تقابل در این مورد نه تنها باعث از بین رفتن توان بر سر محاسبات توابع شده، بلکه این کار به‌صورتی کاملا غیرسازنده صورت می‌گیرد.

مثال ١٦–٣ نشان می‌دهد که پس از ملاحظات دقیق‌تر، نتایج هیجان‌انگیز مثال ١٦–٢ سراب به نظر می‌رسند. با این حال، همه چیز از دست‌رفته نیست. هنگامی که می‌خواهیم یک مسئله بهینه‌سازی دنیای واقعی را با استفاده از الگوریتم تکاملی حل نماییم، باید دامنه جستجو را تعریف نماییم. این کار از این جهت صورت می‌گیرد که مطمئن باشیم راه‌حل در درون دامنه جستجو قرار می‌گیرد. این بدین معنی است که ما معمولاً فضای جستجو را بزرگتر از آنچه که باید در نظر می‌گیریم. این بزرگی از آن جهت است که نمی‌دانیم راه‌حل دقیقاً در کجا واقع شده است. اما برای نمونه در مورد دو مثال قبلی، ممکن است حدس بزنیم که راه‌حل برای مثال در مرکز فضای جستجو واقع شده است. بنابراین، یک وضعیت واقع‌گرایانه‌تر نسبت به دو مثال ١٦–٢ و ١٦–٣ شاید آن است که راه‌حل مسئله گرینوانک را به‌گونه‌ای اتفاقی شیفت دهیم که به آن اجازه رسیدن به کران‌های دامنه را بدهد، اما تمایل این شیفت به‌گونه‌ای باشد که بخواهد راه‌حل را بیشتر در همان حوالی مرکز دامنه نگه دارد. حال به مثال بعد توجه کنید.

مثال ١٦–٤

در این مثال یک بار دیگر تابع ۲۰ بعدی گرینوانک را بهینه می‌نماییم. در اینجا نیز از همان پارامترهای مثال ١٦–٢ و ١٦–٣ استفاده می‌نماییم. با این حال، در اینجا راه‌حل مسئله گرینوانک را به‌صورت اتفاقی و به‌گونه‌ای شیفت می‌دهیم که راه‌حل تعبیری از یک بردار با توزیع نرمال بوده و انحراف از معیار هر عنصر از این بردار برابر ۲۰۰ باشد.

$$i \in [1,n] \text{ برای } \quad r_i \leftarrow 200N(0,1)$$
$$r_i \leftarrow \max(min(r_i, 600), -600)$$
$$f(x) = 1 + \sum_{i=1}^{n} (x_i - r_i)^2/4000 - \prod_{i=1}^{n} \cos\left((x_i - r_i)/\sqrt{i}\right) \qquad (۲۱-۱۶)$$

$N(0,1)$ یک عدد اتفاقی با توزیع نرمال، میانگین صفر و واریانس واحد می‌باشد. این بدین معنی است که $200N(0,1)$ دارای انحراف معیاری برابر ۲۰۰ خواهد بود. عملیات min/max در معادله (۱۶-۲۱) برای اطمینان از قرار گرفتن راه‌حل شیفت‌یافته مسئله گرینوانک در درون دامنه جستجو $[-600, +600]$، می‌باشند. پس از ۲۰ شبیه‌سازی مونت کارلو، که در آن از مقادیر متفاوت $\{r_i\}$ برای هر بار شبیه‌سازی استفاده نموده‌ایم، کمترین مقادیر هزینه پیدا شده توسط BBO و OBBOها به قرار زیرند:

BBO: ۹,۵

OBBO بازتابی: ۱۱,۲

OBBO کوشی: ۹,۹

فرا OBBO: ۱۱,۹

OBBO کوشی بازتابی: ۶,۰

عملکرد BBO و OBBO کوشی به لحاظ آماری یکی است در حالی که عملکرد OBBO بازتابی و فرا OBBO بدتر از عملکرد BBO می‌باشد. با این حال، OBBO کوشی بازتابی به طرز قابل توجهی بهتر از BBO است. این به دلیل آن است که OBBO کوشی بازتابی تمایل دارد ذرات را به سمت مرکز دامنه ببرد. شاید به نظر برسد که به دلیل مشابه OBBO کوشی نیز باید بهتر از BBO عمل کند. با این حال، OBBO کوشی ذرات را در حالی به سمت مرکز دامنه سوق می‌دهد که به‌صورت همزمان آن‌ها را از ذره فعلی x دور می‌سازد (شکل ۱۶-۴ را ببینید). این باعث افت عملکرد در نسل‌های بعدی که بیشتر ذرات دارای هزینه‌های کمتر هستند، می‌شود. OBBO کوشی بازتابی دارای عملکرد بهتری است چرا که نه تنها ذرات را به سمت مرکز دامنه سوق داده، بلکه تمایل دارد ذرات را در همان مکان اصلیشان در فضای جستجو نگه دارد. این موضوع پس از چند نسل اولیه بسیار مفید خواهد بود.

۱۶-۳ احتمالات تقابل

بخش ۱۶-۲ نشان داد که چگونه می‌توان از OBL در BBO جهت بهبود عملکرد آن استفاده نمود. این بخش احتمال نزدیکتر شدن به راه‌حل یک مسئله‌ی بهینه‌سازی هنگام استفاده از انواع مختلف مقابله را مورد

مطالعه قرار می‌دهد: تقابل بازتابی، تقابل کوشی و تقابل کوشی بازتابی. این بخش حاوی مطالب ریاضی بسیاری است. بنابراین، خوانندگانی که تنها به دنبال مطالب کاربردی هستند می‌توانند با خیال راحت از این بخش صرف نظر کرده و یا تنها نتایج آن را در انتهای این بخش و در جدول ۱۶–۱ مطالعه نمایند.

در این بخش فرضیات زیر را در نظر می‌گیریم.

۱. فرض می‌کنیم که فضای جستجو یک بعدی است. کاملاً واضح است که این فرض بسیار محدود کننده است، اما به هر حال یک نقطه‌ی شروع به حساب می‌آید. می‌توان این حالت یک بعدی را به حالت‌های دیگر با تعداد ابعاد بیشتر بسط داد.

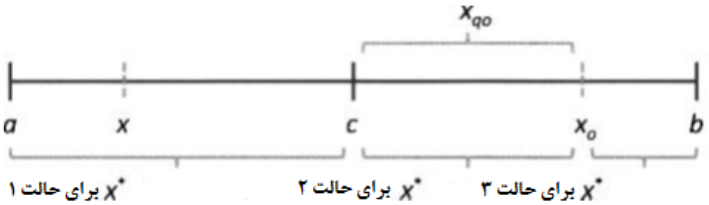
۲. فرض می‌کنیم که راه‌حل x^* مسئله بهینه‌سازی ناشناخته است و معادل عددی اتفاقی با توزیع یکنواخت بر روی دامنه‌ی x می‌باشد. این فرض بر پایه‌ی ادله‌ی ناکافی قرار دارد. بنابراین قاعده، در صورت عدم برخورداری از دانش قبلی، فرض بر آن است که تمامی اتفاق‌ها در فضای جستجو دارای احتمالات یکسانی هستند [دمبسکی و مارکس، ۲۰۰۹b]، [دمبسکی، و مارکس، ۲۰۰۹a].

فرض کنید ذره‌ای دلخواه مانند x در اختیار داریم. می‌توان بدون از دست رفتن کلیت فرض کرد که x در نیمه‌ی پایینی فضای جستجو قرار دارد. بیایید فرض کنیم که مخالف کوشی آن x_{qo} نسبت به مخالف آن x_o به راه‌حل بهینه x^* نزدیکتر است. شکل ۱۶–۶ ذره‌ی دلخواه x، مخالفش x_o و مخالف کوشی آن x_{qo} را نشان می‌دهد. ذره‌ی بهینه می‌تواند در یکی از سه ناحیه‌ی زیر قرار داشته باشد.

۱. حالت اول: $x^* \in [a, c]$

۲. حالت دوم: $x^* \in [c, x_o]$

۳. حالت سوم: $x^* \in [x_o, b]$



شکل ۱۶–۶ x یک ذره‌ی تکاملی بوده و x_o و x_{qo} به ترتیب مخالف و مخالف کوشی آن می‌باشند. راه‌حل بهینه x^* نیز دارای توزیع یکنواخت بر روی $[a, b]$ می‌باشد؛ بنابراین می‌تواند در هر یک از نواحی بالا قرار گیرد.

حالت اول

برای حالت اول کاملا واضح است که x_{qo} نسبت به x_o به راه‌حل بهینه x^* نزدیکتر است. بنابراین،

$$\Pr\left(|x_{qo}-x^*|<|x_o-x^*|\right)=1 \quad \text{برای حالت ۱} \qquad (۲۲-۱۶)$$

حالت دوم

برای حالت دوم، x^* و x_{qo} مستقل بوده و دارای توزیع یکنواخت بر روی $[c,x_o]$ می‌باشند. ما می‌توانیم از قضیه احتمال کل [میتزنماکر و اپفال[1]، ۲۰۰۵] و این حقیقت که $x_o-x^*>0$ است استفاده کرده و احتمال نزدیکتر بودن x_{qo} نسبت به x_o به x^* را محاسبه نماییم:

$$\begin{aligned}&\Pr\left(|x_{qo}-x^*|<|x_o-x^*|\right)\\&=\Pr\left(|x_{qo}-x^*|<x_0-x^*|x_{qo}-x^*<0\right)\Pr\left(x_{qo}-x^*<0\right)+\\&\Pr\left(|x_{qo}-x^*|<x_0-x^*|x_{qo}-x^*>0\right)\Pr\left(x_{qo}-x^*>0\right)\\&=\Pr\left(x_{qo}>2x^*-x_0|x_{qo}<x^*\right)\Pr\left(x_{qo}<x^*\right)+\\&\Pr\left(x_{qo}<x_0|x_{qo}>x^*\right)\Pr\left(x_{qo}x^*\right)\end{aligned} \qquad (۲۳-۱۶)$$

عبارت موجود در سمت راست معادله‌ی بالا را در نظر بگیرید. اولاً، از آنجا که x_{qo} و x^* هر دو دارای توزیع یکنواخت بر روی $[c,x_o]$ می‌باشند می‌توان دید که

$$\begin{aligned}&\Pr\left(x_{qo}<x^*\right)=\frac{1}{2}\\&\Pr\left(x_{qo}>x^*\right)=\frac{1}{2}\\&\Pr\left(x_{qo}<x_0|x_{qo}>x^*\right)=1\end{aligned} \qquad (۲۴-۱۶)$$

می‌توانیم از قضیه‌ی بیز برای نوشتن اولین عبارت در سمت راست معادله‌ی (۱۶-۲۳) به‌صورت زیر استفاده نماییم

$$\begin{aligned}&\Pr\left(x_{qo}>2x^*-x_0|x_{qo}<x^*\right)\Pr\left(x_{qo}<x^*\right)\\&=\Pr\left(x_{qo}>2x^*-x_0,x_{qo}<x^*\right)\\&=\Pr\left(2x^*-x_0<x_{qo}<x^*\right)\\&=\int_c^{x_o}\int_{x_{qo}}^{(x_{qo}+x_0)/2}f(x^*)f\left(x_{qo}\right)d\,x^*dx_{qo}\end{aligned} \qquad (۲۵-۱۶)$$

[1] Opfal

که در آن فرض بر آن است که x_{qo} و x^* مستقل بوده و به‌ترتیب دارای تابع چگالی احتمال $f(x_{qo})$ و $f(x^*)$ می‌باشند. با فرض توابع چگالی احتمال یکنواخت، انتگرال بالا را می‌توان به‌صورت زیر نوشت

$$\begin{aligned}&\Pr(x_{qo} > 2x^* - x_0 | x_{qo} < x^*)\Pr(x_{qo} < x^*)\\&= \int_c^{x_o}\int_{x_{qo}}^{\frac{x_{qo}+x_0}{2}} \frac{1}{(x_o-c)^2}\, d\,x^* dx_{qo}\\&= \int_c^{x_o} \frac{x_o - x_{qo}}{2(x_o-c)^2} dx_{qo}\\&= 1/4\end{aligned} \qquad (۱۶-۲۶)$$

با جایگزین نمودن معادلات (۱۶-۲۴) و (۱۶-۲۶) در معادله‌ی (۱۶-۲۳) به عبارت زیر خواهیم رسید

$$\text{برای حالت ۲ } \Pr(|x_{qo} - x^*| < |x_o - x^*) = 3/4 \qquad (۱۶-۲۷)$$

حالت سوم

برای این حالت، از شکل ۱۶-۶ کاملا واضح است که x_o نسبت به x_{qo} به x^* نزدیکتر است. بنابراین،

$$\text{برای حالت ۳ } \Pr(|x_{qo} - x^*| < |x_o - x^*|) = 0 \qquad (۱۶-۲۸)$$

نتایج نهایی

بگذارید از علامت $\mathcal{E}$ برای نشان دادن حالتی که در آن x_{qo} نسبت به x_o به x^* نزدیکتر است استفاده نماییم:

$$\mathcal{E} = \{|x_{qo} - x^*| < |x_o - x^*|\} \qquad (۱۶-۲۹)$$

حال می‌توانیم نتایج حالت‌های اول، دوم و سوم را با هم ترکیب نماییم:

$$\begin{aligned}\Pr(\mathcal{E}) = {} & \Pr(\mathcal{E}|x^* \in [a,c])\Pr(x^* \in [a,c]) + \\ & \Pr(\mathcal{E}|x^* \in [c,x_o])\Pr(x^* \in [c,x_o]) + \\ & \Pr(\mathcal{E}|x^* \in [x_o,b])\Pr(x^* \in [x_o,b])\\ = {} & (1)\left(\frac{1}{2}\right) + \left(\frac{3}{4}\right)\left(\frac{x_o-c}{b-a}\right) + 0\end{aligned} \qquad (۱۶-۳۰)$$

اگر x دارای توزیع یکنواخت بر روی نیمه‌ی پایینی فضای جستجو باشد، آنگاه x_o دارای توزیع یکنواخت بر روی نیمه‌ی بالایی فضای جستجو خواهد بود. بنابراین امید ریاضی x_o برابرست با

$$E(x_o) = (\mathrm{c} + \mathrm{b})/2 \qquad (۱۶-۳۱)$$

حال با محاسبه‌ی امید ریاضی معادله‌ی (۱۶-۳۰) مقدار زیر به دست خواهد آمد

$$E\left(\Pr\left(|x_{qo}-x^*|<|x_o-x^*|\right)\right)=\frac{1}{2}+\frac{3}{4}\frac{\frac{b-c}{2}}{b-a} \qquad (۳۲-۱۶)$$
$$=1/2+3/16=11/16$$

در استنتاج بالا فرض بر آن است که $x\in[a,c]$ اما این فرض کلیت نتایج را تحت تأثیر قرار نمی‌دهد بدین معنی که اگر فرض بر $x\in[c,b]$ بود نیز همین نتایج به دست می‌آمد. بنابراین، به قضیه‌ی زیر می‌رسیم.

قضیه ۱۶-۱. فرض کنید که یک ذره تکاملی مانند x و یک راه‌حل مانند x^* از یک مسئله‌ی بهینه‌سازی تک بعدی مستقل بوده و دارای توزیع یکنواخت بر روی فضای جستجو می‌باشند. در این صورت احتمال نزدیکتر بودن مخالف کوشی x به x^* نسبت به مخالف آن برابر $11/16$ خواهد بود.

این نتایج اولین بار در [ارگزار و همکاران، ۲۰۰۹]، [ارگزار، ۲۰۱۱] ارائه شدند. برخی نتایج دیگر نیز در این مقالات ارائه شده‌اند که خلاصه‌ی آن‌ها در جدول ۱۶-۱ آورده شده است. ردیف اول این جدول نشان می‌دهد که یک ذره‌ی تکاملی و مخالف آن هر دو دارای احتمال یکسانی برای نزدیک بودن به راه‌حل بهینه می‌باشند. این موضوع به دلیل تقارنی است که بین x و x_o وجود دارد.

اگرچه که جدول ۱۶-۱ به مسائل تک بعدی محدود است، اما تعمیم روش ارائه شده در این بخش به مسائل با ابعاد بالاتر سرراست بوده و به‌عنوان زمینه‌ای برای تحقیقات آتی رها می‌شود. برخی نتایج آزمایشی از ابعاد بالاتر در [ارگزار و همکاران، ۲۰۰۹] نشان داده شده است. بنابراین نتایج به نظر می‌رسد با افزایش تعداد ابعاد، احتمالات به یک مجانب نزدیک می‌شوند. همچنین مسئله‌ی ۱۶-۱۲ را ببینید.

توجه داشته باشید که ما x^* را به‌عنوان راه‌حلی دلخواه برای مسئله‌ی بهینه‌سازی در نظر گرفتیم. به همین ترتیب می‌توانستیم آن را به‌عنوان بدترین ذره‌ی جمعیت در نظر بگیریم. ویژگی کلیدی OBL آن است که پس از تولید شدن جمعیت مخالف، بهترین N ذره از میان N ذره‌ی جمعیت اصلی و N ذره‌ی جمعیت مخالف برای نسل بعد نگه داشته می‌شوند. تنها دلیل موفقیت OBL آن است که مخالف کوشی و مخالف کوشی بازتابی دارای احتمال بیشتری نسبت به یک ذره‌ی دلخواه مانند x برای نزدیک بودن به یک نقطه‌ی دلخواه در فضای جستجو می‌باشند.

جدول ۱۶-۱ احتمالات تک بعدی یک سری نقاط مخالف مشخص برای نزدیکتر بودن به راه‌حل بهینه، نسبت به نقاط دیگر.

رویداد	احتمال
$\lvert x_o - x^* \rvert < \lvert x - x^* \rvert$	$1/2$
$\lvert x_{qo} - x^* \rvert < \lvert x - x^* \rvert$	$9/16$
$\lvert x_{qr} - x^* \rvert < \lvert x - x^* \rvert$	$11/16$
$\lvert x_{qo} - x^* \rvert < \lvert x_o - x^* \rvert$	$11/16$
$\lvert x_{qr} - x^* \rvert < \lvert x_o - x^* \rvert$	$9/16$
$\lvert x_{qo} - x^* \rvert < \lvert x_{qr} - x^* \rvert$	$1/2$

در استدلال ما فرض شده است که x دارای توزیع یکنواخت بر روی فضای جستجو می‌باشد. ما انتظار داریم که با پیشرفت الگوریتم تکاملی به سمت نسل‌های آخر، بیشتر ذرات به راه‌حل بهینه نزدیکتر شوند. این بدین معنی است که x دیگر نمی‌تواند دارای توزیع یکنواخت باشد. این موضوع می‌تواند به این اشاره داشته باشد که OBL باید دارای تأثیر بیشتری در ابتدای فرایند جستجو باشد. بنابراین، هنگام پیاده‌سازی OBL می‌توان از نرخ جهش بیشتری در ابتدای فرایند جستجو (نسبت به انتهای آن) استفاده نمود. این مانند همان منطقی است که معمولاً در ذوب فلزات استفاده می‌نماییم (فصل ۹ را ببینید). ما همچنین معمولاً از استدلالی مشابه در فرایند جهش استفاده می‌نماییم، بدین ترتیب که در ابتدای فرایند جستجو از نرخ‌های جهش بیشتری نسبت به اواخر فرایند جستجو استفاده می‌نماییم [هاوپت و هاوپت، ۲۰۰٤، بخش ۵-۹].

۱۶-۴ نرخ جهش

این بخش به معرفی مفهوم نرخ جهش می‌پردازد، مفهومی که می‌تواند باعث بهبود عملکرد OBL شود. این ایده از درک این موضوع که OBL به منابع محاسباتی نیاز دارد، نشئت می‌گیرد. هر ذره‌ی مخالفی که تولید می‌شود نیاز به یک محاسبه‌ی اضافی تابع برازندگی دارد و محاسبه تابع برازندگی می‌تواند در مسائل دنیای واقعی به لحاظ محاسباتی بسیار پرهزینه باشد (فصل ۲۱ را ببینید). ما نمی‌خواهیم راه‌حل‌های مخالف را در حین پیاده‌سازی الگوریتم تکاملی به صوررت دلخواه تولید نماییم بلکه تمایل داریم که این راه‌حل‌های مخالف را تنها در صورت اطمینان از به صرفه بودن آن‌ها تولید نماییم.

توجه داشته باشید که مخالف یک ذره با برازندگی بالا دارای احتمال کمتری جهت برازندگی نسبت به مخالف ذره‌ای با برازندگی کم است. بدین معنی که اگر ذره‌ای تکاملی به راه‌حل بهینه نزدیک باشد، تولید ذره‌ی مخالف آن به صرفه نخواهد بود. عکس این مطلب نیز صادق است، بدین ترتیب که اگر ذره‌ای تکاملی از راه‌حل بهینه دور باشد، تولید مخالف آن به صرفه خواهد بود. صدالبته که نمی‌توان فهمید که آیا یک ذره به راه‌حل بهینه نزدیک است یا در نقطه‌ای دور از آن قرار گرفته است. اما از مقادیر نسبی برازندگی هر ذره

در جمعیت تکاملی خبر داریم. بنابراین شاید بهتر باشد OBL به‌گونه‌ای پیاده‌سازی شود که احتمال تولید مخالف یک ذره تابعی از مقدار برازندگی آن ذره باشد. منطق OBBO در شکل ۱۶–۵ می‌تواند با چیزی مانند آنچه که در شکل ۱۶–۷ نشان داده شده است، جایگزین شود.

پارامتر $\alpha \geq 0$ در شکل ۱۶–۷ فشار تولید ذرات مخالف را کنترل می‌کند. با توجه به اینکه μ_k با برازندگی z_k متناسب است، می‌توان دید که منطق تقابل برازندگی–محور احتمال تولید مخالف ذرات با برازندگی کمتر را، بیشتر می‌کند. مقدار کم α باعث تولید تعداد زیادی ذرات مخالف می‌شود. به‌صورت حدی وقتی α به سمت صفر میل می‌کند، منطق تقابل برازندگی–محور معادل تقابل استاندارد از شکل ۱۶–۵ خواهد بود و برای هر ذره تکاملی یک مخالف تولید می‌گردد. با بزرگتر شدن α ذرات مخالف کمتری تولید خواهد شد. با $\alpha \to \infty$، هیچ ذره‌ی مخالفی تولید نشده و الگوریتم OBBO به یک BBO استاندارد بدل خواهد شد. تولید ذرات مخالف یک ریسک است، چرا که به دلیل محاسبه‌ی تابع برازندگی هر یک از ذرات مخالف، تلاش محاسباتی بیشتری لازم خواهد بود. آیا این کار به صرفه است؟ پارامتر α تعادل مورد نیاز را فراهم می‌آورد.

$\alpha =$ فشار تقابل $\in [0,1]$
$r_1 \leftarrow U[0,1]$
اگر $r_1 < J_r$ آنگاه
$m = 0$
برای هر ذره‌ی z_k
$r_2 \leftarrow U[0,1]$
اگر $r_2 > \alpha\mu_k$ آنگاه
$m \leftarrow m + 1$
مخالف $\bar{z}_m \leftarrow z_k$
پایان اگر
بهترین N ذره از $\{z_k\} \leftarrow \{z_k\} \cup \{\bar{z}_m\}$
پایان اگر

شکل ۱۶–۷ منطق تقابل برازندگی–محور. این منطق می‌تواند جایگزین منطق استاندارد OBBO از شکل ۱۶–۵ شود.

یک راه دیگر برای پیاده‌سازی تقابل برازندگی–محور تولید ذرات مخالف تنها برای جزیی از ذرات با برازندگی کم می‌باشد. این جزء با ρ مشخص می‌گردد. این ایده بسیار شبیه ایده‌ی نشان داده شده در بالا

می‌باشد ولی بسیار قاطع‌تر بوده و می‌تواند مانند شکل ۱۶–۸ پیاده‌سازی شود. توجه داشته باشید که در این شکل، $\rho \in [0,1]$ می‌باشد.

ایده‌ی ارائه شده در این بخش تلاشی است برای هوشمندتر و مؤثرتر ساختن OBL. محققین خلاق می‌توانند از ایده‌های دیگری برای بهتر ساختن OBL استفاده نمایند. ما همچنین می‌توانیم از ایده‌های موجود در جهش هدایت شده در تحقیقات الگوریتم تکاملی کلی برای بهبود OBL استفاده نماییم [ژانگ و همکاران، ۲۰۰۵]. ما منطق نرخ جهش را در مثالی در بخش بعد نشان خواهیم داد.

$\rho =$ نرخ پرش $\in [0,1]$
$r \leftarrow U[0,1]$
اگر $r < J_r$ آنگاه
$m = 0$
برای هر ذره‌ی z_k
اگر z_k در کسر ρ از کم برازنده‌ترین ذرات جمعیت واقع شد آنگاه
$m \leftarrow m + 1$
$\bar{z}_m \leftarrow$ مخالف z_k
پایان اگر
ذره‌ی بعد
$\{z_k\} \leftarrow$ بهترین N ذره از $\{z_k\} \cup \{\bar{z}_m\}$
پایان اگر

شکل ۱۶–۸ منطق تقابل نسبی برازندگی–محور. این منطق می‌تواند جایگزین منطق OBBO از شکل ۱۶–۵ شود. اگر $\rho = 1$ باشد، این منطق به منطق تقابل استاندارد از شکل ۱۶–۵ بدل خواهد شد.

۱۶–۵ بهینه‌سازی ترکیبی مقابله‌ای

این بخش OBL را به مسائل بهینه‌سازی ترکیبی بسط می‌دهد. مطمئناً اگر بخواهیم OBL را به مسائل ترکیبی بسط دهیم، باید در تعریف تقابل که در بخش ۱۶–۱ ارائه شده است، تجدید نظر نماییم. کارهای اولیه در این زمینه در [ارگزار و سایمون، ۲۰۱۱] ارائه شده است.

یک مسئله ترکیبی مسئله‌ای است که در آن می‌خواهیم بهترین راه برای مرتب‌سازی یک مجموعه از گره‌ها را پیدا کنیم. مسئله فروشنده دوره‌گرد (TSP) یک مثال بسیار خوب از یک مسئله ترکیبی می‌باشد (بخش ۲–۵ و فصل ۱۸ را ببینید). یک TSP می‌تواند یک مسئله مسیر–باز یا مسیر–بسته باشد. یک مسئله مسیر–

بسته مسئله‌ای است که در آن راه‌حل، یک مسیر بسته ایجاد می‌نماید. مسیر بسته نیز مسیری است که از یک شهر شروع شده و به همان شهر ختم می‌شود. مسئله‌ی مسیر–باز نیز مسئله‌ای است که در آن راه‌حل از هر مسیر دقیقاً یکبار می‌گذرد و بدین ترتیب شهر ابتدایی و شهر انتهایی متفاوت خواهد بود. در این بخش مسائل مسیر–باز را در نظر می‌گیریم.

پیش از ارائه تعریف تقابل در یک الگوریتم تکاملی ترکیبی، ابتدا مثالی ساده را جهت معرفی برخی تعاریف ارائه می‌دهیم. فرض کنید می‌خواهیم یک TSP چهار شهری را با استفاده از یک الگوریتم تکاملی حل نماییم. این شهرها عبارتند از A، B، C و D. یکی از راه‌حل‌های نامزد عبارتست از:

$$A \to B \to C \to D \qquad (۳۳–۱۶)$$

۱. یک *ساق*، سفرِ بین دو شهر مجاور است. می‌توان دید که معادله‌ی (۱۶–۳۳)، دارای سه ساق است: $A \to B$، $B \to C$ و $C \to D$.

۲. *همجواری*، تعداد ساق‌هایی است که بین دو شهر قرار دارد. در معادله‌ی (۱۶–۳۳) همجواری دو شهر A و B برابر یک، همجواری دو شهر A و C برابر دو و همجواری دو شهر A و D برابر ۳ می‌باشد.

۳. همجواری کل یک مسیر به‌صورت مجموع همجواری‌های موجود بین هر دو شهر مجاور تعریف می‌گردد. در معادله‌ی (۱۶–۳۳) همجواری کل برابر سه است چرا که $A \to B$، $B \to C$ و $C \to D$ هر یک دارای همجاوری برابر یک می‌باشند. اگر تعداد شهرهای در مسیر برابر N باشد، همجواری کل یک مسیر همواره برابر $N - 1$ خواهد بود.

٤. *همجواری نسبی* مسیر که با β نشان داده می‌شود، به‌صورت مجموع همجواری‌های موجود بین هر دو شهر مجاور تعریف می‌شود به‌طوری که همجواری‌ها از یک مسیر دیگر مانند α گرفته شده باشد. برای مثال، فرض کنید مسیرهای زیر را در اختیار داریم:

$$\begin{aligned} \alpha &: \ D \to C \to A \to B \\ \beta &: \ B \to D \to A \to C \end{aligned} \qquad (۳٤–۱۶)$$

همجواری β نسبت به α برابر شش است. β دارای سه ساق است: ساق اول $B \to D$ است که این دو شهر در α دارای همجواری برابر سه می‌باشند. ساق دوم $D \to A$ است که این دو شهر در α دارای همجواری دو بوده و در نهایت ساق سوم $A \to C$ است که این دو شهر دارای همجواری یک در α هستند.

یک راه برای تعریف مخالف مسیر α آن است که مسیری مانند β بیابیم به‌طوری که همجواری نسبی دارای بیشترین مقدار باشد. این کار یک کار حسی است و از این حقیقت ناشی می‌شود که همجواری α

نسبت به α برابر $N-1$ بوده و دارای کمترین مقدار می‌باشد. با استفاده از این تعریف، مخالف معادله‌ی (۱۶-۳۳) برابرست با:

$$C \rightarrow A \rightarrow D \rightarrow B \qquad (۱۶-۳۵)$$

این مسیر نسبت به مسیر معادله‌ی (۱۶-۳۳) دارای همجواری نسبی ۷ (بیشترین همجواری ممکن) می‌باشد.

با این حال، پیدا کردن مسیری که مقدار همجواری نسبی را ماکزیمم نماید خود یک مسئله‌ی بهینه‌سازی ترکیبی است. این بدین معنی است که اگر بخواهیم TSP را با استفاده از OBL حل نماییم، باید یک مسئله‌ی ترکیبی را حل نماییم که خود شامل چند مسئله‌ی ترکیبی در هر یک از نسل‌هایش می‌شود. بنابراین، مخالف greedy یک ذره‌ی ترکیبی را معرفی می‌نماییم. مخالف greedy، شهر اولیه را بدون تغییر نگه می‌دارد و سپس شهری را که دارای بیشتری همجواری نسبی است به‌عنوان شهر دوم در نظر می‌گیرد. پس از آن نیز شهر سوم جدید را به‌گونه‌ای انتخاب می‌نماییم که دارای بیشترین همجواری نسبی از شهر دوم جدید باشد. این کار تا آنجا تکرار می‌شود که مسیر greedy تکمیل گردد. این فرایند در شکل ۱۶-۹ نشان داده شده است.

با استفاده از شکل ۱۶-۹، مخالف greedy مسیر معادله‌ی (۱۶-۳۳) به‌صورت زیر به دست می‌آید:

$$A \rightarrow D \rightarrow B \rightarrow C \qquad (۱۶-۳۶)$$

این مسیر دارای همجواری نسبی شش نسبت به مسیر معادله‌ی (۱۶-۳۳) می‌باشد، که این مقدار از مخالف نشان داده شده در معادله‌ی (۱۶-۳۵) یک واحد کمتر می‌باشد. شکل ۱۶-۱۰ راهی ساده‌تر را برای پیاده‌سازی مخالف greedy نشان می‌دهد. این الگوریتم‌ها هیچ کدام مخالف راه‌حل نامزد TSP را به دست نمی‌دهند اما تلاشی هستند برای به دست آوردن بهترین مخالف ممکن با صرف تلاش محاسباتی کم.

راه‌حل نامزد $= \alpha = \{\alpha_1, \alpha_2, \dots, \alpha_N\}$

مجاورت میان گره‌های α_i و α_j $= p(\alpha_i, \alpha_j) = |i-j|$

$\beta_1 \leftarrow \alpha_1$

$\beta \leftarrow \{\beta_1\}$

برای $k = 2$ تا N

$\beta_k \leftarrow arg\max_a p(\alpha_{k-1}, a) : a \notin \beta$

$\beta \leftarrow \beta \cup \beta_k$

بعدی k

شکل ۱۶-۹ شبه کد بالا الگوریتمی برای یافتن مخالف greedy مسیر α را نشان می‌دهد. در این الگوریتم N برابر تعداد گره‌ها در هر راه‌حل نامزد است.

راه‌حل نامزد $= \{\alpha_1, \alpha_2, \dots, \alpha_N\} = \alpha$
برای $k = 1$ تا N
اگر k فرد است
$m \leftarrow (k+1)/2$
در غیر این صورت
$m \leftarrow N + 1 - k/2$
پایان اگر
$\beta_k \leftarrow \alpha_m$
k بعدی

شکل ۱۶–۱۰ شبه کد بالا الگوریتمی ساده برای یافتن مخالف greedy مسیر α را نشان می‌دهد. در این الگوریتم N برابر تعداد گره‌ها در هر راه‌حل نامزد است. این الگوریتم معادل الگوریتم شکل ۱۶–۹، اما قدری ساده‌تر است.

مثال ۱۶–۵

در این مثال به بررسی کاربرد OBL در TSP می‌پردازیم. در این مثال ما از برش inverover و همچنین از الگویتم BBO برای انتخاب جمعیت مهاجرت کننده استفاده می‌نماییم. همچنین برای تابع محک از تابع Ulysses16 TSP که شامل ۱۶ شهر می‌شود (ضمیمه ج.۶ را ببینید)، استفاده کرده و تعداد محاسبات تابع برازندگی را برابر ۱۰۰۰۰ در نظر می‌گیریم. به یاد آورید که یک مسئله‌ی TSP با ۱۶ شهر دارای $16!/2 \approx 10^{13}$ راه‌حل ممکن است. جدول ۱۶–۲ مقدار میانگین و انحراف استاندارد کوتاه‌ترین مسیر یافت شده توسط ترکیبات متفاوت BBO/OBL را پس از ۴۰ شبیه‌سازی مونت کارلو، نشان می‌دهد. نتایج حاکی از آن هستند که با افزایش نرخ جهش J_r و نسبت جهش ρ، عملکرد بهبود می‌یابد (برای تعریف نسبت جهش ρ به شکل ۱۶–۸ مراجعه کنید). با این که در جدول نشان نداده‌ایم، اگر J_r و ρ زیادی افزایش یابند، عملکرد افت پیدا می‌کند.

جدول ۱۶–۲ مثال ۱۶–۵: نتایج بهینه‌سازی زیست‌جغرافی–محور تقابلی برای حل Ulysses16 TSP. نتایج مقدار میانگین و انحراف استاندارد بهترین راه‌حل را بر روی ۴۰ شبیه‌سازی مونت کارلو نشان می‌دهد. $J_r = 0$ معادل BBO استاندارد، بدون OBL می‌باشد. در کل، با افزایش نرخ جهش J_r و نسبت جهش ρ، عملکرد بهبود می‌یابد.

	$\rho = 0.1$	$\rho = 0.2$	$\rho = 0.3$	$\rho = 0.4$
$J_r = 0.0$	7266 ± 353	7266 ± 353	7266 ± 353	7266 ± 353
$J_r = 0.1$	7153 ± 289	7284 ± 244	7122 ± 296	7127 ± 270
$J_r = 0.2$	7160 ± 297	7100 ± 324	7047 ± 251	6910 ± 315
$J_r = 0.3$	7180 ± 267	6976 ± 336	6945 ± 270	6869 ± 319
$J_r = 0.4$	7127 ± 201	7005 ± 326	6910 ± 265	6776 ± 207

۱۶–۶ یادگیری دوگانه

یادگیری مقابله–محور مشابه یادگیری دوگانه است که اولین بار در دهه ۱۹۹۰ در [کولارد[1] و آوراند[2]، ۱۹۹٤]، [کولارد و گاسپار[3]، ۱۹۹٦] ارائه شد و سپس در اوایل دهه ۲۰۰۰ در [یانگ، ۲۰۰۳a] و [یانگ، ۲۰۰۳b] دوباره کشف شد. سپس، مقاله‌ی [یانگ و یاوو، ۲۰۰۵] پیشنهاد گرفتن دوگان بدترین ذره موجود در جمعیت را مطرح نمود. وارد نمودن ایده‌های یادگیری دوگانه در الگوریتم OBBO از شکل ۱٦–۵، منطق دوگانی شکل ۱٦–۱۱ را نتیجه می‌دهد. توجه داشته باشید که شکل ۱٦–۱۱ می‌تواند جایگزین بخش "opposition logic" از شکل ۱٦–۵ شود.

{بدترین ذرات در جمعیت N_d} ← {w_k}
از {w_k} برای ایجاد جمعیتی از N_d مخالف استفاده کن: {$\overline{w}_k$}
برای $i = 1$ تا N_d
  اگر $\overline{w}_k$ بهتر از w_k بود آنگاه w_k را با $\overline{w}_k$ جایگزین کن
i بعدی

شکل ۱٦–۱۱ شبه کد بالا منطق دوگانی را به تصویر می‌کشد. این منطق می‌تواند جایگزین "منطق مقابله" از شکل ۱٦–۵ شود. N_d تعداد دوگان‌هایی بوده که می‌توانیم در یک نسل تولید کنیم.

تعداد دوگان‌ها که با N_d مشخص می‌شوند را می‌توان برای به دست آوردن بهترین عملکرد الگوریتم تکاملی، تعدیل نمود. [یانگ و یاوو، ۲۰۰۵] استفاده از شِمای تعدیل زیر را که در هر نسل اجرا کرده و می‌توانیم به انتهای منطق دوگانی از شکل ۱٦–۱۱ اضافه نماییم، پیشنهاد می‌نماید:

$$\begin{aligned} N_v &\leftarrow |\overline{w}_k : f(\overline{w}_k) > f(w_k)| \\ s &\leftarrow (\delta N_d - N_v)/N \\ N_d &\leftarrow \beta^s N_d \\ N_d &\leftarrow \max(N_d, N_{d,min}) \\ N_d &\leftarrow \min(N_d, N_{d,max}) \end{aligned} \qquad (۳۷–۱٦)$$

در معادله‌ی (۱٦–۳۷)، $f(.)$ تابع برازندگی است، بنابراین، مقدار بزرگتر از $f(.)$ نشانگر ذره‌ای با عملکرد و برازندگی بهتر است. N_v نشانگر تعداد دوگان‌های معتبر از نسل قبل می‌باشد. این مقدار با مقدار $\overline{w}_k$، که تعداد دوگان‌هایی را که از ذرات اصلی بهتر بودند را نشان می‌دهد، برابر است. پارامتر $\delta \in (0,1)$ یک مرز

[1] Collard
[2] Aurand
[3] Gaspar

تصمیم است. اگر نسبت ذرات معتبر از δ بیشتر باشد، آنگاه می‌خواهیم دوگان‌های بیشتری را در نسل بعد تولید نماییم؛ اما اگر نسبت دوگان‌های معتبر از δ کمتر باشد، آنگاه می‌خواهیم تعداد دوگان‌های کمتری را در نسل بعد تولید نماییم. پارامتر $\beta \in (0,1)$ ثابتی است که سرعت اقتباس را کنترل می‌کند. $N_{d.min}$ و $N_{d.max}$ به ترتیب کمترین و بیشترین مقادیر مجاز N_d می‌باشند. مقادیر زیر برای ثابت‌های موجود در معادله‌ی (۱۶-۳۷) پیشنهاد می‌شوند [یانگ و یاوو، ۲۰۰۵]:

$$\begin{aligned} &N_d = 0.5N \quad \text{آغازین} \\ &\delta = 0.9 \\ &\beta = 0.5 \\ &N_{d,min} = 1 \\ &N_{d,max} = 0.5N \end{aligned} \qquad (۱۶-۳۸)$$

که در آن N اندازه جمعیت است. یادگیری دوگانه را می‌توان به PBIL برای حل مسائل بهینه‌سازی پویا نیز بسط داد [یانگ و یاوو، ۲۰۰۵]، [یانگ و یاوو، ۲۰۰۸b]. در PBIL، یک بردار احتمال دوگانی p_d متقارن بردار p است به‌صورتی که احتمال $p_d = p - 1$ برابر ۵۰٪ است.

۱۶-۷ نتیجه‌گیری

یادگیری مقابله-محور (OBL) یک ایده‌ی کاملاً نو و جدید در زمینه‌ی بهینه‌سازی است و جای زیادی برای تعمیم و گسترش دارد. OBL انطباقی یکی از زمینه‌های جالب برای تحقیق و کار بیشتر است. انطباق را می‌توان به چند روش پیاده‌سازی نمود. برای مثال، از آنجا که جمعیت الگوریتم تکاملی با گذشت نسل‌ها به سمت راه‌حل‌های خوب همگرا می‌شود، شاید بهتر باشد OBL را بیشتر در مراحل اولیه و کمتر در مراحل آخر الگوریتم تکاملی پیاده‌سازی نماییم. این کار را می‌توان با قرار دادن نرخ جهش J_r و نسبت جهش ρ به‌صورت تابعی کاهشی از شماره‌ی نسل، انجام داد. همچنین، ما در این فصل درجه‌ی تقابل را به ۰ یا ۱ محدود کرده‌ایم. می‌توان OBL انطباقی را با کاهش احتمالاتی درجه‌ی تقابل با شماره‌ی نسل، پیاده‌سازی نمود.

راه‌های دیگر پیاده‌سازی انطباق در یک الگوریتم OBL شامل عوض کردن نوع تقابل با افزایش شماره‌ی نسل و یا بر اساس برازندگی ذره می‌شود. ذرات با برازندگی کم باید بیشتر از ذرات با بزارندگی زیاد مورد تغییرات شدید تقابلی واقع شوند. بنابراین شاید باید فراتقابل برای ذرات با برازندگی کم حفظ شود.

با اینکه مدل‌سازی ریاضی زیادی برای OBL با استفاده از نظریه احتمال انجام شده است، اما OBL تا به حال به‌عنوان یک الگوریتم بهینه‌سازی مدل‌سازی ریاضی نشده است. زمینه‌های مهم برای تحقیقات آتی در زمینه OBL شامل اقتباس مدل‌های ریاضی الگوریتم تکاملی می‌شود (فصل ٤ و بخش ٧-٦ را ببینید).

از دیگر زمینه‌ها برای انجام تحقیقات می‌توان به کاوش رابطه‌ی میان OBL و تکامل از طریق جستجو برای بدعت [لهمن و استنلی، ۲۰۱۱] اشاره نمود. همچنین، از آنجا که OBL بر پایه‌ی تکامل اجتماعی قرار دارد، می‌توان مدل‌های فرهنگی بیشتری را در OBL ترکیب نمود (فصل ۱۵ را ببینید). مطالب آموزشی بیشتر در زمینه‌ی OBL را می‌توانید در [تیزهوش، ۲۰۰۵] و [تیزهوش و همکاران، ۲۰۰۸] بیابید.

مسائل

تمارین نوشتاری

١٦-١ معادله‌ی (١٦-٤) مخالف modulo را تعریف می‌نماید. یک تعریف معادل ارائه دهید که در آن از تابع modulo استفاده نشده باشد.

١٦-٢ مثالی از یک دامنه‌ی دو بعدی ارائه دهید که در آن مخالف یک نقطه مانند x که در دامنه قرار دارد، بتواند خارج از دامنه واقع شود.

١٦-٣ نقطه‌ی $(x,y) = (2,2)$ را در نظر بگیرید. دامنه‌ی x برابر $[1,5]$ بوده و دامنه‌ی y برابر $[1,7]$ می‌باشد. مخالف، مخالف کوشی، فرامخالف و مخالف کوشی بازتابی این نقطه را به دست آورید.

١٦-٤ نقطه‌ی $(x,y) = (2,2)$ را در نظر بگیرید. دامنه‌ی x برابر $[1,5]$ بوده و دامنه‌ی y برابر $[1,7]$ می‌باشد. نقطه‌ی $(2,5)$ چه نوع مخالفی برای این نقطه است؟

١٦-٥ توضیح دهید که چگونه می‌توان الگوریتم OBBO از شکل ١٦-٥ را اصلاح نمود تا شامل نرخ جهش انطباقی شود.

١٦-٦ چه تضادی میان فرض مثال ١٦-٤ و فرض دوم از بخش ١٦-٣ وجود دارد؟ کدام فرض منطقی‌تر به نظر می‌رسد؟

١٦-٧ فرض کنید نرخ مهاجرت از شهر μ_k در الگوریتم BBO از شکل ١٦-٧ یک متغیر اتفاقی با توزیع یکنواخت بر روی $[0,1]$ باشد.

الف) احتمال تولید ذره‌ی مخالف برای یک ذره‌ی اتفاقی مانند z_k چه قدر خواهد بود؟

ب) آیا مقدار احتمالی که شما به دست آورده‌اید با $\alpha \to 0$ و $\alpha \to \infty$، درکی حسی به دست می‌دهد؟

۱۶–۸ در شکل‌های ۱۶–۹ و ۱۶–۱۰، به‌صورت دلخواه شهر آغازین را در مخالف greedy مسیر α، که با β نمایش داده‌ایم، برابر با شهر آغازین α در نظر گرفته‌ایم. با این حال، همجواری β نسبت به α به این شهر آغازین بستگی دارد. حال مسیر $\alpha = \{A \to B \to C \to\to D \to E\}$ را در نظر بگیرید.

الف) اگر شهر آغازین برابر A باشد، مخالف greedy این مسیر (β) دارای چه میزان همجواری نسبت به α خواهد بود؟

ب) اگر شهر آغازین برابر B باشد، مخالف greedy این مسیر (β) دارای چه میزان همجواری نسبت به α خواهد بود؟

۱۶–۹ بیشترین و کمترین مقدار s در معادله‌ی (۱۶–۳۷) چه قدر است؟

۱۶–۱۰ فرض کنید از منطق دوگانی انطباقی معادله‌ی (۱۶–۳۷) و ثابت‌های پیشنهادی استفاده می‌نماییم. همچنین فرض کنید پس از نسل اول $N_v = 0.1N$ باشد. مقدار N_d در طول نسل دوم چه قدر خواهد بود؟

تمارین کامپیوتری

۱۶–۱۱ برنامه‌ای کامپیوتری بنویسید که برای مقادیر دلخواه a و b $(a < b)$ مقدار $x^* \sim U[a, b]$، مخالف استاندارد (x_o) و مخالف کوشی (x_{qo}) نقطه‌ی x را محاسبه نماید. بررسی کنید و ببینید کدام مخالف به x^* نزدیکتر است. برنامه را چندهزار بار اجرا کنید تا درستی قضیه ۱۶–۱ را متوجه شوید.

۱۶–۱۲ مثال ۱۶–۱۱ را با $n = 1$ تا 20 که در آن n ابعاد مسئله است، حل نمایید. احتمال اینکه $\left\|x_{qo} - x^*\right\|_2 < \|x_o - x^*\|_2$ باشد را به‌عنوان تابعی از n رسم کنید. نتایج به دست آمده را توضیح دهید.

۱۶–۱۳ مثال ۱۶–۴ را با منطق تقابل نسبی برازندگی–محور از شکل ۱۶–۸ تکرار کنید. از $\rho = 0.1, 0.5$ و 1.0 استفاده نمایید. میانگین کمترین هزینه‌ی پیدا شده توسط OBBO بعد از ۲۰ بار شبیه‌سازی مونت کارلو برای این مقادیر از ρ چه قدر است؟ نتایج به دست آمده را توضیح دهید.

فصل هفدهم

دیگر الگوریتم‌های تکاملی

این فصل به مرور برخی از الگوریتم‌های تکاملی که در فصل‌های پیش فرصت کافی برای بحث در مورد آن‌ها را نداشتیم، می‌پردازد. برخی از الگوریتم‌های این فصل در مرز تیره‌ی میان الگوریتم‌های تکاملی و غیرتکاملی واقع شده‌اند. بنابراین این فصل جای خوبی برای خلاصه‌سازی آن‌ها است. برخی دیگر از الگوریتم‌های موجود در این فصل کاملاً تکاملی اما جدید هستند و به همین دلیل میزان تأثیر آن‌ها بر آینده‌ی نظریه و عمل الگوریتم تکاملی مشخص نیست. هنگام تصمیم‌گیری در مورد الگوریتم‌هایی که باید در این کتاب مورد بحث واقع می‌شدند، کاملاً واضح بود که الگوریتم‌های همچون GAها، EP، ES و GP را می‌توان به دلیل اهمیت بنیادین و قدمتشان در بخش II قرار داد. تصمیم در مورد الگوریتم‌های بخش III کمی سخت‌تر بود. الگوریتم‌های تکاملی بحث شده در فصل‌های گذشته عقاید و گرایشات نویسنده در مورد اهمیت هر الگوریتم را بازتاب می‌دهند.

چندین الگوریتم تکاملی وجود دارند که در این کتاب دارای فصل مخصوص به خود نیستند اما باید حداقل کمی در مورد آن‌ها بحث کنیم. این کار هدف این فصل است. الگوریتم‌های ارائه شده در این فصل الزاماً کم اهمیت‌تر، کم اثرتر و یا کم استفاده‌تر از سایر الگوریتم‌های تکاملی معرفی شده در این کتاب نیستند. قرار گرفتن آن‌ها در این فصل تنها بیانگر تجربه‌ی محدود و تمایلات موضوعی نویسنده کتاب می‌باشد.

۱۷–۱ جستجوی ممنوعه

جستجوی ممنوعه (TS[1]) در [گلاور[2] و مک‌میلان، ۱۹۸۶] معرفی گردید. کالاها، سخنرانی‌ها و اعمال ممنوعه می‌توانند ریشه در فرهنگ، سیاست، اخلاق و یا مذهب داشته باشند. TS صرفاً یک روش جمعیت–محور برای بهینه‌سازی نیست، بلکه می‌توان آن را یک الگوریتم تکاملی در نظر گرفت چرا که ریشه در دنیای طبیعی داشته و یک فرایند جستجوی دوره‌ای (نسلی) است. ایده‌ی اصلی TS آن است که اگر ناحیه‌ای از فضای جستجو یکبار مورد کاوش قرار گرفته است، آنگاه این ناحیه ممنوعه خواهد بود و الگوریتم جستجو نباید دوباره به این ناحیه باز گردد. به‌طور مشابه، اگر یک استراتژی جستجوی خاص پیش از این در فرایند جستجو مورد استفاده واقع شده باشد، آنگاه این استراتژی ممنوعه خواهد بود و الگوریتم جستجو نباید دوباره از این استراتژی استفاده نماید.

شکل ۱۷–۱ یک الگوریتم بنیادین TS را نشان می‌دهد. در این الگوریتم، T لیستی از ویژگی‌های ممنوعه می‌باشد و x_0 بهترین راه‌حل نامرد حاضر است. هنگامی که از x_0 فرزندانی تولید می‌نماییم، به الگوریتم اجازه نمی‌دهیم که ویژگی‌های موجود در لیست T را در فرزند ایجاد نماید. هنگامی که یک ذره‌ی بهبود یافته مانند

---

[1] Tabu Search
[2] Glover

x' یافت می‌شود، ویژگی‌هایی را از x' به لیست T اضافه می‌نماییم. ما به‌صورت متناوب و احتمالاً بر اساس اینکه یک ویژگی چه مدت است در لیست T به سر می‌برد، ویژگی‌ها را از این لیست خارج می‌نماییم. این کار، تغییرات تدریجی ممنوعات را، به همان شکل که می‌توان در جوامع انسانی مشاهده نمود، شبیه‌سازی می‌کند. توجه داشته باشید که در شکل ۱۷-۱، آزمایش $x' \notin T$ (ویژگی‌های x')، به‌صورت عمدی مبهم رها شده است. جزییات این آزمایش به مسئله، روش تولید همسایگان x_0، عملکرد کاربر و دیگر جزییات بستگی دارد.

```
راه‌حل نامزد x0 را مقداردهی اولیه کن
T ← ∅
تا زمانی که شرایط توقف برآورده نشده است
    فرزندان ← ∅
    تا زمانی که |فرزندان| < M
        یک همسایه مانند x' برای x0 ایجاد کن
        اگر x' ∉ T (ویژگی‌های x')
            فرزندان ← فرزندان ∪ x'
        پایان اگر
    پایان حلقه
    x' ← argmin(f(x) : x ∈ فرزندان)
    اگر f(x') < f(x0)
        T ← T ∪ (ویژگی‌هایی از x')
        x0 ← x'
    پایان اگر
    ویژگی‌های قدیمی را از T حذف کن
نسل بعدی
```

شکل ۱۷-۱ طرح کلی یک الگوریتم جستجوی ممنوعه (TS) برای پیدا کردن مینیمم تابع $f(x)$. هر دوره شامل تولید M فرزند می‌شود. M پارامتری است که توسط کاربر تعیین می‌گردد.

می‌توان از تنوعات زیادی در الگوریتم شکل ۱۷-۱ استفاده نمود. برای مثال، می‌توان درجه‌ی ممنوعیت را تغییر داد. همچنین می‌توان به جای لیستی از ویژگی‌های ممنوعه، لیستی از استراتژی‌های ممنوعه را تشکیل داد. معمولاً از TS برای تقویت سایر الگوریتم‌های تکاملی استفاده می‌شود. هدف از طرح کلی خلاصه‌ی TS در این بخش، فراهم آوردن اطلاعات کافی برای خوانندگان جهت پیاده‌سازی یک الگوریتم TS ساده، یادگیری ایده‌های اساسی TS و یادگیری جزییات بیشتر از سایر منابع می‌باشد. برای مطالعه بیشتر در مورد TS می‌توانید به [ریوز، ۱۹۹۳، فصل ۳]، [گلاور و لاگونا[1]، ۱۹۹۸]، [گندروو[2]، ۲۰۰۳] و [گندروو و پتوین[3]، ۲۰۱۰] مراجعه نمایید.

۱۷-۲ الگوریتم تجمع مصنوعی ماهی‌ها

الگوریتم تجمع مصنوعی ماهی‌ها (AFSA[4])، که اولین بار در [لی و همکاران، ۲۰۰۳] معرفی گردید و برخی اوقات با نام الگوریتم مدرسه مصنوعی ماهی‌ها شناخته می‌شود، بر پایه‌ی رفتار تجمعی ماهی‌ها قرار دارد. مکان یک ماهی مصنوعی در فضای جستجو با x_i نشان داده می‌شود، که در آن $i \in [1, N]$ اندیس ماهی بوده و N تعداد ماهی‌ها در جمعیت می‌باشد. ما دامنه‌ی جستجو برای هر بعد را با $[l_k, u_k]$ نشان می‌دهیم به‌طوری که $k \in [1, n]$ بوده و n بعد فضای جستجو می‌باشد. ماهی‌ها دارای یک میدان دید هستند و می‌توانند ماهی‌های دیگری که در این میدان قرار دارند را ببینند. ماهی‌هایی که خارج از میدان دید یک ماهی باشند برای ماهی قابل مشاهده نخواهند بود. برد دید ماهی به‌صورت زیر تعریف می‌شود

$$v = \delta \max_k (u_k - l_k) \qquad (۱-۱۷)$$

که در آن δ یک پارامتر میزان‌سازی بوده و معمولاً در طول فرایند بهینه‌سازی به تدریج کاهش می‌یابد. [فراندز[5] و همکاران، ۲۰۰۹] نشان داده است که برای مسائل با ابعاد بین دو و چهار، δ باید بین ۱ تا ۱۰ واقع شود تا الگوریتم عملکرد خوبی داشته باشد. با این حال این مقدار باید برای مسائل با ابعاد بالاتر تنظیم شود. اندیس ماهی‌هایی که که در برد دید ماهی x_i قرار دارند به‌صورت زیر نمایش داده می‌شود:

$$V_i = \{j \neq i : \left|\left|x_i - x_j\right|\right|_2 \leq v \qquad (۲-۱۷)$$

[1] Laguna
[2] Gendreau
[3] Potvin
[4] Artificial Fish Swarm Algorithm
[5] Fernandes

گوییم یک ماهی در یک محیط پرازدحام واقع شده است اگر تعداد ماهی نسبتاً زیادی در برد دید آن واقع شده باشد:

$$\begin{aligned} &\frac{|V_i|}{N} > \theta \quad \Rightarrow \quad \text{برد دید } x_i \text{ شلوغ است} \\ &\frac{|V_i|}{N} \leq \theta \quad \Rightarrow \quad \text{برد دید } x_i \text{ شلوغ نیست} \end{aligned} \tag{۳-۱۷}$$

که در آن θ یک پارامتر میزان‌سازی است. در [فرناندز و همکاران، ۲۰۰۹] نشان داده شده است که $\theta \approx 1$ عملکرد خوبی را برای مسائل با ابعاد پایین نتیجه می‌دهد. یک ماهی در AFSA دارای پنج رفتار متمایز است: رفتار اتفاقی، رفتار تعقیبی، رفتار تجمعی، رفتار کاوشی و رفتار جست و خیزی (جهش).

۱-۲-۱۷ رفتار اتفاقی

گاهی اوقات ماهی‌ها دارای رفتاری اتفاقی هستند. بدین معنی که در جهت‌های اتفاقی در فضای جستجو حرکت می‌کنند. شکل ۱۷-۲ شبه کدی برای حرکت اتفاقی نشان می‌دهد. حرکت اتفاقی زمانی اتفاق می‌افتد که ماهی هیچ ماهی دیگری را در برد دید خود مشاهده نکرده و یا فرایند بهینه‌سازی دچار رکود شده باشد. رکود یعنی بهترین ذره در جمعیت نتوانسته باشد در طول m نسل گذشته بهبود قابل توجهی حاصل کند:

$$arg\min_x f_{t-m}(x) - arg\min_x f_t(x) < \eta \implies \text{رکود} \tag{۴-۱۷}$$

که در آن $f_t(x)$ مقدار تابع بهینه‌سازی برای ذره‌ی x در نسل tاُم، m یک پارامتر میزان‌سازی صحیح و مثبت و η یک پارامتر میزان‌سازی غیرمنفی است. در معادله‌ی (۱۷-۴) فرض می‌کنیم که مسئله‌ی بهینه‌سازی یک مسئله‌ی مینیمم‌سازی است. در [فرناندز و همکاران، ۲۰۰۹] نشان داده شده است که $m \approx 10n$ و $\eta \approx 10^{-4}$ عملکرد خوبی را برای مسائل محک با ابعاد کم به دست می‌دهد. n ابعاد مسئله است.

برای $k = 1$ تا n
$r \leftarrow U[0,1]$
اگر $r < 1/2$ آنگاه
$\rho \leftarrow U[0,1]$
$y_i(k) \leftarrow x_i(k) + \rho \min\big(v, u_k - x_i(k)\big)$
در غیر این صورت
$\rho \leftarrow U[0,1]$
$y_i(k) \leftarrow x_i(k) + \rho \min(v, x_i(k) - l_k)$

```
پایان اگر
بَعد بعد
```

شکل ۱۷-۲ رفتار اتفاقی در یک الگوریتم تجمعی مصنوعی ماهی‌ها. این کد حرکت اتفاقی ماهی x_i به محل جدید y_i، که در آن n ابعاد مسئله‌ی بهینه‌سازی است، را نشان می‌دهد. $U[0,1]$ عددی اتفاقی با توزیع یکنواخت بر روی $[0,1]$ بوده و v برد دید تعریف شده در معادله‌ی (۱۷-۱) می‌باشد.

۱۷-۲-۲ رفتار تعقیبی

گاهی یک ماهی به سمت یک ماهی دیگر در برد دید خود که در محلی با تمرکز غذایی بالا قرار دارد، حرکت می‌کند. رفتار تعقیبی یک ماهی به‌صورت زیر تعریف می‌شود:

$$j^* \leftarrow arg\min_j\{f(x_j) : j \in V_i\}$$
$$y_i \leftarrow x_i + r(x_{j^*} - x_i) \qquad (۵-۱۷)$$

که در آن $r \in [0,1]$ یک متغیر اتفاقی با توزیع یکنواخت و y_i محل جدید x_i می‌باشد. در اینجا نیز فرض می‌کنیم مسئله‌ی بهینه‌سازی یک مسئله‌ی مینیمم‌سازی است، بنابراین j^* اندیس ماهی است که در برد دید x_i قرار داشته و دارای بهترین عملکرد در مسئله‌ی بهینه‌سازی ما می‌باشد. اگر یک ماهی در برد دید هیچ ماهی دیگری قرار نداشته باشد آنگاه نمی‌تواند در رفتار تعقیبی شرکت کند. همچنین، x_i تنها در صورتی ماهی دیگری را تعقیب خواهد نمود که بهترین ماهی x_{j^*} در برد دیدش بوده و دارای عملکردی بهتر از خود x_i باشد.

۱۷-۲-۳ رفتار تجمعی

ماهی‌ها موجوداتی اجتماعی هستند و به همین دلیل گاهی تجمعاتی را تشکیل می‌دهند. در این مورد، یک ماهی مانند x_i به سمت مرکز ماهی‌هایی که در برد دیدش قرار دارند حرکت می‌کند. رفتار تجمعی برای ماهی x_i را می‌توان به‌صورت زیر توصیف نمود:

$$c_i \leftarrow \frac{1}{|V_i|}\sum_{j \in V_i} x_j$$
$$y_i \leftarrow x_i + r(c_i - x_i) \qquad (۶-۱۷)$$

که در آن $r \in [0,1]$ یک متغیر اتفاقی با توزیع یکنواخت و y_i محل جدید x_i می‌باشد. اگر یک ماهی در برد دید هیچ ماهی دیگری قرار نداشته باشد آنگاه نمی‌تواند در رفتار تجمعی شرکت کند. رفتار تجمعی تنها وقتی صورت می‌پذیرد که محدوده‌ی دیداری ماهی خالی و یا پرازدحام نبوده و $f(c_i)$ بهتر از $f(x_i)$ باشد.

۱۷–۲–۴ رفتار کاوشی

وقتی یک ماهی، ماهی دیگری را می‌بیند که غذای بیشتری دارد به سویش حرکت می‌کند. رفتار کاوشی برای ماهی x_i را می‌توان به‌صورت زیر تعریف نمود:

$$\begin{aligned} &j \leftarrow \text{عدد صحیح اتفاقی} \in V_i \\ &y_i \leftarrow x_i + r(x_j - x_i) \end{aligned} \qquad (۷–۱۷)$$

که در آن $r \in [0,1]$ یک متغیر اتفاقی با توزیع یکنواخت و y_i محل جدید x_i می‌باشد. رفتار کاوشی، حرکت یک ماهی به سمت یک ماهی اتفاقی دیگر است که در برد دیدش قرار دارد. اگر یک ماهی در برد دید هیچ ماهی دیگری قرار نداشته باشد آنگاه نمی‌تواند در رفتار کاوشی شرکت کند. رفتار کاوشی هنگامی اتفاق می‌افتد که محدوده‌ی دید ماهی پرازدحام بوده و یا محدوده‌ی دید ماهی پرازدحام نبوده اما $f(c_i)$ از معادله‌ی (۱۷–٦) بدتر از $f(x_i)$ بوده و یا محدوده‌ی دید ماهی پرازدحام نبوده و $f(x_{j^*})$ در معادله‌ی (۱۷–۵) بدتر از $f(x_i)$ می‌باشد.

۱۷–۲–۵ رفتار جست و خیزی

گاهی اوقات یک ماهی به‌صورت اتفاقی در فضای جستجو، خیزش می‌کند. این مشابه مواقعی است که یک ماهی از آب بیرون پریده و در یک مکان متفاوت فرود می‌آید. جست و خیز برای یک ماهی اتفاقی هنگامی اتفاق می‌افتد که فرایند بهینه‌سازی، مانند آنچه که در معادله‌ی (۱۷–٤) نشان داده شد، دچار رکود شده باشد. شکل ۱۷–۳ شبه کدی را برای رفتار جست و خیزی ماهی نشان می‌دهد.

برای $k = 1$ تا n
$r \leftarrow U[0,1]$
$\rho \leftarrow U[0,1]$
اگر $r < 1/2$ آنگاه
$x_i(k) \leftarrow x_i(k) + \rho(u_k - x_i(k))$

در غیر این صورت
$x_i(k) \leftarrow x_i(k) + \rho(x_i(k) - l_k)$
بُعد بعدی

شکل ۱۷-۳ رفتار افت و خیزی در یک الگوریتم تجمع مصنوعی ماهی‌ها. این کد خیزش ذره‌ی x_i در یک الگوریتم تجمع مصنوعی ماهی‌ها را، که در آن n ابعاد مسئله‌ی بهینه‌سازی بوده و $U[0,1]$ یک عدد اتفاقی با توزیع یکنواخت بر روی $[0,1]$ می‌باشد، نشان می‌دهد.

۱۷-۲-۶ خلاصه‌ای از الگوریتم تجمع مصنوعی ماهی‌ها

AFSA از روش انتخابی greedy استفاده می‌کند. بدین معنی که، پس از رفتارهای اتفاقی، تعقیبی، تجمعی و کاوشی، ماهی x_i تنها در صورت آنکه مکان جدید y_i بهتر از مکان قبلی باشد، به سمت آن حرکت می‌کند. شکل ۱۷-٤ شبه کدی را برای AFSA نشان می‌دهد که شباهت‌هایی به بهینه‌سازی تجمع ذرات دارد. محققان انواع و ترکیبات زیادی از AFSA را ارائه کرده‌اند [نشاط[1] و همکاران، ۲۰۱۲]. تحلیل و مدل‌سازی ریاضی AFSA، ترکیب نمودن ویژگی‌های ماهی‌های بیولوژیکی و روشن‌سازی رابطه‌ی میان AFSA و PSO از جمله زمینه‌های مفید و مهم برای تحقیقات آتی در زمینه‌ی AFSA می‌باشد.

اندازه جمعیت $= N$
جمعیتی ابتدایی از راه‌حل‌های نامزد را مقداردهی کن: $\{x_i\}$ برای $i \in [1, N]$
تا زمانی که شرایط توقف برآورده نشده است
برای هر ذره‌ی x_i
ماهی موجود در برد دید x_i را همان‌طور که در معادله‌ی (۱۷-۲) نشان داده شده است، پیدا کن
اگر $V_i = \emptyset$
حرکت اتفاقی مانند آنچه که در شکل $(17-2)$ نشان داده شده است $\leftarrow y_i$
در غیر این صورت و اگر برد دید x_i شلوغ است (معادله‌ی (۱۷-۳) را ببینید)
حرکت کاوشی مانند آنچه که در معادله‌ی $(17-7)$ نشان داده شده است $\leftarrow y_i$
در غیر این صورت
اگر $f(c_i) < f(x_i)$ (معادله‌ی (۱۷-٦) را ببینید) آنگاه

[1] Neshat

حرکت تجمعی مانند آنچه که در معادله‌ی (17 – 6) نشان داده شده است ← y_i

در غیر این صورت

حرکت کاوشی مانند آنچه که در معادله‌ی (17 – 7) نشان داده شده است ← y_i

پایان اگر

اگر $f(x_{j^*}) < f(x_i)$ (معادله‌ی (۱۷–۵) را ببینید) آنگاه

حرکت تعقیبی مانند آنچه که در معادله‌ی (17 – 5) نشان داده شده است ← y_i

در غیر این صورت

حرکت کاوشی مانند آنچه که در معادله‌ی (7 – 7) نشان داده شده است ← y_i

پایان اگر

$y_i \leftarrow argmin\{f(x_i), f(y_i)\}$

پایان اگر

ذره‌ی بعد

$x_i \leftarrow argmin\{f(x_i), f(y_i)\}$ برای $i \in [1, N]$

اگر الگوریتم مانند آنچه که در معادله‌ی (۱۷–٤) نشان داده شده است دچار رکود شده است

$j \leftarrow$ عدد صحیح اتفاقی $\in [1, N]$

حرکت جهشی مطابق شکل $x_j \leftarrow 3 - 17$

پایان اگر

نسل بعدی

شکل ۱۷–٤ یک الگوریتم تجمع مصنوعی ماهی‌ها (AFSA) برای مینیمم‌سازی تابع n بعدی $f(x)$ که در آن x_i، راه‌حل نامزد iاُم می‌باشد.

۱۷–۳ بهینه‌ساز جستجوی گروهی

بهینه‌ساز جستجوی گروهی (GSO[1])، که به نام بهینه‌سازی جستجوی گروهی نیز شناخته می‌شود، بر پایه‌ی رفتار تغذیه‌ای حیوانات می‌باشد [هی[2] و همکاران، ۲۰۰۹]. اساس این الگوریتم مانند اساس الگوریتم

[1] Group Search Optimizer
[2] He

تجمع ماهی (بخش ۱۷-۲) و بهینه‌سازی کاوش باکتریایی (بخش ۱۷-۶) می‌باشد، با این تفاوت که GSO بر پایه‌ی رفتار مشاهده شده از حیوانات زمینی است.

برخی حیوانات تلاش‌های خود را بر پیدا کردن غذا متمرکز می‌کنند. این حیوانات، حیوانات تولیدکننده[1] نام دارند. سایر حیوانات تلاش خود را بر دنبال کردن حیوانات دیگر برای بهره‌گیری از موفقیت آن‌ها در پیدا کردن غذا متمرکز می‌کنند. این نوع حیوانات، ملحق‌شونده[2] نامیده می‌شوند. GSO شامل نوع سومی از حیوانات تحت عنوان حیوانات تکاور[3] نیز می‌شود. این حیوانات به‌صورت اتفاقی در فضای جستجو برای پیدا کردن منابع پرسه می‌زنند. هر ذره در فضای جستجوی n بعدی دارای مکانی است که با x_i نمایش داده می‌شود. همچنین هر ذره دارای زاویه‌ی جهتی است که با $\phi_i = [\phi_{i,1} \dots \phi_{i,n-1}]$ نمایش داده می‌شود.

تولیدکنندگان

در GSO فرض بر آن است که تنها یک تولیدکننده در جمعیت وجود دارد. در هر نسل فرض بر آن است که ذره با کمترین هزینه، ذره‌ی تولیدکننده است. در هر نسل، تولیدکننده سه نقطه از محیط اطراف خود را برای پیدا کردن نقطه‌ای با تابع هزینه‌ی کمتر نسبت به محل فعلی خود، مورد پویش قرار می‌دهد. این عمل مانند جستجوی محلی است. اگر این عمل را با x_p نمایش دهیم، آنگاه سه نقطه عبارتند از

$$\begin{aligned} x_z &= x_p + r_1 l_{max} D(\phi_p) \\ x_r &= x_p + r_1 l_{max} D(\phi_p + r_2\, \theta_{max}/2) \\ x_l &= x_p + r_1 l_{max} D(\phi_p - r_2\, \theta_{max}/2) \end{aligned} \qquad (۸-۱۷)$$

که در آن r_1 یک متغیر اتفاقی با میانگین صفر، واریانس واحد و توزیع یکنواخت، $r_2 \in [0,1]$ متغیری اتفاقی با توزیع یکنواخت، ϕ_p زاویه‌ی جهت x_p، l_{max} یک پارامتر میزان‌سازی برای تعیین میزان دید تولیدکننده، θ_{max} یک پارامتر میزان‌سازی برای تعیین حد چرخش زاویه‌ی تولیدکننده و $D(.)$، مبدل مختصات قطبی به کارتزین است که به‌صورت زیر تعریف می‌گردد

$$\begin{aligned} &D(\phi_p) = [d_1 \dots d_n] \\ &d_1 = \prod_{q=1}^{n-1} cos\phi_{p,q} \\ &j \in [2, n-1] \text{ برای } d_j = sin\phi_{p,j-1} \prod_{q=j}^{n-1} cos\phi_{p,q} \\ &d_n = sin\phi_{p,n-1} \end{aligned} \qquad (۹-۱۷)$$

[1] Producer
[2] Scrounger
[3] Ranger

اگر تولیدکننده مقدار تابع هزینه‌ی بهتری در یکی از سه نقطه‌ی تعریف شده در معادله‌ی (۱۷–۸) بیابد، بلافاصله به آن نقطه می‌رود، در غیر این صورت، در مکان خود باقی مانده و مقدار زاویه جهت ϕ_p خود را به‌صورت اتفاقی به یک مقدار جدید تغییر می‌دهد. اگر تولیدکننده پس از a_{max} نسل نتواند نقطه‌ی بهتری را بیابد، مقدار زاویه جهت خود را به همان مقداری که a_{max} نسل گذشته بود، بر می‌گرداند. با این حال، دلیل تأثیر این آخرین استراتژی بر روی عملکرد بهینه‌سازی معلوم نیست و به نظر می‌آید که بتوان از آن صرف نظر کرد.

ملحق‌شوندگان

ملحق‌شوندگان معمولاً به سوی تولیدکننده حرکت می‌کنند. اما این حرکت یک حرکت مستقیم نیست و الگویی زیگزاگی دارد. این کار به آن‌ها اجازه می‌دهد ضمن حرکت به سوی تولید کننده به جستجوی مقادیر تابع هزینه‌ی کم بپردازند. حرکت یک ملحق‌شونده به‌صورت زیر مدل می‌شود

$$x_i \leftarrow x_i + r_3 o(x_p - x_i) \qquad (۱۷–۱۰)$$

که در آن r_3 یک متغیر اتفاقی برداری n بعدی بوده و هر یک از این ابعاد دارای توزیع یکنواخت بر روی $[0,1]$ می‌باشند. علامت o نیز نشانگر ضرب عنصر به عنصر است.

تکاوران

تکاوران به‌صورت اتفاقی در فضای جستجو حرکت کرده و به دنبال مقادیر کم از تابع هزینه می‌گردند. حرکت یک تکاور به‌صورت زیرمدل می‌شود

$$\begin{aligned} \phi_i &\leftarrow \phi_i + \rho\alpha_{max} \\ x_i &\leftarrow x_i + \alpha_{max} l_{max} r_1 D(\phi_i) \end{aligned} \qquad (۱۷–۱۱)$$

که در آن α_{max} یک پارامتر میزان‌سازی بوده و میزان بیشترین حد مجاز برای چرخش تکاور را تعیین می‌نماید. $\rho \in [-1,1]$ یک متغیر اتفاقی با توزیع یکنواخت بوده و l_{max} نیز یک پارامتر میزان‌سازی است که بیشترین مسافتی که یک تکاور می‌تواند در یک نسل طی کند را تعیین می‌کند. این پارامتر همان پارامتر l_{max} در معادله‌ی (۱۷–۸) می‌باشد. در نهایت، r_1 متغیری اتفاقی با میانگین صفر، واریانس واحد و توزیع یکنواخت می‌باشد.

خلاصه

شکل ۱۷–۵ طرح کلی GSO را به تصویر می‌کشد و همچنین نشان می‌دهد که GSO چندین پارامتر میزان‌سازی دارد. توجه داشته باشید که در شکل ۱۷–۵، یک ذره حدود ۸۰٪ ذرات ملحق‌شونده، حدود ۲۰٪ تکاور و یک ذره تولیدکننده می‌باشد. [هی و همکاران، ۲۰۰۹] تأثیر این تنظیمات و برخی دیگر از پارامترهای میزان‌سازی را مورد مطالعه قرار داده و مقادیر زیر را پیشنهاد می‌دهد:

$$\begin{aligned} a_{max} &= round\sqrt{n+1} \\ \theta_{max} &= \pi / a_{max}^2 \\ \alpha_{max} &= \theta_{max}/2 \\ l_{max} &= \left|\left|U-L\right|\right|_2 \end{aligned} \qquad (۱۲–۱۷)$$

که در آن بردارهای n بعدی U و L به ترتیب کران بالا و پایین فضای جستجو می‌باشند.

اندازه جمعیت $= N$

جمعیتی ابتدایی از راه‌حل‌های نامزد را مقداردهی کن: $\{x_i\}$ برای $i \in [1,N]$

اندازه‌ی زاویه‌ی ϕ_i را برای هر راه‌حل نامزد به‌صورت اتفاقی مقداردهی اولیه کن

تا زمانی که شرایط توقف برآورده نشده است

تولیدکننده را پیدا کن: $x_p \leftarrow arg\min_{x_i}\{f(x_i) : i \in [1,N]\}$

نتایج معادله‌ی $\{x_z, x_r, x_l\} \leftarrow (17-8)$

اگر $\min\{f(x_z), f(x_r), f(x_l)\} < f(x_p)$ آنگاه

$x_p \leftarrow argmin\{f(x_z), f(x_r), f(x_l)\}$

در غیر این صورت

$\rho \leftarrow U[-1,1]$

$\phi(x_p) \leftarrow \phi(x_p) + \rho\alpha_{max}$

پایان اگر

برای هر $x_i \neq x_p$

$r_2 \leftarrow U[0,1]$

اگر $r_2 < 0.8$

بگذارید x_i از معادله‌ی (۱۷–۱۰) مقدار بگیرد

در غیر این صورت

بگذارید x_i از معادله‌ی (۱۷–۱۱) برد بپذیرد

شکل ۱۷-۵ یک بهینه‌سازی جستجوی گروهی (GSO) برای مینیمم‌سازی تابع n بعدی $f(x)$ که در آن x_i نشانگر iاُمین راه‌حل نامزد می‌باشد.

GSP مشابه PSO است. هرچند، یک تفاوت آن است که در PSO ذرات دارای حافظه‌ای از مکان قبلی خود در فضای جستجو می‌باشند. یک تفاوت دیگر نیز آن است که در PSO، هر ذره از یک استراتژی جستجوی واحد استفاده می‌نماید. یک ویژگی متمایزکننده‌ی GSO رفتار متنوع و تغییرکننده‌ی آن است. البته این نوع رفتار را می‌توان در PSO گربه‌ماهی نیز مشاهده نمود (بخش ۱۱-۷ را ببینید). زمینه‌های امیدبخش برای تحقیقات آتی در حوزه‌ی GSO شامل مدل‌سازی و تحلیل ریاضی، انطباق آنلاین پارامترهای میزان‌سازی و ترکیب نمودن سایر ویژگی‌های الهام گرفته شده از طبیعت، می‌شود.

۱۷-۴ الگوریتم قورباغه جهنده

الگوریتم قورباغه جهنده (SFLA[1]) در [اوساف[2] و لانسی[3]، ۲۰۰۳] و [اوساف و همکاران، ۲۰۰۶] به‌عنوان ترکیبی از PSO و تکامل پیچیده‌ی به‌هم‌آمیخته (SCE[4]) معرفی گردید. ایده‌ی اصلی SCE، تکامل زیرجمعیت‌ها به‌صورت مستقل و تعامل آن‌ها با یکدیگر به‌صورت دوره‌ای می‌باشد [دوان[5] و همکاران، ۱۹۹۲]، [دوان و همکاران، ۱۹۹۳]. SCE از انتخاب احتمالاتی برای انتخاب والدین در هر نسل استفاده کرده و همچنین جهت جلوگیری از بروز رکود، به‌صورت اتفاقی ذرات جدیدی تولید می‌نماید. SFLA شامل ایده‌هایی از PSO و SCE می‌شود.

شکل ۱۷-۶ استراتژی جستجوی جهانی الگوریتم SFLA را نشان می‌دهد. کار این الگوریتم با تولید N راه‌حل نامزد اتفاقی شروع می‌شود. سپس این N ذره به m زیرجمعیت تقسیم می‌گردند. معمولاً N مضربی از m بوده و بدین ترتیب همه‌ی m زیرجمعیت‌ها شامل تعداد یکسانی ذره می‌شوند. سپس، در هر یک از زیرجمعیت‌ها یک الگوریتم جستجوی محلی اجرا می‌شود. در ابتدای نسل بعد، این زیرجمعیت‌ها به یکدیگر

[1] Shuffled Frog Leaping Algorithm
[2] Eusaff
[3] Lansey
[4] Shuffled Complex Evolution
[5] Duan

برخورد نموده و بدین ترتیب برخی ذرات به‌صورت اتفاقی از زیرجمعیت خود جدا شده و به یک زیرجمعیت دیگر می‌روند. پارامترهای میزان‌سازی معمول برای SFLA شامل اندازه‌ی جمعیت N در حدود ۲۰۰ و حدود ۲۰ زیرجمعیت می‌شود [البلتاگی[1] و همکاران، ۲۰۰۵].

عبارت "جستجوی محلی را انجام بده" در شکل ۱۷-٦ به معنی اجرای الگوریتم شکل ۱۷-۷ است. در طول جستجوی محلی، هر زیرجمعیت به‌طور مستقل به اجرای یک جستجوی تکاملی برای i_{max} دوره می‌پردازد. در هر دوره، x_w که بدترین ذره‌ی موجود در زیرجمعیت می‌باشد، به‌روزرسانی می‌شود:

$$x_w \leftarrow x_w + r(x_b - x_w) \qquad (۱۳-۱۷)$$

که در آن $r \in [0,1]$، یک متغیر اتفاقی با توزیع یکنواخت بوده و x_b بهترین ذره‌ی موجود در زیرجمعیت می‌باشد. اگر معادله‌ی (۱۷-۱۳) باعث بهبود x_w نگردد، از معادله‌ی زیر برای به‌روزرسانی آن استفاده خواهیم کرد:

$$x_w \leftarrow x_w + r(x_g - x_w) \qquad (۱٤-۱۷)$$

که در آن $r \in [0,1]$، یک متغیر اتفاقی با توزیع یکنواخت بوده و x_g بهترین ذره‌ی جهانی در میان همه‌ی m زیرجمعیت‌ها می‌باشد. اگر معادله‌ی (۱۷-٤) نیز باعث بهبود x_w نگردد، آنگاه x_w را با یک ذره‌ی تولید شده به‌صورت اتفاقی جایگزین می‌نماییم.. مقدار معمول برای حد تعداد دوره‌ها در شکل ۱۷-۷، $i_{max} = 20$ می‌باشد [البلتاگی و همکاران، ۲۰۰۵]. تحقیقات نویدبخش در زمینه‌ی SFLA شامل مدل‌سازی و تحلیل ریاضی و ترکیب نمودن سایر ویژگی‌های الهام گرفته شده از طبیعت، می‌شود.

```
جمعیتی ابتدایی از راه‌حل‌های نامزد را مقداردهی کن: {x_i} برای i ∈ [1, N]
تا زمانی که شرایط توقف برآورده نشده است
    جمعیت را به‌صورت اتفاقی به m زیرجمعیت تقسیم کن
    برای هر زیرجمعیت m تا i = 1
        جستجوی محلی در iامین زیرجمعیت را انجام بده (شکل ۱۷-۷)
    زیرجمعیت بعدی
نسل بعدی
```

شکل ۱۷-٦ شبه کد بالا طرح کلی استراتژی جستجوی جهانی الگوریتم قورباغه جهنده (SFLA) را نشان می‌دهد.

[1] Elbeltagi

بهترین ذره در کل جمعیت (x_g) را پیدا کن
برای i_{max} تا $i = 1$
بهترین و بدترین ذرات زیرجمعیت (به ترتیب x_b و x_w) را پیدا کن
از معادله‌ی (۱۷–۱۳) برای به‌روزرسانی x_w استفاده کن
اگر به‌روزرسانی باعث بهبود x_w نشد
از معادله‌ی (۱۷–۱٤) برای به‌روزرسانی x_w استفاده کن
اگر به‌روزسانی باعث بهبود x_w نشد
ذره‌ی تولید شده به صورت اتفاقی ← x_w
پایان اگر
پایان اگر
دوره‌ی بعد

شکل ۱۷–۷ شبه کد بالا استراتژی جستجوی محلی الگوریتم قورباغه‌ی جهنده (SFLA) را نشان می‌دهد.

۱۷–۵ الگوریتم کرم شب‌تاب

الگوریتم کرم شب‌تاب در [یانگ، ۲۰۰۸b، فصل ۸] و [یانگ، ۲۰۱۰b] معرفی گردید. این الگوریتم بر پایه‌ی جذب شدن کرم‌های شب‌تاب به یکدیگر قرار دارد. میزان جذابیت بر پایه‌ی میزان روشنایی کرم شب‌تاب قرار دارد و به همین دلیل به‌صورت نمایی با فاصله کاهش می‌یابد. یک کرم شب‌تاب تنها جذب کرم‌هایی می‌شود که روشناییشان از خودش بیشتر باشد.

شکل ۱۷–۸ شبه کد الگوریتم کرم شب‌تاب را نشان می‌دهد. با $\gamma \to 0$، همه‌ی کرم‌های شب‌تاب به یک میزان جذب یکدیگر می‌شوند و این به معنای انتشار یکنواخت نور در اتمسفر می‌باشد. این نوع رفتار را می‌توان در خلاء مشاهده نمود. با $\gamma \to \infty$، کرم‌های شب‌تاب اصلا جذب یکدیگر نشده و این متناظر جستجوی اتفاقی است. این نوع رفتار را می‌توان در مه غلیظ مشاهده نمود. پارامترهای β_0 و α، مصالحه‌ی میان انتفاع (جذب شدن به سایر کرم‌های شب‌تاب) و کاوش (جستجوی اتفاقی) را تعیین می‌نمایند. پارامترهای میزان‌سازی معمول به قرار زیراند:

$$\gamma_i = \frac{\gamma_0}{\max_j \left\|x_i - x_j\right\|_2} \quad \text{که در آن } \gamma_0 = 0.8 \qquad (۱۵–۱۷)$$

$$\alpha = 0.01$$
$$\beta = 1$$

هر کرم شب‌تاب x_i در هر زمان، روشنایی خود را با یک کرم شب‌تاب دیگر مانند x_j مقایسه می‌کند. اگر x_j درخشنده‌تر از x_i باشد، آنگاه x_i حرکتی انجام می‌دهد که هم دارای مؤلفه‌ی اتفاقی بوده و هم دارای مؤلفه‌ای در جهت x_j می‌باشد. مقدار αr در شکل ۱۷–۸ یک مؤلفه‌ی اتفاقی است و معمولاً به دلیل کوچک بودن α، مقدار کوچکی است (معادله‌یی ۱۷–۱۵ را ببینید). مقدار $\beta_0 e^{-\gamma_i r_{ij}^2}(x_j - x_i)$، مؤلفه‌ی جهت‌یافته است. همان‌طور که پیش از این نیز گفته شد، این مؤلفه تابعی نمایی از فاصله r_{ij} میان x_j و x_i می‌باشد. اگرچه که نمایی بودن این تابع ریشه‌ای بیولوژیکی دارد، ممکن است بخواهیم از توابع دیگری که با افزایش فاصله کاهش می‌یابند، استفاده نماییم.

یکی از مواردی که در شکل ۱۷–۸ به چشم می‌آید آن است که بهترین ذره‌ی موجود در جمعیت هیچگاه به‌روزرسانی نمی‌شود. شاید اگر بهترین ذره را به‌صورت دوره‌ای و به منظور یافتن ذره‌ای بهتر به‌روزرسانی نماییم بتوانیم عملکرد الگوریتم را بهبود بخشیم. با این حال ممکن است این روش پرریسک بوده و خروجی چندان خوبی نداشته باشد چرا که در این روش ممکن است به تعداد زیادی محاسبه‌ی تابع نیاز باشد تا محل بهتری در فضای جستجو یافت شود.

تنوعات بسیاری برای الگوریتم کرم شب‌تاب وجود دارد که در [لوکاسیک[1] و زک[2]، ۲۰۰۹]، [یانگ، ۲۰۰۹b] و [یانگ، ۲۰۱۱a] مورد بحث قرار گرفته‌اند. برای مثال، پارامتر α معمولاً تابعی کاهشی با زمان است و هدف آن کاهش میزان کاوش با بهینه‌تر شدن جمعیت می‌باشد. یک نسخه از این الگوریتم برای مسائل ترکیبی در [صیادی[3] و همکاران، ۲۰۱۰] ارائه شده است. الگوریتم کرم شب‌تاب مانند AFSA، که در بخش ۱۷–۲ مورد بحث قرار گرفت، بسیار مشابه PSO است (فصل ۱۱ را ببینید). می‌توان با اعمال اصلاحاتی به الگوریتم شکل ۱۷–۸ آن را به حالت خاصی از PSO تبدیل نمود (مسئله‌ی ۱۷–۵ را ببینید).

یک جمعیت اتفاقی مانند $\{x_i\}$ را برای $i \in [1, N]$ تولید کن
تا زمانی که شرایط توقف برآورده نشده است
  برای هر ذره‌ی x_i
    برای هر ذره‌ی $x_j \neq x_i$
      اگر $f(x_j) < f(x_i)$
        برای هر بعد $k \in [1, n]$

[1] Lukasik
[2] Zak
[3] Sayadi

$\rho \leftarrow U[0,1]$
اگر $\rho < 1/2$
$r_k \leftarrow (u_k - x_i(k))U[0,1]$
در غیر این صورت
$r_k \leftarrow (x_i(k) - l_k)U[0,1]$
پایان اگر
بُعد بعدی
فاصله‌ی میان x_j و x_i $\leftarrow r_{ij}$
$x_i \leftarrow x_i + \beta_0 e^{-\gamma_i r_{ij}^2}(x_i - x_j) + \alpha r$ (این یک عملیات برداری است)
پایان اگر
x_j بعدی
x_i بعدی
نسل بعدی

شکل ۱۷–۸ یک الگوریتم کرم شب‌تاب برای مینیمم‌سازی تابع n بعدی $f(x)$. در این الگوریتم x_i، iاُمین راه‌حل نامزد بوده و $x_i(k)$، kاُمین عنصر x_i است. $U[0,1]$ یک عدد اتفاقی با توزیع یکنواخت بر روی $[0,1]$ بوده و l_k و u_k به ترتیب حد پایین و بالای kاُمین بعد فضای جستجو می‌باشند.

۱۷–۶ بهینه‌سازی کاوش باکتریایی

الگوریتم بهینه‌سازی کاوش باکتریایی (BFOA[1]) در [پاسینو[2]، ۲۰۰۲] و بر پایه‌ی رفتار باکتری اشریشا کلی[3]، که عموماً با نام لاتین E. coli شناخته می‌شود، می‌باشد. BFOA بر پایه‌ی این فرضیه است که انتخاب طبیعی به گسترش ژن‌هایی کمک می‌کند که در رفتارهای موفقیت‌آمیز کاوش غذا را نمایندگی می‌نمایند. صدالبته که کاوش غذا در میان همه‌ی گونه‌ها و نه فقط باکتری‌ها معمول است. گاهی حیوانات به‌صورت تعاملی به کاوش غذا می‌پردازند و گاهی نیز به تنهایی این کار را انجام می‌دهند. اگر این کار را به تنهایی انجام دهند، می‌توانند تمامی غذایی را که پیدا می‌کنند برای خودشان نگه دارند. اما اگر به‌صورت گروهی به کاوش بپردازند، می‌توانند راحت‌تر با مهاجمان مقابله نمایند. حیوانات باید احتمال موفقیت در کاوش را با ریسک‌های احتمالی از سوی مهاجمان متعادل نمایند.

[1] Bacterial Foraging Optimization Algorithm
[2] Passino
[3] Escherichia coli bacteria

اگر یک حیوان ناحیه‌ای جغرافیایی با غذای بسیار بیابد، باید میزان انتفاع خود از آن منابع را با احتمال پیدا کردن منابع بهتر در یک مکان دیگر متعادل نماید. این نوع دیگری از رفتار متعادل‌کننده‌ی ریسک است. هنگامی که یک حیوان منابع غذایی خود در یک محل را تخلیه می‌کند، زمانی بسیار مناسب برای کاوش سایر نواحی که ممکن است دارای منابع بیشتری باشد پیش می‌آید.

کاوش همچنین شامل رفتارهایی غیر از جستجو برای غذا نیز می‌شود. کاوش به معنای دنبال کردن و حمله به حیوان مورد شکار و همچنین خوردن شکار نیز می‌باشد. اگر شکار بزرگتر از مهاجم باشد، آنگاه مهاجما باید با یکدیگر همکاری کرده و به شکار حمله نمایند. اگر شکار کوچکتر از مهاجم باشد، آنگاه شاید برای مهاجم بهتر باشد خودش به تنهایی اقدام به حمله و خوردن شکار نمایند. برخی کاوش‌کنندگان به‌صورت پیوسته در محیط حرکت کرده و به دنبال شکار می‌گردند. برخی کاوشگران نیز در یک محل ثابت باقی مانده و به انتظار ورود شکار به فاصله حمله می‌نشینند. برخی دیگر از کاوشگران ترکیبی از این روش‌ها را مورد استفاده قرار می‌دهند. BFOA به‌صورت خاص بر اساس رفتار کاوشی باکتری مدل شده است اما نظریه‌ی کاوش به‌صورت گسترده‌ای مورد مطالعه قرار گرفته است [استفنز[1] وو کربس[2]، ۱۹۸۶]، [گیرالدو[3] و کاراکو[4]، ۲۰۰۰] و کاربردهای بالقوه‌ی بسیاری در نظریه‌ی بهینه‌سازی دارد [کوئیجانو[5]، ۲۰۰۶].

BFAO بر اساس سه رفتار باکتری قرار دارد. اول آنکه باکتری‌ها خود را در محیط به پیش رانده و جلو می‌برند. این رفتار کموتاکسی[6] نام دارد. دوم آنکه باکتری‌ها بازتولید می‌نمایند. سوم و آخر آنکه باکتری‌ها بر اثر اتفاقات محیطی نابود و یا پراکنده می‌شوند.

کموتاکسی

اولین رفتار باکتری که با نام کموتاکسی یا خودرانی شناخته می‌شود را می‌توان به دو رفتار تقسیم نمود. اول آنکه باکتری می‌تواند در جهت‌های اتفاقی جست‌وخیز کند. دوم آنکه باکتری‌ها می‌توانند خود را در جهت یک منبع غذایی افزایشی حرکت دهند. این نوع دوم از رفتار خودرانی نه تنها از منبع غذایی، بلکه از حضور باکتری‌های دیگر نیز تأثیر می‌پذیرد. حضور باکتری‌های دیگر می‌تواند هم باعث جذب و هم باعث دفع باکتری شود. جذب از این رو صورت می‌گیرد که حضور باکتری‌های دیگر در یک محل به معنای وجود

[1] Stephens
[2] Krebs
[3] Giraldeau
[4] Caraco
[5] Quijano
[6] Chemotaxis

منبع غذایی در آن محل می‌باشد. دفع نیز از این جهت صورت می‌گیرد که حضور سایر باکتری‌ها در یک محل خاص به معنای وجود رقابت برای به دست آوردن منبع غذایی در آن محل می‌باشد.

فرض کنید می‌خواهیم مینیمم یک تابع مانند $f(x)$ را بیابیم. در BFOA، رفتار خودرانی به‌صورت زیر مدل می‌شود

$$x \leftarrow x + c\Delta \qquad (۱۶-۱۷)$$

که در آن x محل یک ذره در جمعیت بوده و c اندازه‌ی گام است. Δ نیز یک تابع واحد در جهتی خاص در فضای جستجو می‌باشد. هنگام جست‌وخیز باکتری، Δ یک بردار واحد اتفاقی است. ترکیب جاذبه و دافعه‌ی حاصل از وجود باکتری‌های دیگر، تابع هزینه‌ی مؤثر $f'(x)$ را نتیجه می‌دهد:

$$f'(x) = f(x) + \sum_{i=1}^{N} [h \exp\left(-w_r ||x - x_i||_2^2\right) - d \exp\left(-w_a ||x - x_i||_2^2\right)] \qquad (۱۷-۱۷)$$

که در آن N اندازه‌ی جمعیت بوده و h، w_r، d و w_a پارامترهای میزان‌سازی مرتبط با نیروی دافعه و جاذبه‌ای می‌باشند که باکتری‌ها به یکدیگر وارد می‌کنند. اگر ذره‌ی x در جهتی حرکت کند که $f'(x)$ کاهش یابد، به حرکت خود در آن جهت ادامه می‌دهد، هرچند که یک حد بالا مانند N_s (یک پارامتر میزان‌سازی دیگر) برای تعداد حرکات ممکن در یک جهت خاص در نظر گرفته می‌شود.

بازتولید

هر ذره رفتار خودرانی را برای N_c دوره ادامه می‌دهد. بدین معنی که ابتدا در یک جهت اتفاقی حرکت کرده، سپس اگر حرکت اتفاقی باعث کاهش $f'(x)$ شود به حرکت خود در آن جهت ادامه می‌دهد. N_c، دوره‌ی زندگی هر باکتری را مشخص می‌کند. پس از آن، سلامت هر باکتری بر اساس مقدار میانگین $f'(x)$ در طول N_c دوره‌ی قبل اندازه‌گیری می‌شود. نیمه‌ی سالم‌تر جمعیت باکتری‌ها با دو کلون به ازای هر باکتری تولید مثل کرده و بدین ترتیب، N باکتری جدید برای نسل بعد تولید می‌گردد.

حذف شدن و پراکندگی

پس از بازتولید، گام حذف شدن=پراکندگی انجام می‌شود. هر باکتری با احتمال p_e (یک پارامتر میزان‌سازی دیگر) به یک مکان اتفاقی در فضای جستجو پراکنده می‌شود.

خلاصه

شکل ۱۷–۹ یک BFOA بنیادین را نشان می‌دهد. پارامترهای میزان‌سازی معمول به قرار زیر می‌باشند [پاسینو، ۲۰۰۲]:

$c = 0.1$ اندازه‌ی گام
$N = 50$ اندازه‌ی جمعیت
$N_c = 100$ اندازه‌ی گام
$N_s = 4$ تعداد گام‌های کموتاکسیس
$N_r = 4$ تعداد گام‌های کاهش هزینه
$N_e = 2$ تعداد گام‌های حذف- پراکندگی (۱۸–۱۷)
$d = 1$ عمق نیروی جاذبه
$h = 1$ عمق نیروی دافعه
$w_a = 0.2$ پهنای نیروی جاذبه
$w_r = 10$ پهنای نیروی دافعه
$p_e = 0.25$ احتمال حذف- پراکندگی

تعداد نسل‌ها در BFOA مانند سایر الگوریتم‌های تکاملی چندان واضح نیست. بیرونی‌ترین حلقه‌ی شکل ۱۷–۹، N_e بار اجرا می‌شود که این مقدار معمولاً برابر دو است. بهترین راه برای اندازه‌گیری تلاش محاسباتی الگوریتم تکاملی، نه تعداد نسل‌ها بلکه تعداد ارزیابی تابع است.

توجه داشته باشید که در شکل ۱۷–۹ بازتولید به‌صورت کلونی‌سازی صورت می‌گیرد. شاید بتوانیم با استفاده از عملیات بازترکیب پیچیده‌تر عملکرد را بهبود ببخشیم (بخش ۸–۸ را ببینید)، هر چند که این کار ممکن است باعث انحراف از اساس باکتریایی BFAO شود. همچنین، در شکل ۱۷–۹ برای تولید نسل بعد نیمه‌ی برتر جمعیت را کلون می‌نماییم. می‌توان به جای این کار، بهترین B ذره را کلون نمود. در این صورت B یک پارامتر میزان‌سازی خواهد بود.

BFOA زمینه‌ای تحقیقاتی با امکانات بسیار است. کاوش باکتریایی و حیوانی دارای جنبه‌های بسیاری هستند که می‌توان آن‌ها را برای بهبود عملکرد بهینه‌سازی مدل کرد. انطباق خودکار پارامترهای بهینه‌سازی می‌تواند برای BFOA بسیار مهم باشد، چرا که در این الگوریتم پارامترهای میزان‌سازی بسیاری وجود دارد

و این کار می‌تواند باعث بهبود عملکرد شود [دسگوپتا[1] و همکاران، ۲۰۰۹]. BFOA را می‌توان با سایر الگوریتم‌های تکاملی ترکیب نمود، همان‌طور که پیش از این با PSO [بیسواس[2] و همکاران، ۲۰۰۷b] و DE [بیسواس و همکاران، ۲۰۰۷a] ترکیب شده است. مقداری تحلیل ریاضی از BFOA در [داس و همکاران، ۲۰۰۹] ارائه شده است، هرچند که هنوز جای کار بسیاری در این زمینه وجود دارد. توجه داشته باشید که مدل کموتاکسی باکتریایی یک الگوریتم تکاملی مشابه اما مجزا است و بر پایه‌ی رفتار باکتری قرار دارد [مولر[3] و همکاران، ۲۰۰۲].

پارامترهای نشان داده شده در معادله‌ی (۱۷–۱۸) را مقداردهی اولیه کن
یک جمعیت اتفاقی مانند $\{x_i\}$ را برای $i \in [1, N]$ تولید کن
برای $l = 1$ تا N_e (گام‌های حذف- پراکندگی)
  برای $k = 1$ تا N_r (گام‌های بازتولید)
    برای $j = 1$ تا N_c (گام‌های کموتاکسی)
      برای هر یک از ذرات x_i $i \in [1, N]$
        هزینه‌ی مؤثر را مطابق معادله‌ی (۱۷–۱۷) محاسبه کن
        یک بردار یکه‌ی اتفاقی n بعدی مانند Δ تولید کن
        برای $m = 1$ تا N_s (گام‌های کاهش هزینه)
          $\hat{x}_i \leftarrow x_i + c\Delta$
          اگر $f'(\hat{x}_i) < f'(x_i)$
            $x_i \leftarrow \hat{x}_i$
          در غیر این صورت
            $m \leftarrow N_s$ (از حلقه‌ی کاهش هزینه خارج شو)
          پایان اگر
        m بعدی
      ذره‌ی بعدی
    j بعدی

[1] Desgupta
[2] Biswas
[3] Muller

برای هر یک از ذرات x_i، $i \in [1, N]$

میانگین $f'(x_i)$ در طول حلقه‌ی گام‌های کموتاکسی $\leftarrow F_i$

ذره‌ی بعد

بدترین $N/2$ ذره بر اساس $\{F_i\}$ را از بین ببر

بهترین $N/2$ ذره بر اساس$\{F_i\}$ را کلون ساز

k بعدی

برای هر یک از ذرات x_i، $i \in [1, N]$

عدد اتفاقی $r \leftarrow U[0,1]$

اگر $r < p_e$ آنگاه

نقطه‌ای اتفاقی در فضای جستجو $\leftarrow x_i$

پایان اگر

ذره‌ی بعد

l بعدی

شکل ۱۷-۹ یک الگوریتم بهینه‌سازی کاوش باکتریایی (BFOA) برای مینیمم‌سازی تابع n بعدی $f(x)$ که در آن x_i، iاُمین راه‌حل نامزد است.

۱۷-۷ الگوریتم کلونی مصنوعی زنبورها

الگوریتم کلونی مصنوعی زنبورها (ABC[1]) بر پایه‌ی رفتار زنبورها بوده و اولین بار در [باستورک[2] و کارابوگا[3]، ۲۰۰۶]، [کارابوگا و باستورک، ۲۰۰۷] منشر گردید. ABC بر پایه‌ی جستجوی منبع غذای بهینه توسط زنبورها است. محل یک منبع غذایی مشابه یک نقطه در فضای جستجوی یک مسئله‌ی بهینه‌سازی می‌باشد. میزان شهد موجود در یک محل نیز مشابه برازندگی یک راه‌حل نامزد است. در ABC، سه نوع مختلف از زنبورها شبیه‌سازی می‌شوند.

نوع اول زنبورهای کاوشگر[4] یا کارگر هستند که بین کندویشان و منبع غذایی در حال رفت و آمد هستند. هر کاوشگر با یک محل خاص در ارتباط است و این محل را هنگام رفت و آمد به خاطر می‌سپارد. هنگامی

[1] Artificial Bee Colony
[2] Basturk
[3] Karaboga
[4] Forager Bees

که یک زنبور کاوشگر شهد خود را به کندو می‌برد، دوباره به منبع غذایی بازمی‌گردد، اما در همین حین به جستجوی محلی نیز می‌پردازد تا شاید منبع غذایی بهتری در همان نزدیکی پیدا کند.

نوع دوم زنبورها، زنبورهای ناظر[1] هستند که با محل به خصوصی در ارتباط نبوده، اما بر رفتار زنبورهای کاوشگر هنگام برگشت به کندو نظارت می‌کنند. زنبورهای ناظر بر میزان شهد آورده شده توسط زنبورهای کاوشگر به کندو (که به معنای میزان برازندگی محل کاوشگر در فضای جستجو می‌باشد) نظارت کرده و از این اطلاعات برای تصمیم‌گیری در مورد محل مورد کاوش برای شهد، استفاده می‌نمایند. محل جستجوی زنبور ناظر به‌صورت احتمالاتی و بر اساس نظارت آن‌ها بر زنبورهای کاوشگر تعیین می‌گردد.

سومین نوع زنبورها، زنبورهای پیش‌آهنگ[2] هستند که به کاوش پرداخته و مانند زنبورهای ناظر، با منبع غذایی خاصی در ارتباط نمی‌باشند. اگر یک زنبور پیش‌آهنگ ببیند که یک کاوشگر دچار رکود شده و میزان شهدی که به کندو برمی‌گرداند افزایش نیافته، آنگاه زنبور پیش‌آهنگ به‌صورت اتفاقی به دنبال منبع شهدی جدید در فضای جستجو خواهد گشت. رکود به معنای عدم موفقیت کاوشگر در افزایش شهد، پس از چند رفت و برگشت است.

این ایده‌ها به الگوریتم ABC منجر می‌شود. این الگوریتم در شکل ۱۷-۱۰ خلاصه شده است. این شکل نشان می‌دهد که تقسیم‌بندی میان زنبورهای کاوشگر، ناظر و پیش‌آهنگ تنها یک قیاس است. ایده‌ی کلیدی الگوریتم ABC، آن است که رفتارهای کاوشی، نظارت و پیش‌آهنگی برای پیدا کردن بهینه‌ی جهانی شبیه‌سازی شده‌اند.

شکل ۱۷-۱۰ نشان می‌دهد که هر کاوشگر به‌صورت اتفاقی به اصلاح مکان خود در فضای جستجو می‌پردازد. اگر این اصلاح اتفاقی باعث بهبود عملکرد شود، آنگاه کاوشگر به محل جدید می‌رود. همچنین، زنبورهای ناظر نیز به‌صورت اتفاقی به اصلاح مکان زنبورهای کاوشگر می‌پردازند. برای این کار، زنبور کاوشگر به‌صورت اتفاقی و با استفاده از چرخ رولت انتخاب می‌شود. در اینجا نیز اگر اصلاح انجام شده باعث بهبود کاوشگر شود، کاوشگر به محل جدید می‌رود. در آخر نیز، یک پیش‌آهنگ، در صورتی که کاوشگری نتوانسته باشد پس از تعداد معلومی از اصلاحات اتفاقی بهبود یابد، آن را جابه‌جا می‌نماید. شمارنده‌ی $T(x_i)$ در شکل ۱۷-۱۰ شمارنده‌ای است که تعداد اصلاحات ناموفق متوالی بر روی هر کاوشگر را می‌شمارد. با توجه به شکل ۱۷-۱۰ می‌توان دید که الگوریتم ABC دارای چندین پارامتر میزان‌سازی است. پارامترهای معمول ABC عبارتند از

[1] Onlooker Bees
[2] Scout Bees

$P_f = P_o = \frac{N}{2}$، حد رکود $L = Nn/2$ (۱۹-۱۷)

در [کارابوگا و باستورک، ۲۰۰۷]، [کارابوگا و باستورک، ۲۰۰۸]، [کارابوگا و آکای[1]، ۲۰۰۹] و [کارابوگا و همکاران، ۲۰۱۳]، انواع بسیاری از الگوریتم ABC مورد بحث قرار گرفته‌اند. همچنین می‌توان با بررسی شکل ۱۷-۱۰، اصلاحات زیادی را به آن اعمال نمود. برای نمونه، در شل ۱۷-۱۰، اگر کاوشگری محل بهتری پیدا کند، مکان خود را قاطعانه تغییر می‌دهد. به جای این نوع به‌روزرسانی قاطعانه می‌توان از به‌روزرسانی احتمالاتی استفاده نمود. همچنین در شکل ۱۷-۱۰، زنبور ناظر با استفاده از چرخ رولت، زنبور کاوشگر را انتخاب می‌نماید. به جای انتخاب چرخ رولت می‌توان از نوع دیگری از انتخاب برازندگی-محور استفاده نمود (بخش ۸-۷ را ببینید).

چندین الگوریتم مشابه ABC تا به حال ارائه شده‌اند. برای اطلاع از آن‌ها می‌توانید به [ترشکو[2]، ۲۰۰۰]، [تئودورووِیچ[3]، ۲۰۰۳]، [بناتچبا[4] و همکاران، ۲۰۰٤] و همچنین منابع [کارابوگا و باستورک، ۲۰۰۸] مراجعه نمایید. یکی از الگوریتم‌هایی که بسیار شبیه ABC است، الگوریتم زنبورها [فام[5] و همکاران، ۲۰۰٦] می‌باشد. اما از میان الگوریتم‌های تکاملی ذکر شده در این کتاب، الگوریتم DE بیش از همه به ABC شبیه است (فصل ۱۲ را ببینید). تحقیقات آتی در زمینه‌ی ABC می‌تواند شامل بررسی جنبه‌های مشترک آن با DE و سایر الگوریتم‌های زنبور-محور و همچنین ترکیب نمودن ویژگی‌های الهام گرفته شده از زیست با این الگوریتم، شود.

اندازه‌ی جمعیت $= N$

عدد صحیح L (حد رکود) را مقداردهی اولیه کن

اندازه‌ی جمعیت کاوشگر را مقداردهی اولیه کن؛ $P_f < N$

اندازه‌ی جمعیت ناظر را مقداردهی اولیه کن؛ $P_o = N - P_o$

جمعیتی اتفاقی از کاوشگران را مقداردهی اولیه کن؛ $\{x_i\}$ برای $i \in [1, P_f]$

شمارنده‌های تعداد تلاش‌های کاوشگران را مقداردهی اولیه کن؛ $T(x_i) = 0$ برای $i \in [1, P_f]$

تا زمانی که شرایط توقف برآورده نشده است

زنبورهای کاوشگر

[1] Akay
[2] Tereshko
[3] Teodorovic
[4] Benatchba
[5] Pham

برای هر کاوشگر x_i، $i \in [1, P_f]$

$k \neq i$ به‌طوری که $k \leftarrow$ عدد صحیح اتفاقی $\in [1, N]$

$s \leftarrow$ عدد صحیح اتفاقی $\in [1, n]$

$r \leftarrow U[-1,1]$

$v_i(s) \leftarrow x_j(s) + r(x_j(s) - x_k(s))$

اگر $f(v_i)$ بهتر از $f(x_i)$ باشد، آنگاه

$x_i \leftarrow v_i$

$T(x_i) \leftarrow 0$

در غیر این صورت

$T(x_i) \leftarrow T(x_i) + 1$

پایان اگر

کاوشگر بعدی

زنبورهای ناظر

برای هر ناظر v_i، $i \in [1, P_o]$

یک کاوشگر مانند x_j را انتخاب کن به‌طوری که $\Pr(x_j) \propto$ برازندگی (x_j)

$k \neq j$ به‌طوری که $k \leftarrow$ عدد صحیح اتفاقی $\in [1, P_f]$

$s \leftarrow$ عدد صحیح اتفاقی $\in [1, n]$

$r \leftarrow U[-1,1]$

$v_i(s) \leftarrow x_j(s) + r(x_j(s) - x_k(s))$

اگر $f(v_i)$ بهتر از $f(x_j)$ باشد، آنگاه

$x_j \leftarrow v_i$

$T(x_j) \leftarrow 0$

در غیر این صورت

$T(x_j) \leftarrow T(x_j) + 1$

پایان اگر

ناظر بعدی

زنبورهای پیش‌آهنگ

برای هر کاوشگر x_i، $i \in [1, P_f]$

اگر $T(x_i) > L$، آنگاه

ذره‌ی تولید شده به صورت اتفاقی $\leftarrow x_i$

$T(x_i) \leftarrow 0$

پایان اگر
کاوشگر بعدی
نسل بعد

شکل ۱۷-۱۰ یک الگوریتم کلونی مصنوعی زنبورها (ABC) برای بهینه‌سازی تابع n بعدی $f(x)$ که در آن x_i اُمین راه‌حل نامزد است.

۱۷-۸ الگوریتم جستجوی گرانشی

الگوریتم جستجوی گرانشی (GSA[1]) که بر پایه‌ی قوانین گرانش بنا شده است، در [راشدی[2] و همکاران، ۲۰۰۹] معرفی گردید. GSA مشابه بهینه‌سازی نیروی مرکزی[3]، که یک الگوریتم بهینه‌سازی تکاملی قاطع بر پایه‌ی گرانش است، می‌باشد [فارماتو[4]، ۲۰۰۷]، [فارماتو، ۲۰۰۸]. سایر الگوریتم‌های مشابه شامل بهینه‌سازی گرانش فضایی[5] [سیاو[6] و همکاران، ۲۰۰۵] و بهینه‌سازی تشعشع یکپارچه[7] [چاونگ[8] و جیانگ[9]، ۲۰۰۷] می‌شود. GSA از این جهت که هر ذره‌ی موجود در جمعیت دارای مکان و سرعت است شبیه PSO است، اما ذرات در GSA دارای شتاب نیز می‌باشند. ذرات بر اساس مقدار وزنشان، که متناسب با مقدار برازندگی‌شان است، یکدیگر را جذب می‌کنند. شکل ۱۷-۱۱ الگوریتم GSA را نشان می‌دهد.

[1] Gravitational Search Algorithm
[2] Rashedi
[3] Central Force Optimization
[4] Farmato
[5] Space Gravitational Optimization
[6] Hsiao
[7] Integrated Radiation Optimization
[8] Chaung
[9] Jiang

یک جمعیت اتفاقی از ذرات را مقداردهی اولیه کن؛ $\{x_i\}$ برای $i \in [1,N]$

سرعت هر ذره را مقداردهی اولیه کن؛ v_i برای $i \in [1,N]$

ثابت جهانی گرانش G_0 و نرخ زوال α را مقداردهی اولیه کن

عدد نسل، $t = 0$ و حد نسل t_{max} را مقداردهی اولیه کن

تا زمانی که شرایط توقف برآورده نشده است

$G \leftarrow G_0 \exp(-\frac{\alpha t}{t_{max}})$ ثابت گرانش

برای هر ذره‌ی x_i، $i \in [1,N]$

$m_i \leftarrow \frac{f(x_i)-\max_k f(x_k)}{\min_k f(x_k)-\max_k f(x_k)} \in [0,1]$

$M_i \leftarrow \frac{m_i}{\sum_{k=1}^{N} m_k}$ برازندگی نرمالیزه شده

ذره‌ی بعد

برای هر ذره‌ی x_i، $i \in [1,N]$

$R_{ik} \leftarrow ||x_k - x_i||_2$ فاصله برای $k \in [1,N]$

$F_{ik} \leftarrow \frac{GM_iM_k}{R_{ik}+\epsilon}(x_k - x_i)$ بردار نیرو برای $k \in [1,N]$

$r_k \leftarrow U[0,1]$ عدد اتفاقی برای $k \in [1,N]$

$a_i \leftarrow \frac{1}{M}\sum_{k=1,k\neq i}^{N} r_k F_{ik}$ بردار شتاب

$r \leftarrow U[0,1]$ عدد اتفاقی

$v_i \leftarrow rv_i + a_i$ بردار سرعت

$x_i \leftarrow x_i + v_i$ بردار مکان

ذره‌ی بعد

شماره‌ی نسل را یک واحد افزایش بده: $t \leftarrow t + 1$

نسل بعد

شکل ۱۷-۱۱ یک الگوریتم جستجوی گرانشی برای مینیمم‌سازی $f(x)$ که در آن x_i، iاُمین راه‌حل نامزد است و ϵ یک ثابت بسیار کوچک برای جلوگیری از وقوع تقسیم بر صفر می‌باشد.

در بیرونی‌ترین حلقه (حلقه‌ی نسل) از شکل ۱۷-۱۱، ابتدا مقدار ثابت گرانشی G به‌روزرسانی می‌شود. نامتغیر بودن G در طبیعت نسبت به زمان بحث‌برانگیز است، به‌صورتی که برخی دانشمندان معتقدند که این ثابت در واقع متغیر با زمان است [ژوفری[1] و همکاران، ۲۰۰۶]. کاهش تدریجی G در GSA باعث کاهش کاوش با جلو رفتن الگوریتم می‌شود. سپس، مقادیر برازندگی را به‌گونه‌ای قرار می‌دهیم که بدترین ذره دارای برازندگی $m_i = 0$ و بهترین ذره دارای برازندگی $m_i = 1$ باشد. مقادیر برازندگی متناظر با وزن گرانشی می‌باشند. بعد از آن، مقادیر نرمالیزه‌ی برازندگی $\{M_i\}$ را به دست می‌آوریم. مجموع مقادیر ترمالیزه برابر ۱ است. سپس، برای هر جفت ذره نیروی جاذبه محاسبه می‌نماییم. بزرگی و دامنه‌ی این نیرو متناسب با مقدار برازندگی و فاصله‌ی بین آن‌ها می‌باشد. در آخر نیز از بردار شتاب برای به روزرسانی مکان و سرعت هر ذره استفاده می‌نماییم. پارامترهای میزان‌سازی معمول در شکل ۱۷-۱۱ $G_0 = 100$ و $\alpha = 20$ می‌باشد.

محققین اصلاحات و تعامیم بسیاری را برای GSA ارائه کرده‌اند. از جمله‌ی آن‌ها می‌توان به راه‌های مختلف برای جایگزین نمودن G در هر نسل اشاره نمود. معادله‌ی شتاب را می‌توان به‌گونه‌ای به‌روزرسانی نمود که تنها بهترین راه‌حل‌ها هر ذره را جذب نمایند:

$$a_i \leftarrow \frac{1}{M_i} \sum_{k \in B, k \neq i} r_k F_{ik} \qquad (۲۰-۱۷)$$

که در آن B مجموعه‌ای شامل بهترین ذرات بوده و اندازه‌ی آن یک پارامتر میزان‌سازی محسوب می‌شود. به علاوه، می‌توان از انواع مختلفی از جرم مؤثر برای نیروی فعال گرانشی، نیروی غیرفعال گرانشی و اینرسی استفاده نمود [راشدی و همکاران، ۲۰۰۹]. نوعی از GSA برای جستجو در دامنه‌های گسسته در [راشدی و همکاران، ۲۰۱۰] ارائه شده است. با توجه به شباهت میان GSA و PSO، به نظر می‌رسد بتوان بسیاری از تعامیم ارائه شده برای PSO را در GSA نیز پیاده‌سازی نمود (فصل ۱۱ را ببینید).

۹-۱۷ جستجوی هارمونی

جستجوی هارمونی (HS[2]) در [گیم[3] و همکاران، ۲۰۰۱] معرفی شده و توضیحات بیشتر در مورد آن در [لی و گیم، ۲۰۰۶] ارائه شده است. HS بر پایه‌ی فرایند موسیقایی قرار دارد. هر نوازنده‌ای در یک گروه موسیقی نت خاصی را در یک دامنه‌ی مجاز به صدا در می‌آورد. اگر همه‌ی نت‌ها با هم هارمونی خوبی ایجاد کند، تجربه‌ی مثبتی در حافظه‌ی جمعی کر (هم‌سرایان) ذخیره شده و احتمال به دست آوردن پیوسته‌ی

[1] Jofre
[2] Harmony Seacrh
[3] Geem

هارمونی خوب افزایش می‌یابد. در HS، یک کر به مثابه‌ی یک راه‌حل نامزد است و یک نوازنده به مثابه‌ی یک متغیر مستقل یا ویژگی راه‌حل می‌باشد.

شکل ۱۷–۱۲ طرح کلی الگوریتم HS را نشان می‌دهد [عمران و مهدوی[1]، ۲۰۰۸]. جستجوی هارمونی معمولاً از نمادگذاری‌های متفاوتی نسبت به نمادگذاری‌های استاندارد الگوریتم‌های تکاملی استفاده می‌کند. برای مثال، از *بردار هارمونی* برای اشاره به ذرات الگوریتم تکاملی با راه‌حل نامزد x، از *اندازه‌ی حافظه‌ی هارمونی* برای اشاره به اندازه‌ی جمعیت N، از *نرخ تفکر حافظه‌ی هارمونی* مشابه نرخ برش c در GAها، از *نرخ تنظیم تن صدا* برای اشاره به نرخ جهش p_m و از *فاصله‌ی پهنای باند* برای اشاره به انحراف استاندارد σ در جهش گاوسی، استفاده می‌شود. مقادیر معمول برای این پارامترها عبارتند از:

- $c = 0.9$
- افزایش خطی p_m از 0.01 در اولین نسل تا 0.99 در آخرین نسل
- کاهش نمایی σ از ۵٪ دامنه‌ی جستجو تا ۰٫۰۱٪ فضای جستجو

از شکل ۱۷–۱۲ می‌توان دید که HS در هر نسل یک فرزند تولید می‌نماید. برای هر ویژگی راه‌حل، یک عدد اتفاقی مانند r_c تولید می‌نماییم. اگر r_c کمتر از نرخ برش c_1 باشد، آنگاه ویژگی راه‌حل در فرزند برابر یک ویژگی راه‌حل اتفاقی از جمعیت قرار داده می‌شود. این گام مانند برش یکنواخت جهانی است (بخش ۸–۸–٦ را ببینید). با این حال، اگر r_m بزرگتر از نرخ برش باشد، آنگاه ویژگی راه‌حل در فرزند برابر یک عدد اتفاقی در دامنه‌ی جستجو قرار داده می‌شود. این گام مانند جهش یکنواخت متمرکز در مرکز محدوده‌ی جستجو می‌باشد (بخش ۸–۹–۲ را ببینید). اگر ویژگی راه‌حل فرزند به جای یک عدد اتفاقی از جمعیت گرفته شده باشد، جهش گاوسی متمرکز را بر روی ویژگی راه‌حل اجرا می‌نماییم (بخش ۸–۹–۳ را ببینید). در آخر، اگر فرزند بهتر از بدترین ذره‌ی موجود در جمعیت باشد، آنگاه فرزند جایگزین آن ذره در جمعیت می‌شود. این آخرین گام همان استراتژی است که در EP از آن استفاده می‌نماییم (بخش ۵–۱ را ببینید).

$p_m =$ نرخ جهش $\in [0,1]$

$\sigma^2 =$ واریانس جهش گاوسی

$c =$ نرخ برش $\in [0,1]$

جمعیت راه‌حل‌های نامزد را مقداردهی اولیه کن؛ $\{x_k\}$ برای $k \in [1, N]$

تا زمانی که شرایط توقف برآورده نشده است

[1] Mahdavi

فرزند $= [0\ \ 0\ \ \ldots\ \ 0] \in R^n$

برای هر یک از ویژگی‌های راه‌حل $s = 1, \ldots, n$

$r_c \leftarrow U[0,1]$

اگر $r_c < c$، آنگاه

$j \leftarrow$ عدد صحیح اتفاقی $\in [1, n]$

فرزند$(s) \leftarrow x_j(s)$

$r_m \leftarrow U[0,1]$

اگر $r_m < p_m$، آنگاه

فرزند$(s) \leftarrow$ فرزند$(s) + N(0, \sigma^2)$

پایان اگر

در غیر این صورت

فرزند$(s) \leftarrow U[x_{min}(s), x_{max}(s)]$

پایان اگر

ویژگی راه‌حل بعد

$m \leftarrow arg\ \max_k (f(x_k): k \in [1, N])$

اگر $f\left(\text{فرزند}\right) < f(x_\square)$، آنگاه

$x_m \leftarrow$ فرزند

پایان اگر

نسل بعد

شکل ۱۷-۱۲ طرح کلی الگوریتم جستجوی هارمونی (HS) با اندازه‌ی جمعیتی برابر N برای مینیمم‌سازی تابع n بعدی $f(x)$. $\{x_k\}$ جمعیت تمام ذرات بوده و x_k، kاُمین ذره بوده و $x_k(s)$، sاُمین ویژگی x_k می‌باشد.

به‌طور خلاصه، به نظر می‌رسد که ایده‌ی جدیدی در الگوریتم HS وجود ندارد. HS ترکیبی است از ایده‌های پیشین در الگوریتم‌های تکاملی مانند بازترکیب یکنواخت جهانی، جهش یکنواخت، جهش گاوسی و جایگزین نمودن بدترین ذره در هر نسل. سهم و مشارکت HS در الگوریتم‌های تکاملی در دو زمینه است. اول آنکه نحوه‌ی ترکیب این ایده‌ها توسط HS بدیع است. ثانیا، انگیزه‌ی موسیقایی این الگوریتم نو و جدید می‌باشد. با این حال، مقالات کمی در زمینه‌ی HS به بحث در مورد انگیزش موسیقایی یا تعامیم آن پرداخته‌اند. بیشتر نشریات و مقالات به ترکیب HS با سایر الگوریتم‌های تکاملی، پارامترهای میزان‌سازی HS و یا اعمال

HS به مسائل خاص می‌پردازند. اگر بتوان تعامیم با انگیزش موسیقایی بیشتری به HS اعمال نمود، می‌توان آن را به‌عنوان الگوریتم تکاملی متمایزی به حساب آورد. چنین تحقیقاتی به مطالعه‌ی نظریه موسیقی، مطالعه فرایند ساخت و تنظیم موسیقی و اعمال خلاقانه‌ی این نظریات به الگوریتم HS، نیاز دارد. برای مطالعه‌ی بیشتر در این زمینه می‌توانید به [گیم، ۲۰۱۰a]، [گیم، ۲۰۱۰b] و کتاب [گیم، ۲۰۱۰c] مراجعه نمایید.

۱۷–۱۰ بهینه‌سازی تعلیم و تعلم محور

بهینه‌سازی تعلیم و تعلم محور (TLBO[1]) در [راو[2] و همکاران، ۲۰۱۱] معرفی و در [راو و همکاران، ۲۰۱۲]، [راو و پاتل[3]، ۲۰۱۲] و [راو و ساوسانی[4]، ۲۰۱۲، فصل ٦] مورد بحث واقع شده است. TLBO بر پایه‌ی فرایند تعلیم و تعلم در کلاس درس واقع شده است. در هر نسل، بهترین راه‌حل نامزد در جمعیت به‌عنوان معلم در نظر گرفته شده و سایر راه‌حل‌های نامزد به‌عنوان دانش‌آموز در نظر گرفته می‌شوند. دانش‌آموزان بیشتر مواقع از معلم آموزش می‌پذیرند، اما از یکدیگر نیز یاد می‌گیرند. در TLBO، یک موضوع درسی به مثابه‌ی یک متغیر مستقل یا ویژگی راه‌حل می‌باشد. مرحله‌ی معلم شامل اصلاح نمودن هر متغیر مستقل $x_i(s)$ در هر راه‌حل نامزد x_i، به‌صورت زیر می‌باشد:

$$c_i(s) \leftarrow x_i(s) + r\left(x_t(s) - T_f \bar{x}(s)\right) \qquad (۲۱–۱۷)$$

که در آن $\bar{x}(s) = \frac{1}{N}\sum_{k=1}^{N} x_k(s)$

فرایند بالا برای همه‌ی $i \in [1, N]$ و همه‌ی $s \in [1, n]$ انجام شده به‌صورتی که N اندازه‌ی جمعیت بوده و n ابعاد مسئله می‌باشد. همچنین، x_t بهترین ذره‌ی موجود در جمعیت (معلم)، r عددی اتفاقی با توزیع یکنواخت بر روی $[0,1]$ و T_f فاکتور آموزش بوده که با احتمال مساوی دارای مقدار ۱ یا ۲ می‌باشد. فرزند c_i در صورتی که بهتر از والد x_i باشد جایگزین آن می‌شود. در کل، معادله‌ی (۱۷–۲۱) مقدار $x_i(s)$ را در جهت بهترین ذره $x_t(s)$ می‌شود. این را می‌توان با گرفتن امید ریاضی از معادله‌ی (۱۷–۲۱) مشاهده نمود:

$$\bar{c}_i(s) = \bar{x}(s) + \frac{1}{2}\left(x_t(s) - \frac{3}{2}\bar{x}(s)\right) \qquad (۲۲–۱۷)$$
$$= \frac{x_t(s)}{2} + \frac{\bar{x}(s)}{4}$$

با توجه به معادله‌ی بالا، $c_i(s)$ به $x_t(s)$ نزدیکتر است تا $x_i(s)$. معادله‌ی (۱۷–۲۲) همچنین نشان می‌دهد که $c_i(s)$ نسبت به جمعیت والد به صفر نزدیکتر است. این ویژگی ممکن است برتری ناعادلانه‌ای را در

[1] Teaching-learning Based Optimization
[2] Rao
[3] Patel
[4] Savsani

مسائلی که راه‌حلشان به صفر نزدیک است، نثار TLBO کند. مسائل محک بسیاری دارای راه‌حل‌هایی در $x^* = 0$ هستند، بنابراین تحقیقات آتی در زمینه‌ی TLBO باید با دقت بسیاری به بررسی عملکرد آن در مسائل با راه‌حل غیرصفر پرداخته و الگوریتم را جهت از بین بردن این گرایش ذاتی، تنظیم نمایند.

پس از کامل شدن مرحله‌ی معلم، مرحله‌ی دانش‌آموز آغاز می‌شود. مرحله‌ی دانش‌آموز مستلزم تنظیم هر ذره بر اساس یک ذره‌ی اتفاقی دیگر می‌باشد:

$$c_i(s) \leftarrow \begin{cases} x_i(s) + r\big(x_i(s) - x_k(s)\big) & \text{اگر } x_k \text{ بهتر از } x_i \text{ است} \\ x_i(s) + r\big(x_k(s) - x_i(s)\big) & \text{در غیر این صورت} \end{cases} \qquad (۱۷-۲۳)$$

برای تمامی $i \in [1, N]$ و $s \in [1, n]$ که در آن k عددی صحیح و اتفاقی در $[1, N]$ بوده به‌طوری که $k \neq i$ و r عددی اتفاقی با توزیع یکنواخت بر روی $[0,1]$ می‌باشد. مرحله‌ی دانش‌آموز $x_i(s)$ را به‌گونه‌ای تنظیم می‌نماید که در صورت بدتر بودن x_k آن را از x_k دور و در صورت بهتر بودن x_k آن را به x_k نزدیک نماید (عبارت دوم از معادله‌ی (۱۷-۲۳)).

شکل ۱۷-۱۳ طرح کلی الگوریتم TLBO را به تصویر می‌کشد. بازرسی شکل ۱۲-۲ نشان می‌دهد که از میان تمامی الگوریتم‌های تکاملی بحث شده در این کتاب، شبیه‌ترین آن‌ها به TLBO، تکامل دیفرانسیلی (DE) می‌باشد. فرض کنید در شکل ۱۲-۲ نرخ برش را برابر $c = 1$ قرار می‌دهیم. همچنین فرض کنید به جای پارامتر ثابت اندازه‌ی گام F از پارامتر اندازه‌ی گامی اتفاقی که برای هر متغیر مستقل متفاوت است استفاده می‌کنیم، به‌طوری که در هر زمان یک متغیر مستقل در بردار جهش v مقداردهی شود. در ضمن فرض کنید که به جای آنکه منتظر تولید تمام فرزندان شویم، هر ذره مانند x_i را بلافاصله بعد از تولید فرزندش با آن جایگزین می‌کنیم. این تغییرات در DE الزاماً ناچیز نیستند بلکه صرفاً سرراست می‌باشند. با اعمال این تغییرات، الگوریتم DE از شکل ۱۲-۲ به الگوریتم DE اصلاح شده‌ی شکل ۱۷-۱٤ تبدیل می‌شود.

جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن؛ $\{x_k\}$ برای $k \in [1, N]$
تا زمانی که شرایط توقف برآورده نشده است
  برای هر x_i؛ $i \in [1, N]$
    کامنت: فاز معلم
    $x_t \leftarrow arg\min_x \big(f(x): x \in \{x_k\}_{k=1}^N\big)$
    $T_f \leftarrow$ عدد صحیح اتفاقی $\in \{1,2\}$
    برای هر یک از ویژگی‌های راه‌حل $s \in [1, n]$
      $\bar{x}(s) \leftarrow \frac{1}{N}\sum_{k=1}^N x_k(s)$

$r \leftarrow U[0,1]$
$c_i(s) \leftarrow x_i(s) + r\left(x_t(s) - T_f \bar{x}(s)\right)$
ویژگی راه‌حل بعدی
$x_i \leftarrow arg\min_{x_i, c_i}\left(f(x_i), f(c_i)\right)$
کامنت: فاز دانش‌آموز
$k \leftarrow$ عدد صحیح اتفاقی $\in [1, N]: k \neq i$
اگر $f(x_i) < f(x_k)$، آنگاه
برای هر یک از ویژگی‌های راه‌حل $s \in [1, n]$
$r \leftarrow U[0,1]$
$c_i(s) \leftarrow x_i(s) + r\left(x_i(s) - x_k(s)\right)$
ویژگی راه‌حل بعدی
در غیر این صورت
برای هر یک از ویژگی‌های راه‌حل $s \in [1, n]$
$r \leftarrow U[0,1]$
$c_i(s) \leftarrow x_i(s) + r\left(x_k(s) - x_i(s)\right)$
ویژگی راه‌حل بعدی
پایان اگر
$x_i \leftarrow arg\min_{x_i, c_i}\left(f(x_i), f(c_i)\right)$
ذره‌ی بعد
نسل بعد

شکل ۱۷-۱۳ طرح کلی الگوریتم تعلیم و تعلم محور (TLBO) با اندازه‌ی جمعیتی برابر N برای مینیمم‌سازی تابع n بعدی $f(x)$. x_t بهترین ذره در جمعیت بوده و با نام *معلم* شناخته می‌شود.

حال فرض کنید به جای تولید اتفاقی r_1، r_2 و r_3 در شکل ۱۷-۱٤، آن‌ها را به قرار زیر مقداردهی کنیم:

$$\begin{aligned} r_1 &= i \\ r_2 &= arg\min_i\{f(x_i) : i \in [1, N]\} \\ x_{r3}(s) &= \text{میانگین } s\text{اُمین ویژگی راه‌حل در جمعیت} \end{aligned} \qquad (۲٤-۱۷)$$

جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن؛ $\{x_k\}$ برای $k \in [1, N]$
تا زمانی که شرایط توقف برآورده نشده است
برای هر x_i؛ $i \in [1, N]$

$r_1 \leftarrow$ عدد صحیح اتفاقی $\in [1, N] : r_1 \neq i$

$r_2 \leftarrow$ عدد صحیح اتفاقی $\in [1, N] : r_1 \neq \{i, r_1\}$

$r_3 \leftarrow$ عدد صحیح اتفاقی $\in [1, N] : r_1 \neq \{i, r_1, r_2\}$

برای هر یک از ویژگی‌های راه‌حل $s \in [1, n]$

$r \leftarrow U[0,1]$

$v(s) \leftarrow x_{r1}(s) + r\big(x_{r2}(s) - x_{r3}(s)\big)$

ویژگی راه‌حل بعدی

$x_i \leftarrow arg\min_{x_i, v}\big(f(x_i), f(v)\big)$

ذره‌ی بعدی

نسل بعد

شکل ۱۷-۱٤ یک الگوریتم تکامل دیفرانسیلی اصلاح شده برای مینیمم‌سازی $f(x)$.

با اعمال این تغییرات اضافی، الگوریتم DE از شکل ۱۷-۱٤ به الگوریتم شکل ۱۷-۱۵ بدل می‌شود. این الگوریتم معادل مرحله‌ی معلم از الگوریتم TLBO در شکل ۱۷-۱۳ با $T_f = 1$ می‌باشد.

به‌طور مشابه، به جای تولید اتفاقی r_1، r_2 و r_3 در شکل ۱۷-۱٤، آن‌ها را به قرار زیر مقداردهی کنیم:

$$\begin{aligned} r_1 &= i \\ r_2 &= arg\min_{i,k}\{f(x_i), f(x_k)\} \\ r_3 &= arg\max_{i,k}\{f(x_i), f(x_k)\} \end{aligned} \qquad (۲۵-۱۷)$$

که در آن k عددی صحیح و اتفاقی در $[1, N]$ بوده به‌طوری که $k \neq i$. بدین ترتیب مرحله‌ی دانش‌آموز از TLBO به دست می‌آید.

به‌طور خلاصه، به نظر می‌رسد ایده‌ی جدیدی در TLBO معرفی نشده است. TLBO تغییریافته‌ی DE است. خود DE نیز نوعی الگوریتم ژنتیک است (بخش ۱۲-٤ را ببینید). سهم و مشارکت TLBO در الگوریتم‌های تکاملی، اجرای DE در دو مرحله است؛ مرحله‌ی معلم و مرحله‌ی دانش‌آموز. همچنین، انگیزه‌ی تعلیم و تعلمی TLBO نوعی بدعت است. با این حال، مقالاتی که در زمینه‌ی TLBO منتشر شده‌اند تا به حال از نظریه‌ی تعلیم و تعلم برای بهبود الگوریتم TLBO استفاده نکرده‌اند. اگر از تعامیم تعلیم و تعلم محور بیشتری به TLBO اعمال شود، می‌توان آن را به‌عنوان یک الگوریتم تکاملی متمایز به حساب آورد. این کار همچنین باعث بهبود عملکرد TLBO خواهد شد. این نوع از تحقیق مستلزم مطالعه‌ی نظریه‌ی یادگیری، سبک‌های یادگیری و اعمال خلاقانه‌ی این نظریات به الگوریتم TLBO می‌باشد. یک موضوع تحقیقاتی مهم

دیگر در زمینه‌ی TLBO، بررسی عملکرد آن در مورد مسائلی است که جوابشان در مرکز دامنه‌ی جستجو واقع نشده است تا بتوان آن را جهت حذف نمودن این گرایش تنظیم نمود. برای نقدهای بیشتر در مورد TLBO می‌توانید به [کرپینسک[1] و همکاران، ۲۰۱۲] و برای اطلاع از جواب‌هایی که به این نقدها داده شده می‌توانید به [واومیر[2]، ۲۰۱۳] مراجعه نمایید.

جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن؛ $\{x_k\}$ برای $k \in [1, N]$

تا زمانی که شرایط توقف برآورده نشده است

برای هر x_i؛ $i \in [1, N]$

$$x_t \leftarrow arg\min_x f(x) : x \in \{x_k\}_{k=1}^{N}$$

برای هر یک از ویژگی‌های راه‌حل $s \in [1, n]$

$$\bar{x}(s) \leftarrow \frac{1}{N}\sum_{k=1}^{N} x_k(s)$$

$$r \leftarrow U[0,1]$$

$$v(s) \leftarrow x_i(s) + r(x_t(s) - \bar{x}(s))$$

ویژگی راه‌حل بعدی

$$x_i \leftarrow arg\,min\big(f(x_i), f(v)\big)$$

ذره‌ی بعدی

نسل بعد

شکل ۱۷-۱۵ یک الگوریتم دیفرانسیل تکاملی اصلاح شده‌ی دیگر برای مینیمم‌سازی $f(x)$. این الگوریتم معادل مرحله‌ی معلم از بهینه‌سازی تعلیم و تعلم محور می‌باشد.

۱۷-۱۱ نتیجه‌گیری

محققین الگوریتم‌های تکاملی زیادی را از زمان اولین الگوریتم ژنتیک در سال ۱۹۵۳ ارائه نموده‌اند [دایسون، ۱۹۹۸، صفحه‌ی ۱۱۱]. به نظر می‌رسد که هر فرایند طبیعی را می‌توان به‌عنوان نوعی الگوریتم بهینه‌سازی در نظر گرفت [آلکساندر، ۱۹۹۶]. در اسن فصل دیدیم که بسیاری از فرایندهای بهینه‌سازی ویژگی‌های الگوریتمی مشابهی دارند. به همین دلیل، فهمیدن اینکه کجا یک الگوریتم تکاملی تمام شده و دیگری آغاز می‌شود، کار سختی است. چه مواقعی یک الگوریتم تکاملی جدید کلاس خود را داشته و چه مواقعی باید آن را در کلاس سایر الگوریتم‌های تکاملی موجود قرار داد؟ یکی از چالش‌های جامعه‌ی محققین

[1] Crepinsek
[2] Waghmare

پیدا کردن این تعادل و تشویق به انجام تحقیقات جدید در حین بالا نگه داشتن استانداردهای معرفی و ایجاد الگوریتم‌های جدید می‌باشد.

ما در این فصل چند الگوریتم تکاملی جدید را مورد بحث قرار دادیم. برخی از آن‌ها محبوب و پراستفاده‌اند اما نمی‌توان آن‌ها را در جای دیگری از این کتاب جای داد. برخی دیگر نسبتاً جدید بوده و میزان استفاده‌ی از آن‌ها توسط مهندسین و دانشمندان علوم کامپیوتر هنوز چندان معلوم نیست. الگوریتم‌های تکاملی بسیار دیگری نیز وجود دارند که وقت کافی برای بحث در مورد آن‌ها را نداشتیم. برخی از این الگوریتم‌ها عبارتند از:

- الگوریتم جامعه و تمدن[1] [ری[2] و لیو[3]، ۲۰۰۳]
- جستجوی سیستم باردارشده[4] [کاوه[5] و طلاطهاری[6]، ۲۰۱۰]
- بهینه‌سازی علف مهاجم[7] [مهرابیان[8] و لوکاس[9]، ۲۰۰۶]
- جستجوی فاخته[10] [یانگ، ۲۰۰۹a]
- قطرات آب هوشمند[11] [شاه‌حسینی[12]، ۲۰۰۷]
- دینامیک تشکیل رودخانه[13] [رابانال[14] و همکاران، ۲۰۰۷]
- جستجوی انتشار اتفاقی[15] [بیشاپ، ۱۹۸۹]
- انطباق گاوسی[16] [جلستروم[17]، ۱۹۶۹]
- الگوریتم بحران بزرگ مهبانگ[18] [ارول[19] و اکسین[20]، ۲۰۰۶]

[1] Society and Civilization Algorithm
[2] Ray
[3] Liew
[4] Charged System Search
[5] Kaveh
[6] Talatahari
[7] Invasive Weed Optimization
[8] Mehrabian
[9] Lucas
[10] Cuckoo Search
[11] Intelligent Water Drops
[12] Shah-Hosseini
[13] River Foundation Dynamic
[14] Rabanal
[15] Stochastic Diffusion Search
[16] Gaussian Adaptation
[17] Kjellstrom
[18] Big Bang Big Crunch Algorithm
[19] Erol
[20] Eksin

- الگوریتم رقابتی امپریالیست[۱] [آتش‌پزگارگری[۲] و لوکاس، ۲۰۰۷]
- بهینه‌سازی چرخ جیرجیری[۳] [ژوزلین[۴] و کلمنتس[۵]، ۱۹۹۹]
- تکامل گرامری[۶] [اُنیل[۷] و رایان[۸]، ۲۰۰۳]
- بهینه‌سازی تجمع کرم شب‌تاب[۹] [کیرشناناند[۱۰] و قوزه[۱۱]، ۲۰۰۹]
- بهینه‌سازی واکنش شیمیایی[۱۲] [لام[۱۳] و لی، ۲۰۱۰]
- گله‌ی کریل[۱۴] [گاندومی[۱۵] و علوی[۱۶]، ۲۰۱۲]
- الگوریتم الهام گرفته شده از خفاش[۱۷] [یانگ، ۲۰۱۰c]
- آستانه پذیرش[۱۸] [دیوک[۱۹] و شویر[۲۰]، ۱۹۹۰]
- الگوریتم طوفان بزرگ و سفر رکورد-به-رکورد[۲۱] [دیوک، ۱۹۹۳]
- مدل کموتاکسی باکتری[۲۲] [مولر و همکاران، ۲۰۰۲] که در بخش ۱۷-۶ نیز به‌طور گذرا به آن اشاره کردیم
- چند الگوریتم زنبور مصنوعی که در بخش ۱۷-۷ به آن‌ها اشاره نمودیم
- چند الگوریتم گرانش و نیرو محور که در بخش ۱۷-۸ به آن‌ها اشاره نمودیم

بدون شک الگوریتم‌های تکاملی دیگری هستند که یا به لیست بالا تعلق دارند و یا حداقل مستحق بحث‌های بیشتر می‌باشند و تنها به دلیل ناآگاهی نویسنده از قلم افتاده‌اند. الگوریتم‌های موجود در لیست بالا به همراه دیگر الگوریتم‌های بحث شده در این فصل می‌توانند گستره‌ی وسیعی از بهره‌وری و سودمندی را

---

[1] Imperialist Competitive Algorithm
[2] Atashpaz-Gargari
[3] Squeaky Wheel Optimization
[4] Joslin
[5] Clements
[6] Grammatical Evolution
[7] O'Neill
[8] Ryan
[9] Glow worm Swarm Optimization
[10] Kirshnanand
[11] Ghose
[12] Chemical Reaction Optimization
[13] Lam
[14] Krill Herd
[15] Gandomi
[16] Alavi
[17] Bat-inspired Algorithm
[18] Threshold Accepting
[19] Dueck
[20] Scheuer
[21] Great Deluge Algorithm and record-to-record Travel
[22] Bacterial Chemotaxis Model

برای دانشجویان و محققین علاقه‌مند به ارمغان آورند. همچنین انواع دیگری از روش‌های محاسباتی وجود دارند که معمولاً به‌عنوان الگوریتم تکاملی طبقه‌بندی نمی‌شوند. اما گاهی مرز میان یادگیری ماشین و بهینه‌سازی کمی فازی به نظر می‌رسد. به همین دلیل است که الگوریتم‌هایی مانند شبکه‌های عصبی [فاوست، ۱۹۹٤]، منطق فازی[1] [روس[2]، ۲۰۱۰]، سیستم دفاعی مصنوعی[3] [هافمیر[4] و فارست[5]، ۲۰۰۰]، زندگی مصنوعی[6] [آدامی[7]، ۱۹۹۷]، محاسبات غشایی[8] [پاوون[9]، ۲۰۰۳] و بسیاری الگوهای محاسباتی دیگر در این کتاب مورد بحث واقع نشده‌اند.

مسائل

تمارین نوشتاری

۱۷-۱ تعریف رکود در معادله‌ی (۱۷-٤) را برای حالتی که مسئله‌ی بهینه‌سازی یک مسئله‌ی ماکزیمم‌سازی باشد، بازنویسی کنید.

۱۷-۲ الگوریتم بنویسید که نسبت به رفتار جست و خیزی مصنوعی ماهی‌ها از شکل ۱۷-۳ ساده‌تر باشد اما همان عملکرد را داشته باشد.

۱۷-۳ آیا بهینه‌ساز جستجوی گروهی نخبه‌گراست؟

۱۷-٤ الگوریتم قورباغه جهنده در هر نسل چند ارزیابی تابع انجام می‌دهد؟

۱۷-۵ حالت خاصی را ارائه دهید که در آن بتوان الگوریتم کرم شب‌تاب از شکل ۱۷-۸ را به‌عنوان حالتی خاص یا تعمیمی از بهینه‌سازی تجمع ذرات از شکل ۱۱-۱ به حساب آورد.

۱۷-٦ بخش ۸-۷ برخی گزینه‌های ممکن را برای فرایند انتخاب در الگوریتم‌های تکاملی مورد بحث قرار می‌دهد. بهینه‌سازی کاوش باکتریایی از چه نوع انتخابی استفاده می‌کند؟

۱۷-۷ شبه کدی برای تولید یک بردار واحد اتفاقی برای بهینه‌سازی کاوش باکتریایی بنویسید.

۱۷-۸ حالت خاصی را ارائه دهید که بتوان در آن الگوریتم کلونی مصنوعی زنبورها از شکل ۱۷-۱۰ را به‌عنوان حالتی خاص یا تعمیمی از الگوریتم دیفرانسیل تکاملی از شکل ۱۲-۲ به حساب آورد.

---

[1] Fuzzy Logic
[2] Ross
[3] Artificial Immune Systems
[4] Hofmeyr
[5] Forrest
[6] Artificial Life
[7] Adami
[8] Membrane Computing
[9] Paun

۱۷-۹ حالت خاصی را ارائه دهید که بتوان در آن الگوریتم جستجوی گرانشی از شکل ۱۷-۱۱ را به‌عنوان حالتی خاص یا تعمیمی از بهینه‌سازی تجمع ذرات از شکل ۱۱-۱ به حساب آورد.

۱۷-۱۰ احتمال اینکه یک فرزند خاص در جستجوی هارمونی کاملاً از ویژگی‌های راه‌حل جهش‌نیافته و از قبل موجود جمعیت والدین تشکیل شود، چه قدر است؟

تمارین کامپیوتری

۱۷-۱۱ یکی از الگوریتم‌های تکاملی معرفی شده در این فصل را شبیه‌سازی کنید. برخی از پارامترها را تغییر دهید تا تأثیر آن‌ها را بر عملکرد الگوریتم تکاملی مشاهده نمایید.

۱۷-۱۲ بهینه‌سازی تعلیم و تعلم محور (TLBO) از شکل ۱۷-۱۳ دارای دو مرحله‌ی معلم و دانش‌آموز است. شبیه‌سازی‌هایی برای سه نوع مختلف از TLBO بنویسید: اولین نوع، الگوریتم اصلی شکل ۱۷-۱۳ می‌باشد، نوع دوم تنها از مرحله‌ی معلم استفاده می‌کند، و نوع سوم تنها از مرحله‌ی دانش‌آموز بهره می‌برد. تابع ۱۰ بعدی اکلی را با اندازه‌ی جمعیتی برابر ۱۰۰ و حد تعداد ارزیابی تابع برابر ۱۰۰۰۰، بهینه نمایید. بهترین عملکرد به دست آمده از سه نوع مختلف TLBO که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است را گزارش کنید. این کار را برای تابع ۱۰ بعدی روزنبروک تکرار کرده و نتایج را توضیح دهید.

فصل هجدهم

بهینه‌سازی ترکیبی

تا به اینجا، تأکید این کتاب بر مسائل بهینه‌سازی با دامنه‌ی پیوسته بوده است. این فصل به بحث در مورد مسائل بهینه‌سازی گسسته می‌پردازد، مسائلی با فرم $min_x f(x)$ که در آن دامنه‌ی x گسسته است. یک مسئله‌ی بهینه‌سازی گسسته که با نام مسئله‌ی بهینه‌سازی ترکیبی نیز شناخته می‌شود، را می‌توان پیدا کردن بهترین شی از میان مجموعه‌ای از اشیاء نامزد تصور نمود:

$$\min_x f(x) \quad \text{به‌طوری که} \quad x \in \{x_1, x_2, \dots, x_{N_x}\} \qquad (۱-۱۸)$$

N_x کاردینالیتی فضای جستجو است. به‌صورت نظری می‌توان با ارزیابی $f(x)$ برای تمامی N_x راه‌حل ممکن اقدام به حل معادله‌ی (۱۸-۱) نمود. این روش برای حل یک مسئله‌ی بهینه‌سازی جستجوی کامل یا brute force نام دارد. با این حال، در اکثر موارد فضای جستجو به قدری بزرگ است که نمی‌توان برای حل مسئله‌ی بهینه‌سازی تمامی راه‌حل‌های ممکن را امتحان نمود.

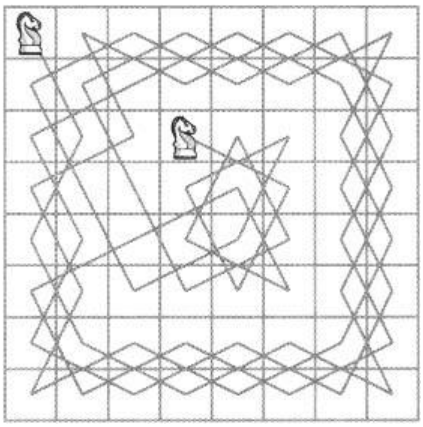
مسئله‌ی سفر شوالیه[۱،۲] یک مسئله‌ی بهینه‌سازی ترکیبی کلاسیک است. این مسئله اولین بار با دیدی ریاضی و توسط لئونارد اویلر[۳] در سال ۱۷۵۹ مورد بحث قرار گرفت [بال[٤] و کوکستر[٥]، ۲۰۱۰]. چطور یک شوالیه (اسب) را بر روی یک صفحه‌ی شطرنج خالی حرکت دهیم به‌طوری که هر اسب از هر خانه دقیقاً یکبار عبور نماید؟ یک مسیر بسته مسیری است که در آن اسب به خانه‌ی ابتدایی خود باز می‌گردد و به همین دلیل در این نوع مسیر به ٦٤ حرکت نیاز دارد، در غیر این صورت مسیر باز بوده و به ٦٣ حرکت نیاز خواهد بود.

مسئله شوالیه‌ی باز را می‌توان به‌صورت $min_x f(x)$ نشان داد به‌طوری که x شامل مکان اولیه و یک سری از ٦٣ حرکت می‌شود. در این صوررت $f(x)$ تعداد خانه‌هایی را که اسب موفق به عبور از آن‌ها نشده است را اندازه خواهد گرفت. کاردینالیتی N_x (که بزرگی فضای جستجو است) بیشتر از 3.3×10^{13} می‌باشد [لوبینگ[٦] و وگنر[٧]، ۱۹۹۵]. ما نمی‌خواهیم با استفاده از روش brute force به حل این مسئله بپردازیم اما چون دارای هوش انسانی هستیم، در حل این مسئله چندان دچار مشکل نخواهیم شد. می‌توان دید که کاردینالیتی فضای جستجو الزاماً نشانگر میزان سختی آن نیست. شکل ۱۸-۱ راه‌حلی را برای مسئله‌ی شوالیه باز نشان می‌دهد. این مسئله همچنین بر روی صفحه‌های شطرنجی با اندازه‌های دیگری به غیر از 8×8 نیز مورد مطالعه قرار گرفته است.

---

[۱] در اینجا شوالیه به معنای مهره‌ی اسب در شطرنج است.

[2] The Knight's Tour Problem
[3] Leonahrd Euler
[4] Ball
[5] Coxeter
[6] Lobbing
[7] Wegener



شکل ۱۸-۱ راه‌حلی برای مسئله‌ی سفر شوالیه باز [فیلی[1]، ۲۰۰۶، صفحه‌ی ۲۳۷].

مروری بر فصل

بیشتر این فصل به مسئله‌ی فروشنده‌ی دوره‌گرد (TSP)، که به احتمال زیاد معروف‌ترین مسئله‌ی بهینه‌سازی ترکیبی است که مطالعات زیادی بر روی آن صورت گرفته است، اختصاص یافته است. بخش ۱۸-۱ دیدی کلی از TSP را ارائه می‌دهد. بخش ۱۸-۲ چند روش غیرتکاملی معروف برای حل TSP را مورد بحث قرار می‌دهد. این روش‌ها از این جهت حائز اهمیت هستند که از ابتکار به کار رفته در آن‌ها برای شروع و یا بهبود جمعیت تکاملی استفاده خواهد شد. در بخش ۱۸-۳ روش‌های مختلف برای ارائه نمودن راه‌حل‌های نامزد TSP و نحوه‌ی ترکیب نمودن راه‌حل‌های نامزد در یک الگوریتم تکاملی برای به دست آوردن فرزندان مورد بحث قرار خواهد گرفت. بخش ۱۸-٤ روش‌هایی را برای جهش راه‌حل‌های نامزد TSP ارائه خواهد داد. بخش ۱۸-۵ مسائل ذکر شده در بخش‌های قبل را به یکدیگر پیوند داده و یک الگوریتم تکاملی پایه‌ای را برای حل TSP ارائه می‌دهد. بخش ۱۸-٦ مسئله‌ی رنگ‌آمیزی گراف[2] را، که یکی دیگر از مسائل بهینه‌سازی ترکیبی معروف می‌باشد، مطرح می‌نماید. توجه داشته باشید که ضمیمه‌ی ج.٦ مسائل محک TSP و سایر مسائل محک ترکیبی را مورد بحث قرار می‌دهد.

۱۸-۱ مسئله‌ی فروشنده‌ی دوره‌گرد

مسئله‌ی شوالیه که پیش از این به بحث در مورد آن پرداختیم، مسئله‌ی سختی نیست اما به مسئله‌ی پیچیده‌تر به اسم فروشنده‌ی دوره‌گرد منجر می‌گردد: کوتاه‌ترین مسیر برای گذشتن از همه‌ی N شهر به‌صورتی

[1] Fealy
[2] Graph Coloring Problem

که از هر شهر دقیقاً یکبار رد شویم، کدام مسیر است؟ مانند مسئله‌ی سفر شوالیه، در TSP نیز اگر به شهر ابتدایی بازگردیم TSP بسته و در غیر این صورت باز خواهد بود. TSP اولین بار در یک جزوه‌ی دستنویس به زبان آلمانی که در سال ۱۸۳۲ منتشر شده بود، مشاهده گردید. عنوان این جزوه، "فروشنده‌ی دوره‌گرد – او چگونه باید باشد و چگونه باید رفتار کند تا به نظم رسیده و در کسب و کار خود راضی و خوشحال باشد – توسط یک فروشنده‌ی دوره‌گرد پیر"، بود. ریاضیدان اتریشی کارل منگر[1] نام "مسئله‌ی پیام‌رسان[2]" را بر این مسئله نهاد و اولین کسی بود که در اواخر دهه‌ی ۱۹۲۰ و اوایل دهه‌ی ۱۹۳۰ به بحث فنی در مورد این مسئله پرداخت [شریور[3]، ۲۰۰۵].

TSP دارای کاربردهایی در روباتیک، ساخت بوردهای الکترونیک، جوشکاری، تولید، حمل‌ونقل و بسیاری زمینه‌های دیگر می‌باشد. همان‌طور که در بخش ۲–۵ بحث نمودیم، یک مسئله‌ی TSP باز با n شهر دارای $(n-1)!$ راه‌حل ممکن است. این عدد حتی برای مقادیر متوسط n بسیار بزرگ خواهد شد. برای مثال، تعداد راه‌حل‌های ممکن برای یک TSP با ۵۰ شهر برابر $49! = 6.1 \times 10^{62}$ است. این در حالی است که ۵۰ شهر برای یک TSP چندان زیاد نیست. یک بورد الکترونیکی ممکن است ده‌ها هزار سوراخ داشته باشد و یک دریل باید طوری برنامه‌ریزی شود که به هر یک از این سوراخ‌ها دقیقاً یکبار سر زده و در عین حال نوعی تابع هزینه (مانند زمان یا انرژی) را مینیمم نماید.

به‌صورت کلی، فرض می‌کنیم شهرهای یک TSP با n شهر را به‌صورت *شهر ۱*، *شهر ۲*، ... ، *شهر* n، نمایش می‌دهند. همچنین فرض می‌کنیم که فاصله‌ی بین شهر i و شهر j، که با $D(i,j)$ نمایش داده می‌شود، برای تمامی $i,j \in [1,n]$ مشخص بوده و $D(i,j) = D(j,i)$ می‌باشد. این یک مسئله‌ی TSP متقارن نامیده می‌شود چرا که فاصله از شهر i تا j با فاصله از شهر j تا i برابر است. می‌توان مواردی را در نظر گرفت که در آن‌ها $D(i,j) \neq D(j,i)$ باشد (برای مثال ممکن است هزینه‌ی سربالایی رفتن بیشتر از هزینه‌ی سرپایینی رفتن باشد). چنین مسائلی TSP نامتقارن نام دارند، اما ما در این فصل بیش از این به این نوع مسائل نمی‌پردازیم.

یک مسیر معتبر در یک TSP باز، مسیری است که در آن از هر n شهر یکبار عبور صورت بپذیرد. یک مسیر معتبر در یک TSP بسته مسیری است که در آن شهر ابتدایی و انتهایی یکی بوده و از همه‌ی $(n-1)$ شهر دیگر دقیقاً یکبار عبور شود. یک مثال از یک مسیر معتبر در یک TSP باز چهار شهری به قرار زیر است:

$$\text{مسیر معتبر چهار شهره باز} : 3 \rightarrow 2 \rightarrow 4 \rightarrow 1 \qquad (۲–۱۸)$$

[1] Karl Menger
[2] The Messenger Problem
[3] Schrijver

یک مثال از یک مسیر معتبر در یک TSP بسته‌ی چهار شهری نیز به قرار زیر است:

$3 \to 2 \to 4 \to 1 \to 3$: مسیر معتبر چهار شهره بسته (۳-۱۸)

یک شاخه، یا لبه، یک تکه از مسیر است. معادله‌ی (۱۸-۲) دارای سه شاخه است: $2 \to 3$، $4 \to 2$ و $1 \to 4$. معادله‌ی (۱۸-۳) دارای چهار شاخه یا لبه است: به‌صورت کلی، یک مسئله‌ی باز با n شهر دارای $(n-1)$ شاخه و یک مسئله‌ی بسته با n شهر دارای n شاخه می‌باشد.

در TSP، در تلاش برای مینیمم‌سازی مسافت کل هستیم. فرض کنید که n شهر در یک TSP باز به‌صورت $x_1 \to x_2 \to \cdots \to x_n$ مرتب شده‌اند. در این صورت، مسافت کل برابرست با

$$D_T = \sum_{i=1}^{N-1} D(x_i, x_{i+1}) \qquad (۴-۱۸)$$

توجه داشته باشید که از واژه‌ی "مسافت" با مفهومی کلی استفاده می‌نماییم. بنابراین، این واژه ممکن است به معنای فاصله‌ی فیزیکی، هزینه‌ی اقتصادی، یا هر کمیت دیگری که در صدد مینیمم‌سازی آن هستیم، باشد.

۱۸-۲ مقداردهی اولیه‌ی TSP

این بخش چند روش غیرتکاملی ابتکاری ساده و محبوب را که می‌توان برای حل TPSها از آن‌ها استفاده نمود، معرفی می‌نماید [نمهاوزر[1] و ولسی[2]، ۱۹۹۹]. ما می‌توانیم از این روش‌ها نه تنها برای جستجوی راه‌حل‌های TSP، بلکه برای مقداردهی اولیه‌ی جمعیت الگوریتم تکاملی، که به منظور حل TSP طراحی گردیده است، استفاده نماییم. اگر الگوریتم تکاملی را به جای مقداردهی اولیه‌ی اتفاقی، به‌صورت هوشمندانه مقداردهی نماییم، شانس خود را برای پیدا کردن راه‌حلی خوب به مقدار زیادی افزایش خواهیم داد. این موضوع نه تنها در مورد TSP بلکه در مورد هر مسئله‌ای که بخواهیم با یک الگوریتم تکاملی حلش نماییم، صادق است (بخش ۸-۱ را ببینید).

الگوریتم‌های این بخش *greedy* نام دارند چرا که همگی تغییراتی افزایشی را در راه‌حل‌هایشان ایجاد می‌نمایند که بهترین تغییر فوری در عملکرد را وعده می‌دهد. بدین معنی که، همه‌ی این الگوریتم‌ها راه‌حل‌ها را بر اساس بیشترین بهبود فوری می‌سازند. روش ارائه شده در بخش ۱۸-۲-۱ راه‌حل‌ها را با اضافه نمودن دوره‌ای شهری که کمترین فاصله را با شهر قبلی دارد، ایجاد می‌کند. روش ارائه شده در بخش ۱۸-۲-۲ نیز

[1] Nemhauser
[2] Wolsey

راه‌حل‌ها را با اضافه نمودن دوره‌ای کوتاه‌ترین شاخه‌ی بعدی می‌سازد. روش ارائه شده در بخش ۱۸-۲-۳ به‌صورت دوره‌ای شهری را که دارای کوتاه‌ترین مسیر با هر یک از شهرهای قبلی باشد، اضافه می‌نماید. در آخر، بخش ۱۸-۲-٤ به بحث در مورد استفاده از اتفاقی‌سازی روش‌های مقداردهی *greedy* می‌پردازد.

۱۸-۲-۱ مقداردهی اولیه‌ی نزدیکترین همسایه

یک روش ساده و حسی برای مقداردهی اولیه یک راه‌حل نامزد، استراتژی نزدیکترین همسایه است. این استراتژی در یک TSP با n شهر به‌صورت زیر تعریف می‌گردد.

۱. مقدار $i = 1$ را قرار بده.
۲. یکی از شهرها $s(1) \in [1, n]$ را به‌صورت اتفاقی انتخاب کن.
۳. $s(i+1) \leftarrow argmin_{\sigma}\{D(s(i), \sigma) : \sigma \notin s(k)$ برای $k \in [1, i]$. بدین معنی که نزدیکترین شهر به $s(i)$ را که هنوز به عنصری از s تخصیص داده نشده است را پیدا و آن را به $s(i+1)$ تخصیص بده.
٤. i را یکی اضافه کن.
۵. اگر $i = n$، به عملیات خاتمه بده، در غیر این صورت به گام ۳ بازگرد.

در انتهای این فرایند دارای مسیری باز به‌صورت $s(1) \rightarrow s(2) \ldots \rightarrow s(n)$، که حدسی مناسب برای یک راه‌حل TSP را به دست می‌دهد، خواهیم بود. اگر یک مسیر بسته بخواهیم، می‌توان به سادگی $s(1)$ را به انتهای مسیر افزود.

به دلیل انتخاب اتفاقی شهر اول، با تکرار این الگوریتم راه‌حل‌های متفاوتی به دست خواهیم آورد. برای نمونه، ماتریس فاصله‌ی زیر را در نظر بگیرید

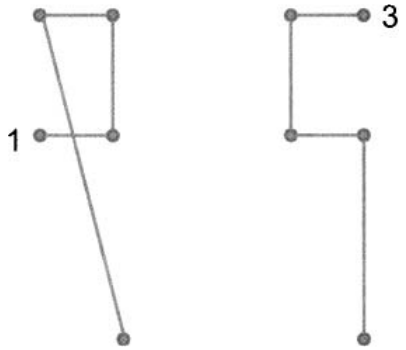
$$D = \begin{bmatrix} \times & 3 & 2 & 9 & 3 \\ 3 & \times & 5 & 8 & 11 \\ 2 & 5 & \times & 4 & 6 \\ 9 & 8 & 4 & \times & 10 \\ 3 & 11 & 6 & 10 & \times \end{bmatrix} \qquad (۵-۱۸)$$

که در آن D_{ij}، که می‌توان آن را به‌صورت $D(i,j)$ نیز نمایش داد، نشانگر فاصله‌ی بین شهر i و j می‌باشد. اگر شهر ۱ را به‌عنوان شهر ابتدایی انتخاب نماییم، الگوریتم نزدیکترین همسایه مسیر $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 5$ را با هزینه‌ی کل ۲۵ نتیجه خواهد داد. اگر شهر ۲ را به‌عنوان شهر ابتدایی انتخاب نماییم، مسیر $2 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5$ با هزینه‌ی کل ۱۹ به دست خواهد آمد.

توجه داشته باشید که ماتریس D که یک ماتریس $n \times n$ فاصله می‌باشد، متقارن است چرا که فاصله از شهر i تا j با فاصله از شهر j تا i برابرست. همچنین، یک ماتریس $n \times n$ دارای $n(n-1)/2$ درایه بالای قطر اصلی می‌باشد. بنابراین، یک TSP با n شهر دارای $n(n-1)/2$ شاخه‌ی متمایز خواهد بود.

اگر بخواهیم یک الگوریتم تکاملی را برای حل TSP به‌صورت هوشمندانه مقداردهی اولیه نماییم، می‌توانیم از مقداردهی نزدیکترین همسایه فقط برای یک ذره از جمعیت و یا برای چندین ذره و یا حتی برای کل جمعیت استفاده نماییم. با این حال، اگر ذرات زیادی را با این روش تولید نماییم، احتمالا ذرات تکراری بسیاری خواهیم داشت. ما همچنین می‌توانیم از نوعی الگوریتم نزدیکترین همسایه‌ی احتمالاتی استفاده نماییم، بدین ترتیب که در آن احتمال تخصیص داده شدن یک شهر به $s(i+1)$ به‌صورت معکوس با فاصله‌ی آن با $s(i)$ متناسب باشد. در آخر نیز می‌توانیم الگوریتم نزدیکترین همسایه را به الگوریتم نزدیکترین دو–همسایه تعمیم دهیم. در این روش، با فرض معلوم بودن $s(i)$، شهری را به $s(i+1)$ تخصیص می‌دهیم که به کوتاهترین فاصله‌ی ترکیبی $D\big(s(i), s(i+1)\big) + D(s(i+1), \sigma)$ منجر شود. σ می‌تواند هر شهری باشد به شرطی که برای $k \leq i+1$، مخالف $s(k)$ باشد.

پیدا کردن مثالی که مقداردهی اولیه‌ی نزدیکترین همسایه در آن عملکرد ضعیفی داشته باشد کار سختی نیست. شکل ۱۸–۲ یک TSP ساده با ۵ شهر را نشان می‌دهد. در سمت چپ شکل ما با شهر ۱ آغاز کرده و نتیجه‌ی ضعیفی را از الگوریتم نزدیکترین همسایه به دست آورده‌ایم. در سمت راست شکل، از شهر ۳ شروع کرده‌ایم و با استفاده از الگوریتم نزدیکترین همسایه به بهینه‌ی جهانی دست پیدا نموده‌ایم. در شکل ۱۸–۲، عملکرد مقداردهی اولیه‌ی نزدیکترین همسایه بسیار به شهر آغازین وابسته است. اما در کل، مقداردهی اولیه‌ی نزدیکترین همسایه به دلیل اینکه هنگام برنامه‌ریزی مسیر به بیشتر از یک شهر در پیش رو نگاه نمی‌کند، معمولاً الگوریتمی ناموفق است.



شکل ۱۸–۲ مقداردهی اولیه‌ی نزدیکترین همسایه برای یک TSP با پنج شهر. بنا بر شهر آغازین، یا نتیجه‌ی سمت راست و یا نتیجه‌ی سمت چپ حاصل خواهد شد.

۱۸–۲–۲ مقداردهی اولیه‌ی کوتاهترین شاخه (لبه)

یک راه ساده‌ی دیگر برای مقداردهی یک راه‌حل نامزد در TSP، استفاده از الگوریتم greedy کوتاه‌ترین شاخه (لبه) می‌باشد. فرض کنید یک TSP با n شهر و با ماتریس فاصله‌ی D از معادله‌ی (۱۸–۵) در اختیار داریم. L_k را به‌صورت شاخه‌ای که با kاُمین عدد کوچک در D مرتبط است، تعریف می‌نماییم. بدین معنی که، $\{L_k\}$ دارای $n(n-1)/2$ شاخه‌ای است که بر اساس فاصله به‌صورت صعودی مرتب شده‌اند. مقداردهی اولیه‌ی کوتاه‌ترین شاخه برای یک TSP بسته به‌صورت زیر خواهد بود:

۱. T را به‌عنوان مجموعه‌ای از شاخه‌ها در مسیر تعریف کن و آن را برابر مجموعه‌ای تهی قرار بده.
۲. کوتاه‌ترین شاخه در $\{L_k\}$ را که شرایط الف تا ج را برآورده می‌سازد پیدا کن:

الف) در T نباشد؛
ب) اگر به T اضافه شود باعث ایجاد مسیری بسته با تعداد شاخه‌ی کمتر از n نشود؛
ج) اگر دو شهر i و j را به هم متصل ساخته و به T اضافه شود، T دارای بیشتر از دو شاخه بین i و j نشود.

۳. اگر T دارای n شاخه است، الگوریتم به پایان رسیده، در غیر این صورت به گام ۲ بازگرد.

مقداردهی اولیه‌ی کوتاهترین شاخه، مسیری ایجاد می‌کند که شامل شاخه‌هایی می‌شود که بین نزدیکترین شهرها به یکدیگر الگوریتم کوتاهترین شاخه احتمالاتی نبوده و به همین دلیل فارق از تعداد دفعات اجرا، همیشه یک نتیجه را به دنبال خواهد داشت. بنابراین، در کل باید از این الگوریتم تنها برای ایجاد یک ذره در جمعیت الگوریتم تکاملی استفاده نمود. تنها حالت استثنا در جایی است که بیش از یک جفت شهر دارای فواصل یکسانی هستند. در این صورت می‌توان از یک فرایند اتفاقی برای انتخاب مسیرهای برابر به لحاظ هزینه در گام ۲ استفاده نمود. بدین صورت و بنا بر نتیجه‌ی فرایند اتفاقی، مسیرهای متفاوتی حاصل خواهد شد.

به‌عنوان مثالی از مقداردهی اولیه‌ی کوتاهترین شاخه، ماتریس فاصله‌ی معادله‌ی (۱۸–۵) را در نظر بگیرید. الگوریتم کوتاهترین شاخه به‌صورت زیر پیش خواهد رفت:

۱. کوتاهترین شاخه بین شهر ۱ و ۳ است، بنابراین این شاخه را در T قرار می‌دهیم.
۲. کوتاه‌ترین شاخه از بین شاخه‌های باقی مانده بین شهرهای ۱ و ۲ و شهرهای ۱ و ۵ است. یکی از این دو شاخه را به‌صورت اتفاقی انتخاب می‌نماییم. فرض کنید شاخه‌ی بین شهرهای ۱ و ۵ انتخاب شده و در T قرار می‌گیرد.

۳. کوتاه‌ترین شاخه از بین شاخه‌های باقی مانده بین شهرهای ۱ و ۲ است، اما شهر ۱ دارای دو شاخه در T است. بنابراین، به دنبال کوتاهترین شاخه‌ی بعدی، که شاخه‌ی بین شهرهای ۳ و ٤ می‌باشد گشته، و آن را در T قرار می‌دهیم.

٤. کوتاه‌ترین شاخه‌ی باقی مانده بین شهرهای ۳ و ۲ است، اما شهر ۳ دارای دو شاخه در T می‌باشد. به دنبال کوتاه‌ترین شاخه‌ی بعدی می‌گردیم. این شاخه بین شهرهای ۳ و ۵ است. اما باز دوباره، شهر ۳ دارای دو شاخه در T است. بنابراین باز به دنبال کوتاه‌ترین شاخه‌ی بعدی می‌گردیم که بین شهرهای ۲ و ٤ است و آن را در T قرار می‌دهیم.

۵. تنها شاخه‌ی باقی مانده که شروط الگوریتم کوتاه‌ترین شاخه را برآورده می‌سازد، شاخه‌ی بین شهرهای ۲ و ۵ است. بنابراین این شاخه را به T اضافه کرده و مسیر کامل می‌شود.

الگوریتم بالا مسیر بسته‌ی $1 \to 5 \to 2 \to 4 \to 3 \to 1$ را نتیجه می‌دهد. اگر بخواهیم از این الگوریتم برای پیدا کردن مسیر باز استفاده نماییم، الگوریتم را پس از به دست آوردن $(n-1)$ شاخه و قرار دادن آن‌ها در T متوقف می‌نماییم. در این صورت مسیر $2 \to 4 \to 3 \to 1 \to 5$ به دست خواهد آمد.

۱۸-۲-۳ مقداردهی اولیه‌ی الحاقی

مقداردهی اولیه‌ی الحاقی ابتدا با یک زیر-مسیر شروع شده و سپس در هر زمان یک شهر به آن اضافه می‌شود به‌طوری که اضافه شدن این شهر کمترین مقدار افزایش در مسافت را نتیجه دهد [روزنکراتز[1] و همکاران، ۱۹۷۷]. زیر-مسافت اولیه معمولاً یک تک شاخه است که در میان شاخه‌ها کوتاه‌ترین می‌باشد. در این حالت، الگوریتم الحاق نزدیکترین را خواهیم داشت که برای TSP باز به‌صورت زیر خواهد بود.

۱. T را به‌عنوان شاخه‌های موجود در مسیر تعریف نمایید و آن را با کوتاهترین شاخه‌ی موجود در ماتریس فاصله مقداردهی اولیه نمایید.

۲. $c \leftarrow argmin_c\{D(c,k): (c \notin T)$ و $(k \in T\}$. بدین معنی که از میان تمام شاخه‌هایی که در T نیستند، شاخه‌ای را انتخاب کن که به T نزدیکترین است.

۳. $\{k,j\} \leftarrow argmin_{k,j}\{(D(k,c)+D(c,j)-D(k,j) : D(k,j) \in T\}$. بدین معنی که شاخه‌ی $D(k,j)$ از T را به‌گونه‌ای انتخاب کن که اختلاف میان مسافت زیر-مسیر $k \to c \to j$ و مسافت $k \to j$ کمترین شود.

٤. $D(k,j)$ را از T حذف و $D(k,c)$ و $D(c,j)$ را به T اضافه کن.

[1] Rosenkratz

۵. اگر T دارای $(n-1)$ شاخه است الگوریتم به پایان رسیده، در غیر این صورت به گام ۲ بازگرد.

اگر یک مسیر بسته بخواهیم، می‌توان به سادگی یک شاخه‌ی دیگر به T اضافه کرد تا مسیر کامل شود. می‌توان الگوریتم الحاق نزدیکترین را با تغییر مقداردهی اولیه در گام ۱ به‌گونه‌ای اصلاح کرد که T به‌صورت اتفاقی یا پوسته‌ای محدب از شهرها و یا به‌صورت‌های متنوع مقداردهی شود. این کار به ما اجازه خواهد داد با استفاده از الگوریتم الحاقی بیش از یک ذره‌ی تکاملی را مقداردهی اولیه نماییم.

به‌عنوان مثالی از مقداردهی اولیه‌ی الحاق نزدیکترین، ماتریس فاصله‌ی معادله‌ی (۱۸-۵) را در نظر بگیرید. الحاق نزدیکترین به‌صورت زیر عمل خواهد کرد.

۱. کوتاهترین شاخه بین شهرهای ۱ و ۳ قرار دارد. بنابراین، این شاخه را در T قرار می‌دهیم.
۲. شهرهایی که هنوز در T نیستند عبارتند از شهرهای ۲، ٤ و ۵. میان این شهرها، نزدیک‌ترینشان به T (که در واقع نزدیکترین شهر به یکی از دو شهر ۱ یا ۳ می‌باشد)، شهر ۲ یا ۵ است، چرا که هر دوی این شهرها ۳ واحد از شهر ۱ فاصله دارند. فرض کنید شهر ۵ به‌صورت اتفاقی انتخاب شده و در T قرار می‌گیرد. حال شاخه‌ی بین شهرهای ۱ و ۳ را از T حذف می‌نماییم و شاخه‌های بین شهرهای ۱ و ۵ و شهرهای ۳ و ۵ را در T قرار می‌دهیم.
۳. شهرهایی که هنوز در T قرار نگرفته‌اند شهرهای ۲ و ٤ می‌باشند. از میان این شهرها نزدیک‌ترینشان به T (که در واقع نزدیکترین شهر به یکی از سه شهر ۱، ۳ یا ۵ می‌باشد)، شهر ۲ است که ۳ واحد با شهر ۱ فاصله دارد. در اینجا دو گزینه در پیش روی ما خواهد بود:

الف) می‌توان شاخه‌ی بین شهر ۱ و ۵ را از T حذف نمود و به جای آن شاخه‌های بین شهرهای ۱ و ۲ و شهرهای ۲ و ۵ را به آن اضافه نمود. این کار مسافت را از ۳ واحد (مسافت شاخه‌ی بین شهرهای ۱ و ۵) به ١٤ واحد (مجموعه مسافت‌های شاخه‌های بین شهرهای ۱ و۲ و شهرهای ۲ و ۵) افزایش خواهد داد. این به معنای افزایش ۱۱ واحدی خواهد بود.

ب) می‌توان شاخه‌ی بین شهر ۳ و ۵ را از T حذف نمود و به جای آن شاخه‌های بین شهرهای ۵ و ۲ و شهرهای ۲ و ۳ را به آن اضافه نمود. این کار مسافت را از ٦ واحد (مسافت شاخه‌ی بین شهرهای ۳ و ۵) به ١٦ واحد (مجموعه مسافت‌های شاخه‌های بین شهرهای ۵ و۲ و شهرهای ۲ و ۳) افزایش خواهد داد. این به معنای افزایش ۱۰ واحدی خواهد بود.

ما گزینه‌ی ب را انتخاب خواهیم نمود چرا که کمترین میزان افزایش را به دنبال دارد. حال T شامل شاخه‌ی بین شهرهای ۱ و ۵، ۵ و ۲ و شهرهای ۲ و ۳ می‌شود.

٤. تنها شهری که هنوز در T قرار ندارد شهر ٤ است. بنابراین، باید شاخه‌ای را به T اضافه نماییم که شامل شهر ٤ می‌شود. برای این کار سه گزینه محتمل است.

الف) می‌توان شاخه‌ی بین شهرهای ۱ و ۵ را از T حذف نمود و به جای آن شاخه‌های بین شهرهای ۱ و ٤ و شهرهای ٤ و ۵ را به آن اضافه نمود. این کار مسافت را از ۳ واحد (مسافت شاخه‌ی شهرهای ۱ و ۵) به ۱۹ واحد (مجموع مسافت‌های شاخه‌های شهرهای ۱ و ٤ و شهرهای ٤ و ۵) افزایش خواهد داد. این به معنای افزایش ١٦ واحدی است.

ب) می‌توان شاخه‌ی بین شهرهای ۲ و ۵ را از T حذف نمود و به جای آن شاخه‌های بین شهرهای ۵ و ٤ و شهرهای ٤ و ۲ را به آن اضافه نمود. این کار مسافت را از ۱۱ واحد (مسافت شاخه‌ی شهرهای ۲ و ۵) به ۱۸ واحد (مجموع مسافت‌های شاخه‌های شهرهای ۵ و ٤ و شهرهای ٤ و ۲) افزایش خواهد داد. این به معنای افزایش ۷ واحدی است.

ج) می‌توان شاخه‌ی بین شهرهای ۲ و ۳ را از T حذف نمود و به جای آن شاخه‌های بین شهرهای ۲ و ٤ و شهرهای ٤ و ۳ را به آن اضافه نمود. این کار مسافت را از ۵ واحد (مسافت شاخه‌ی شهرهای ۲ و ۳) به ۱۲ واحد (مجموع مسافت‌های شاخه‌های شهرهای ۲ و ٤ و شهرهای ٤ و ۳) افزایش خواهد داد. این به معنای افزایش ۷ واحدی است.

می‌توان هر یک از گزینه‌های ب یا ج را انتخاب نمود چرا که هر دو کمترین میزان افزایش در مسافت را نتیجه می‌دهند. فرض کنید گزینه‌ی ج به‌صورت اتفاقی انتخاب شود. در این صورت T شامل شاخه‌های بین شهرهای ۱ و ۵، ۵ و ۲، ۲ و ٤ و شهرهای ۳ و ٤ خواهد شد.

الگوریتم بالا مسیر $3 \to 4 \to 2 \to 5 \to 1$ را نتیجه می‌دهد.

۱۸-۲-۴ مقداردهی اولیه‌ی احتمالاتی (اتفاقی)

تا به اینجه مقداردهی اولیه‌ی نزدیک‌ترین همسایه تنها روش مقداردهی اتفاقی است که در مورد آن بحث کرده‌ایم. با این حال، هر یک از روش‌های مقداردهی را می‌توان به‌گونه‌ای اصلاح نمود که شامل یک عنصر اتفاقی شود. به علاوه، می‌توان مقداردهی اولیه‌ی نزدیک‌ترین همسایه را نیز به‌گونه‌ای اصلاح نمود که اتفاقی‌تر از الگوریتم ارائه شده در بخش ۱۸-۲-۱ باشد. اضافه نمودن رفتار اتفاقی به الگوریتم‌های مقداردهی اولیه علاوه بر جذاب نگاه داشتن آن‌ها، یکی از بنیادی‌ترین ویژگی‌های الگوریتم‌های تکاملی را در آن‌ها به وجود می‌آورد. همچنین این کار به ما اجازه می‌دهد از روش مقداردهی اولیه برای ایجاد بیش از یک ذره‌ی تکاملی استفاده نماییم.

در مقداردهی اولیه‌ی نزدیک‌ترین همسایه (بخش ۱۸-۲-۱) می‌توان گام ۳ "نزدیکترین شهر به $s(i)$ را پیدا کن"، را با انتخاب شهرها، به‌صورتی که احتمال انتخاب شدن با مقدار فاصله از $s(i)$ رابطه‌ی معکوس داشته باشد، جایگزین نمود. این کار بیشترین احتمال برای انتخاب شدن شهری که به $s(i)$ نزدیک‌ترین می‌باشد را نتیجه خواهد داد، اما از سویی نیز برای تمام شهرهایی که در TSP هستند احتمال انتخاب شدن غیرصفری را نتیجه می‌دهد.

در مقداردهی اولیه‌ی کوتاه‌ترین شاخه (بخش ۱۸-۲-۲) می‌توان انتخاب کوتاه‌ترین شاخه‌ای که شرایط را برآورده می‌سازد را با انتخاب شاخه با احتمالی که با طول شاخه فاصله‌ی عکس دارد، جایگزین نمود (این شاخه از میان شاخه‌هایی انتخاب می‌شود که شرایط را برآورده می‌سازند). این کار بیشترین احتمال را برای انتخاب شدن کوتاهترین شاخه را نتیجه داده، اما در عین حال احتمال انتخاب غیرصفری را برای سایر شاخه‌های موجود در TSP نیز نتیجه خواهد داد.

در مقداردهی اولیه‌ی الحاقی (بخش ۱۸-۲-۳) می‌توان خاصیت اتفاقی بودن را به دو گام از گام‌های این الگوریتم افزود. اول، در گام ۲ به جای انتخاب شهری که به T نزدیکترین است، می‌توان شهرها را با احتمالی که با فاصله‌شان از T نسبت عکس دارد انتخاب نمود. این کار بیشترین احتمال برای انتخاب شدن شهری که به T نزدیکترین است را نتیجه می‌دهد اما در عین حال برای سایر شهرها نیز احتمال انتخاب شدنی غیرصفر را به دنبال خواهد داشت. دوم آنکه در گام ۳، به جای انتخاب شاخه‌ای که اختلاف در زیر-شاخه کمترین شود، می‌توان شاخه را با احتمالی که با میزان اختلاف در فواصل زیر-مسیر نسبت عکس دارد، انتخاب نمود. با این کار، بیشترین احتمال برای انتخاب شاخه با کمترین اختلاف مسافت به دست خواهد آمد اما در عین حال احتمال انتخاب سایر شاخه‌ها نیز غیرصفر خواهد بود.

۱۸-۳ نحوه‌ی نمایش و برش TSP

این بخش به بحث در مورد روش‌های مختلف نمایش راه‌حل‌های نامزد در TSP می‌پردازد. ما نحوه‌ی نمایش مسیر (بخش ۱۸-۳-۱)، نمایش مجاورت (بخش ۱۸-۳-۲)، نمایش ترتیبی (۱۸-۳-۳) و نمایش ماتریسی (۱۸-۳-٤) را مورد بحث قرار خواهیم داد. همچنین، چگونگی ترکیب راه‌حل‌های نامزد از طریق برش را در هر یک از این نحوه‌های نمایش مورد بحث قرار خواهیم داد.

۱۸–۳–۱ نمایش مسیر[1]

نمایش مسیر یکی از طبیعی‌ترین راه‌ها برای نمایش یک راه‌حل در TSP می‌باشد. در نمایش مسیر بردار زیر یک مسیر n شهری به‌صورت $x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_n$ را نمایندگی می‌کند:

$$x = [\, x_1 \quad x_2 \quad \ldots \quad x_n] \tag{۶–۱۸}$$

بخش‌های زیر روش‌هایی برای ترکیب والدین و تولید فرزندان در این نوع نمایش، را مورد بحث قرار می‌دهند.

۱۸–۳–۱–۱ برش تطبیق‌یافته جزیی: برش تطبیق‌یافته جزیی (PMX[2]) [گلدبرگ و لینگل[3]، ۱۹۸۵] بر پایه‌ی برش کلاسیک تک–نقطه‌ای که بیشتر در GAها مورد استفاده قرار می‌گیرد (بخش ۸–۸ را ببینید)، بنا شده است. به‌عنوان مثال، دو بردار والد زیر را در نظر بگیرید

$$\begin{matrix} P_1 = [2 & 3 & 4 & 5 & 6 & 1] \\ P_2 = [3 & 2 & 6 & 1 & 4 & 5] \end{matrix} \tag{۷–۱۸}$$

اگر برش تک–نقطه‌ای را در نقطه‌ی میانی این دو بردار انجام دهیم، فرزندان زیر را به دست خواهیم آورد

$$\begin{matrix} c_1 = [2 & 3 & 4 & 1 & 4 & 5] \\ c_2 = [3 & 2 & 6 & 5 & 6 & 1] \end{matrix} \tag{۸–۱۸}$$

این دو فرزند غیر معتبر هستند چرا که c_1 دو بار از شهر ٤ رد شده و از شهر ٦ رد نمی‌شود. c_2 نیز دو بار از شهر ٦ رد شده و از شهر ٤ عبور نمی‌کند. این مشکل را می‌توان به سادگی و با جایگزین کردن یکی از ٤ها در c_1 با ٦، و جایگزین کردن یکی از ٦ها در c_2 با ٤، حل نمود. برای مثال، می‌توان فرزندان معادله‌ی (۸–۱۸) را به‌صورت زیر اصلاح نمود:

$$\begin{matrix} c_1 = [2 & 3 & \mathbf{6} & 1 & 4 & 5] \\ c_2 = [3 & 2 & 6 & 5 & 4 & 1] \end{matrix} \tag{۹–۱۸}$$

در معادله‌ی بالا شهرهایی که پررنگ نوشته شده‌اند، شهرهایی هستند که به‌صورت اتفاقی تعویض شده‌اند تا مسیرهای معتبر به دست آیند.

---

[1] Path Representation
[2] Partially Matched crossover
[3] Lingle

۱۸-۳-۱-۲ برش ترتیبی: برش ترتیبی (OX[1]) قسمتی از مسیر از یک والد را در فرزند کپی می‌نماید [دیویس، ۱۹۸۵]. این کار فرزندی را ایجاد می‌نماید که دارای یک مسیر جزیی است. سپس، برش ترتیبی فرزند را با کپی کردن تکه‌ی باقی‌مانده از یک والد دیگر، کامل می‌نماید. در این فرایند ترتیب نسبی شهرها از والد ۱ و ۲ حفظ می‌شود. برای مثال، فرض کنید والدین زیر را در اختیار داریم

$$P_1 = [9 \quad 2 \quad 3 \quad 8 \quad 4 \quad 5 \quad 6 \quad 1 \quad 7] \\ P_2 = [4 \quad 5 \quad 2 \quad 1 \quad 8 \quad 7 \quad 6 \quad 9 \quad 3] \qquad (۱۰-۱۸)$$

ما یک زیر–مسیر را به‌صورت اتفاقی از والد ۱ انتخاب می‌نماییم. فرض کنید زیربردار $[6\ 5\ 4\ 8]$ از P_1 انتخاب شده است. در این صورت فرزند جزیی زیر به دست خواهد آمد

$$c_1 = [- \quad - \quad - \quad 8 \quad 4 \quad 5 \quad 6 \quad - \quad -] \qquad (۱۱-۱۸)$$

می‌بینیم که c_1 هنوز به شهرهای ۱، ۲، ۳، ۷ و ۹ نیاز دارد. ترتیب این شهرها در P_2 به‌صورت $\{2, 1, 7, 9, 3\}$ می‌باشد. بنابراین، این شهرها را با همان ترتیب که در P_2 دارند به c_1 کپی می‌نماییم. در این صورت خواهیم داشت

$$c_1 = [2 \quad 1 \quad \mathbf{7} \quad 8 \quad 4 \quad 5 \quad 6 \quad 9 \quad 3] \qquad (۱۲-۱۸)$$

در برش ترتیبی، معمولاً فرزند دوم را با جابه‌جا کردن نقش‌های P_1 و P_2 تولید می‌نماییم. در مثال بالا، نسخه‌ی ابتدایی فرزند دوم با کپی شدن یک زیرمسیر از P_2 تولید می‌شود

$$c_2 = [- \quad - \quad - \quad 1 \quad 8 \quad 7 \quad 6 \quad - \quad -] \qquad (۱۳-۱۸)$$

سپس شهرهای ۹، ۲، ۳، ٤ و ٥ با همان ترتیبی که در P_1 دارند به فرزند دوم کپی شده تا این فرزند نیز تکمیل شود:

$$c_2 = [9 \quad 2 \quad \mathbf{3} \quad 1 \quad 8 \quad 7 \quad 6 \quad 4 \quad 5] \qquad (۱٤-۱۸)$$

۱۸-۳-۱-۳ برش چرخشی: برش چرخشی (CX[2]) که در [الیور[3] و همکاران، ۱۹۸۷] معرفی شده است، یک فرزند را از دو والد و به‌گونه‌ای تولید می‌نماید که در حین کامل شدن فرزند با اطلاعات والد دوم،

[1] Order crossover
[2] Cycle crossover
[3] Oliver

بیشترین میزان اطلاعات ترتیب از والد اول حفظ شود. چگونگی این نوع برش با یک مثال بسیار روشن خواهد شد. فرض کنید والدین زیر را در اختیار داریم

$$P_1 = [2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 1]$$
$$P_2 = [4 \quad 5 \quad 2 \quad 1 \quad 6 \quad 3] \qquad (۱۵-۱۸)$$

در این صورت، یک فرزند به‌صورت زیر به وجود می‌آید:

۱. یک اندیس اتفاقی میان ۱ و n انتخاب نمایید. فرض کنید عدد ٤ انتخاب می‌شود. $P_1(4) = 5$، بنابراین، فرزند ابتدا به‌صورت $c = [- - - 5 - -]$ مقداردهی می‌شود.

۲. $P_2(4) = 1$ و شهر ۱ در مکان ششم P_1 قرار دارد. بنابراین فرزند به‌صورت $c = [- - - 5 - 1]$ در می‌آید.

۳. $P_2(6) = 3$ و شهر ۳ در مکان دوم P_1 قراار دارد. بنابراین، فرزند به‌صورت $c = [- 3 - 5 - 1]$ در خواهد آمد.

٤. $P_2(2) = 5$ اما فرزند شهر ۵ را دارد. بنابراین، شهرهای باقی‌مانده را از P_2 به فرزند کپی می‌نماییم. در این صورت فرزند به‌صورت $c = [4\ 3\ 2\ 5\ 6\ 1]$ در خواهد آمد.

ما معمولاً فرزند دوم را با جابه‌جا کردن نقش‌های P_1 و P_2 ایجاد می‌نماییم. شکل ۱۸-۳ عملیات برش چرخشی را نشان می‌دهد. برش چرخشی همواره فرزندانی معتبر را نتیجه می‌دهد.

والد $P_1 = 1$، والد $P_2 = 2$

عددی صحیح و اتفاقی از $s \leftarrow [1, n]$

$r \leftarrow P_1(s)$

فرزند را با مسیری تهی مقداردهی اولیه کن: $C(i) = 0$ برای $i \in [1, n]$

$C(s) \leftarrow r$

تا زمانی که $C(i)$ برای برخی $i \in [1, n]$ برابر صفر است

  $r \leftarrow P_2(s)$

  اگر برای تمامی $i \in [1, n]$، $C(i) \neq r$ باشد، آنگاه

    $s \leftarrow \{i: P_1(i) = r)\}$

    $C(s) \leftarrow r$

  در غیر این صورت

    برای $i = 1$ تا n

اگر $C(i) = 0$ آنگاه $C(i) \leftarrow P_2(i)$

i بعدی

پایان اگر

شهر بعدی

شکل ۱۸-۳ برش چرخشی برای TSP با n شهر.

۱۸-۳-۱-٤ برش ترتیب-محور: برش ترتیب-محور (OBX[1]) نسخه‌ی اصلاح‌شده‌ی برش چرخشی است [سیسوردا، ۱۹۹۱]. برش ترتیب-محور چند محل را در P_1 به‌صورت اتفاقی انتخاب کرده، شهرهای متناظر با آن محل‌ها را در P_2 یافته و آن شهرها را بر اساس ترتیبی که در P_2 دارند، در P_1 مرتب می‌نماید. نتیجه‌ی به دست آمده فرزند خواهد بود. برای مثال، فرض کنید دو والد زیر را در اختیار داریم

$$P_1 = [2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 1]$$
$$P_2 = [4 \quad 5 \quad 2 \quad 1 \quad 6 \quad 3] \qquad (۱۶-۱۸)$$

حال برش ترتیب-محور تعداد مشخصی از نقاط را در P_1 به‌صورت اتفاقی انتخاب می‌نماید. فرض کنید محل‌های ۱، ۳ و ٤ انتخاب شده‌اند. شهرهای موجود در این نقاط در P_2 عبارتند از شهرهای ٤، ۲ و ۱. فرزند ابتدا همه‌ی شهرهای P_1، غیر از شهرهای ٤، ۲ و ۱ را شامل می‌شود:

$$c_1 = [- \quad 3 \quad - \quad 5 \quad 6 \quad -] \qquad (۱۷-۱۸)$$

سپس، شهرهای ٤، ۲ و ۱ را با همان ترتیبی که در P_2 دارند، به فرزند کپی می‌نماییم. بدین ترتیب خواهیم داشت

$$c_1 = [4 \quad 3 \quad 2 \quad 5 \quad 6 \quad 1] \qquad (۱۸-۱۸)$$

معمولاً فرزند دوم را با تعویض نقش‌های P_1 و P_2 ایجاد می‌نماییم. در مثال بالا فرزند دوم در ابتدا دارای همه‌ی شهرهای P_2 به غیر از شهرهای ۲، ٤ و ٥ خواهد بود (چرا که این شهرها در مکان‌های ۱، ۳ و ٤ والد P_1 قرار دارند):

$$c_2 = [- \quad - \quad - \quad 1 \quad 6 \quad 3] \qquad (۱۹-۱۸)$$

[1] Order-Based crossover

سپس شهرهای ۲، ٤ و ۵ را با همان ترتیبی که در P_1 دارند به فرزند دوم کپی می‌نماییم

$$c_2 = [2\ 4\ 5\ \ 1\ \ 6\ \ 3] \qquad (۲۰-۱۸)$$

۱۸-۳-۱-۵ برش Inver-Over: با داشتن دو والد P_1 و P_2، برش inver-over به‌صورت زیر عمل خواهد کرد [تائو[1] و میکالویچ، ۱۹۹۸].

۱. یک محل اتفاقی مانند s را در P_1 انتخاب کن. فرض کن $P_1(s) = r$.

۲. فرض کن که r در محل kاُم از P_2 واقع شده است. بدین معنی که $P_2(k) = r$. شهر آخر (e) را به‌صورت $e = P_2(k+1)$ قرار بده.

۳. ترتیب شهرها میان $P_1(s+1)$ و e در P_1 را عوض کن تا فرزند به دست آید.

برای مثال فرض کنید دو والد زیر را در اختیار داریم.

$$P_1 = [2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 1]$$
$$P_2 = [4\ \ 5\ \ 2\ \ 1\ \ 6\ \ 3] \qquad (۲۱-۱۸)$$

یک مکان اتفاقی مانند s را از P_1 انتخاب می‌نماییم. فرض کنید $s = 4$. می‌توان دید که $P_1(4) = 5$. همچنین می‌توان دید که شهر ۵ در مکان دوم P_2 قرار دارد، بدین معنی که $P_2(2) = 5$. بنابراین $e = P_2(3) = 2$ را به‌عنوان شهر انتهایی قرار می‌دهیم. سپس با معکوس نمودن ترتیب شهرها بین $P_1(5)$ و شهر ۲ در P_1 فرزند زیر را به دست می‌آوریم

$$c = [6\ 5\ 4\ \ 3\ \ 2\ \ 1] \qquad (۲۲-۱۸)$$

اگر در گام ۲، $k = n$ باشد، آنگاه $P_2(k+1)$ تعریف نشده خواهد بود. در چنین حالتی می‌توان از روش‌های خاصی برای ادامه فرایند برش استفاده نمود. برای مثال، می‌توان $e = P_2(k-1)$ قرار داد و یا می‌توان به گام ۱ بازگشت و یک s اتفاقی جدید انتخاب نمود.

[1] Tao

۱۸–۳–۲ نمایش مجاورت[1]

در نمایش مجاورت [گرفنستت[2] و همکاران، ۱۹۸۵]، اگر یک مسیر که با بردار x نمایش داده می‌شود دارای راهی مستقیم از شهر i به شهر j باشد، آنگاه iاُمین عنصر x برابر j خواهد بود. برای مثال، بردار زیر را در نظر بگیرید

$$x = [2 \quad 4 \quad 8 \quad 3 \quad 9 \quad 7 \quad 1 \quad 5 \quad 6] \tag{۱۸–۲۳}$$

در این صورت بردار x دارای تعابیر زیر خواهد بود.

- $x(1) = 2$، بنابراین مسیر دارای شاخه‌ای از شهر ۱ به شهر ۲ است.
- $x(2) = 4$، بنابراین مسیر دارای شاخه‌ای از شهر ۲ به شهر ٤ است.
- $x(3) = 8$، بنابراین مسیر دارای شاخه‌ای از شهر ۳ به شهر ۸ است.
- $x(4) = 3$، بنابراین مسیر دارای شاخه‌ای از شهر ٤ به شهر ۳ است.
- $x(5) = 9$، بنابراین مسیر دارای شاخه‌ای از شهر ۵ به شهر ۹ است.
- $x(6) = 7$، بنابراین مسیر دارای شاخه‌ای از شهر ٦ به شهر ۷ است.
- $x(7) = 1$، بنابراین مسیر دارای شاخه‌ای از شهر ۷ به شهر ۱ است.
- $x(8) = 5$، بنابراین مسیر دارای شاخه‌ای از شهر ۸ به شهر ۵ است.
- $x(9) = 6$، بنابراین مسیر دارای شاخه‌ای از شهر ۹ به شهر ٦ است.

با ترکیب موارد بالا، مسیری که توسط x نمایندگی می‌شود به‌صورت زیر به دست خواهد آمد

$$1 \to 2 \to 4 \to 3 \to 8 \to 5 \to 9 \to 6 \to 7 \to 1 \tag{۱۸–۲٤}$$

یک بردار مانند x که از نمایش مجاورت برای یک TSP با n شهر استفاده می‌نماید دارای خواص زیر است.

- برای تمامی $i \in [1,n]$ ،$x(i) \neq i$
- برای تمامی $j \in [1,n]$، دقیقاً یک $i \in [1,n]$ وجود دارد به‌طوری که $x(i) = j$

خواص بالا برای آنکه x یک مسیر معتبر را به دست بدهد لازم‌اند اما کافی نیستند. برای مثال بردار زیر نامعتبر است

---

[1] Adjacency Representation
[2] Grefenstette

$$x = [2 \quad 1 \quad 8 \quad 3 \quad 9 \quad 7 \quad 4 \quad 5 \quad 6] \tag{۱۸–۲۵}$$

اگر از شهر ۱ شروع کنیم، توالی $1 \to 2 \to 1 \to 2 \to \cdots$ را تا بینهایت تکرار خواهیم کرد و به هیچ شهر دیگری نخواهیم رسید. بخش‌های زیر روش‌هایی برای ترکیب والدین و تولید فرزندان در این نوع نمایش، را مورد بحث قرار می‌دهند.

۱۸–۳–۲–۱ برش کلاسیک: ابتدا روش برشی را مورد بحث قرار می‌دهیم که با نمایش مجاورت همخوانی نداشته و کار نمی‌کند. این همان برش تک–نقطه‌ای در GAهاست (بخش ۸–۸ را ببینید). برای مثال، دو بردار والد زیر را در نظر بگیرید

$$\begin{aligned} P_1 &= [2 \quad 4 \quad 1 \quad 3] \\ P_2 &= [4 \quad 3 \quad 1 \quad 2] \end{aligned} \tag{۱۸–۲۶}$$

اگر برش تک–نقطه‌ای را در نقطه‌ی میانی این دو بردار انجام دهیم، فرزندان زیر به دست خواهند آمد

$$\begin{aligned} c_1 &= [2 \quad 4 \quad 1 \quad 2] \\ c_2 &= [4 \quad 3 \quad 1 \quad 3] \end{aligned} \tag{۱۸–۲۷}$$

c_1 نماینده‌ی مسیر $1 \to 2 \to 4 \to 2$ است. این مسیر نامعتبر است چرا که هیچگاه از شهر ۳ عبور نمی‌کند. توجه داشته باشید که مسیر c_1 از شهر ۲ دو بار عبور می‌نماید. c_2 نیز نماینده‌ی مسیر $1 \to 4 \to 3 \to 1$ بوده و چون از شهر ۲ عبور نمی‌کند نامعتبر است. همچنین این مسیر از شهر ۳ دو بار عبور می‌نماید.

۱۸–۳–۲–۲ برش شاخه‌های متناوب[1]: برش شاخه‌های متناوب با یک برش کلاسیک آغاز شده و سپس مسیرهای نامعتبر را اصلاح می‌نماید [گرفنستت و همکاران، ۱۹۸۵]. برای نمونه، می‌دانیم که c_1 در معادله‌ی (۱۸–۲۷) به دلیل داشتن دو مقدار ۲ و هیچ مقدار از ۳ نامعتبر است. می‌توان یا جایگزین نمودن یکی از مقادیر ۲ با یک مقدار ۳ این مسیر را اصلاح نمود. اگر اولین ۲ را با ۳ جایگزین نماییم خواهیم داشت

$$c_1' = [3 \quad 4 \quad 1 \quad 2] \tag{۱۸–۲۸}$$

این کار عملی نخواهد بود چرا که به چرخه‌ی $1 \to 3 \to 1 \to \cdots$ منتهی می‌شود. در نتیجه می‌توان دومین ۲ را با ۳ عوض کرد. در این صورت خواهیم داشت

$$c_1'' = [2 \quad 4 \quad 1 \quad 3] \tag{۱۸–۲۹}$$

[1] Alternating Edges Crossover

که این بردار مسیر معتبر $3 \to 4 \to 2 \to 1$ را نمایندگی می‌کند. مثال بالا یک برش تک–نقطه‌ای را نشان می‌دهد اما می‌توان برش شاخه‌های متناوب را می‌توان برای برش دو–نقطه‌ای و یا برش‌های چندین–نقطه‌ای نیز به کار برد. اگرچه برش شاخه‌های متناوب مسیرهای معتبری را ایجاد می‌کند، معمولاً مسیرهای خوب را مختل کرده و به همین دلیل در عمل کارکرد مطلوبی ندارد.

۱۸–۳–۲–۳ برش ابتکاری[1]: دلیل نام‌گذاری این نوع برش این است که از حس مشترک برای ترکیب نمودن راه‌حل‌های نامزد استفاده می‌نماید [گرفنستت و همکاران، ۱۹۸۵]. در این نوع برش بهترین شاخه‌ها از دو والد برای ایجاد فرزند ترکیب می‌شوند. مراحل برش ابتکاری به‌صورت زیر است:

۱. یک شهر اتفاقی مانند r را به‌عنوان شهر آغازین انتخاب کن.
۲. شاخه‌های از والدین را که از r خارج می‌شوند مقایسه کن. کوتاه‌ترین شاخه را برای فرزند انتخاب کن.
۳. شهر واقع شده در سر دیگر شاخه که در گام ۲ انتخاب گردید، نقطه‌ی آغازین برای انتخاب شهر بعدی خواهد بود.
٤. اگر شهر انتخاب شده قبلاً در فرزند وجود داشته باشد، شهر انتخاب شده را با یک شهر اتفاقی که در فرزند وجود ندارد، جایگزین کن.
۵. گام ۲ را آنقدر ادامه بده تا فرزند کامل شود.

شکل ۱۸–٤ یک شبه کد برای برش ابتکاری را ارائه می‌دهد. توجه داشته باشید که برش ابتکاری را می‌توان با هر نوع نمایش بردار–محور از TSP به کار برد. شکل ۱۸–٤ را می‌توان به راحتی به بیش از دو والد تعمیم داد. همچنین، می‌توان اصلاحات دیگری را نیز به شکل ۱۸–٤ افزود. برای مثال، به جای انتخاب قاطعانه‌ی r_{min} از $\{r_1, r_2\}$ برای مینیمم ساختن $[d(r,r_1), d(r,r_2)]$، می‌توان r_{min} را به‌صورت احتمالاتی و اتفاقی تعیین نمود. این کار می‌تواند برای مثال مستلزم قرار دادن $r_{min} = r_i$ با احتمالی که با $d(r,r_i)$ برای $i \in [1,2]$ نسبت معکوس دارد، باشد.

والد $P_1 = 1$، والد $P_2 = 2$
شهری اتفاقی در $r \leftarrow [1,n]$
فرزند $C \leftarrow \{r\}$

[1] Heuristic Crossover

تا زمانی که $|C| < n$

از r_i برای نشان دادن شهری که به دنبال r در والد i قرار دارد استفاده کن $(i \in [1,2])$

فاصله از r تا r_i برای $d(r, r_i) = i \in [1,2]$

$r_{min} \leftarrow \min_{\{r_1, r_2\}} [d(r, r_1), d(r, r_2)]$

$r \leftarrow r_{min}$

اگر $r \in C$، آنگاه

شهری اتفاقی در $[1, n]$ به طوری که $r \leftarrow r \notin C$

پایان اگر

$C \leftarrow \{C, r\}$

شهر بعدی

شکل ۱۸-٤ برش ابتکاری برای TSP با n شهر.

به‌عنوان مثالی از برش ابتکاری، فرض کنید یک TSP چهار شهری با ماتریس فاصله‌ی زیر در اختیار داریم.

$$D = \begin{bmatrix} - & 13 & 9 & 15 \\ 13 & - & 4 & 7 \\ 9 & 4 & - & 12 \\ 15 & 7 & 12 & - \end{bmatrix} \qquad (۳۰-۱۸)$$

که در آن D_{ij} فاصله‌ی بین دو شهر i و j است. همچنین فرض کنید دو والد زیر را در اختیار داریم

$$P_1 = [2 \quad 4 \quad 1 \quad 3] \\ P_2 = [4 \quad 3 \quad 1 \quad 2] \qquad (۳۱-۱۸)$$

در این مورد برش ابتکاری به‌صورت زیر عمل خواهد کرد.

۱. یک شهر مانند $r \in [1,4]$ را به‌صورت اتفاقی انتخاب می‌نماییم. فرض کنید که $r = 2$ انتخاب شود.

۲. می‌بینیم که P_1 دارای شاخه‌ی $2 \rightarrow 4$ بوده و $d(2,4) = 7$. همچنین می‌توان دید که P_2 دارای شاخه‌ی $2 \rightarrow 3$ بوده و $d(2,3) = 7$. بنابراین، شاخه‌ی $2 \rightarrow 3$ را برای فرزند انتخاب می‌نماییم. بدین ترتیب $C = \{2,3\}$ خواهد بود.

۳. می‌توان دید که هر دو والد دارای شاخه‌ی $3 \rightarrow 1$ می‌باشند. بنابراین فرزند به‌صورت $C = \{2,3,1\}$ در خواهد آمد.

٤. می‌بینیم که P_1 دارای شاخه‌ی $2 \to 1$ بوده و $d(1,2) = 13$. همچنین می‌توان دید که P_2 دارای شاخه‌ی $4 \to 1$ بوده و $d(1,4) = 15$. بنابراین شاخه‌ی $2 \to 1$ را برای فرزند انتخاب می‌نماییم. اما شهر ۲ قبلاً در C وجود دارد، بنابراین یک شهر اتفاقی که در C وجود ندارد را انتخاب می‌نماییم. فرض کنید که شهر ٤ (که در واقع تنها شهری است که در C وجود ندارد) انتخاب می‌شود. بدین ترتیب $C = \{2,3,1,4\}$ خواهد شد.

۵. حال C کامل شده است و نمایش مجاورت آن به‌صورت $C = [4\ 3\ 1\ 2]$ خواهد بود.

۱۸–۳–۳ نمایش ترتیبی[1]

در نمایش ترتیبی [گرفنستت و همکاران، ۱۹۸۵]، یک مسیر با n شهر به‌صورت بردار زیر نمایش داده می‌شود

$$x = [x_1 \quad x_2 \quad \dots \quad x_n] \tag{۳۲–۱۸}$$

که در آن $x_i \in [1, n-1]$ می‌باشد. بدین معنی که

$$\begin{aligned} &x_1 \in [1, n] \\ &x_2 \in [1, n-1] \\ &x_3 \in [1, n-2] \\ &. \\ &. \\ &. \\ &x_n = 1 \end{aligned} \tag{۳۳–۱۸}$$

فرض کنید لیستی ترتیبی از شهرها در اختیار داریم

$$L = \{1 \quad 2 \quad \dots \quad n\} \tag{۳٤–۱۸}$$

بدین معنی که برای $i \in [1, n]$، $L(i) = i$. در نمایش ترتیبی، x_1 نمایشگر شهر اول در مسیر می‌باشد. x_2 نیز اندیس شهر دوم مسیر در مجموعه‌ی $L_2 = L \backslash x_1$ را به دست می‌دهد[2]. x_3 نیز اندیس شهر سوم مسیر در مجموعه‌ی $L_3 = L \backslash \{x_1, x_2\}$ را به دست می‌دهد. در کل x_k نیز اندیس شهر kاُم مسیر در مجموعه‌ی $L_k = L \backslash \bigcup_{i=1}^{k-1} x_i$ را به دست می‌دهد. توجه داشته باشید که هر فرم برداری از معادله‌ی (۱۸–۳۳)، یک مسیر معتبر به دست خواهد داد.

---

[1] Ordinal Representation

[2] توجه داشته باشید که ما از نماد $A \backslash B$ برای اشاره به مجموعه‌ی $\{x : x \in A \text{ و } x \notin B\}$ استفاده می‌نماییم. بنابراین، $A \backslash B$ به معنای مجموعه‌ی تمام عضوهایی از A است که عضو B نیستند.

برای مثال فرض کنید یک مسیر ٦ شهری به‌صورت زیر نمایش داده شود

$$x = [5 \quad 2 \quad 4 \quad 1 \quad 2 \quad 1] \tag{۳۵-۱۸}$$

با توجه به داشتن لیست ترتیبی $L = \{1, 2, 3, 4, 5, 6\}$، مسیر نمایش داده شده توسط x را به‌صورت زیر می‌سازیم.

۱. $x_1 = 5$ و $L(5) = 5$، بنابراین شهر ۵ اولین شهر در مسیر خواهد بود. حال ۵ را از L حذف کرده و $L_2 = \{1, 2, 3, 4, 6\}$ را ایجاد می‌نماییم.

۲. $x_2 = 2$ و $L_2(2) = 2$، بنابراین شهر ۲ دومین شهر در مسیر خواهد بود. حال ۲ را از L_2 حذف کرده و $L_3 = \{1, 3, 4, 6\}$ را ایجاد می‌نماییم.

۳. $x_3 = 4$ و $L_3(4) = 6$، بنابراین شهر ٦ سومین شهر در مسیر خواهد بود. حال ٦ را از L_3 حذف کرده و $L_4 = \{1, 3, 4\}$ را ایجاد می‌نماییم.

٤. $x_4 = 1$ و $L_4(1) = 1$، بنابراین شهر ۱ چهارمین شهر در مسیر خواهد بود. حال ۱ را از L_4 حذف کرده و $L_5 = \{3, 4\}$ را ایجاد می‌نماییم.

۵. $x_5 = 2$ و $L_5(2) = 4$، بنابراین شهر ٤ پنجمین شهر در مسیر خواهد بود. حال ٤ را از L_5 حذف کرده و $L_6 = \{3\}$ را ایجاد می‌نماییم.

٦. $x_6 = 1$ و $L_6(1) = 3$، بابراین شهر ۳ ششمین شهر در مسیر است.

بدین ترتیب مسیر $3 \to 4 \to 1 \to 6 \to 2 \to 5$ خواهد بود.

فرض کنید می‌خواهیم از برش تک-نقطه‌ای برای ترکیب نمودن والدین و به دست آوردن فرزندان استفاده نماییم. برای این کار والدین زیر را در نظر بگیرید.

$$\begin{aligned} P_1 &= [5 \quad 2 \quad 4 \quad 1 \quad 2 \quad 1] \\ P_2 &= [1 \quad 5 \quad 3 \quad 3 \quad 1 \quad 1] \end{aligned} \tag{۳٦-۱۸}$$

اگر نقطه‌ی برش را در نقطه‌ی میانی والدین انتخاب نماییم، فرزندان زیر به دست خواهند آمد.

$$\begin{aligned} c_1 &= [5 \quad 2 \quad 4 \quad 3 \quad 1 \quad \mathbf{1}] \\ &= 5 \to 2 \to 6 \to 4 \to 1 \to 3 \\ c_2 &= [1 \quad 5 \quad 3 \quad 1 \quad 2 \quad 1] \\ &= 1 \to 6 \to 4 \to 2 \to 5 \to 3 \end{aligned} \tag{۳۷-۱۸}$$

هر دو فرزند مسیرهایی معتبر هستند. اگرچه نمایش ترتیبی در ابتدا کمی عجیب به نظر می‌رسد، فایده‌ی آن این است که برش تک-نقطه‌ای بر روی آن همواره مسیرهای معتبری را نتیجه خواهد داد.

۱۸–۳–۴ نمایش ماتریسی[1]

در نمایش ماتریسی، یک مسیر با n شهر به‌صورت یک ماتریس M با ابعاد $n \times n$ نمایش داده می‌شود. درایه‌های این ماتریس تنها ۱ یا ۰ هستند [فاکس[2] و مک‌ماهون[3]، ۱۹۹۱]. $M_{ik} = 1$ خواهد بود اگر و فقط اگر شهر i در مسیر قبل از شهر k قرار داشته باشد. برای مثال، ماتریس زیر را در نظر بگیرید

$$M = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (۳۸–۱۸)$$

یک‌های موجود در سطر اول نشان‌دهنده‌ی آنند که شهر ۱ قبل از شهرهای ۲، ٤ و ۵ قرار دارد. یک‌های موجود در سطر دوم نیز حاکی از آنند که شهر ۲ قبل از شهرهای ٤ و ۵ قرار دارد. یک‌های سطر سوم نشان‌دهنده‌ی آنند که شهر ۳ قبل از شهرهای ۱، ۲، ٤ و ۵ قرار دارد. همچنین نحوه‌ی قرارگیری یک‌ها در سطر چهارم گویای آن است که شهر ٤ تنها قبل از شهر ۵ قرار دارد. در آخر سطر پنجم تماماً دارای درایه‌های صفر است، بدین معنی که شهر آخر بوده و قبل از هیچ شهری واقع نشده است. بنابراین و با توجه به آنچه که گفته شد، معادله‌ی (۱۸–۳۸) مسیر $3 \to 1 \to 2 \to 4 \to 5$ را نمایندگی می‌کند.

یک تعبیر دیگر از معادله‌ی (۱۸–۳۸) آن است که سطری که دارای بیشترین تعداد یک است، اولین شهر است، سطری که دارای دومین تعداد یک است شهر دوم بوده و الی آخر. بنابراین، شهری که از نظر تعداد یک در مقام kاُم باشد، شهر kاُم خواهد بود.

حال باید به چند ویژگی ماتریس M که دارای ابعاد $n \times n$ است و یک مسیر معتبر را نمایندگی می‌کند، دقت نماییم.

- دقیقاً یک سطر از M دارای $(n-1)$تا یک بوده، دقیقاً یک سطر از M دارای $(n-2)$تا یک بوده و الی آخر.
- با استفاده از خاصیت بالا می‌توان تعداد یک‌های موجود در M را محاسبه نمود:

$$\text{تعداد یک‌ها} = \sum_{i}^{n} (n-i) = n(n-1)/2 \qquad (۳۹–۱۸)$$

- هیچ شهری در مسیر پیش از خودش قرار ندارد، بدین معنی که به ازای هر $i \in [1,n]$، $M_{ii} = 0$.

[1] Matrix Representation
[2] Fox
[3] McMahon

- اگر شهر i قبل از شهر j و شهر j نیز قبل از شهر k قرار داشته باشد، آنگاه شهر i قبل از شهر k قرار خواهد داشت:

$$\left(\mathrm{M_{ij}} = 1 \text{ و } \mathrm{M_{jk}} = 1\right) \Rightarrow \mathrm{M_{ik}} = 1 \qquad (۴۰-۱۸)$$

بخش‌های زیر به بحث در مورد چند روش برای ترکیب نمودن ماتریس‌های والد جهت ایجاد فرزندان می‌پردازند. می‌توان برای این کار از اشتراک (تلاقی) دو ماتریس و یا اجتماع آن‌ها استفاده نمود.

۱۸-۳-٤-۱ برش اشتراکی[1]: برش اشتراکی را با یک مثال توضیح می‌دهیم. فرض کنید که معادله‌ی (۱۸-۳۸) والد M_1 و معادله‌ی زیر والد دوم را نشان بدهد

$$M_2 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad (۴۱-۱۸)$$

والد M_2 معرف مسیر $4 \to 1 \to 2 \to 5 \to 3$ می‌باشد. اشتراک دو ماتریس M_1 و M_2 را با اجرای AND منطقی بین درایه‌های متناظر این دو ماتریس، به دست می‌آوریم. این کار فرزندی به دست می‌دهد که به‌صورت جزیی تعریف شده است

$$M_c = M_1 \wedge M_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (۴۲-۱۸)$$

این فرزند معرف یک مسیر معتبر نیست، اما حاکی از آن است که شهر ۱ قبل از شهرهای ۲ و ۵ بوده و شهر ۲ و ٤ نیز قبل از شهر ۵ می‌باشند. دلیل وجود این ترتیب آن است که این ترتیب در هر دو والد وجود دارد. در حقیقت، این تنها ترتیبی است که در هر دو والد مشترک است. حال می‌توانیم به‌صورت شبه اتفاقی، یک‌هایی را به ماتریس M_c اضافه نماییم تا یک مسیر معتبر به دست آید (بدین معنی که ماتریس نهایی تمام خواصی را که در بالا برشمردیم برآورده سازد). برای مثال، ممکن است یک‌ها را به‌گونه‌ای به M_c اضافه نماییم که تبدیل ماتریس زیر شود

[1] Intersection Crossover

$$M_c = \begin{bmatrix} 0 & 1 & \mathbf{1} & \mathbf{1} & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & 0 & 1 \\ 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \end{bmatrix} \qquad (۱۸-۴۳)$$

در ماتریس بالا یک‌های اضافه شده به‌صورت پررنگ نمایش داده شده‌اند. حالا M_c تمام خواص لازم برای یک مسیر معتبر را برآورده می‌سازد. مسیر متناظر با این ماتریس مسیر $1 \to 4 \to 5 \to 2 \to 3$ می‌باشد.

۱۸-۳-٤-۲ برش اجتماع[1]: برش اجتماع را با یک مثال توضیح می‌دهیم. فرض کنید معادلات (۱۹-۳۸) و (۱۸-٤۱) معرف والدین M_1 و M_2 باشد. اجتماع دو ماتریس M_1 و M_2 را با اجرای OR منطقی بین درایه‌های متناظر این دو ماتریس، به دست می‌آوریم. این کار فرزندی به دست می‌دهد که به‌صورت جزیی تعریف شده است.

$$M_c = M_1 \vee M_2 = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad (۱۸-٤٤)$$

حال یک "نقطه‌ی برش" اتفاقی را، که M_c را به ٤ ربع (نه الزاماً با اندازه‌های مساوی) تقسیم می‌کند، انتخاب می‌نماییم. فرض کنید این نقطه در محل تلاقی سطر و ستون دوم انتخاب شود. قسمت بالا-چپ و پایین-راست M_c را بدون تغییر نگه می‌داریم، اما فرض می‌کنیم قسمت بالا-راست و پایین-چپ دارای درایه‌هایی تعریف نشده باشند:

$$M_c = M_1 \vee M_2 = \begin{bmatrix} 0 & 1 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ \times & \times & 0 & 1 & 1 \\ \times & \times & 1 & 0 & 1 \\ \times & \times & 1 & 0 & 0 \end{bmatrix} \qquad (۱۸-٤۵)$$

حال باید تغییرات لازم برای از بین بردن تناقضات در M_c را انجام دهیم. برای مثال، $M_{c34} = 1$ و $M_{c43} = 1$، بنابراین باید یکی از این درایه‌ها تبدیل به صفر شود. به همین ترتیب، $M_{c35} = 1$ و $M_{c53} = 1$، بنابراین یکی از این درایه‌ها باید تبدیل به صفر شود. با انجام این دو کار، فرزند تصحیح شده (اما ناقص) زیر به دست خواهد آمد

[1] Union Crossover

$$M_c = \begin{bmatrix} 0 & 1 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ \times & \times & 0 & 0 & 0 \\ \times & \times & 1 & 0 & 1 \\ \times & \times & 1 & 0 & 0 \end{bmatrix} \quad (۱۸-۴۶)$$

حال به‌صورت شبه اتفاقی یک‌هایی را به درایه‌های غیرقطری M_c اضافه می‌نماییم تا معرف یک مسیر معتبر شود (بدین معنی که خواصی که پیش از این برشمردیم را برآورده سازد). برای مثال، ممکن است یک‌ها را به‌گونه‌ای به M_c اضافه نماییم که ماتریس M_c به‌صورت زیر در بیاید

$$M_c = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad (۱۸-۴۷)$$

حال M_c تمام خواص یک مسیر معتبر را برآورده ساخته و مسیر $1 \to 2 \to 4 \to 5 \to 3$ را نمایندگی می‌نماید.

۱۸-۴ جهش TSP

این بخش چند روش برای ایجاد جهش در راه‌حل‌های TSP را مورد بحث قرار می‌دهد. در این بخش ما خود را به نمایش مسیر محدود می‌نماییم (بخش ۱۸-۳-۱ را ببینید). با این حال، روش‌های متنوعی برای سایر نمایش‌ها وجود دارد. همچنین، هر نوع نمایشی را می‌توان ابتدا به نمایش مسیر تبدیل کرده و سپس یکی از روش‌های این بخش را برای ایجاد جهش در آن استفاده نمود.

۱۸-۴-۱ واژگونی[1]

عمل واژگونی، ترتیب شهرهای بین دو اندیس اتفاقی را عکس می‌کند [فوگل، ۱۹۹۰]. برای مثال، راه‌حل x می‌تواند به ترتیب زیر جهش کرده و به x_m تبدیل شود:

$$\begin{aligned} x &= 1 \to \underbrace{5 \to 4 \to 7 \to 6} \to 2 \to 3 \\ x_m &= 1 \to \overbrace{6 \to 7 \to 4 \to 5} \to 2 \to 3 \end{aligned} \quad (۱۸-۴۸)$$

در بالا نقاط ابتدایی و پایانی جهش به‌صورت اتفاقی انتخاب می‌شوند. عمل واژگونی گاهاً با نام جهش ۲-opt نیز شناخته می‌شود [بیر و اشوفل، ۲۰۰۲]. به‌طور کلی، برای پیاده‌سازی عمل واژگونی در یک TSP

[1] Inversion

با n شهر، $n(n-1)/2$ راه وجود دارد. از میان تمامی واژگونی‌های محتمل برای یک TSP با n شهر، کم هزینه‌ترینشان همواره مسیری را نتیجه می‌دهد که دارای شاخه‌های متقاطع نیست [باک و همکاران، ۱۹۹۷a].

۱۸-۴-۲ الحاق[1]

عمل الحاق، شهری که در مکان i قرار دارد را به مکان k جابه‌جا می‌کند، به‌طوری که i و k به‌صورت اتفاقی انتخاب می‌شوند [فوگل، ۱۹۸۸]. برای مثال، فرض کنید مسیر x که در معادله‌ی (۱۸-٤۸) نشان داده شده است را در اختیار داریم. همچنین فرض کنید که i و k به‌صورت اتفاقی برابر ٤ و۲ انتخاب می‌شوند. در این صورت شهر ۷ را که در مکان ٤ قرار دارد به مکان ۲ انتقال می‌دهیم تا مسیر جهش‌یافته حاصل شود

$$x_m = 1 \to 7 \to 5 \to 4 \to 6 \to 2 \to 3 \qquad (۱۸-٤۹)$$

عمل الحاق گاهاً با نام جهش or-opt نیز شناخته می‌شود [بیر و اشوفل، ۲۰۰۲].

۱۸-۴-۳ جابه‌جایی[2]

جابه‌جایی تعمیمی از الحاق است [مایکلویچ، ۱۹۹٦، فصل ۱۰]. در عمل جابه‌جایی، یک توالی از q شهر که از مکان iاُم شروع می‌شود، به مکان kاُم منتقل می‌شود. q، i و k به‌صورت اتفاقی انتخاب می‌گردند. برای مثال، فرض کنید مسیر x را، که در معادله‌ی (۱۸-٤۸) نشان داده شده است، در اختیار داریم. فرض کنید به‌صورت اتفاقی $q = 2$، $i = 4$ و $k = 2$ انتخاب شده‌اند. حال توالی دو شهر را که در مکان ٤ آغاز می‌شوند (شهر ۷ و ٦)، انتخاب کرده و آن‌ها را به مکان ۲ انتقال می‌دهیم. در این صورت خواهیم داشت:

$$x_m = 1 \to 7 \to 6 \to 5 \to 4 \to 2 \to 3 \qquad (۱۸-۵۰)$$

عمل جابه‌جایی گاهی با نام شیفت دادن[3] نیز شناخته می‌شود [بیر و اشوفل، ۲۰۰۲]. می‌توان عمل جابه‌جایی را با عمل واژگونی ترکیب نمود، بدین ترتیب که قبل از انتقال دادن توالی شهرها به مکان جدید، ترتیب آن‌ها را عکس نمود.

[1] Insertion
[2] Displacement
[3] Shifting

۱۸–۴–۴ تبادل هم‌پاسخ[1]

تبادل هم‌پاسخ، جای شهرهایی که در مکان‌های i و k واقع شده‌اند را با یکدیگر عوض می‌نماید (i و k به‌صورت اتفاقی انتخاب می‌شوند) [بنژاف، ۱۹۹۰]. برای مثال، فرض کنید مسیر x که در معادله‌ی (۱۸–٤۸) نشان داده شده است را در اختیار داریم. همچنین فرض کنید که i و k به‌صورت اتفاقی به ترتیب برابر ۵ و ۱ انتخاب شده‌اند. حال جای شهرهای واقع در مکان‌های ۱ و ۵ را با هم تعویض می‌نماییم تا مسیر جهش‌یافته‌ی زیر حاصل شود

$$x_m = 6 \rightarrow 5 \rightarrow 4 \rightarrow 7 \rightarrow 1 \rightarrow 2 \rightarrow 3 \qquad (۵۱–۱۸)$$

تبادل هم‌پاسخ گاهاً با نام جهش ۲–تعویضه[2] نیز شناخته می‌شود [بیر و اضوفل، ۲۰۰۲]. می‌توان این روش را با جابه‌جا نمودن یک توالی از شهرها به جای دو شهر تنها، تعمیم داد. همچنین، می‌توان این روش را با عمل واژگونی نیز ترکیب نمود بدین ترتیب که ترتیب یکی از توالی‌ها را پیش از تعویض با توالی دیگر، عکس نمود.

۱۸–۵ الگوریتمی تکاملی برای مسئله‌ی فروشنده‌ی دوره‌گرد

با داشتن زمینه‌ای که از بخش‌های قبل به دست آوردیم، می‌توانیم یک الگوریتم تکاملی پایه برای حل TSP را ارائه نماییم. این الگوریتم در شکل ۱۸–۵ نشان داده شده است. در پیاده‌سازی شکل ۱۸–۵ گزینه‌های زیادی پیش رو داریم.

- همان‌طور که در بخش ۱۸–۲ بحث نمودیم، برای مقداردهی اولیه‌ی جمعیت گزینه‌های زیادی را در اختیار داریم.
- همان‌طور که در بخش ۱۸–۳ نشان دادیم، برای انجام عمل برش نیز چند گزینه را در اختیار داریم. همچنین می‌توان از بیش از یک روش برش استفاده نمود و از یک نسل به نسل دیگر روش برش را به‌صورت احتمالاتی تغییر داد. به علاوه، می‌توان دید که کدام روش برش بهترین نتیجه را به دست داده و بدین ترتیب تناوب روش‌های برش را بر اساس میزان برازندگی فرزندان تنظیم نمود.
- همان‌طور که در بخش ۱۸–٤ نشان دادیم، برای عمل جهش نیز چند گزینه در پیش رو داریم. همچنین می‌توان از بیش از یک روش جهش استفاده نمود و مانند روش‌های برش، روش جهش را نیز از یک نسل به نسل دیگر به‌صورت احتمالاتی تغییر داد. به علاوه، می‌توان دید که کدام روش جهش بهترین

[1] Reciprocal Exchange
[2] 2-Excchange Mutation

نتیجه را به دست داده و بدین ترتیب تناوب روش‌های جهش را بر اساس میزان برازندگی فرزندان تنظیم نمود.

- در شکل ۱۸–۵ باید روش انتخاب والدین را تعیین نماییم. برای این کار می‌توانیم از هر یک از روش‌های انتخابی که در بخش ۸–۷ در مورد آن‌ها بحث کردیم استفاده نماییم.

```
نرخ جهش = p_m
N راه‌حل نامزد را مقداردهی اولیه کن (بخش ۱۸–۲ را ببینید)
راه‌حل‌های نامزد را با استفاده از نحوه‌ی نمایش مطلوب نشان دهید
    نمایش مسیر، مجاورت، ترتیبی و یا ماتریسی
مسافت مسیر را برای هر یک از راه‌حل‌های نامزد محاسبه کن
تا زمانی که شرایط توقف برآورده نشده است
    برای k = 1 تا N
        والدینی را از {x_i} جهت تولید یک فرزند انتخاب کن
        فرزند C_k را با استفاده از روش‌های برشی که پیش از این بحث شد تولید کن:
        اگر از نمایش مسیر استفاده می‌شود:
            از روش برش بحث شده در بخش ۱۸–۳–۱ استفاده کن
        در غیر این صورت اگر از نمایش مجاورت استفاده می‌شود:
            از روش برش بحث شده در بخش ۱۸–۳–۲ استفاده کن
        در غیر این صورت اگر از نمایش ترتیبی استفاده می‌شود:
            از روش برش بحث شده در بخش ۱۸–۳–۳ استفاده کن
        در غیر این صورت اگر از نمایش ماتریسی استفاده می‌شود:
            از روش برش بحث شده در بخش ۱۸–۳–٤ استفاده کن
        پایان اگر
        r ← U[0,1]
        اگر r < p_m آنگاه
            C_k را با استفاده از یکی از روش‌های بخش ۱۸–٤ دچار جهش کن
        پایان اگر
```

مسافت مسیر را برای C_k محاسبه کن
فرزند بعدی
ذرات تکراری در $\{x_i\} \cup \{C_i\}$ را جایگزین کن
بهترین N ذره از $\{x_i\} \cup \{C_i\}$ $\leftarrow \{x_i\}$
نسل بعد

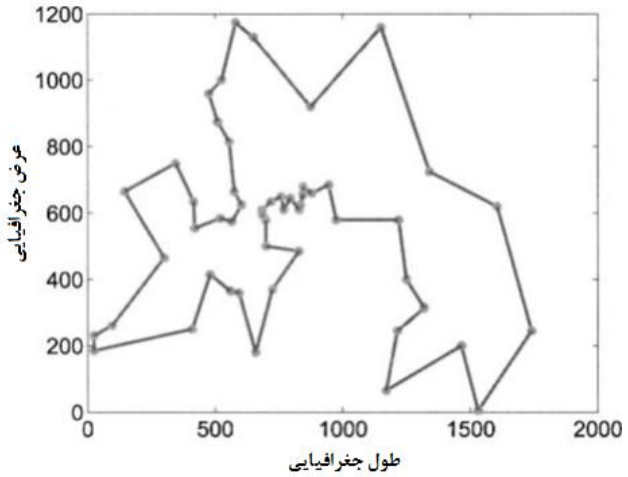
شکل ۱۸-۵ یک الگوریتم تکاملی برای حل مسئله‌ی فروشنده‌ی دوره‌گرد.

- با مسائلی مانند TSP که اطلاعات خاص مسئله (فواصل بین شهرها) از قبل مهیاست، معمولاً با ترکیب الگوریتم تکاملی با الگوریتمی که از اطلاعات مکانی استفاده می‌نماید، نتایج بسیار بهتری به دست می‌آوریم. برای نمونه، بعد از ایجاد فرزند C_k در شکل ۱۸-۵، می‌توانیم یک زیر-مسیر از C_k را به‌صورت اتفاقی انتخاب کرده و ترتیب آن را با استفاده از یکی از روش‌های ابداعی بخش ۱۸-۲ (و یا در صورت کوچک بودن زیر-مسیر با استفاده از روش brute force)، عوض نماییم [جایالاکشمی[1] و همکاران، ۲۰۰۱]. این نوع روش‌ها، الگوریتم‌های تکاملی ترکیبی[2] نام دارند چرا که عمل‌گرهای استاندارد الگوریتم تکاملی را با الگوریتم‌های غیرتکاملی که مختص TSP می‌باشند، ترکیب می‌نماید. در مقالات و کتب مختلف، انواع زیادی از الگوریتم‌های تکاملی ترکیبی برای TSP معرفی شده‌اند.
- عبارت "ذرات تکراری را جایگزین کن" در شکل ۱۸-۵ معمولاً به دلیل اینکه در بهینه‌سازی ترکیبی چند ذره‌ی برتر جمعیت تمایل به حاکمیت بر جمعیت را دارند (که این خود به دلیل گسسته بودن فضای جستجو می‌باشد)، معمولاً لازم است. این بدین معنی است که جمعیت به سمت داشتن چند ذره‌ی (و گاهی تنها یک ذره) خاص همگرا می‌شود. ما در بخش ۸-٦، تنوع را برای بهینه‌سازی پیوسته مورد بحث قرار دادیم. در مورد بهینه‌سازی ترکیبی تنوع به ملاحظات بیشتری نیاز دارد. ما می‌توانیم از روش‌های متنوعی برای جایگزینی ذرات تکراری استفاده نماییم. می‌توان این ذرات را با ذرات تصادفی و یا ذرات تولید شده به یکی از روش‌های ابتکاری بخش ۱۸-۲ جایگزین نمود. همچنین می‌توان آن‌ها را با یکی از روش‌های بخش ۱۸-٤ دچار جهش نمود. امکان استفاده از ترکیبی از این روش‌ها نیز وجود دارد.

[1] Jayalakshmi
[2] Hybrid EAs

مثال ۱۸-۱

در این مثال، TSP برلین۵۲ که مجموعه‌ای از ۵۲ مکان در برلین آلمان می‌باشد را بررسی می‌نماییم. این مسئله بر روی وبسایت TSPLIB نیز در دسترس است (ضمیمه‌ی ج.٦ را ببینید). شکل ۱۸-٦، ۵۲ شهر و کوتاهترین مسیر بسته را نشان می‌دهد. مقادیر طول و عرض جغرافیایی نرمالیزه شده‌اند. اندازه‌ی کوتاهترین مسیر برابر ۷۵٤۲ واحد است.



شکل ۱۸-٦ مثال ۱۸-۱: شهرهای TSP برلین۵۲ و کوتاه‌ترین مسیر بسته که ۷۵٤۲ واحد می‌باشد.

در این مثال، الگوریتم TSP تکاملی از شکل ۱۸-۵ را برای TSP برلین۵۲، با پارامترهای زیر پیاده‌سازی می‌نماییم:

- اندازه‌ی جمعیت را $N = 53$ (یکی بیشتر از تعداد شهرها) در نظر می‌گیریم.
- جمعیت اولیه را با تولید N مسیر اتفاقی ایجاد می‌نماییم.
- از نمایش مسیر استفاده می‌نماییم.
- در هر نسل، برازندگی مسیر x را به‌صورت زیر تعریف می‌نماییم:

$$f(x) = \max_z D(z) + \min_z D(z) - D(x) \qquad (۵۲-۱۸)$$

که در آن مینیمم و ماکزیمم بر روی کل جمعیت گرفته شده و $D(z)$ معرف اندازه‌ی مسیر z می‌باشد. این معادله، فاصله را به برازندگی تبدیل می‌نماید، به‌طوری که مسیرهای کوتاه دارای برازندگی زیاد و مسیرهای طولانی دارای برازندگی کم می‌باشند. همچنین این تبدیل به‌گونه‌ای است که مقادیر برازندگی همواره مثبت هستند.

- در الگوریتم شکل ۱۸–۵، والدین را با استفاده از انتخاب چرخ رولت انتخاب می‌نماییم.
- از برش تطبیق‌یافته‌ی جزیی (PMX) استفاده می‌نماییم (بخش ۱۸–۳–۱ را ببینید).
- از نرخ جهش $p_m = 5\%$ و جهش واژگونی استفاده می‌نماییم (بخش ۱۸و٤ ررا ببینید).
- ذرات تکراری را با استفاده از یک فرایند دو مرحله‌ای جایگزین می‌نماییم. ابتدا، جمعیت والد/فرزند را بررسی کرده و ذرات تکراری را دچار جهش می‌نماییم. سپس، دوباره جمعیت والد/فرزند را بررسی کرده و ذرات تکراری را با مسیرهای اتفاقی جایگزین می‌نماییم.
- ۲۰ شبیه‌سازی مونت کارلو و برای هر شبیه‌سازی ۳۰۰ نسل را در نظر می‌گیریم. میانگین فاصله بهترین ذرات ۲۰ شبیه‌سازی (D^*) را محاسبه می‌نماییم.

ما چند آزمایش در این مثال انجام می‌دهیم. ابتدا، از ۵ روش متفاوت برش استفاده کرده و نتایج زیر را به دست می‌آوریم.

$$\begin{aligned} &D^* = 8724 && \text{برش تطبیق‌یافته جزیی:} \\ &D^* = 8393 && \text{برش ترتیبی:} \\ &D^* = 9493 && \text{برش چرخشی:} \\ &D^* = 17109 && \text{برش ترتیب محور:} \\ &D^* = 10595 && inver - over \text{ برش:} \end{aligned} \qquad (۵۳–۱۸)$$

می‌توان دید که برش ترتیبی بهتر از سایر برش‌ها عمل می‌نماید.

حال از برش ترتیبی استفاده کرده و سه روش جهش مختلف را امتحان می‌نماییم. در این صورت نتایج زیر به دست خواهد آمد

$$\begin{aligned} &D^* = 8393 && \text{جهش واژگونی:} \\ &D^* = 9776 && \text{جهش الحاقی:} \\ &D^* = 10036 && \text{جهش تبادل هم‌پاسخ:} \end{aligned} \qquad (۵٤–۱۸)$$

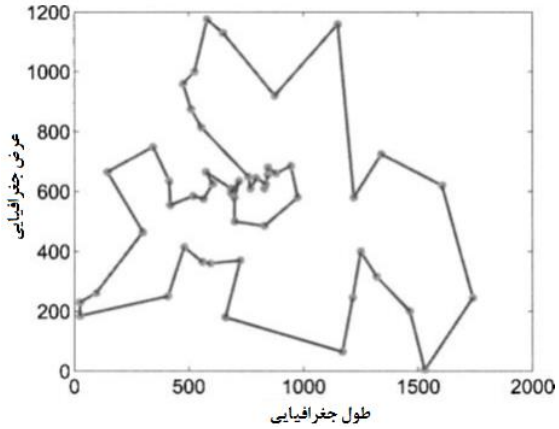
می‌توان دید که جهش واژگونی بهتر از سایر روش‌ها عمل می‌نماید.

در آخر نیز ار برش ترتیبی و جهش واژگونی استفاده کرده و سه روش مقداردهی اولیه‌ی مختلف را امتحان می‌نماییم. در روش اول، کل جمعیت اولیه را با مسیرهای اتفاقی ایجاد می‌نماییم. در روش دوم، دو ذره را با استفاده از مقداردهی اولیه‌ی نزدیک‌ترین همسایه (بخش ۱۸–۲–۱ را ببینید) ایجاد کرده و باقی

$(N-2)$ ذره را به‌صورت اتفاقی ایجاد می‌نماییم. در روش سوم نیز کل جمعیت اولیه را با استفاده از روش مقداردهی اولیه‌ی نزدیک‌ترین همسایه ایجاد می‌نماییم. در این صورت نتایج زیر به دست خواهد آمد

$$\begin{aligned} &D^* = 8393 &&\text{:جهش واژگونی} \\ &D^* = 9776 &&\text{:جهش الحاقی} \\ &D^* = 10036 &&\text{:جهش تبادل هم‌پاسخ} \end{aligned} \qquad (۵۴-۱۸)$$

می‌توان دید که ایجاد کل جمعیت با استفاده از روش مقداردهی اولیه‌ی نزدیک‌ترین همسایه بهترین گزینه است. با این حال، حتی در صورت ایجاد چند ذره با استفاده از این روش، بیشتر فواید این روش نصیبمان خواهد شد. شکل ۱۸-۷ نتایج معمول یک شبیه‌سازی از الگوریتم تکاملی با برش ترتیبی، جهش واژگونی و مقداردهی اولیه‌ی نزدیک‌ترین همسایه برای کل جمعیت، را نشان می‌دهد.



شکل ۱۸-۷ مثال ۱۸-۱: نتیجه‌ی معمول الگوریتم تکاملی برای مسئله‌ی برلین ۵۲. مسافت مسیر نشان داده شده برابر ۸۰۳۶ واحد است که این مقدار ۶٫۵٪ بدتر از مسیر بهینه می‌باشد.

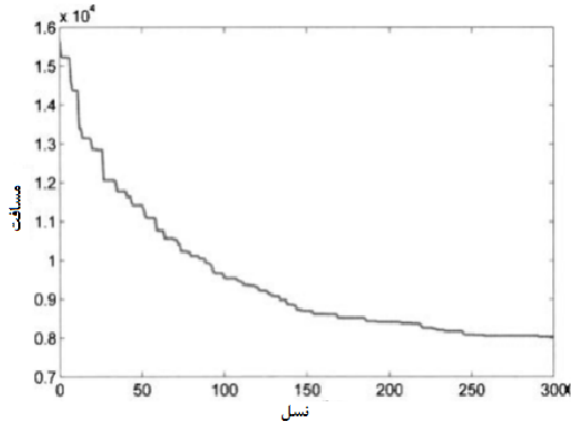
مثال ۱۸-۱ به بررسی چند گزینه‌ی محتمل برای الگوریتم تکاملی پرداخت، اما با این حال راه‌حل بهینه‌ی جهانی را به دست نداد. این موضوع چندان تعجب‌برانگیز نیست چرا که کاردینالیتی فضای جستجو از مرتبه‌ی $\frac{51!}{2} = 10^{66}$ می‌باشد. ما هر شبیه‌سازی را برای ۳۰ نسل و با اندازه‌ی جمعیتی برابر ۵۳ ذره انجام دادیم. در نتیجه، در هر شبیه‌سازی، ۱۵۹۰۰ راه‌حل بالقوه مورد ارزیابی قرارگرفته است. این مقدار کسر ناچیزی از فضای جستجو است، اما با این حال به فاصله‌ی کمتر از ۱۰٪ از جواب بهینه‌ی کل دست پیدا کردیم. با این حال، مسئله برلین ۵۲ نوعی TSP آسان تلقی می‌شود. شکل ۱۸-۶ مکان قرارگیری شهرها را نشان می‌دهد و

نظر نمی‌رسد که پیدا کردن بهترین مسیر کار چندان سختی برای یک انسان و یا یک برنامه‌ی کامپیوتری باشد. این نتایج خط تأکیدی بر یکی از نکاتی که پیش از این به آن اشاره کردیم، می‌کشد:

با مسائلی مانند TSP که اطلاعات خاص مسئله (فواصل بین شهرها) از قبل مهیاست، معمولاً با ترکیب الگوریتم تکاملی با الگوریتمی که از اطلاعات مکانی استفاده می‌نماید، نتایج بسیار بهتری به دست می‌آوریم.

هر پیاده‌سازی از الگوریتم تکاملی برای TSP باید این نصیحت را جدی بگیرد. یک محقق الگوریتم تکاملی باید ابتکارات غیرتکاملی TSP را مورد مطالعه قرار داده و آن‌ها را در الگوریتم تکاملی جای دهد تا نتایج مناسبی به دست آورد.

در آخر نیز باید اشاره کنیم که در مثال ۱۸–۱ باید بسیار بیشتر از ۱۰۰ نسل در هر شبیه‌سازی در نظر بگیریم تا بتوانیم نتایج خوبی به دست آوریم. شکل ۱۸–۸ یک نمودار معمول از راه‌حل کمترین فاصله در جمعیت الگوریتم تکاملی را به‌عنوان تابعی از شماره‌ی نسل نشان می‌دهد. این نمودار نشان می‌دهد که بهترین راه‌حل حتی بعد از ۲۰۰ نسل نیز هنوز در حال بهبود است. بنابراین اگر اجازه می‌دادیم که شبیه‌سازی چند صد نسل بیشتر از ۳۰۰ نسل ادامه پیدا کند، جواب بسیار بهتری به دست می‌آوردیم. اما معمولاً برای به دست آوردن نتایج رقابتی و عالی، شبیه‌سازی باید برای ده‌ها هزار نسل ادامه پیدا کند. صدالبته که الگوریتم‌های تکاملی که دارای محدودیت توان پردازشی هستند برای به دست آوردن راه‌حل‌های زمان–حقیقی در TSP جذاب و ضروری هستند، اما بهای عملکرد سریع‌تر نتایج بدتر می‌باشد.



شکل ۱۸–۸ مثال ۱۸–۱: رفتار همگرایی نوعی الگوریتم تکاملی برای مسئله‌ی برلین۵۲. الگوریتم تکاملی پس از ۳۰۰ نسل تقریباً همگرا شده است اما به نظر می‌رسد که در صورت ادامه‌ی شبیه‌سازی، بهترین ذره برای چند صد نسل آتی نیز بهبود می‌یافته است.

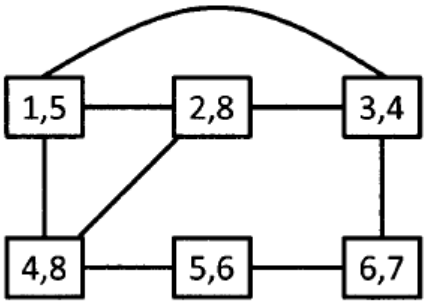
۱۸–۶ مسئله‌ی رنگ‌آمیزی گراف

گراف مجموعه‌ای از گره‌ها است که به یکدیگر متصل می‌باشند. هر گره دارای یک اندیس بی‌همتا و یک یک مقدار وزن ناهمتا می‌باشد [پاردالوس[1] و ماوریدو[2]، ۱۹۹۸]. شکل ۱۸–۹ مثالی از یک گراف را نمایش می‌دهد.

مسائل رنگ‌آمیزی گراف بسیاری وجود دارند که با وجود شباهتی که به هم دارند، بسیار متمایز از هم هستند. مسئله‌ی رنگ‌آمیزی گراف کلاسیک به یکی از دو صورت زیر تعریف می‌گردد:

۱. کمترین تعداد رنگ‌های لازم n برای رنگ کردن تمامی گره‌های یک گراف را به‌گونه‌ای تعیین کنید که هیچ دو گره‌ی به هم متصلی دارای یک رنگ نباشند.
۲. فرض کنید n رنگ در اختیار دارید، همه‌ی گره‌های موجود در گراف را به‌گونه‌ای رنگ نمایید که هیچ دو گره‌ی به هم متصلی دارای رنگ‌های یکسان نباشد.

توجه داشته باشید که مسئله‌ی نوع اول، که با نام "مسئله‌ی n–رنگ‌آمیزی نیز شناخته می‌شود، را می‌توان با حل پی‌درپی مسئله‌ی نوع دوم برای مقادیر کوچکتر n، حل نمود.



شکل ۱۸–۹ مثالی از یک گراف. هر گره دارای برچسبی به فرم (i,w) می‌باشد. i اندیس یکتا و w وزن غیریکتا می‌باشد.

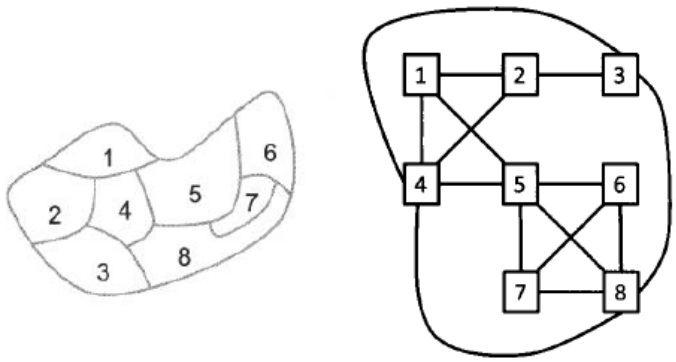
"مسئله‌ی رنگ‌آمیزی گراف وزن‌دهی شده" تعمیمی از تعریف دوم که در بالا ارائه شده است می‌باشد. این مسئله شامل تخصیص دادن یکی از n رنگ به هر گره است به‌طوری که گره‌های به هم متصل دارای رنگ‌های یکسان نبوده و جمع وزن گره‌های رنگ شده ماکزیمم شود. در این بخش بر این نوع مسئله تأکید خواهیم نمود. توجه داشته باشید مسئله‌ی نوع دوم که در بالا مطرح گردید نوع خاصی از مسئله‌ی رنگ‌آمیزی گراف وزن‌دهی شده است به‌طوری که می‌توان برای حل آن، در مسئله‌ی رنگ‌آمیزی گراف وزن‌دهی شده به هر گره وزن ۱ را اختصاص داده و سپس آن را حل نمود.

[1] Pardalos
[2] Mavridou

در مسئله‌ی رنگ‌آمیزی گراف وزن‌دهی شده، برازندگی یک راه‌حل نامزد به‌صورت جمع وزن‌های گره‌های رنگ شده اندازه‌گیری شده و در این صورت هدف مسئله، رنگ‌آمیزی گره‌ها به نحوی است که برازندگی ماکزیمم گردد. مسئله‌ی رنگ‌آمیزی گراف وزن‌دهی شده دارای کاربردهایی در زمانبندی، شبکه‌های کامپیوتری، تشخیص خطا و عیب‌یابی، تطبیق الگو، نظریه‌ی ارتباطات، بازی‌ها و بسیاری زمینه‌های دیگر می‌باشد [اوفوکتپ[1] و بکاک[2]، ۲۰۰۵]. هنگامی که با یک مسئله‌ی بهینه‌سازی عملی مواجه می‌شویم، اگر بتوانیم آن را به معادلی از مسئله‌ی رنگ‌آمیزی گراف تبدیل نماییم، خواهیم توانست از ابزاری که برای حل مسئله‌ی رنگ‌آمیزی گراف در اختیار داریم برای حل مسئله‌ی بهینه‌سازی‌مان استفاده نماییم.

دلیل این که این مسائل "مسائل رنگ‌آمیزی گراف" گاهاً "مسائل رنگ‌آمیزی نقشه" نامیده می‌شوند آن است که یک نقشه را می‌توان به‌صورت یک گراف نمایش داد[3]. به‌عنوان مثالی از تبدیل یک نقشه به گراف، شکل ۱۸–۱۰ نقشه‌ای را نشان می‌دهد که در آن هر ناحیه با یک رنگ برچسب‌گذاری شده است. برای تبدیل نقشه به گراف، ابتدا باید توجه کنیم که با توجه به نقشه، ناحیه‌ی ۱ دارای مرز مشترک با نواحی ۲، ٤ و ۵ می‌باشد. بنابراین، گراف سمت راست نشان می‌دهد که گره‌ی ۱ به گره‌های ۲، ٤ و ۵ متصل است. سپس، می‌بینیم که ناحیه‌ی ۲ دارای مرز مشترک با نواحی ۱، ۳ و ٤ بوده و به همین علت در گراف سمت راست، گره‌ی ۲ به گره‌های ۱، ۳ و ٤ متصل است. این فرایند را آنقدر ادامه می‌دهیم تا گراف کامل شود. می‌توان دید که مسئله‌ی رنگ‌آمیزی نقشه به‌گونه‌ای که نواحی مجاور دارای رنگ‌های یکسان نباشد، معادل مسئله‌ی رنگ‌آمیزی گراف است. توجه داشته باشید که عکس این قضیه صحیح نیست. به این معنی که یک گراف لزوما معرف یک نقشه نخواهد بود. برای مثال، یک گراف کامل با ۵ گره، که در آن همه‌ی گره‌ها به هم متصل هستند، را نمی‌توان به یک نقشه تبدیل نمود.

[1] Ufuktepe
[2] Becak
[3] قضیه‌ی مشهور چهار–رنگ بیان می‌دارد که برای رنگ‌آمیزی یک نقشه به‌گونه‌ای که هیچ دو شهر مجاوری دارای رنگ‌های یکسان نباشند، بیش از چهار رنگ نیاز نیست.



شکل ۱۸–۱۰ شکل سمت چپ یک نقشه را نشان می‌دهد. شکل سمت راست گراف معادل نقشه را نشان می‌دهد. یک نقشه را همواره می‌توان به یک گراف معادل تبدیل نمود در حالی که عکس این موضوع درست نیست.

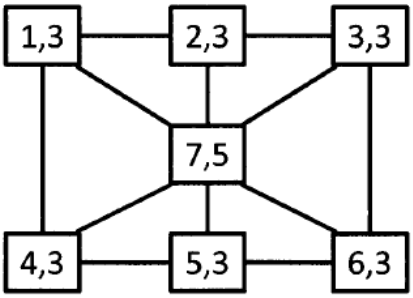
مسئله‌ی رنگ‌آمیزی گراف تک–رنگه را برای گراف شکل ۱۸–۹ در نظر بگیرید. گره‌های ۲ و ٤ دارای بیشترین وزن هستند اما نمی‌توان هر دوی آن‌ها را رنگ نمود چرا که به هم متصل هستند. چه گره‌هایی را باید رنگ نماییم تا بیشترین مقدار برازندگی به دست آید؟ یک الگوریتم مشهور، الگوریتم greedy است که در شکل ۱۸–۱۱ نشان داده شده است. الگوریتم greedy ساده است. این الگوریتم ابتدا گره‌ها را بر اساس وزن به‌صورت نزولی مرتب کرده و سپس اولین رنگ مجاز را به گره‌ها به ترتیبی که دارند، اختصاص می‌دهد.

با داشتن گراف شکل ۱۸–۹، الگوریتم greedy گره‌ها را به شکل $\{2, 4, 6, 5, 1, 3\}$ مرتب می‌نماید. توجه کنید که جای گره‌های ۲ و ٤ را می‌توان تعویض نمود چرا که دارای وزن یکسانی هستند. برای مسئله‌ی تک–رنگی، گره‌های ۲ و ٦ را رنگ می‌نماییم. در این صورت برازندگی برابر ۱۵ خواهد شد. برای مسئله‌ی دو–رنگی، (مثلاً سبز و قرمز)، رنگ قرمز را به گره‌های ۲ و ٦ و رنگ سبز را به گره‌های ٤ و ۳ اختصاص می‌دهیم. در این صورت برازندگی برابر ۲۷ خواهد شد.

الگوریتم greedy ساده و سریع است و معمولاً بسیار خوب کار می‌کند. با این حال، پیدا کردن موردی که الگوریتم greedy در آن دچار شکست شود کار چندان سختی نیست. برای نمونه، گراف شکل ۱۸–۱۲ را در نظر بگیرید. هنگامی که از الگوریتم greedy برای مسئله‌ی تک–رنگی این گراف استفاده می‌نماییم، تنها گره‌ی ۷ رنگ خواهد شد. در این صورت برازندگی برابر ۵ خواهد بود. کاملاً واضح است که با رنگ کردن گره‌های ۱، ۳ و ۵ برازندگی بهتری (که برابر ۹ است)، به دست خواهد آمد.

```
N گره‌ی مرتب شده به ترتیب نزولی وزن = {x_i}
n رنگ = {C_k}
برای i = 1 تا N
    برای k = 1 تا n
        در صورت مجاز بودن، C_k را به x_i تخصیص بده و از این حلقه خارج شو
    رنگ بعدی
گره‌ی بعدی
```

شکل ۱۸-۱۱ یک الگوریتم greedy برای حل نمودن مسئله‌ی رنگ‌آمیزی گراف با N گره و n رنگ.



شکل ۱۸-۱۲ هنگامی که از الگوریتم greedy برای رنگ نمودن این گراف با یک رنگ استفاده می‌نماییم، تنها گره‌ی ۷ رنگ می‌شود.

الگوریتم‌های تکاملی و مسائل رنگ‌آمیزی گراف

چگونه می‌توان از یک الگوریتم تکاملی برای حل مسئله‌ی رنگ‌آمیزی گراف استفاده نمود؟ یک راه برای این کار، تعریف ذرات تکاملی به‌صورت لیستی از گره‌ها می‌باشد. سپس می‌توان از الگوریتم greedy برای رنگ نمودن گره‌ها بر اساس ترتیبشان استفاده نمود. در این صورت هر ذره دارای یک مقدار برازندگی خواهد بود. می‌توان از هر نوعی از انتخاب‌های برازندگی-محور استفاده نمود و پس از آن نیز می‌توان از هر نوع روش بازترکیبی که در بخش ۱۸-۳ و هر نوع روش جهشی که در بخش ۱۸-٤ در مورد آن‌ها آموختیم، برای تولید فرزندان استفاده نماییم. این روش مسئله‌ی رنگ‌آمیزی گراف را به فرم TSP تبدیل کرده و بدین ترتیب خواهیم توانست از تمام مطالبی که در مورد TSP در بخش‌های قبل آموختیم، استفاده نماییم.

همانند TSP، هر الگوریتم تکاملی رنگ‌آمیزی گرافی باید ابتکارات غیرتکاملی را برای دستیابی به نتایج مطلوب، شامل شود. مطالب زیادی در مورد مسئله‌ی رنگ‌آمیزی گراف وجود دارد. [جنسن[1] و تافت[2]، ۱۹۹٤] زمینه‌ی مناسبی را در مورد جنبه‌های نظری و تحلیلی این مسئله به همراه الگوریتم‌های ابتکاری، فراهم می‌نماید. به علاوه، روش‌های الگوریتم تکاملی محور بسیاری برای حل مسئله‌ی رنگ‌آمیزی گراف وجود دارد. الگوریتم‌های تکاملی ترکیبی که جستجوی تکاملی را با بهینه‌سازی محلی ادغام می‌نمایند، در زمره‌ی بهترین الگوریتم‌های رنگ‌آمیزی گراف قرار دارند. برای مطالعه‌ی بیشتر به [گالینیر[3] و همکاران، ۲۰۱۳] مراجعه نمایید.

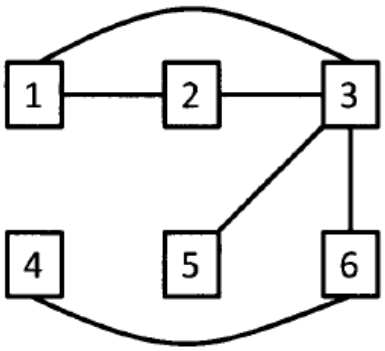
مثال ۱۸-۲

این مسئله نشان می‌دهد که چگونه می‌توان یک مسئله‌ی زمانبندی را به یک مسئله‌ی رنگ‌آمیزی گراف تبدیل نمود. فرض کنید می‌خواهیم وقایع ۱، ۲، ۳، ٤، ٥ و ٦ را زمانبندی نماییم به‌صورتی که هیچ یک از زوج وقایع زیر در یک زمان رخ ندهند:

$$(1,2), (1,3), (3,5), (3,6), (4,6)$$

این مسئله را می‌توان به‌صورت گراف شکل ۱۸-۱۳ ارائه نمود. همه‌ی گره‌ها دارای وزن‌های یکسانی بوده و به همین دلیل این وزن‌ها در شکل نشان داده نشده‌اند. گره‌های متناظر وقایعی که نمی‌توانند در یک زمان رخ دهند در گراف به یکدیگر وصل شده‌اند. این گراف دارای راه‌حل تک-رنگی نیست. با این حال، می‌توان با رنگ نمودن گره‌های ۱، ٥ و ٦ با یک رنگ و رنگ نمودن گره‌ی ۲ با رنگی دیگر و همچنین گره‌های ۳ و ٤ با رنگ سوم، به یک راه‌حل دو رنگی دست یافت. به تعبیری دیگر، می‌توان وقایع ۱، ٥ و ٦ را در زمان اول، واقعه‌ی ۲ را در زمان دوم و وقایع ۳ و ٤ را در زمان سوم، زمانبندی نمود. بدین ترتیب می‌توان دید که می‌توان از الگوریتم رنگ‌آمیزی گراف برای زمانبندی عملیات در یک کارخانه که در آن اعمال خاصی از منابع یکسانی استفاده می‌نمایند، بهره برد.

[1] Jensen
[2] Toft
[3] Galinier



شکل ۱۸-۱۳ می‌توان مسئله‌ی زمانبندی از مثال ۲ را به‌صورت این گراف نشان داد.

۱۸-۷ نتیجه‌گیری

ما مسئله‌ی فروشنده‌ی دوره‌گرد (TSP) و برخی از معمول‌ترین نحوه‌های نمایش آن و عملگرهایش را خلاصه نمودیم. همچنین، در مورد مسئله‌ی رنگ‌آمیزی گراف بحث کرده و نشان دادیم که چطور می‌توان آن را به یک TSP تبدیل نمود. محققین عملگرهای TSP بسیاری را ارائه داده‌اند که در این فصل مجالی برای پرداختن به آن‌ها نبود. [لارانگا و همکاران، ۱۹۹۹b] مرور بسیار مناسبی از TSP نحوه‌های نمایش آن و عملگرهایش ارائه می‌دهد. TSP دارای تاریخچه‌ی بسیار طولانی است و محققین از روش‌های بسیار دیگری، به غیر از الگوریتم‌های تکاملی، برای حل آن استفاده نموده‌اند. کتب بسیار خوبی نیز به TSP اختصاص یافته‌اند که از جمله‌ی آن‌ها می‌توان به [اپلگیت[1] و همکاران، ۲۰۰۷] و [لاولر[2] و همکاران، ۱۹۸۵]، اشاره نمود. برای مجموعه مقالات کنفرانسی که به الگوریتم‌های تکاملی ترکیبی اختصاص یافته‌اند می‌توانید به [هائو[3] و میدندورف، ۲۰۱۲] مراجعه نمایید.

این فصل تنها به دو مسئله‌ی بهینه‌سازی ترکیبی پرداخته است (TSP و رنگ‌آمیزی گراف)، اما مسائل بهینه‌سازی ترکیبی مشهور بسیار دیگری نیز وجود دارند. این مسائل شامل مسئله‌ی درخت پوشای مینیمم[4]، مسئله‌ی زمانبندی کار کارگاهی[5]، مسئله‌ی کوله‌پشتی[6] و مسئله‌ی بسته‌بندی صندوق[7] می‌شوند. الگوریتم‌های تکاملی به همه‌ی این مسائل اعمال شده‌اند اما هنوز جای زیادی برای تحقیقات بیشتر وجود دارد. برخی انواع الگوریتم‌های تکاملی و همچنین برخی الگوریتم‌های تکاملی جدید هنوز به این مسائل بهینه‌سازی ترکیبی

[1] Applegate
[2] Lawler
[3] Hao
[4] Minimum Spanning Tree Problem
[5] Job Shop Scheduling
[6] Knapsack Problem
[7] Bin Packing Problem

اعمال نشده‌اند. راه‌های کارامد برای ترکیب الگوریتم‌های تکاملی با ابتکارات غیرتکاملی می‌توانند زمینه‌ی مناسب و مفیدی برای تحقیقات آتی باشند. در آخر، به نتایج نظری بیشتری برای اندازه‌گیری عملکرد الگوریتم‌های تکاملی بر روی مسائل ترکیبی نیاز است.

بحث ما در مورد TSPها در این فصل تنها به TSPهای متقارن محدود گشت. این نوع از مسائل آن‌هایی هستند که در آن‌ها فاصله از شهر i به k با فاصله از شهر k به i برابر است. ضمیمه‌ی ج.٦ چندی دیگر از انواع مسائلی که با TSP در ارتباط هستند را مورد بحث قرار می‌دهد. این مسائل شامل TSP نامتقارن، مسئله‌ی مرتب‌سازی متوالی[1]، مسئله‌ی مسیریابی وسیله نقلیه تقویت‌شده[2] و مسئله‌ی مسیر همیلتون[3]، می‌شود.

یک نوع جالب دیگر از TSP، نوع *به اندازه‌ی کافی نزدیک*[4] می‌باشد. در این مسئله، گرافی به ما داده می‌شود که در آن هر گره مانند i دارای یک شعاع "به اندازه‌ی کافی نزدیک" مانند r_i می‌باشد. هدف، یافتن چرخه‌ای همیلتونی با کمترین فاصله است که از فاصله‌ای کمتر از r_i از هر گره‌ی i می‌گذرد [یوآن[5] و همکاران، ۲۰۰۷]. این مسئله ارتباط نزدیکی با TSP دارد و با اینکه دارای عناصر ترکیبی می‌باشد، اما در واقع یک مسئله‌ی بهینه‌سازی پیوسته است. در آخر نیز به TSP دابینز[6] اشاره می‌نماییم. این مسئله نوعی TSP برای یک وسیله‌ی نقلیه است که دارای محدودیت‌های حرکتی است. برای نمونه، یک وسیله‌ی نقلیه ممکن است مجبور باشد از مجموعه‌ای از مکان‌ها در حین طی کردن کوتاه‌ترین مسافت ممکن، عبور نماید، بی‌آنکه مجاز به تغییر مسیر ناگهانی باشد [ساولا[7] و همکاران، ۲۰۰۸].

[1] Sequential Ordering Problem
[2] Capacitated Vehicle Routing Problem
[3] Hamiltonian Path Problem
[4] Close_enough TSP
[5] Yuan
[6] Dubins TSP
[7] Savla

مسائل

تمارین نوشتاری

۱-۱۸ اگر ماتریس معادله‌ی (۱۸-۵) را در اختیار داشته باشیم و از مقداردهی اولیه‌ی نزدیک‌ترین همسایه استفاده نماییم، چه مسافت و چه مسیری در TSP حاصل خواهد شد؟

۲-۱۸ فرض کنید یک TSP با n شهر در اختیار داشته و می‌خواهید با استفاده از استراتژی نزدیک‌ترین همسایه که در بخش ۱۸-۲-۱ با آن آشنا شدید، M مسیر را در جمعیت اولیه تعیین نمایید. احتمال اینکه با این استراتژی M شهر اولیه‌ی متفاوت را انتخاب نمایید چه قدر خواهد بود؟ اگر $n = 100$ و $M = 10$ باشد این احتمال چه قدر خواهد بود؟

۳-۱۸ یک TSP پنج شهری باز را به‌گونه‌ای فرمول‌بندی کنید که در آن الگوریتم نزدیک‌ترین دوهمسایه که در بخش ۱۸-۲-۱ توصیف گردید، از الگوریتم نزدیک‌ترین همسایه بهتر عمل نماید.

۴-۱۸ در دومین گام مثال مقداردهی اولیه‌ی کوتاه‌ترین شاخه (که در بخش ۱۸-۲-۲ معرفی گردید)، باید از انتخابی اتفاقی استفاده می‌نمودیم چرا که دو شاخه دارای اندازه‌ی یکسانی بودند. فرض کنید که به جای شاخه‌ای که در این مثال انتخاب شد، شاخه‌ی دیگر را انتخاب نماییم. در این صورت، مسیر بسته‌ی نهایی چه خواهد بود و مسافت کل چه تغییری خواهد نمود؟

۵-۱۸ در دومین گام مثال مقداردهی اولیه‌ی الحاقی (که در بخش ۱۸-۲-۳ معرفی گردید)، باید از انتخابی اتفاقی استفاده می‌نمودیم چرا که دو شهر دارای مسافت یکسانی تا T بودند. فرض کنید به جای شهری که در این مثال انتخاب شد، شهر دیگر را انتخاب نماییم. در این صورت مسیر بسته‌ی نهایی چه خواهد بود؟

۶-۱۸ مسیر باز $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$ را در نظر بگیرید. نمایش‌های مسیر، مجاورت، ترتیبی و ماتریسی این مسیر چه خواهند بود؟

۷-۱۸ رتبه‌ی نمایش ماتریسی از یک مسیر TSP چیست؟

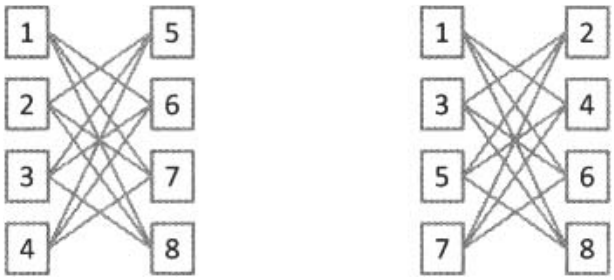
۸-۱۸ ما مسئله‌ی دو-رنگی شکل ۱۸-۹ را با استفاده از الگوریتم greedy حل نمودیم. در این الگوریتم ابتدا گره‌ها به‌صورت $\{2, 4, 6, 5, 3, 1\}$ مرتب گردیدند و سپس اولین رنگ به گره‌های ۲ و ۶، دومین رنگ به گره‌های ۳ و ۴ اختصاص یافت تا برازندگی برابر با ۲۷ به دست آید. با این حال، می‌توانستیم گره‌ها را به ترتیب $\{4, 2, 6, 5, 1, 3\}$ مرتب نماییم چرا که گره‌های ۲ و ۴ دارای وزن‌های یکسان می‌باشند. با این ترتیب، رنگ‌ها چگونه اختصاص یافته و برازندگی چه قدر خواهد شد؟

۱۸-۹ توضیح دهید که چگونه یک جدول 9×9 سودوکو را می‌توان به‌صورت یک مسئله‌ی رنگ‌آمیزی گراف در آورد. توضیح: گراف دارای ۸۱ گره خواهد بود.

۱۸-۱۰ مسئله‌ی رنگ‌آمیزی گراف را برای گراف سمت چپ شکل ۱۸-۱٤ در نظر بگیرید. در این گراف همه‌ی گره‌ها دارای وزن یکسان هستند.

الف) از الگوریتم greedy برای پیدا کردن کمترین تعداد رنگ‌ها جهت رنگ‌آمیزی گره‌ها استفاده نمایید.

ب) این کار را برای گراف سمت راست شکل تکرار نمایید. این گراف همان گراف سمت چپ است و تنها ترتیب گره‌ها عوض شده است.



شکل ۱۸-۱٤ مسئله‌ی ۱۸-۱۰: گراف‌های سمت چپ و راست معادل‌اند و تنها در ترتیب گره‌ها با هم متفاوت‌اند.

تمارین کامپیوتری

۱۸–۱۱ مثال ۱۸–۱ را با یک TSP دیگر از وبسایت TSPLIB تکرار نمایید.

۱۸–۱۲ مثال ۱۸–۱ را با برش ترتیبی، مقداردهی اولیه‌ی اتفاقی کل جمعیت و منطق مقابله–محور، که در شکل ۱۶–۱۰ نشان داده شده است، جهت عمل جهش تکرار نمایید. فاصله‌ی بهترین راه‌حل پیدا شده با میانگین‌گیری بر روی ۲۰ شبیه‌سازی مونت کارلو چه قدر است؟ این نتایج را با نتایج به دست آمده از سه روش جهش متفاوت که در مثال ۱۸–۱ استفاده شده‌اند، مقایسه نمایید.

فصل نوزدهم

بهینه‌سازی مقید

واجب است راهی برای اعمال قیدها و محدودیت‌ها (که معمولاً در هر کاربردی در دنیای حقیقی وجود دارند) به تابع برازندگی پیدا کنیم.

کارلوس ای. کوئلو کوئلو [کوئلو کوئلو، ۲۰۰۲]

تمامی مسائل بهینه‌سازی که در دنیای واقعی وجود دارند، مقید هستند. این فصل روش‌های مختلفی را برای رفع و رجوع کردن این محدودیت‌ها مورد بحث قرار خواهد داد. یک مسئله‌ی بهینه‌سازی مقید را می‌توان به‌صورت زیر نوشت

$$\min_x f(x) \quad \text{به‌طوری که } g_i(x) \le 0 \text{ برای } i \in [1,m] \text{ و } h_i(x) = 0 \text{ برای } j \in [1,p] \qquad (۱-۱۹)$$

این مسئله دارای $(m+p)$ قید (محدودیت) است به‌طوری که m قید به‌صورت نامعادله و p قید به‌صورت تساوی می‌باشند. مجموعه‌ای از تمامی xها که تمامی $(m+p)$ را برآورده می‌سازد، مجموعه‌ی امکان‌پذیر[1] و مجموعه‌ی تمامی xهایی که این قیود را نقض می‌نماید مجموعه‌ی امکان ناپذیر[2] نامیده می‌شود:

$$\mathcal{F} = \left\{x: g_i(x) \le 0 \text{ برای } i \in [1,m] \text{ و } h_j(x) = 0 \text{ برای } j \in [1,p]\right\} \text{ مجموعه‌ی امکان‌پذیر} \qquad (۲-۱۹)$$

$$\overline{\mathcal{F}} = \{x: x \notin \mathcal{F}\} \text{ مجموعه‌ی امکان‌ناپذیر}$$

از این پس، از عبارت *الگوریتم‌های تکاملی مقید* برای اشاره به الگوریتم‌هایی که مختص حل معادله‌ای به فرم (۱۹-۱) می‌باشند استفاده خواهیم کرد.[3]

مروری بر فصل

الگوریتم‌های تکاملی مقید می‌توانند در دسته‌بندی‌های مختلفی قرار بگیرند.

۱. *روش‌های تابع مجازات*[4]، تابع هزینه‌ی یک ذره‌ی الگوریتم تکاملی را بر اساس میزان تخطی آن از قیود، اصلاح می‌نمایند. روش‌های تابع مجازاتی که راه‌حل‌های امکان‌ناپذیر را مجاز شمرده و یا حتی تشویق می‌نمایند، روش‌های خارجی[5] نام دارند. در این حالت، این روش‌ها هزینه و یا انتخاب راه‌حل‌های نامزد امکان‌ناپذیر را مجازات می‌کنند. روش‌های تابع مجازاتی که راه‌حل‌های امکان‌ناپذیر

---

[1] Feasible Set
[2] Infeasible Set
[3] عبارت الگوریتم تکامیلی مقید ار لحاظ دستور زبان چندان درست نیست چرا که به لحاظ دستور زبانی، این عبارت به معنی الگوریتم‌هایی می‌بشد که خود مقید هستند، نه الگوریتم‌هایی که برای حل مسائل مقید طراحی شده‌اند. اما این عبارت راحت؛ خلاصه و مشهور است و به همین دلیل اطمینان داریم که خواننده در درک معنی آن دچار اشتباه نخواهد شد.
[4] Penalty Function Approaches
[5] Exterior Approaches

در جمعیت را مجاز نمی‌شمرند، روش‌های نقطه‌ی داخلی[1] نام دارند. در بخش ۱۹-۱، روش‌های کلی پیاده‌سازی روش‌های توابع مجازات را مورد بررسی قرار خواهیم داد و در بخش ۱۹-۲ نیز چگونگی پیاده‌سازی روش‌های مختلف تابع مجازات در الگوریتم‌های تکاملی مقید را نشان خواهیم داد.

۲. *نمایش‌های خاص*[2] روش‌هایی برای ارائه نمودن مسائل مقید (بسته به نوع مسئله) هستند به‌طوری که ارائه غیرمقید بوده اما راه‌حل‌های نامزد مقید باقی می‌مانند. عملگرهای خاص[3] روش‌های خاص مسئله برای انجام فرایندهای انتخاب، بازترکیب و جهش هستند به‌گونه‌ای که این قیود همواره توسط فرزندان برآورده شوند. این دو روش به راه‌حل‌های نامزد امکان‌ناپذیر اجازه‌ی ظهور در جمعیت را نمی‌دهند. این روش‌ها را در بخش ۱۹-۳ مورد بحث قرار خواهیم داد.

۳. *الگوریتم‌های بازسازی*[4] ذرات تکاملی امکان‌ناپذیر را به‌گونه‌ای اصلاح می‌نمایند که این ذرات امکان‌پذیر شوند. این الگوریتم‌ها قویاً به مسئله بستگی دارند. این روش‌ها ممکن است در حین اصلاح برخی ذرات امکان‌ناپذیر، به برخی ذرات امکان‌ناپذیر دیگر اجازه دهند در جمعیت باقی بمانند. تنها روش اصلاحی که در این فصل مورد بحث قرار خواهیم داد، الگوریتم ژنوکوپ[5] از بخش ۱۹-۳-۲ می‌باشد.

۴. *روش‌های ترکیبی*[6] خواص روش‌های بالا و یا الگوریتم‌های بهینه‌سازی مقید غیرتکاملی را با یکدیگر ترکیب می‌نماید. برای مثال، بسیاری از الگوریتم‌های تکاملی مقید، از یکی از روش‌های بالا به همراه جستجوی محلی استفاده می‌نمایند. در این فصل، برخی روش‌های بنیادین الگوریتم‌های تکاملی مقید را ارائه خواهیم داد، اما در مورد چگونگی ترکیب آن‌ها بحث نخواهیم کرد. با این حال، مقالات بسیاری شامل مثال‌هایی از چنین ترکیبی هستند. پس از آنکه خواننده با روش‌های بنیادین الگوریتم‌های تکاملی مقید در این فصل آشنا بشود، باید به سراغ بررسی الگوریتم‌های ترکیبی، که در منابع گوناگون وجود دارند، رفته و یا با ترکیب بهترین ویژگی‌های الگوریتم‌های تکاملی مقید مختلف، الگوریتم ترکیبی خود را ایجاد نماید.

[1] Interior Point Approaches
[2] Special Representations
[3] Special Operators
[4] Repair Algorithms
[5] Genocop Algorithm
[6] Hybrid Methods

در این کتاب تمامی روش‌هایی که در تمامی این سال‌ها برای رفع و رجوع قیود ارائه شده‌اند را پوشش نخواهیم داد، اما بخش ۱۹-٤ چند روش دیگر برای بهینه‌سازی مقید را به طرح می‌کشد. این روش‌ها شامل استفاده از الگوریتم‌های فرهنگی و بهینه‌سازی چندهدفه می‌شود.

یکی از مسائل عمده‌ای که باید در یک الگوریتم بهینه‌سازی مقید حل نماییم، این است که ذرات را چگونه رتبه‌بندی نماییم. برخی راه‌حل‌ها دارای هزینه‌ی بالا هستند اما قیود را برآورده می‌سازند، در حالی که برخی راه‌حل‌های دیگر دارای هزینه‌ی کم هستند اما از قیود تخطی می‌کنند. بخش ۱۹-۵ به خلاصه‌سازی روش‌های رتبه‌بندی که پیش از این مورد بررسی قرار گرفته‌اند پرداخته و چند روش رتبه‌بندی دیگر را نیز ارائه می‌نماید. بخش ۱۹-٦ همه‌ی این مطالب را همراه با مطالب ارائه شده در این فصل جمع‌بندی کرده و با استفاده از چند محک، مقایسه‌ای از الگوریتم‌های BBO مقید مختلف ارائه می‌دهد.

۱۹-۱ روش‌های تابع مجازات

روش‌های تابع مجازات، راه‌حل‌هایی که قیود را تقریباً یا کاملاً نقض می‌کنند، مجازات می‌نمایند. روش‌های تابع مجازات برای مسائل بهینه‌سازی مقید کلی ابتدا توسط ریچارد کورانت[1] در سال ۱۹٤۳ ارائه گردید [کورانت، ۱۹٤۳]. معمولاً از این روش‌ها به‌عنوان محبوب‌ترین الگوریتم‌ها برای بهینه‌سازی مقید نام برده می‌شود، اما روش‌های دیگری برای الگوریتم‌های تکاملی مقید نیز وجود دارند که به سرعت در حال مشهور شدن هستند.

یک روش تابع مجازات را می‌توان به دو صورت مختلف طراحی نمود. راه اول آن است که ذرات امکان‌پذیری را که در حال نزدیک شدن به مرز قیود می‌باشند، مجازات نماییم. این روش‌ها را روش‌های نقطه‌ی داخلی یا روش‌های مانع[2] می‌نماییم. این روش، که به‌صورت مختصر در بخش ۱۹-۱-۱ مورد بحث قرار خواهد گرفت، اجازه‌ی وجود هیچ گونه ذره‌ی امکان‌ناپذیری را در جمعیت نمی‌دهد.

راه دوم آن است که به برخی ذرات امکان‌ناپذیر اجازه‌ی وجود در جمعیت را بدهیم، اما آن‌ها را به لحاظ هزینه و یا انتخاب برای ایجاد نسل بعد، تنبیه و مجازات نماییم. این روش، که چند مثال از آن در بخش ۱۹-۱-۲ ارائه خواهد شد، ذرات امکان‌پذیر را مجازات نمی‌کند حتی اگر بسیار به مرز قیود نزدیک باشند. این نوع روش‌ها را روش‌های خارجی می‌نامند.

[1] Richard Courant
[2] Barrier Methods

۱۹–۱–۱ روش‌های نقطه‌ی داخلی

روش‌های نقطه‌ی داخلی تنها به ذرات امکان‌پذیر اجازه‌ی وجود در جمعیت را می‌دهند. این روش‌ها، هر چه ذرات به مرز قیود نزدیکتر شوند، آن‌ها را بیشتر از لحاظ هزینه تنبیه خواهند نمود و بدین ترتیب ذرات را به ماندن در درون قیود تشویق می‌نمایند. در اینجا ایده‌ی روش‌های نقطه‌ی داخلی را با یک مثال ساده تشریح می‌کنیم.

مثال ۱۹–۱

مسئله‌ی اسکالر زیر را در نظر بگیرید

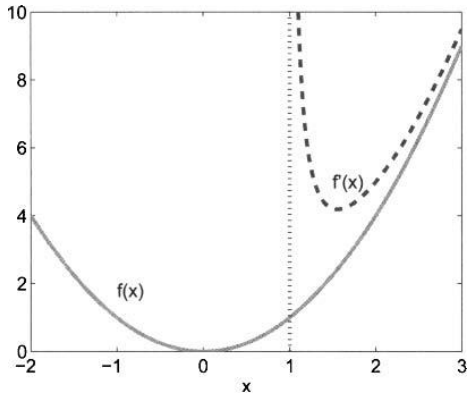
$$\min f(x) \quad \text{به‌طوری که} \quad x \geq c \quad \text{که در آن} \quad f(x) = x^2 \qquad (۳–۱۹)$$

می‌توان این مسئله را به‌گونه‌ای اصلاح نمود که مقادیر مجاز x با حرکت به سمت قیود، مجازات شوند. تابع اصلاح شده، تابع مانع نام دارد. برای مثال، می‌توان مسئله‌ی مقید معادله‌ی (۱۹–۳) را به مسئله‌ی نامقید زیر تبدیل نمود

$$\min f'(x) \quad \text{که در آن} \quad f'(x) = x^2 + (x - c + \delta)^{-\alpha} \qquad (٤–۱۹)$$

که در آن $\delta > 0$ یک ثابت کوچک و $\alpha > 0$ یک ثابت دیگر است. با میل α به سمت صفر، $argmin_x f'(x) \to argmin_x f(x)$، اما در عین حال یک $f'(x)$ با رفتار بدتر به دست خواهیم آورد، به این معنی که $f'(x)$ کمتر هموار خواهد بود.

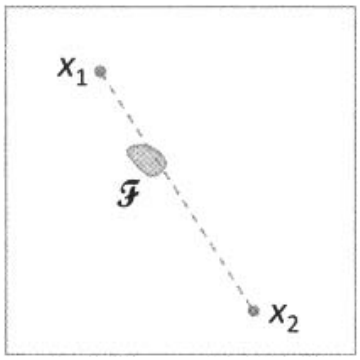
شکل ۱۹–۱، $f(x)$ و $f'(x)$ را برای $c = 1$، $\delta = 0.01$ و $\alpha = 1$ نشان می‌دهد. صدالبته که برای این مثال ساده باید از این که $x \geq c$ است اطمینان حاصل نماییم.. بنابراین، روش نقطه‌ی درونی کمک چندانی به ما نکرد. اما این مثال نشان داد که چگونه روش‌های نقطه‌ی درونی مانع از شکسته شدن قیود توسط ذرات تکاملی امکان‌پذیر پس از برش و جهش می‌شوند.



شکل ۱۹-۱ مثال ۱۹-۱: می‌خواهیم $f(x)$ را به‌گونه‌ای مینیمم کنیم که $x \geq 1$. از یک روش نقطه‌ی درونی برای تبدیل مینیمم‌سازی مقید $f(x)$ به مینیمم‌سازی نامقید $f(x)$ استفاده می‌نماییم.

روش‌های نقطه‌ی درونی چندان در الگوریتم‌های تکاملی مقید مورد استفاده واقع نمی‌شوند. این موضوع به این دلیل است که برای بسیاری از مسائل بهینه‌سازی مقید، پیدا نمودن راه‌حل‌های نامزدی که تمام قیود را برآورده سازند خود کاری بس چالش‌برانگیز است. همچنین، ممکن است راه‌حل‌های امکان‌ناپذیر حاوی اطلاعات ارزشمندی در جستجو برای یک بهینه مقید باشند. برای مثال، یک مسئله با یک ناحیه‌ی امکان‌پذیر کوچک ممکن است با ترکیب دو ذره‌ی امکان‌ناپذیر، قابل حل باشد (شکل ۱۹-۲ را ببینید).

با این حال، مسائل بهینه‌سازی بسیاری وجود دارند که پیدا نمودن راه‌حل‌های امکان‌پذیر در آن‌ها نسبتاً ساده است. در نتیجه، با توجه به مقالات گسترده‌ای که به روش‌های نقطه‌ی درونی برای الگوریتم‌های بهینه‌سازی کلی اختصاص یافته است، اندکیِ روش‌های نقطه‌ی درونی برای الگوریتم‌های تکاملی مقید، تعجب‌آور است [رایت، ۱۹۸۷]. این ممکن است نشانگر ناحیه‌ای از قلم افتاده در تحقیقات الگوریتم‌های تکاملی مقید باشد.



شکل ۱۹-۲ مثالی از یک مجموعه‌ی کوچک امکان‌پذیر $\mathcal{F}$ در یک فضای جستجوی بزرگ. ممکن است یافتن یک راه‌حل نامزد مانند $x \in \mathcal{F}$ سخت باشد، اما ممکن است پیدا کردن دو ذره‌ی امکان‌ناپذیر مانند x_1 و x_2 و سپس ترکیب آن‌ها برای تولید یک ذره‌ی امکان‌پذیر، کار آسان‌تری باشد.

۱۹-۱-۲ روش‌های خارجی

روش‌های خارجی به ذرات امکان‌ناپذیر اجازه‌ی وجود در جمعیت را می‌دهند، اما هزینه و احتمال انتخاب شدنشان را مجازات می‌نمایند. این بخش مروری گسترده از روش‌های خارجی برای بهینه‌سازی مقید را به دست می‌دهد.

۱۹-۱-۲-۱ *رویکردهای مجازات مرگ*[1] رویکردهای مجازات مرگ، روش‌های خارجی هستند که که به ذرات امکان‌ناپذیر اجازه‌ی وجود در جمعیت را می‌دهند، اما برای مدت مختصری از زمان. یک رویکرد مجازات مرگ شدت روش تابع مجازات را تا بیشترین حد افزایش می‌دهد. با این رویکرد، هر ذره‌ی امکان‌ناپذیری مانند $\bar{x}$ را بلافاصله از جمعیت حذف می‌نماییم. اگر $\bar{x}$ از عمل بازترکیب به دست آید، آن را نپذیرفته و عمل بازترکیب را آن قدر ادامه می‌دهیم تا ذره‌ای امکان‌پذیر به دست آید. به‌طور مشابه، اگر $\bar{x}$ از عمل جهش به دست آید، آن را نپذیرفته و عمل جهش را آن قدر ادامه می‌دهیم تا ذره‌ای امکان‌پذیر به دست آید.

مجازات مرگ رویکردی راحت برای بهینه‌سازی مقید است. یکی از فواید این روش آن است که در آن نیاری به محاسبه‌ی هزینه‌ی ذرات امکان‌ناپذیر نیست و این موضوع می‌تواند باعث کاهش تلاش محاسباتی شود. با این حال، برای بسیاری از مسائل، دستیابی به ذرات امکان‌پذیر به خودی خود کاری دشوار است. به همین دلیل ممکن است رد کردن ذرات امکان‌ناپذیر بیش از اندازه سختگیرانه باشد. به جای رد کامل ذرات امکان‌ناپذیر، شاید بهتر باشد که آن‌ها را در جمعیت نگه داشته و به آن‌هایی که قیود را با شدت کمتری نقض

[1] Death Penalty Approaches

می‌نمایند، هزینه‌ی کمتری اختصاص دهیم و به این ترتیب جمعیت را به حرکت به سمت ناحیه‌ی امکان‌پذیر تشویق نماییم. به‌طور خلاصه، میزان تأثیر رویکرد مجازات مرگ به نوع و اطلاعات خاص مسئله بستگی دارد.

۱۹-۱-۲-۲ *رویکردهای مجازات غیر-مرگ*[1] باقی این فصل به بحث در مورد رویکردهای مجازات غیر-مرگ می‌پردازد. این رویکردها، روش‌های خارجی بخشنده‌تری نسبت به رویکردهای مجازات مرگ هستند چرا که به ذرات امکان‌ناپذیر اجازه‌ی باقی ماندن در الگوریتم تکاملی برای کل دوره‌ی الگوریتم تکاملی را می‌دهند. ما مسئله‌ی استاندارد بهینه‌سازی مقید از معادله‌ی (۱۹-۱) را به مسئله‌ی نامقید زیر تبدیل می‌نماییم:

$$\begin{aligned}&\phi(x) = f(x) + \sum_{i=1}^{m} r_i G_i(x) + \sum_{j=1}^{p} c_j L_j(x) \text{ که در آن } \min_x \phi(x)\\&G_i(x) = \left[\max\left(0, g_i(x)\right)\right]^{\beta}\\&L_j(x) = \left|h_j(x)\right|^{\gamma}\end{aligned} \qquad (۵-۱۹)$$

که در آن r_i و c_i ثابت‌هایی مثبت بوده و فاکتورهای مجازات نامیده می‌شوند. β و γ نیز ثابت‌هایی مثبت هستند که معمولاً برابر ۱ یا ۲ قرار داده می‌شوند. $\phi(x)$ تابع هزینه‌ی مجازات شده[2] نامیده می‌شود و آن را به‌صورت جمع وزنی تابع هزینه‌ی اصلی $f(x)$ و دامنه‌های نقض قید $\{G_i(x)\}$ و $\{L_j(x)\}$ به دست می‌آوریم. می‌بینیم که اگر $x \in \mathcal{F}$، آنگاه $\phi(x) = f(x)$. با این حال، اگر $x \notin \mathcal{F}$، آنگاه $\phi(x) > f(x)$ می‌باشد و بزرگی این نامساوی با افزایش شدت نقض قیود، افزایش می‌یابد.

حال که تابع هزینه‌ی مجازات شده $\phi(x)$ را در اختیار داریم، می‌توانیم یک الگوریتم تکاملی که از $\phi(x)$ به‌عنوان تابع هزینه برای انتخاب ذرات برای نسل بعد استفاده می‌نماید، را اجرا نماییم. بنابراین می‌توانیم هر یک از الگوریتم‌های تکاملی نامقید، که در این کتاب مورد بحث قرار دادیم، را به یک الگوریتم تکاملی مقید بسط دهیم. برای این کار به سادگی از $\phi(x)$ به جای تابع هزینه‌ی $f(x)$ استفاده می‌نماییم.

قیود $h_j(x)$ بسیار سختگیرانه هستند. اگر یک جمعیت اولیه را به‌صورت اتفاقی در فضای جستجو تولید نماییم، احتمال به دست آوردن ذراتی که قیود برابری (مساوی) را برآورده سازند، الزاماً صفر خواهد بود. بنابراین، معمولاً قیود سخت مساوی را به قیود نرم‌تری که الزام می‌دارند $h_j(x)$ تقریباً (نه دقیقاً) برابر صفر باشند، تغییر می‌دهیم. بدین ترتیب خواهیم داشت:

$$\left|h_j(x)\right| \leq \epsilon \qquad (۶-۱۹)$$

[1] None-Death-Penalty Approaches
[2] Penalized Cost Function

که در آن ϵ یک ثابت مثبت کوچک می‌باشد. معادله‌ی بالا معادل دو قید زیر است

$$\begin{matrix} h_j(x) - \epsilon \le 0 \\ -h_j(x) - \epsilon \le 0 \end{matrix} \qquad (۷-۱۹)$$

بسته به مقدار ϵ، احتمال معقولی برای به دست آوردن ذراتی که قید نرم معادله‌ی (۱۹-۶) را برآورده سازند، خواهیم داشت. یک راه برای اختصاص ϵ آن است که در مراحل اولیه‌ی الگوریتم تکاملی از مقادیر نسبتاً بزرگی از ϵ استفاده نماییم تا چند ذره‌ی امکان‌پذیر به دست آید، سپس به تدریج و با افزایش شماره‌ی نسل مقدار ϵ را کاهش دهیم [برست، ۲۰۰۹]، [زاوالا[1] و همکاران، ۲۰۰۹]. بسیاری مقالات تحقیقاتی که به مقایسه‌ی الگوریتم‌های بهینه‌سازی مقید پرداخته‌اند از $\epsilon = 0.0001$ استفاده نموده‌اند [لیانگ[2] و همکاران، ۲۰۰۶].

تبدیل قیود تساوی به قیود نامساوی معادله‌ی (۱۹-۵) را به‌صورت زیر در خواهد آورد

$$\phi(x) = f(x) + \sum_{i=1}^{m+p} r_i G_i(x) \text{ که در آن } \min_x \phi(x)$$

$$G_i(x) = \begin{cases} [\max(0, g_i(x))]^{\beta} & \text{برای } i \in [1, m] \\ [\max(0, |h_i(x)| - \epsilon))]^{\beta} & \text{برای } i \in [m+1, m+p] \end{cases} \qquad (۸-۱۹)$$

در معادله‌ی بالا با قرار دادن $\beta = \gamma$، مسئله را ساده کرده‌ایم. مسائل مانند معادله‌ی (۱۹-۸) را می‌توان با روش‌های آماری یا روش‌های پویا حل نمود. روش‌های آماری از مقادیری از r_i، β و ϵ استفاده می‌کنند که مستقل از شماره‌ی نسل الگوریتم تکاملی می‌باشند.

در مقابل، روش‌های پویا از مقادیری از r_i، β و ϵ استفاده می‌کنند که به شماره‌ی نسل وابسته‌اند. پیاده‌سازی روش‌های ایستا ساده‌تر است اما روش‌های پویا به دلیل انعطاف‌پذیری که دارند، عملکرد بهتری از خود نشان می‌دهند. روش‌های پویا ممکن است بتوانند به‌صورت هوشمندانه وزن‌های خود را بر اساس توزیع جمعیت و یا شاخصه‌های مسئله، تطبیق داده و به عملکرد بهتری دست پیدا کنند. روش‌های پویا معمولاً با افزایش شماره نسل r_i، β را افزایش و ϵ را کاهش می‌دهند. این کار باعث افزایش وزن‌های مرتبط به تخطی از قیود شده و به تدریج باعث جذب ذره‌های امکان‌ناپذیر به سمت ناحیه‌ی امکان‌پذیر می‌شود.

[1] Zavala
[2] Liang

۱۹–۲ روش‌های مشهور اداره کردن قید

این بخش به بحث در مورد چند روش مشهور اداره کردن قید که در الگوریتم‌های تکاملی به کار می‌روند، می‌پردازد. تمامی این روش‌ها رویکردهای مجازات غیر–مرگ هستند.

۱۹–۲–۱ روش‌های مجازات ایستا[1]

معادله‌ی (۱۹–۸) در [همایی‌فر[2] و همکاران، ۱۹۹٤] با $\beta = 2$ و r_i به‌صورت تابعی از بزرگی میزان تخطی از قید، ارائه شده است. بدین معنی که r_i تابعی غیرنزولی از $G_i(x)$ می‌باشد. گاهاً، فاکتور مجازات r_i برابر مجموعه‌ای از مقادیر گسسته، بسته به میزان تخطی از قید، قرار داده می‌شود:

$$r_i = \begin{cases} R_{i1} & \text{اگر } G_i(x) \in [0, T_{i1}] \\ R_{i2} & \text{اگر } G_i(x) \in (T_{i1}, T_{i2}] \\ & \vdots \\ R_{iq} & \text{اگر } G_i(x) \in (T_{i,q-1}, \infty) \end{cases} \qquad (۹–۱۹)$$

در معادله بالا q تعداد سطوح قیود بوده و توسط کاربر تعیین می‌گردد. مقادیر R_{ij} و T_{ij} نیز به ترتیب وزن‌ها و مرز قیود می‌باشند که توسط کاربر تعیین می‌گردند. این یک روش ایستا است چرا که مجازات تعیین شده برای قیود تابعی از شماره نسل نیست. در مقالات تحقیقی معمولاً این روش مشهور مورد نقد واقع می‌شود چرا که به پارامترهای میزان‌سازی بسیاری نیاز دارد. در واقع این روش به $(2q-1)(m+p)$ پارامتر میزان‌سازی نیاز دارد، هر چند که می‌توان این عدد را با ترکیب برخی سطوح وزن و مرزهای قیود، کاهش داد. بدین ترتیب الگوریتم ساده‌تر خواهد شد.

۱۹–۲–۲ برتری نقاط امکان‌پذیر[3]

روش برتری نقاط امکان‌پذیر [پوول[4] و اسکولنیک[5]، ۱۹۹۳] تابع هزینه‌ی مجازات شده‌ی معادله‌ی (۱۹–۸) را به‌صورت زیر اصلاح می‌نماید:

---

[1] Static Penalty Methods
[2] Homaifar
[3] Superiority of Feasible Points
[4] Powell
[5] Skolnick

$$\phi'(x) = \phi(x) + \theta(x) \text{ که در آن } \min_x \phi'(x)$$
$$= f(x) + \sum_{i=1}^{m+p} r_i G_i(x) + \theta(x) \qquad (۱۰-۱۹)$$

که در آن $\theta(x)$ یک عبارت اضافی برای اطمینان از $\phi'(x) \leq \phi'(\bar{x})$ برای تمامی $x \in \mathcal{F}$ و تمامی $\bar{x} \notin \mathcal{F}$ می‌باشد. این به معنای $\phi'(x) \leq \phi'(\bar{x})$ برای تمامی xهای امکان‌پذیر و تمامی $\bar{x}$های امکان‌ناپذیر می‌باشد. این کار را می‌توان با فرض $f(x) > 0$ برای همه‌ی xها و با قرار دادن $\theta(x)$ به‌صورت زیر انجام داد:

$$\theta(x) = \begin{cases} 0 & \text{اگر } x \in \mathcal{F} \\ \max f(y) : y \in \mathcal{F} & \text{اگر } x \notin \mathcal{F} \end{cases} \qquad (۱۱-۱۹)$$

یک راه کمتر محتاطانه [مایکلویکز و شوئناور[1]، ۱۹۹۶] برای پیاده‌سازی این روش، که شامل فرض $f(x) > 0$ نمی‌شود، قرار دادن $\theta(x)$ به‌صورت زیر می‌باشد:

$$\theta(x) = \begin{cases} 0 & \text{اگر } x \in \mathcal{F} \\ \max[0, \max_{y \in \mathcal{F}} f(y) - \min_{y \notin \mathcal{F}} \phi(y)] & \text{اگر } x \notin \mathcal{F} \end{cases} \qquad (۱۲-۱۹)$$

این نوع تعریف از $\theta(x)$ باعث می‌شود که برای تمامی xها، $\phi'(x) = \phi(x)$ بشود. البته این موضوع تحت شرایطی است که در آن برای تمامی $x \in \mathcal{F}$ و تمامی $\bar{x} \notin \mathcal{F}$ $\phi(x) \leq \phi(\bar{x})$ می‌باشد. این بدین معنی است که اگر تابع هزینه‌ی مجازات شده از معادله‌ی (۱۹-۸) باعث بشود تمامی ذرات امکان‌پذیر دارای رتبه‌ی بهتری نسبت تمام ذرات امکان‌ناپذیر شوند، آنگاه هیچ تغییری به معادله‌ی (۱۹-۸) اعمال نخواهد شد. با این حال، اگر معادله‌ی (۱۹-۸) برای برخی $x \in \mathcal{F}$ و برخی $\bar{x} \notin \mathcal{F}$ ، $\phi(x) > \phi(\bar{x})$ را نتیجه بدهد، آنگاه معادله‌ی (۱۹-۱۲) مقادیر تابع هزینه‌ی مجازات شده‌ی تمام ذرات امکان‌ناپذیر را به‌گونه‌ای جابه‌جا می‌کند که $min_{\bar{x}}\phi'(\bar{x}) = max_x \phi'(x)$. این بدین معنی است که در این صورت بهترین هزینه‌ی مجازات شده‌ی امکان‌ناپذیر برابر بدترین هزینه‌ی مجازات شده‌ی امکان‌پذیر خواهد بود.

روش برتری نقاط امکان‌پذیر می‌تواند در صورت وجود قیدهای دشوار در مسئله‌ی بهینه‌سازی، جذاب به نظر برسند. اگر برآورده ساختن قیود سخت باشد، این روش فشار بسیار زیادی را بر نقاط امکان‌پذیر جهت باقی ماندن در جمعیت وارد خواهد آورد. بدین ترتیب اطلاعات این نقاط به نسل‌های بعد منتقل خواهد شد.

[1] Schoenaur

۱۹-۲-۳ الگوریتم‌های تکاملی گزینشی[1]

الگوریتم تکاملی گزینشی رویکردی دیگر برای تقویت برتری نقاط امکان‌پذیر ارائه می‌کند [مورالس[2] و کوئزادا[3]، ۱۹۹۸]. الگوریتم تکاملی گزینشی تابع هزینه‌ی مجازات شده را به‌صورت زیر تعریف می‌نماید

$$\phi(x) = \begin{cases} f(x) & \text{اگر } x \in \mathcal{F} \\ K\left(1 - \dfrac{s(x)}{m+p}\right) & \text{اگر } x \notin \mathcal{F} \end{cases} \qquad (۱۹-۱۳)$$

که در آن K یک ثابت بزرگ، $(m+p)$ تعداد کل قیود و $s(x)$ تعداد قیود برآورده شده توسط x می‌باشد. ثابت K توسط کاربر تعیین شده و باید به قدری بزرگ باشد که $\phi(x) < \phi(\bar{x})$ را برای تمامی $x \in \mathcal{F}$ و تمامی $\bar{x} \notin \mathcal{F}$، تضمین نماید. اگر از یک روش رتبه‌بندی برای انتخاب ذرات جهت فرآیند بازترکیب استفاده نماییم، آنگاه حد بالایی برای K وجود نخواهد داشت. با این حال، اگر از روش چرخ رولت و یا هر روش دیگری، که از مقادیر مطلق $\phi(.)$ برای فرآیند انتخاب استفاده می‌نماید، بهره ببریم، آنگاه باید مراقب باشیم که مقدار K را خیلی بزرگ انتخاب ننماییم چرا که می‌خواهیم در حین این که ذرات امکان‌ناپذیر دارای رتبه‌ی بدتری نسبت به ذرات امکان‌پذیر می‌شوند، اما همچنان از احتمال معقولی جهت انتخاب برای عمل بازترکیب برخودار باشند.

الگوریتم تکاملی گزینشی با معادله‌ی (۱۹-۱۰) متفاوت است چرا که الگوریتم تکاملی گزینشی هزینه‌ی $f(x)$ را برای ذرات امکان‌ناپذیر ارزیابی نمی‌کند. این موضوع می‌تواند باعث کاهش قابل توجه در میزان محاسبات شود. همچنین، الگوریتم تکاملی گزینشی تنها تعداد نقض قیود را جهت تعیین تابع هزینه‌ی مجازات شده مد نظر قرار می‌دهد و میزان بزرگی نقض قید را لحاظ نمی‌کند. این در حالی است که معادله‌ی (۱۹-۱۰) تنها میزان بزرگی نقض قید را مد نظر قرار داده و تعداد نقض قیود را لحاظ نمی‌کند. این موضوع یک مزیت محاسباتی دیگر الگوریتم تکاملی گزینشی را در مسائل دنیای واقعی آشکار می‌سازد چرا که شمردن تعداد نقض قیود به مراتب آسان‌تر از اندازه‌گیری سطح این نقایض می‌باشد.

[1] The Eclectic Evolutionary Algorithms
[2] Morales
[3] Quezada

۱۹–۲–۴ مجازات هم‌تکاملی[1]

ترکیب کردن رویکردهای معادلات (۱۹–۱۰) و (۱۹–۱۳) جالب به نظر می‌رسد چرا که برخی اوقات میزان بزرگی نقض قید برای ما مهم است و در برخی اوقات دیگر، تعداد این نقض‌ها. رویکرد هم‌تکاملی که شامل این ایده می‌شود، در [کوئلو کوئلو، ۲۰۰۰b]، [کوئلو کوئلو، ۲۰۰۲] ارائه شده است و از هزینه‌ی مجازات شده‌ی زیر استفاده می‌نماید

$$\phi(x) = f(x) + w_1 \sum_{i=1}^{m+p} G_i(x) + w_2 \left(1 - \frac{s(x)}{m+p}\right) \qquad (۱۹–۱۴)$$

که در آن w_1 و w_2 وزن هستند. این یک رویکرد هم‌تکاملی است چرا که شامل دو جمعیت می‌شود. یک جمعیت P_1 است که شامل راه‌حل‌های نامزد x بوده و بر اساس هزینه‌ی مجازات شده‌ی $\phi(x)$ رشد و تکامل پیدا می‌کند. جمعیت دوم P_2 تنها شامل زوج‌های (w_1, w_2) می‌باشد. یک ذره در P_1 با استفاده از یک ذره‌ی خاص از P_2 رشد پیدا می‌کند. هزینه‌ی یک زوج (w_1, w_2) به‌صورت زیر ارزیابی می‌شود

$$\psi(w) = \frac{1}{M_1(w)} \sum_{x \in \mathcal{F}} \phi(x) - M_1(w) \qquad (۱۹–۱۴)$$
$$M_1(w) = |x: x \in \mathcal{F}|$$

که در آن w به یک زوج مشخص (w_1, w_2) از P_2 اشاره می‌کند. توجه داشته باشید که $M_1(w)$ تعداد ذرات امکان‌پذیر در P_1 پس از رشد آن با استفاده از w می‌باشد. هزینه‌ی $\psi(w)$ مربوط به ذره‌ی w، به میانگین هزینه‌ی مجازات شده‌ی تمام ذرات امکان‌پذیر از P_1 که توسط آن رشد یافته‌اند، بستگی دارد. این هزینه همچنین به تعداد ذرات امکان‌پذیر رشد یافته توسط w بستگی دارد. در صورتی که $M_1(w) = 0$ باشد، معادله‌ی (۱۹–۱۵) تعریف نشده خواهد بود. اگر $M_1(w) = 0$، آنگاه $\psi(w)$ را می‌توان برابر یک هزینه‌ی دلخواه زیاد قرار دارد.

یک الگوریتم تکاملی برای هر ذره و هر نسل در P_2 ، جمعیت P_1 را رشد می‌دهد. از این رو، این رویکرد هم‌تکاملی می‌تواند به دلیل وجود الگوریتم‌های تکاملی تودرتو، به محاسبات زیادی نیاز داشته باشد. اما این رویکرد دارای قابلیت پیاده‌سازی موازی و در نتیجه کاهش تلاش محاسباتی می‌باشد.

شکل ۱۹–۳ طرح کلی الگوریتم مجازات هم‌تکاملی را نشان می‌دهد. حلقه‌ی بیرونی جمعیت P_2 را رشد می‌دهد. برای هر نسل از P_2 (هر دوره‌ی حلقه‌ی بیرونی)، $|P_2|$ الگوریتم تکاملی در حلقه‌ی داخلی اجرا شده تا راه‌حل‌های نامزد x با استفاده از هزینه‌ی مجازات شده‌ی معادله‌ی (۱۹–۱۴)، رشد یابند.

[1] Coevolutionary Penalty

می‌توان به چند طریق شکل ۱۹–۳ را برای دستیابی به عملکرد بهتر، اصلاح نمود. برای مثال، می‌توان از انواع مختلف نخبه‌گرایی برای نگه داشتن بهترین ذرات P_1 از یک تکامل P_1 به دیگری، استفاده نمود. همچنین می‌توان بیش از یک تکامل P_1 را برای هر ذره‌ی P_2 اجرا نمود تا بهترین عملکرد را برای یک w به دست آورد.

شکل ۱۹–۳ یک مثال از هم‌تکاملی است. در اینجا از آن برای بهینه‌سازی مقید استفاده می‌نماییم، اما هم‌تکاملی دارای کاربردهای جذاب بسیار دیگری است. هم‌تکاملی را در بسیاری نقاط طبیعت می‌توان یافت. برای مثال، گل‌ها و زنبورها به‌گونه‌ای تکامل یافته‌اند که برای ادامه‌ی حیات به یکدیگر وابسته‌اند [پایک[1]، ۱۹۷۸]. هم‌تکاملی همچنین به گونه‌های متفاوتی در الگوریتم‌های تکاملی مورد مطالعه قرار گرفته‌اند [پاردیس[2]، ۲۰۰۰] و می‌توانند موضوع بسیار مفید و فعالی برای تحقیقات آتی باشند.

```
جمعیت اتفاقی از وزن‌های نامزد ← P2 = {w}
تا زمانی که شرایط توقف برآورده نشده است
    برای هر w ∈ P2
        جمعیتی اتفاقی از راه‌حل‌های نامزد ← P1 = {x}
        یک الگوریتم تکاملی را جهت مینیمم ساختن معادله‌ی (۱۹–۱٤) نسبت به x اجرا کن
        از معادله‌ی (۱۹–۱۵) جهت محاسبه‌ی ψ(w) استفاده کن
    w بعدی
    از هزینه‌های ψ(w) برای انتخاب، برش و جهش P2 استفاده کن
نسل بعد P2
```

شکل ۱۹–۳ طرح کلی الگوریتم مجازات هم‌تکاملی برای مینیمم‌سازی $f(x)$ به‌طوری‌که برای $i \in [1, m+p]$، $G_i(x) = 0$.

۱۹–۲–۵ روش‌های مجازات پویا

تابع هزینه‌ی مجازات شده‌ی معادله‌ی (۱۹–۸) در [جوینز[3] و هوک[4]، ۱۹۹٤] با $\beta = 1$ یا 2 و $r_i = (ct)^\alpha$ ارائه شده است که در آن c و α ثابت بوده و t شماره‌ی نسل است:

[1] Pyke
[2] Paredis
[3] Joins
[4] Houck

$$\phi(x) = \underset{m+p}{f(x)} + (ct)^{\alpha} M(x)$$
$$M(x) = \sum_{i=1} G_i(x) \qquad (۱۶-۱۹)$$

این یک رویکرد پویاست چرا که مجازات تعیین شده بر روی قیود با افزایش شماره‌ی نسل، افزایش می‌یابد. با این حال، برای موفقیت با این رویکرد، مقادیر هزینه $f(.)$ و مقدار بزرگی نقض قید $M(.)$ باید به‌گونه‌ای نرمالیزه شود که بتوان تابع هزینه‌ی مجازات شده‌ی $\phi(.)$ را با فرض $f(x) > 0$ برای همه‌ی xها، صورت زیر نوشت:

$$\phi(x) = f'(x) + (ct)^{\alpha} M'(x)$$
$$M'(x) = \begin{cases} \dfrac{M(x)}{\max\limits_{x} M(x)} & \text{اگر } \max\limits_{x} M(x) > 0 \\ 0 & \text{اگر } \max\limits_{x} M(x) = 0 \end{cases} \qquad (۱۷-۱۹)$$
$$f'(x) = f(x) / \max_{x} |f(x)|$$

این کار به ما اطمینان می‌دهد که اجزای هزینه‌ی مجازات شده‌ی $\phi(x)$ دارای بزرگی تقریباً یکسانی هستند. یک گزینه‌ی دیگر، ترکیب روش مجازات پویا با روش برتری نقاط امکان‌پذیر، که در بخش ۱۹-۲-۲ توضیح داده شد، می‌باشد. با این رویکرد، هزینه‌ی مجازات شده به‌صورت زیر نوشته خواهد شد

$$\phi(x) = \begin{cases} f'(x) & \text{اگر } x \in \mathcal{F} \\ f'(x) + (ct)^{\alpha} M'(x) + \theta(x) & \text{اگر } x \notin \mathcal{F} \end{cases} \qquad (۱۸-۱۹)$$

که در آن $\theta(x)$ به‌گونه‌ای تعریف شده است که تمام نقاط امکان‌پذیر دارای هزینه‌ی مجازات شده‌ی کمتری نسبت به تمام نقاط امکان‌ناپذیر باشند. در مقالات [جوینز و هوک، ۱۹۹٤] مقادیر معمول $c = 1/2$ و $\alpha = 1$ یا 2 را گزارش نموده‌اند اما مقادیر مناسب c به بیشینه‌ی شماره‌ی نسل بستگی دارد. برای شبیه‌سازی‌های کوتاهتر الگوریتم تکاملی (چند صد نسل یا کمتر)، c باید با مرتبه‌ی بزرگی یک یا دو از ½ بزرگتر باشد. اگر c خیلی کوچک باشد، آنگاه مجازات نقض قید بسیار کوچک بوده و الگوریتم تکاملی مقدار بزرگی را بر روی ذراتی که هزینه‌ی کم اما نقض قید بزرگ دارند، قرار می‌دهد.

۱۹-۲-۵-۱ مجازات‌های پویای نمایی[1] یک تابع مجازات پویای نمایی در [کارلسون[2] و شونکویلر[3]، ۱۹۹۸] به‌صورت زیر ارائه شده است

$$\phi(x) = f(x)\exp\left(\frac{M(x)}{T}\right) \qquad (۱۹-۱۹)$$

در معادله‌ی بالا، $M(x)$ میزان بزرگی نقض قید بوده که در معادله‌ی (۱۹-۱۶) تعریف شده است و T یک تابع یکنوای غیرافزایشی از شماره‌ی نسل t می‌باشد. در [کارلسون و شونکویلر، ۱۹۹۸]، $T = 1/\sqrt{t}$ پیشنهاد شده است. بدین ترتیب $\lim_{t\to\infty} T = 0$ بوده و بدین ترتیب هزینه‌ی مجازات شده‌ی ذرات امکان‌ناپذیر با میل شماره‌ی نسل به سمت بینهایت، به سمت بینهایت میل خواهد نمود.

در معادله‌ی (۱۹-۱۹) فرض بر آن است که برای تمامی xها، $f(x) \geq 0$ می‌باشد. در غیر این صورت، مجازات قید باعث کاهش (منفی‌تر شدن) هزینه می‌شد. اگر این فرض برآورده نشود، می‌توان قبل از مجازات تابع هزینه، آن را جابه‌جا نمود. همچنین می‌توان یک پارامتر میزان‌سازی را به قسمت مجازات $\phi(x)$ اضافه نمود.

$$\phi(x) = f'(x)\exp\left(\frac{\alpha M'(x)}{T}\right)$$
$$f'(x) = f(x) - \min_x f(x) \qquad (۲۰-۱۹)$$

در معادله‌ی بالا میزان بزرگی نقض قید نرمالیزه شده $M'(x)$ در معادله‌ی (۱۹-۱۷) تعریف شده و α نیز یک پارامتر میزان‌سازی برای تنظیم نمودن وزن‌های نسبی نقض قید می‌باشد. می‌توان تحقیق نمود که مقادیری از α که حدود ۱۰ قرار دارند، بسیار خوب جواب می‌دهند.

در مورد روش مجازات افزایشی[4] که در معادله‌ی (۱۹-۱۷) توصیف گردید، می‌توان روش مجازات پویای نمایی را با روش نقاط امکان‌پذیر برتر، که در بخش ۱۹-۲-۲ توضیح داده شد، ترکیب نمود. با این رویکرد، هزینه‌ی مجازات شده به‌صورت زیر نوشته خواهد شد

$$\phi(x) = \begin{cases} f'(x) & \text{اگر } x \in \mathcal{F} \\ f'(x)\exp\left(\frac{M(x)}{T}\right) + \theta(x) & \text{اگر } x \notin \mathcal{F} \end{cases} \qquad (۲۱-۱۹)$$

یا

[1] Exponential Dynamic Penalties
[2] Carlson
[3] Shonkwiler
[4] Additive Penalty Method

$$\phi(x) = \begin{cases} f'(x) & \text{اگر } x \in \mathcal{F} \\ f'(x)\exp\left(\dfrac{\alpha M'(x)}{T}\right) + \theta(x) & \text{اگر } x \notin \mathcal{F} \end{cases} \qquad (۲۲-۱۹)$$

که در آن $\theta(x)$ به‌گونه‌ای تعریف شده است که تمام نقاط امکان‌پذیر دارای هزینه‌ی مجازات شده‌ی کمتری نسبت به تمام نقاط امکان‌ناپذیر باشند.

۱۹-۲-۵-۲ رویکردهای مجازات پویای دیگر[1]فرم‌های پیچیده‌ی دیگری برای توابع مجازات پویا در [کویت[2] و اسمیت[3]، ۱۹۹۶]، [کوییت و همکاران، ۱۹۹۶]، [جوینز و هوک، ۱۹۹٤]، [کازارلیس[4] و پتریدیس[5]، ۱۹۹۸] و [اسمیت و تیت[6]، ۱۹۹۳] ارائه شده‌اند. توابع مجازات پویا در [کوئلو کوئلو، ۲۰۰۲] مورد مطالعه‌ی کامل قرار گرفته‌اند. روش‌های مجازات پویا معمولاً بهتر از روش‌های ایستا عمل کرده اما به میزان‌سازی بیشتری نیاز دارند. میزان‌سازی به مشخصات مسئله وابسته است. مجازات‌های سختگیرانه کاوش مجموعه‌ی امکان‌ناپذیر را نهی می‌کند، اما گاهی برای پیدا کردن راه‌حل خوبی که قیود را برآورده می‌سازد، باید از ذرات امکان‌ناپذیر استفاده نماییم (شکل ۱۹-۲ را به یاد آورید). با این حال، مجازات خیلی کم باعث کاوش بیش از اندازه‌ی مجموعه‌ی امکان‌ناپذیر شده و همگرایی به سمت ذرات امکان‌پذیر بسیار ضعیف خواهد بود. این نکات ما را به سمت بحث در مورد روش‌های مجازات انطباقی[7] در بخش بعد هدایت می‌کند.

۱۹-۲-۶ روش‌های مجازات انطباقی

مشکلی که در مورد روش‌های ایستا و پویا وجود داشت ما را بر آن می‌دارد به دنبال نوع خاصی از روش‌های پویا، که روش‌های انطباقی نام دارند، بگردیم. روش‌های انطباقی از بازخورد (فیدبک) از جمعیت برای تنظیم وزن‌های مجازات، استفاده می‌نمایند. یک رویکرد انطباقی در [حاج-علوعین[8] و بین[9]، ۱۹۹۷] ارائه شده است که وزن‌های مجازات از معادله‌ی (۱۹-۸) را به‌صورت زیر قرار می‌دهد:

[1] Other Dynamic Penalty Approaches
[2] Coit
[3] Smith
[4] Kazarlis
[5] Petridis
[6] Tate
[7] Adaptive Penalty Methods
[8] Haddj-Alouaine
[9] Bean

$$r_i(t+1) = \begin{cases} \frac{r_i(\mathrm{t})}{\beta_1} & \text{اگر مورد اول} \\ \beta_2 r_i(t) & \text{اگر مورد دوم} \\ r_i(t) & \text{در غیر این صورت} \end{cases} \qquad (۱۹-۲۳)$$

که در آن t شماره‌ی نسل، β_1 و β_2 ثابت‌هایی با شرط $\beta_1 > \beta_2 > 1$، *مورد ۱* به معنای امکان‌پذیر بودن بهترین راه‌حل در k نسل قبل و *مورد ۲* به معنای نبود هیچ گونه ذره‌ی امکان‌پذیری در k نسل قبل، می‌باشد. پنجره‌ی نسل k یک پارامتر میزان‌سازی است که بر سرعت انطباق تأثیر می‌گذارد. می‌توان دید که اگر بهترین ذره‌ی موجود در جمعیت امکان‌پذیر باشد، وزن قید کاهش یافته تا ذرات امکان‌ناپذیر بیشتری در جمعیت مجوز حضور پیدا کنند. اگر هیچ ذره‌ی امکان‌پذیری در جمعیت وجود نداشته باشد، وزن قید افزایش یافته تا چند ذره‌ی امکان‌پذیر به دست آوریم. هدف، دستیابی به یک تعادل از مخلوط ذرات امکان‌پذیر و امکان‌ناپذیر می‌باشد تا فضای جستجو به‌صورت کامل مورد کاوش واقع شود و از اطلاعات ذرات امکان‌ناپذیر، با وجود اینکه قیود را برآورده نمی‌سازند، بتوان بهره برد. مقادیر معمول برای این روش عبارتند از $r_i(1) = 1$، $\beta_1 = 4$، $\beta_2 = 3$ و $k = n$ که n ابعاد مسئله (تعداد متغیرهای مستقل در $f(x)$) است [حاج-علوعین و بین، ۱۹۹۳].

۱۹-۲-۷ الگوریتم ژنتیک تبعیضی[1]

GA تبعیضی [له‌ریچ[2] و همکاران، ۱۹۹۵] یک روش هوشمندانه برای مدیریت میزان‌سازی پارامترهای تابع مجازات می‌باشد. میزان کردن پارامترهای r_i در معادله‌ی (۱۹-۸) سخت است. اگر خیلی بزرگ باشند، آنگاه الگوریتم تکاملی مقید بر برآورده ساختن قیود بسیار تمرکز خواهد کرد و چندان به مینیمم‌سازی تابع هزینه نخواهد پرداخت. اگر خیلی کوچک باشند، آنگاه الگوریتم تکاملی مقید بر مینیمم‌سازی تابع هزینه بسیار تمرکز خواهد کرد و چندان به برآورده ساختن قیود نخواهد پرداخت. GA تبعیضی این مسئله را با ایجاد دو لیست رتبه‌بندی شده از ذرات حل می‌نماید: اولین لیست از وزن‌های مجازات کوچک r_{1i} استفاده کرده و لیست دوم از وزن‌های مجازات بزرگ r_{2i} استفاده می‌نماید. ما ذرات را برای نسل بعد به‌صورت متناوب از میان این دو لیست انتخاب می‌نماییم. این تقریباً معادل استفاده از دو زیرجمعیت، یکی با وزن‌های مجازات کوچک و دیگری با وزن‌های مجازات بزرگ، می‌باشد.

[1] Segregated Genetic Algorithm
[2] Le Riche

به نظر می‌رسد این رویکرد جای بسیار زیادی برای تحقیقات داشته باشد. برای مثال، می‌توان از بیش از دو وزن مجازات استفاده نمود. همچنین می‌توان از مفهوم GA تبعیضی برای ترکیب روش‌های اداره کردن قیود، استفاده نمود.

۱۹–۲–۸ فرمول‌بندی برازندگی خودانطباق[1]

روشی به نام فرمول‌بندی برازندگی خودانطباق [فرمانی[2] و رایت، ۲۰۰۳] وجود دارد که مقادیر هزینه‌ی ذرات امکان‌ناپذیر را در دو مرحله مجازات می‌نماید. ابتدا، اگر هر ذره‌ی امکان‌ناپذیری مانند $\bar{x}$ دارای هزینه‌ی مجازات نشده‌ای باشد که از هزینه‌ی بهترین ذره‌ی امکان‌پذیر x بهتر است (بدین معنی که وجود داشته باشد یک $\bar{x} \in \mathcal{F}$ که برای بهترین ذره $x \in \mathcal{F}$ ، $f(\bar{x}) < f(x)$)، آنگاه مقدار هزینه‌ی هر ذره‌ی امکان‌ناپذیر مجازات شده خواهد بود. با این حال، اگر برای تمامی $\bar{x} \in \mathcal{F}$ و برای بهترین ذره‌ی $x \in \mathcal{F}$ ، $f(\bar{x}) > f(x)$ باشد، آنگاه هیچ یک از ذرات امکان‌ناپذیر مجازات نخواهند شد. این کار از مجازات غیرضروری ذرات امکان‌ناپذیر جلوگیری می‌کند و به آن‌ها اجازه می‌دهد دارای مقدار هزینه‌ی مجازات شده‌ی معقولی باشند تا در جمعیت باقی مانده و بتوان از اطلاعات آن‌ها استفاده نمود.

سپس، مرحله‌ی دوم مجازات را آغاز می‌نماییم. تمام ذرات امکان‌ناپذیر به‌گونه‌ایی مجازات می‌شوند که ذره‌ی که بیشترین تخطی از قید را دارد، دارای بدترین هزینه‌ی مجازات شده باشد. برای این کار باید ابتدا امکان‌ناپذیری کل را برای هر ذره‌ی x به‌صورت زیر تعریف نماییم:

$$\iota(X) = \frac{1}{m+p}\sum_{i=1}^{m+p} G_i(x) / \max_{\bar{x} \notin \mathcal{F}} G_i(\bar{x}) \qquad (۱۹–۲٤)$$

که در آن $G_i(.)$ با $\beta = 1$ در معادله‌ی (۱۹–۸) داده شده است. سپس، بهترین ذره (x_b)، بدترین ذرات به لحاظ امکان‌پذیری (x_{wf}) و بدترین ذرات به لحاظ هزینه (x_{wc}) را تعریف می‌نماییم:

[1] Self-Adaptive Fitness Formulation
[2] Farmani

$$x_b = \begin{cases} arg\min_x f(x): x \in \mathcal{F} & \text{اگر } \mathcal{F} \neq \emptyset \\ arg\min_x \iota(x) & \text{در غیر این صورت} \end{cases}$$
$$x_{wf} = \begin{cases} arg\max_x \iota(x): f(x) < f(x_b) & \text{اگر } \exists \bar{x} \notin \mathcal{F} \text{ به طوری که } f(\bar{x}) < f(x_b) \\ arg\max_x \iota(x) & \text{در غیر این صورت} \end{cases} \quad (۱۹-۲۵)$$
$$x_{wc} = arg\max_x f(x)$$

توجه داشته باشید که اگر $\mathcal{F} \neq \emptyset$ باشد، آنگاه $x_b \in \mathcal{F}$ خواهد بود حتی اگر برخی $\bar{x} \notin \mathcal{F}$ وجود داشته باشند که دارای هزینه‌ی کمتری باشند. با این تعاریف، معیار امکان‌ناپذیری به‌صورت زیر به $[0,1]$ نرمالیزه می‌شود

$$\bar{\iota}(x) = \frac{\iota(x) - \iota(x_b)}{\iota(x_{wf}) - \iota(x_b)} \quad (۱۹-۲۶)$$

حال می‌توان مرحله‌ای اول مجازات را به زبان ریاضی به‌صورت زیر بیان نمود

$$\phi(x) = \begin{cases} f(x) + \bar{\iota}(x)\left(f(x_b) - f(x_{wf})\right) & \text{اگر } \exists \bar{x} \notin \mathcal{F} \text{ به طوری که } f(\bar{x}) < f(x_b) \\ f(x) & \text{در غیر این صورت} \end{cases} \quad (۱۹-۲۷)$$

دومین مجازات، $\phi(x)$ را به یک هزینه‌ی مجازات شده‌ی مجازات شده‌تر $\phi'(x)$ نگاشت می‌کند:

$$\phi'(x) = \phi(x) + \gamma|\phi(x)|\left(\frac{\exp(2\tilde{\iota}(x)) - 1}{\exp(2) - 1}\right)$$
$$\gamma = \begin{cases} \dfrac{\left(f(x_{wc}) - f(x_b)\right)}{f(x_b)} & \text{اگر } f(x_{wf}) < f(x_b) \\ 0 & \text{اگر } f(x_{wf}) = f(x_b) \\ \dfrac{\left(f(x_{wc}) - f(x_{wf})\right)}{f(x_{wf})} & \text{اگر } f(x_{wf}) > f(x_b) \end{cases} \quad (۱۹-۲۸)$$

تابع نمایی در معادله‌ی (۱۹-۲۸) مجازات کوچکی را برای ذراتی که تنها کمی از قید تخطی می‌نمایند، نتیجه می‌دهد. فاکتور مقیاس γ این اطمینان را می‌دهد که ذره با بیشترین تخطی از قید دارای بیشترین هزینه‌ی مجازات شده باشد:

$$\phi'(x_{wf}) \geq \phi'(x) \quad \text{برای تمامی } x\text{ها} \quad (۱۹-۲۹)$$

در این روش مجازات دو مرحله‌ای هدف اصلی این است که پس از مجازات، بهترین ذرات موجود در جمعیت شامل برخی ذرات امکان‌پذیر و برخی ذرات امکان‌ناپذیر شوند. ذراتی که دارای هزینه‌ی و تخطی کم از قید می‌باشند به لحاظ هزینه‌ی مجازات شده، رقبای نزدیکی برای ذرات امکان‌پذیر محسوب می‌شوند. این ایده بسیار مؤثر به نظر رسیده و به بسیاری مسائل بهینه‌سازی اعمال شده است. کارهای و تلاش‌های آتی می‌تواند به میزان‌سازی بهتر و ساده‌سازی رویکرد مجازات همزمان با دستیابی به اهداف بنیادین، اختصاص یابند.

۱۹-۲-۹ تابع مجازات خودانطباق[1]

الگوریتم تابع مجازات خود انطباق (SAPF) [تسما[2] و ین[3]، ۲۰۰۶] توابع مجازات را بر اساس توزیع جمعیت تطبیق می‌دهد. اگر تنها چند ذره‌ی امکان‌پذیر وجود داشته باشند، آنگاه هزینه‌های مجازات شده‌ی کمی را به ذرات با تخطی از قید کم تخصیص خواهیم داد هر چند که ممکن است دارای هزینه‌های بالایی باشند. از سوی دیگر، اگر ذرات امکان‌پذیر بسیاری وجود داشته باشند، آنگاه هزینه‌های مجازات شده‌ی کم را تنها به ذرات با هزینه‌های کم اختصاص خواهیم داد. الگوریتم SAPF شامل مراحل زیر می‌شود.

۱. نرمالیزه نمودن مقدار تابع هزینه برای هر ذره‌ی x:

$$f_1(x) = \frac{f(x) - \min\limits_x f(x)}{\max\limits_x f(x) - \min\limits_x f(x)} \qquad (۳۰-۱۹)$$

این کار هزینه‌ی نرمالیزه شده‌ی $f_1(x) \in [0,1]$ را برای تمامی xها نتیجه می‌دهد. در این صورت هزینه‌ی نرمالیزه‌ی بهترین ذره برابر ۰ و هزینه‌ی نرمالیزه‌ی بدترین ذره برابر ۱ خواهد بود.

۲. محاسبه‌ی بزرگی تخطی از قید $\iota(x)$ برای هر ذره به‌صورتی که در معادله‌ی (۱۹-۲٤) نشان داده شده است. با این کار برای تمامی xها، $\iota(x) \in [0,1]$ به دست خواهد آمد. توجه داشته باشید که برای تمامی $x \in \mathcal{F}$، $\iota(x) = 0$ و برای تمامی $\bar{x} \notin \mathcal{F}$، $\iota(x) > 0$ خواهد بود. همچنین، ممکن است $\bar{x}$یی وجود داشته باشد که $\iota(x) = 1$.

۳. تکمیل مقدار فاصله برای هر ذره‌ی x:

[1] Self-Adaptive Penalty Function
[2] Tessema
[3] Yen

$$d(x) = \begin{cases} \iota(x) & \text{اگر } \mathcal{F} = \emptyset \\ \sqrt{f_1^2(x) + \iota^2(x)} & \text{اگر } \mathcal{F} \neq \emptyset \end{cases} \qquad (۳۱-۱۹)$$

اگر هیچ ذره‌ی امکان‌پذیری در جمعیت وجود نداشته باشد، آنگاه مقدار فاصله‌ی x، بدون لحاظ هزینه‌ی آن، برابر کل تخطی از قید x خواهد بود. بین دو ذره‌ی امکان‌پذیر، ذره‌ای که دارای هزینه‌ی کمتر است، فاصله‌ی کمتری خواهد داشت. اگر ذرات امکان‌پذیر در جمعیت وجود داشته باشند، آنگاه فاصله‌ی یک ذره‌ی امکان‌ناپذیر ترکیبی از هزینه و میزان تخطی از قید آن خواهد بود. بنابراین، هنگامی که یک ذره‌ی امکان‌پذیر x را با یک ذره‌ی امکان‌ناپذیر $\bar{x}$ مقایسه می‌نماییم، بسته به هزینه‌های نسبی آن‌ها، هر یک ممکن است دارای فاصله‌ی کوچکتری نسبت به دیگری باشد.

٤. محاسبه‌ی دو تابع هزینه‌ی مجازات شده‌ی دیگر:

$$X(x) = \begin{cases} 0 & \text{اگر } \mathcal{F} = \emptyset \\ \iota(x) & \text{اگر } \mathcal{F} \neq \emptyset \end{cases}$$
$$Y(x) = \begin{cases} 0 & \text{اگر } x \in \mathcal{F} \\ f_1(x) & \text{اگر } x \notin \mathcal{F} \end{cases} \qquad (۳۲-۱۹)$$

هزینه‌ی مجازات شده‌ی $X(x)$ برابر ۰ خواهد بود اگر هیچ ذره‌ی امکان‌پذیری در جمعیت نباشد و برابر $\iota(x)$ خواهد بود اگر ذرات امکان‌پذیر در جمعیت وجود داشته باشند. این هزینه‌ی مجازات شده در واقع ذرات امکان‌ناپذیر را بر اساس میزان بزرگی تخطی از قیدشان و تنها در صورتی که جمعیت دارای ذرات امکان‌پذیر باشد، مجازات می‌نماید. هزینه‌ی مجازات شده‌ی $Y(x)$ برابر ۰ خواهد بود اگر x امکان‌پذیر باشد و برابر هزینه‌ی نرمالیزه‌ی $f_1(x)$ خواهد بود اگر x امکان‌ناپذیر باشد. این تابع ذرات امکان‌ناپذیر را به اندازه‌ای که با مقادیر هزینه‌شان متناسب است، مجازات می‌نماید.

۵. محاسبه‌ی تابع هزینه‌ی مجازات شده

$$\phi(x) = d(x) + (1 - r)X(x) + rY(x) \qquad (۳۳-۱۹)$$

که در آن $r \in [0,1]$، نسبت ذرات امکان‌پذیر در جمعیت می‌باشد. اگر ذرات امکان‌پذیر زیادی در جمعیت باشد، $\phi(x)$ بر $Y(x)$ تاکید خواهد نمود که به معنای اعمال مجازات هزینه–محور بر ذرات امکان‌ناپذیر

می‌باشد. از سوی دیگر، اگر تعداد کمی ذرات امکان‌پذیر در جمعیت وجود داشته باشند، $\phi(x)$ بر $X(x)$ اعمال خواهد شد که به معنای اعمال مجازات تخطی از قید بر ذرات امکان‌ناپذیر می‌باشد.

SAPF در مورد مسائل بهینه‌سازی چندهدفه‌ی مقید نیز به کار رفته است [ین، ۲۰۰۹].

۱۹–۲–۱۰ اداره‌ی تبعیضی و انطباقی قید

الگوریتم تکاملی اداره‌ی تبعیضی و انطباقی قید (ASCHEA[1]) در [حامیدا[2] و شوئناور، ۲۰۰۰] و [حامیدا و شوئناور، ۲۰۰۲] مطرح شده است. ASCHEA بر پایه‌ی دو ایده است. اول آنکه سعی داریم نسبتی خاص از ذرات امکان‌پذیر به ذرات امکان‌ناپذیر را حفظ کنیم. این شبیه روش انطباقی بحث شده در بخش ۱۹–۲–۶ می‌باشد. این کار به ما اجازه می‌دهد تمام فضای جستجو، هم ناحیه‌ی امکان‌پذیر و هم ناحیه‌ی امکان‌ناپذیر، را مورد کاوش قرار دهیم.

دوم آنکه اگر تعداد کمی ذرات امکان‌پذیر در جمعیت وجود داشته باشند، آنگاه ذرات امکان‌پذیر تنها مجاز به بازترکیب با ذرات امکان‌ناپذیر خواهند بود. این موضوع بر مبنای این ایده است که راه‌حل‌های مسائل بهینه‌سازی مقید معمولاً در مرز و یا نزدیک مرز قید واقع می‌شوند [لگوئیزمون[3] و کوئلو کوئلو، ۲۰۰۹]، [ری و همکاران، ۲۰۰۹b]. بنابراین، منطقی است که برای حل مسائل بهینه‌سازی مقید هم ذرات امکان‌پذیر و هم ذرات امکان‌ناپذیر را به سمت مرز قید هدایت کنیم. بازترکیب ذرات امکان‌پذیر با ذرات امکان‌ناپذیر باعث نزدیک شدن فرزندان به مرز قید خواهد شد.

ASCHEA از رویکرد مجازات معادله‌ی (۱۹–۸) با $\beta = 1$ استفاده می‌کند. این الگوریتم همچنین از روش زیر برای به‌روزرسانی وزن‌های مجازات بهره می‌برد:

$$\phi(x) = f(x) + \sum_{i=1}^{m+p} r_i G_i(x)$$

$$r_i(t+1) = \begin{cases} \dfrac{r_i(t)}{\gamma} & \text{اگر } \tau(t) > \tau_d \\ \gamma r_i(t) & \text{در غیر این صورت} \end{cases} \qquad (۳۴–۱۹)$$

که در آن t شماره‌ی نسل، $\gamma > 1$ یک پارامتر میزان‌سازی، τ_d نسبت مشخص ذرات امکان‌پذیر به ذرات امکان‌ناپذیر و $\tau(t)$ نسبت در نسل tاُم است:

[1] Adaptive Segregational Constraint Handling Evolutionary Algorithm
[2] Hamida
[3] Leguizemon

$$t \text{ در نسل } \tau(t) = \frac{|\{x: x \in \mathcal{F}\}|}{|\{\bar{x}: \bar{x} \notin \mathcal{F}\}|} \qquad (۳۵-۱۹)$$

ASCHEA معمولاً با $\tau_d = 1/2$ پیاده‌سازی می‌شود.

دومین ایده‌ی پیاده‌سازی شده در ASCHEA به ذرات امکان‌پذیر اجازه می‌دهد تنها در صورتی با ذرات امکان‌ناپذیر بازترکیب شوند که $\tau(t) < \tau_d$. این کار، فرزندان ذرات امکان‌ناپذیر را به حرکت به سمت ناحیه‌ی امکان‌پذیر تشویق می‌نماید با این امید که تعداد ذرات امکان‌پذیر افزایش یابد.

با این حال، اگر $\tau(t) \geq \tau_d$، آنگاه به اندازه‌ی کافی ذرات امکان‌پذیر در جمعیت وجود خواهد داشت. در این صورت عمل انتخاب را در طی دو مرحله انجام خواهیم داد: ابتدا تعداد مشخصی از ذرات را از مجموعه‌ی امکان‌پذیر $\mathcal{F}$ انتخاب می‌نماییم، سپس باقی ذرات را جهت انجام عمل بازترکیب، بدون لحاظ نمودن امکان‌پذیری، بر اساس مقادیر هزینه‌ی مجازات شده‌ی $\phi(.)$ انتخاب می‌نماییم. کمترین تعداد ذرات امکان‌پذیری که برای بازترکیب انتخاب می‌نماییم معمولاً حدود ۳۰٪ تعداد کل ذراتی است که در طی عمل انتخاب به آن‌ها نیاز داریم. برای مثال، اگر بخواهیم ۱۰۰ ذره را برای بازترکیب انتخاب نماییم، ابتدا ۳۰ ذره‌ی امکان‌پذیر را بر اساس هزینه‌شان انتخاب نموده و سپس باقی ۷۰ ذره را از کل جمعیت و بر اساس هزینه‌ی مجازات شده انتخاب می‌نماییم.

یک الگوریتم مشابه به نام الگوریتم تکاملی منتج از امکان‌ناپذیری (IDEA[1]) نیز وجود دارد که در [ری و همکاران، ۲۰۰۹b]، هم برای بهینه‌سازی تک هدفه و هم برای بهینه‌سازی چندهدفه، مطرح شده است.

۱۹-۲-۱۱ حافظه‌ی رفتاری[2]

حافظه‌ی رفتاری از یک روش تقسیم-و-غلبه[3] برای حل مسائل بهینه‌سازی مقید استفاده می‌نماید [مایکلویکز و همکاران، ۱۹۹۶]، [شوئناور و زانثاکیس[4]، ۱۹۹۳]. با فرض مسئله‌ی معادله‌ی (۱۹-۸) با $m + p$ قید، ابتدا یک جمعیت، که $G_1(x)$ را مینیمم می‌نماید، بدون آنکه هیچ یک از قیود یا تابع هزینه را لحاظ نماییم، رشد می‌دهیم. این الگوریتم تکاملی را پس از آنکه قسمت مشخصی از جمعیت (که توسط کاربر تعیین می‌گردد) قید اول را برآورده سازد، متوقف می‌نماییم. پس از تکمیل این تکامل، از جمعیت نهایی آن برای مقداردهی اولیه‌ی الگوریتمی تکاملی که $G_2(x)$ را مینیمم می‌کند، استفاده می‌نماییم. دو طول الگوریتم

[1] Infeasibility Driven Evolutionary Algorithm
[2] Behavioral Memory
[3] Divide-and-Conquer Method
[4] Zanthakis

تکاملی دوم، هر ذره‌ای که از قید $G_1(x)$ تخطی کند را حذف می‌نماییم، اما سایر قیود و تابع هزینه را لحاظ نمی‌کنیم.

این مراحل را برای هر یک از قیدها تکرار می‌نماییم. iاُمین الگوریتم تکاملی را با نتایج $(i-1)$اُمین الگوریتم تکاملی آغاز می‌نماییم. iاُمین الگوریتم تکاملی جمعیتی را رشد می‌دهد که $G_i(x)$ را مینیمم می‌نماید و در طول این تکامل، ذراتی که از قیود $G_j(x)$ به ازای $j \in [1, i-1]$ تخطی می‌نمایند، حذف می‌گردند. در آخر، پس از آنکه تمامی $m+p$ با استفاده از $m+p$ الگوریتم تکاملی متوالی برآورده گردید، از جمعیت حاصله برای مقداردهی اولیه‌ی یک الگوریتم تکاملی جهت مینیمم‌سازی تابع هزینه، استفاده می‌نماییم. شکل ۱۹-٤ طرح کلی الگوریتم حافظه‌ی رفتاری را نشان می‌دهد.

الگوریتم رفتاری را می‌توان در زمره‌ی رویکردهای تابع مجازات قرار داد چرا که از مجازات مرگ استفاده می‌نماید. با این حال، خطی که در شکل ۱۹-٤ با عبارت "یک الگوریتم تکاملی را برای مینیمم‌سازی ... اجرا کن" آغاز می‌شود، شاید شامل رویکرد تابع مجازات بشود. آن الگوریتم تکاملی می‌تواند شامل هر الگوریتم تکاملی و هر رویکرد اداره‌ی قید باشد. تنها نکته‌ی مهم آن است که سرانجام، ذراتی که قیود قبلی را نقض می‌نمایند، حذف گردند.

جمعیت تولید شده به صورت اتفاقی $\leftarrow \{x\}_0$

برای $i = 1, \ldots, m+p$

iاُمین الگوریتم تکاملی را آغاز کن: $\{x\} \leftarrow \{x\}_{i-1}$

یک الگوریتم تکاملی را جهت مینیمم‌سازی $G_i(x)$ که دارای قید $G_j(x) = 0$ می‌باشد، با استفاده از مجازات مرگ اجرا کن تا از برآورده شدن تمام قیود G_j اطمینان حاسل شود. جمعیت نهایی این الگوریتم تکاملی را با $\{x\}_i$ نمایش بده.

i بعدی

الگوریتم تکاملی نهایی را مقداردهی کن: $\{x\} \leftarrow \{x\}_{m+p}$

یک الگوریتم تکاملی را جهت مینیمم ساختن $f(x)$ که دارای قیود $G_j(x) = 0$ برای $j \in [1, m+p]$ می‌باشد را با استفاده از مجازات مرگ اجرا کن تا تمامی قیود برآورده شوند

شکل ۱۹-٤ طرح کلی الگوریتم حافظه‌ی رفتاری برای مینیمم‌سازی $f(x)$ با قیدهای $G_i(x) = 0$ برای $i \in [1, m+p]$.

حافظه‌ی رفتاری در واقع تعمیمی است از الگوریتم‌های بهینه‌سازی غیر مقید که به تدریج تعداد ارزیابی‌های تابع هزینه را افزایش می‌دهند [دگاریس[1]، ۱۹۹۰]، [گثرکول[2] و راس، ۱۹۹٤]. برای مثال، فرض کنیم می‌خواهیم تابع $f_i(x)$ را نسبت به x مینیمم نماییم. i می‌تواند هر یک از مقادیر مجموعه‌ی بزرگ $\mathcal{I} = \{1, 2, \dots, i_{max}\}$ باشد. یک رویکرد برای این کار این است که ابتدا $f_1(x)$ را با استفاده از یک الگوریتم تکاملی مینیمم نماییم. سپس، با استفاده از آخرین جمعیت به دست آمده، $f_1(x) + f_2(x)$ را مینیمم نماییم. دوباره، با استفاده از آخرین جمعیت به دست آمده در مرحله‌ی دوم، $f_1(x) + f_2(x) + f_3(x)$ را مینیمم نماییم. این فرآیند را آنقدر ادامه می‌دهیم تا از مینیمم‌سازی $f_i(x)$، که بر روی تمام $i \in \mathcal{I}$ میانگین گرفته شده است، رضایت پیدا کنیم. یک رویکرد دیگر آن است که ترکیبی از نمونه‌های اتفاقی $f_i(x)$ را مینیمم نماییم. در این روش تعداد نمونه‌ها را به تدریج و با افزایش شماره نسل، افزایش می‌دهیم. این رویکرد، نمونه‌برداری اتفاقی[3] نام دارد [بنژاف و همکاران، ۱۹۹۸، بخش ۱۰-۵-۱]. معمولاً از این روش در برنامه‌نویسی ژنتیک استفاده می‌شود چرا که محاسبه‌ی هزینه‌ی تنها یک برنامه‌ی کامپیوتری خود به اجرا برای ورودی‌های بسیاری نیاز دارد (فصل ۷ را ببینید).

به نظر می‌رسد که این نوع مینیمم‌سازی $f_i(x)$ برای $i \in \mathcal{I}$ همان شکل مسائل بهینه‌سازی چندهدفه را دارد (فصل ۲۰ را ببینید)، اما مسئله‌ی مورد بحث در اینجا در واقع یک مسئله‌ی بهینه‌سازی تک-هدفه است. همه‌ی توابع $f_i(x)$ مشابه هستند اما برای iهای مختلف، با پارامترهای مختلف ارزیابی می‌شوند. با این حال، مرز میان مسائل بهینه‌سازی تک-هدفه با چند پارامتر و مسائل بهینه‌سازی چندهدفه، مرزی فازی است. یک نفر ممکن است یک مسئله را تک-هدفه بداند و فرد دیگری ممکن است همان مسئله را چندهدفه تلقی نماید.

۱۹-۲-۱۲ رتبه‌بندی اتفاقی[4]

رتبه‌بندی اتفاقی [رونارسون[5] و یائو، ۲۰۰۰]یک عنصر اتفاقی را به الگوریتم‌های تکاملی مقید اضافه می‌نماید. از آنجا که اتفاقی بودن جزء مهمی از الگوریتم‌های تکام می‌باشد، دخیل کردن اتفاقی بودن در رویکردهای اداره‌ی قید منطقی به نظر می‌رسد. رتبه‌بندی اتفاقی گاهاً راه‌حل‌های نامزد را بر اساس هزینه‌شان $f(.)$ و گاهاً بر اساس میزان تخطی آن‌ها از قید رتبه‌بندی می‌نماید. تصمیم اینکه ذرات چگونه رتبه‌بندی

[1] De Garis
[2] Gathercole
[3] Stochastic Sampling
[4] Stochastic Ranking
[5] Runarsson

شوند اتفاقی است. هنگامی که دو ذره‌ی x_1 و x_2 را با یکدیگر مقایسه می‌نماییم، می‌گوییم ذره‌ی x_1 از x_2 بهتر است اگر:

- هر دو راه‌حل امکان‌پذیر بوده و $f(x_1) < f(x_2)$ یا،
- یک عدد تولید شده به‌صورت اتفاقی مانند $r \sim U[0,1]$ از یک پارامتر مشخص تعیین شده توسط کاربر مانند P_f کوچکتر بوده و $f(x_1) < f(x_2)$ و یا،
- هیچ یک از شرایط بالا صدق نکرده و x_1 دارای تخطی از قید کمتری نسبت به x_2 می‌باشد.

در غیر این صورت، x_2 را بهتر از x_1 خواهیم دانست. می‌توان دید که بسته به این که خروجی یک تولید کننده‌ی اعداد اتفاقی چه باشد، ممکن است دو ذره‌ی x_1 و x_2 را بر اساس هزینه‌شان و یا بر اساس میزان تخطی از قیدشان، مقایسه نماییم. پس از آنکه تمامی ذرات موجود در جمعیت مقایسه و دسته‌بندی گردیدند، عمل انتخاب و بازترکیب را برای نسل بعد انجام می‌دهیم. مقادیر احتمال $P_f \in (0.4\ 0.5)$ نتایج خوبی را در مورد بسیاری از مسائل محک به دست می‌دهد [رونارسون و یائو، ۲۰۰۰].

۱۹-۲-۱۳ رویکرد مجازات نیچه[1]

رویکرد جریمه‌ی نیچه [دب و آگراوال، ۱۹۹۹]، [دب، ۲۰۰۰] از سختی میزان‌سازی پارامترهای روش مجازات انگیزش یافته است. این رویکرد بر اساس قوانین زیر، از انتخاب مسابقه‌ای برای انتخاب ذرات جهت عمل بازترکیب استفاده می‌نماید.

- بین دو ذره‌ی امکان‌پذیر، آنی که دارای هزینه‌ی کمتری است برنده‌ی مسابقه خواهد بود.
- بین یک ذره‌ی امکان‌پذیر و یک ذره‌ی امکان‌ناپذیر، ذره‌ی امکان‌پذیر برنده‌ی مسابقه خواهد بود.
- بین دو ذره‌ی امکان‌ناپذیر، آنی که دارای نقض قید کوچکتری است، برنده‌ی مسابقه خواهد بود.

این رویکرد به دلیل سادگی بسیار جذاب است چرا که به هیچ گونه میزان‌سازی پارامترهای مجازات نیاز ندارد. همچنین، مقایسه‌ی دو ذره به هیچ گونه ارزیابی تابع هزینه نیاز ندارد که این موضوع می‌تواند تلاش محاسباتی را کاهش دهد. این رویکرد مجازات نیچه معمولاً نتیجه‌ی خوبی را در مورد مسائل بهینه‌سازی مقید به دست می‌دهد. با این حال، این سادگی مضراتی هم دارد چرا که این رویکرد، یک ذره‌ی امکان‌پذیر با هزینه‌ی بسیار زیاد را از یک ذره‌ای که به مقدار بسیار اندکی امکان‌ناپذیر است ولی هزینه‌ی بسیار کمی هم دارد، بهتر می‌داند. بنابراین، این رویکرد برای مسائلی که در آن‌ها راه‌حل‌ها در مرز قید واقع شده‌اند، همانند اکثر مسائل دنیای واقعی، مناسب نخواهد بود [لگوئیزامون و کوئلو کوئلو، ۲۰۰۹]، [ری و همکاران، ۲۰۰۹b].

[1] Niched-Penalty Approach

قسمت "نیچه"ی این رویکرد تکمیل‌کننده‌ی توانایی اداره‌ی قید آن نیست، اما مقصود از آن حفظ تنوع در جمعیت بوده و به این صورت تشریح می‌گردد که: اگر ذرات در فضای دامنه از هم دور باشند، به آن‌ها اجازه‌ی مسابقه با یکدیگر (برای انتخاب) را نخواهیم داد. پس از آن که ذرات را به‌صورت اتفاقی برای انتخاب مسابقه‌ای انتخاب نمودیم، فاصله‌ی آن‌ها از یکدیگر را محاسبه می‌نماییم. اگر ذرات بسیار از یکدیگر دور باشند، آنگاه به‌صورت اتفاقی ذرات دیگری را برای مسابقه انتخاب خواهیم نمود. این کار اجازه ناپدید شدن خوشه‌های ذرات دور افتاده از جمعیت را نداده و بدین ترتیب تنوع را حفظ می‌نماید.

استراتژی تکامل چند عضوه (MMES[1]) [مزورا[2] و کوئلو کوئلو، ۲۰۰۵] شبیه رویکرد مجازات نیچه می‌باشد. شاخصه‌ی متمایز MMES آن است که هر از چند گاهی (تقریباً ۳٪ نسل‌ها)، انتخاب ذرات امکان‌ناپذیر با کمترین مقدار هزینه و یا با کوچکترین میزان نقض قید برای نسل بعد تضمین شده است. این رویکرد شبه-نخبه‌گرا متضمن شرکت یافتن ذرات امکان‌ناپذیر خوب در قرآیند جستجوی تکاملی است.

۱۹-۳ نمایش‌های ویژه و عملگرهای ویژه

یک نمایش ویژه راهی است برای فرمول‌بندی مسئله به‌گونه‌ای که قیود به‌صورت خودکار توسط راه‌حل‌های نامزد برآورده شوند. یک عملگر ویژه نیز راهی است برای تعریف عملگرهای جهش و بازترکیب در یک الگوریتم تکاملی، به‌گونه‌ای که ذرات فرزند به‌صورت خودکار قیود را برآورده سازند. هر دوی این روش‌ها به مقدار زیادی به نوع مسئله وابسته هستند. نمی‌توان یک کد نمایش یا عملگر ویژه‌ای نوشت که قابل اعمال به گستره‌ی وسیعی از مسائل باشد. با این حال، با این که این وابستگی به نوع مسئله برای نمایش‌ها و عملگرهای ویژه کار طراحی الگوریتم تکاملی را دشوار می‌سازد، اما این کار سود بسیاری خواهد داشت چرا که در این صورت الگوریتم تکاملی حاصل از اطلاعات خاص مسئله استفاده کرده و این کار معمولاً نتایج بهتری را نسبت به استفاده از الگوریتم تکاملی کلی‌تر به دست می‌دهد. این ایده مستقیماً با قضیه‌ی no free lunch در ارتباط است (ضمیمه‌ی ب.۱ را ببینید).

بخش ۱۹-۳-۱ به بحث در مورد نمایش‌های ویژه و رویکرد کدبردار[3] می‌پردازد. بخش ۱۹-۳-۲ نیز عملگرهای ویژه و یک الگوریتم تکاملی مقید محبوب به نام ژنوکوپ[4] را مورد بررسی قرار خواهد داد.

[1] Multimembered Evolutionary Strategy
[2] Mezura
[3] Decoder Approach
[4] Genocop

۱۹–۳–۱ نمایش‌های ویژه

به‌عنوان یک مثال ساده از رویکرد نمایش ویژه در بهینه‌سازی مقید، مسئله‌ی دو بعدی زیر را در نظر بگیرید

$$\min_x f(x) \text{ به‌طوری که } x_1^2 + x_2^2 \le K \qquad (۱۹–۳۶)$$

که در آن K یک ثابت است. این مسئله را می‌توان به مسئله‌ی نامقید زیر تبدیل نمود

$$\min_{\rho,\theta} f(x) \text{ به‌طوری که } \rho \in [0, K] \text{ و } \theta \in [0,2\pi] \qquad (۱۹–۳۷)$$
$$\text{که در آن: } x_1 = \rho\cos\theta \text{ و } x_2 = \rho\sin\theta$$

این تبدیل ساده‌ی کارتزین به قطبی مسئله‌ی مقید از معادله‌ی (۱۹–۳۶) را به مسئله‌ی نامقید معادله‌ی (۱۹–۳۷) تبدیل می‌نماید. مسئله‌ی اصلی معادله‌ی (۱۹–۳۶) دارای یک قید غیر خطی است اما ما آن را به مسئله‌ی معادله‌ی (۱۹–۳۷) تبدیل نموده‌ایم که در آن تنها قیود حاضر، حدودی ساده بر روی دامنه‌ی جسجو می‌باشد.

کدبردارها

یک رویکرد برای حل مسائل بهینه‌سازی مقید کد نمودن قواعدی است که راه‌حل‌های نامزد را تعیین می‌کند، به‌گونه‌ای که مجموعه‌ی قواعد همواره ذرات امکان‌پذیر را نتیجه دهد. این بدین معنی است که به جای آنکه یک راه‌حل نامزد را مستقیماً کد کنیم، مجموعه‌ای از قواعد را که از آن برای به دست آوردن راه‌حل نامزد استفاده می‌نماییم، کد نماییم. در این صورت، هر ذره در جمعیت الگوریتم تکاملی شامل مجموعه‌ی قواعدی برای ساختن راه‌حل نامزد خواهد شد. همچنین می‌توان از این رویکرد برای مسائل بهینه‌سازی نامقید استفاده نمود، اما به نظر می‌رسد که این روش به‌طور خاص به مسائل بهینه‌سازی مقید قابل اعمال باشد چرا که، بسته به نوع خاص مسئله، می‌توان مجموعه‌ای از قواعد را تعیین نمود که همواره قیود مسئله را برآورد سازد.

یک مجموعه قواعد برای ساختن راه‌حل نامزد، یک کدبردار نام دارد [پالمر[1] و کرشنباوم[2]، ۱۹۹٤]، [کوزیل[3] و مایکلویکز، ۱۹۹۹]، و باید چند ویژگی را برآورده سازد:

- برای هر راه‌حل امکان‌پذیر مسئله‌ی بهینه‌سازی، باید حداقل یک کدبردار وجود داشته باشد.

[1] Palmer
[2] Kershenbaum
[3] Koziel

- همه‌ی راه‌حل‌های امکان‌پذیر باید متناظر تعداد یکسانی از کدبردارها باشند تا هیچ گونه گرایشی در جستجو وجود نداشته باشد.
- هر کدبردار باید متناظر یک راه‌حل امکان‌پذیر باشد.
- تبدیل میان کدبردار و راه‌حل نامزد باید نسبت به ارزیابی هزینه سریع باشد.
- تغییرات کوچک در کدبردار باید تغییرات کوچکی را در راه‌حل نامزد ایجاد نماید.

اگر در کاربردی خاص برآورده ساختن تمام این ویژگی‌ها سخت باشد، می‌توان آن‌ها را کمی آسان‌تر گرفت [کوزیل، و مایکلویکز، ۱۹۹۹]، اما این قوانین حداقل خط مشی مفیدی را به دست می‌دهند. برای مثال، ممکن است مسئله‌ای در اختیار داشته باشیم که برآورده ساختن قیود آن بسیار سخت باشد، اما می‌توان مجموعه‌ای از کدبردارها را پیدا کرد که در بسیاریاز مواقع راه‌حل‌های نامزد امکان‌پذیر را نتیجه می‌دهند. اگرچه که چنین مجموعه‌ی کدبرداری کاملاً قواعد بالا را برآورده نمی‌سازد، گاهاً این روش بهتر از کد نمودن مستقیم ذرات در الگوریتم تکاملی خواهد بود.

همین که مجموعه‌ای از قواعد برای ساختن راه‌حل‌های نامزد را به دست آوردیم، جمعیت الگوریتم تکاملی را به‌گونه‌ای تعریف می‌نماییم که شامل مجموعه‌ای از کدبردارها شود. سپس عمل انتخاب، بازترکیب و جهش را بر روی کدبردارها انجام داده و بدین ترتیب متضمن برآورده شدن قیود خواهیم شد.

مثال ۱۹-۲

ناحیه‌ی امکان‌پذیر محدب $\mathcal{F}$ از مسئله‌ی بهینه‌سازی مقید دوبعدی شکل ۱۹-۵ را در نظر بگیرید. نقطه‌ی مرجع r یک نقطه‌ی دلخواه در $\mathcal{F}$ است. هر نقطه مانند $x \in \mathcal{F}$ را می‌توان برای یک مقدار $\alpha \in [0,1]$ و یک b بر روی مرز دامنه‌ی جستجو، به‌صورت $\alpha b + (1-\alpha)r$ نشان داد.

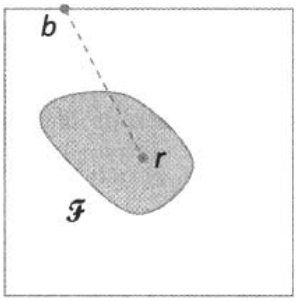
یک کدبردار برای حل این مسئله‌ی بهینه‌سازی می‌تواند به‌صورت زیر عمل نماید:

۱. یک نقطه‌ی امکان‌پذیر برای r می‌یابیم.

۲. با حرکت دادن b دور کل مرز دامنه‌ی جستجو، یک مرز برای $\mathcal{F}$ می‌یابیم. برای هر b ماکزیمم α را به‌گونه‌ای می‌یابیم که $\alpha b + (1-\alpha)r \in \mathcal{F}$. این مقدار ماکزیمم به‌صورت $\alpha_{max}(b) = \max\{\alpha: \alpha b + (1-\alpha)r \in \mathcal{F}\}$ نشان داده می‌شود.

۳. یک جمعیت از ذرات الگوریتم تکاملی را به‌گونه‌ای تعریف می‌نماییم که هر ذره شامل جفت (b, α) شود. پارامتر b می‌تواند هر نقطه‌ای بر روی مرز دامنه‌ی جستجو بوده و α نیز می‌تواند هر عدد حقیقی در بازه‌ی $[0, \alpha_{max}(b)]$ باشد.

٤. یک الگوریتم تکاملی را با انجام عمل بازترکیب بر روی زوج‌های (b, α) اجرا می‌نماییم. تنها آزمایش امکان‌پذیری که باید صورت بپذیرد، $\alpha \in [0, \alpha_{max}(b)]$ می‌باشد.



شکل ۱۹–۵ مثالی از الگوریتم کدبرداری برای بهینه‌سازی بر روی $\mathcal{F}$. با داشتن نقطه‌ی $r \in \mathcal{F}$، هر نقطه مانند $x \in \mathcal{F}$، برای مقداری از $\alpha \in [0, 1]$ و یک b بر روی مرز دامنه‌ی جستجو، به‌صورت یکتا با $\alpha b + (1 - \alpha)r$ نمایش داده خواهد شد.

مثال ۱۹–۲ را باید برای نواحی امکان‌پذیر غیرمحدب اصلاح نمود، اما این مثال چگونگی استفاده از یک کدبردار برای ساده‌سازی قیود را به خوبی نشان می‌دهد.

۱۹–۳–۲ عملگرهای ویژه

در بسیاری از مسائل بهینه‌سازی دنیای واقعی، راه‌حل در مرز قید واقع شده است [لگوئیزامون و کوئلو کوئلو، ۲۰۰۹]، [ری و همکاران، ۲۰۰۹b]. مسئله‌ی بهینه‌سازی خرید مقداری ابزار را در نظر بگیرید. اگر به ما گفته شود که بهترین ابزار ممکن را خریداری کنید، این مسئله ممکن است مقید به یک مرز بالا برای مقدار هزینه باشد. به احتمال زیاد بیشترین مقدار ممکن پول را هزینه خواهیم نمود، چرا که معمولاً ابزار بهتر، هزینه‌ی بیشتری نسبت به ابزار بدتر دارند. این بدین معنی است که بهینه‌ی مقید در مرز قید هزینه واقع خواهد شد. قیودی که برای مثل به رنگ ابزار مربوط می‌شوند، جزیی از خرید نخواهد بود چرا که بهینگی ابزار به رنگ آن ربطی ندارد.

به‌طور مشابه، اگر بخواهیم از نقطه‌ی A به نقطه‌ی B در کمترین زمان ممکن و با وجود حدی بالا برای سوخت، سفر نماییم، احتمالاً از بیشترین مقدار ممکن سوخت استفاده خواهیم کرد چرا که معمولاً با مصرف سوخت بیشتر، سریعتر حرکت خواهیم نمود. این نیز بدین معنی است که بهینه‌ی مقید در مرز قید سوخت

واقع شده است. با تفکر در مورد مسائل دنیای واقعی در خواهیم یافت که بیشتر راه‌حل‌های مسائل بهینه‌سازی در مرز قید واقع می‌شوند.[۱]

این موضوع ایده‌ی حل یک مسئله‌ی بهینه‌سازی با کاوش مرز قید را ایجاد می‌کند. بنابراین می‌توان با اطمینان فضای داخلی مرز قید را نادیده گرفت چرا که انتظار داریم راه‌حل‌های بهینه‌سازی در مرز قید واقع شده باشند. برای مثال، یک مسئله‌ی بهینه‌سازی با قید زیر را در نظر بگیرید:

$$x_1 x_2 x_3 x_4 \geq 0.75 \qquad (۳۸-۱۹)$$

اگر بدانیم که بهینه‌ی مقید در مرز قید واقع شده است، می‌توانیم مسئله بهینه‌سازی را با کاوش ترکیباتی از (x_1, x_2, x_3, x_4) که قید زیر را برآورده می‌سازند، حل نماییم [مایکلویکز و شئوناور، ۱۹۹۶]:

$$x_1 x_2 x_3 x_4 = 0.75 \qquad (۳۹-۱۹)$$

می‌توان یک ذره در الگوریتم تکاملی را به‌صورت زیر مقداردهی اولیه نمود:

$$\begin{aligned} x_1 &= U[-x_{min}, x_{max}] \\ x_2 &= U[-x_{min}, x_{max}] \\ x_3 &= U[-x_{min}, x_{max}] \\ x_4 &= \frac{0.75}{x_1 x_2 x_3} \end{aligned} \qquad (۴۰-۱۹)$$

این کار این اطمینان را به ما می‌دهد که (x_1, x_2, x_3, x_4) معادله‌ی (۱۹-۳۸) را برآورده می‌سازد. حال فرض کنید ذره‌ی دیگری مانند (y_1, y_2, y_3, y_4) در اختیار داریم به‌گونه‌ای که $y_1 y_2 y_3 y_4 = 0.75$. در این صورت، اگر ذره‌ی فرزندی مانند z را تولید نماییم به‌گونه‌ای که $z_i = x_i^{\alpha} y_i^{1-\alpha}$، که در آن $\alpha \in [0,1]$، فرزند همواره قید $z_1 z_2 z_3 z_4 = 0.75$ را برآورده خواهد ساخت. این بدین دلیل است که

$$\begin{aligned} z_1 z_2 z_3 z_4 &= x_1^{\alpha} y_1^{1-\alpha} x_2^{\alpha} y_2^{1-\alpha} x_3^{\alpha} y_3^{1-\alpha} x_4^{\alpha} y_4^{1-\alpha} \\ &= (x_1 x_2 x_3 x_4)^{\alpha} (y_1 y_2 y_3 y_4)^{1-\alpha} \\ &= (0.75)^{\alpha} (0.75)^{1-\alpha} = 0.75 \end{aligned} \qquad (۴۱-۱۹)$$

می‌توان دید که این عملگر برش ویژه به‌گونه‌ای عمل می‌کند که فرزند دو والد امکان‌پذیر، همواره امکان‌پذیر باشد.

به همین ترتیب می‌توان یک عملگر جهش ویژه را برای این مسئله طراحی نمود:

---

[۱] این موضوع در مورد مسائل دنیای واقعی صادق است اما الزاما در مورد مسایل محک صادق نیستند. این تفاوت میان مسائل محک و مسائل دنیای واقعی، به قضیه‌ی no free lunch از ضمیمه‌ی ب مرتبط است.

$$x_i' \leftarrow q x_i \\ x_j' \leftarrow x_j / q \qquad (۴۲-۱۹)$$

که در آن $i \in [1,4]$ و $j \in [1,4]$، دو عدد صحیح اتفاقی متمایز بوده و q یک عدد اتفاقی است. هر ذره‌ی امکان‌پذیری مانند x که به این روش دچار جهش شود، ذره‌ی امکان‌پذیر x' را نتیجه خواهد داد.

این عملگرهای ساده‌ی جهش و بازترکیب استفاده از عملگرهای ویژه برای الگوریتم‌های تکاملی مقید را نشان می‌دهند. توجه داشته باشید که عملگرهای ویژه به نوع مسئله وابسته‌اند. برای هر مسئله‌ی بهینه‌سازی مقید، طراح الگوریتم تکاملی باید عملگرهای خاص مسئله را جهت حفظ امکان‌پذیری، فرمول‌بندی نماید. یک مثال دیگر از عملگر ویژه در [مایکلویکز و شئوناور، ۱۹۹۶] آورده شده است.

۱۹-۳-۳ ژنوکوپ

حال به بحث در مورد الگوریتمی که تحت عنوان الگوریتم ژنتیک برای بهینه‌سازی عددی مسائل مقید (Genocop) شناخته می‌شود [میکلویکز و جنیکو[1]، ۱۹۹۱]، می‌پردازیم.

این الگوریتم شامل چند ویژگی و تکنیک خاص برای بهینه‌سازی مقید می‌شود. ما ژنوکوپ را در این بخش آورده‌ایم چرا که اولین بار با ایده‌ی پیاده‌سازی عملگرهای خاص بر روی ذرات الگوریتم تکاملی جهت اطمینان از برآورده‌سازی قیود خطی، مطرح گردید.

ایده‌ی اصلی ژنوکوپ در آن است که ما گاهاً، بسته به شکل قید، می‌توانیم از عملگرهای خاص مسئله استفاده کرده و ذرات امکان‌ناپذیر را به ذرات امکان‌پذیر تبدیل نماییم. این کار را می‌توان با قیدی که به شکل معادله‌یی (۱۹-۳۸) است، انجام داد. اگر ذره‌ای داشته باشیم که قید را برآورده نمی‌سازد، آنگاه می‌توان چهارمین عنصر آن را مانند خط آخر معادله‌ی (۱۹-۴۰) جایگزین نماییم. این یک رویکرد اصلاحی خواهد بود. متناوباً، اگر سه عنصر اول از ذره‌ی فرزند را ایجاد نماییم، می‌توانیم عنصر چهارم آنگونه که در خط آخر معادله‌ی (۱۹-۴۰) نشان داده شده است، ایجاد نماییم. این یک رویکرد عملگر ویژه خواهد بود.

فرض کنید قید خطی زیر را در اختیار داریم

$$-8x_1 + x_3 \leq 0 \qquad (۴۳-۱۹)$$

اگر ذره‌ای داشته باشیم که این قید را برآورده نسازد، می‌توان به سادگی آن را با قرار دادن x_3 با یک عدد دلخواه که از $8x_1$ کوچکتر است، اصلاح نمود. در این صورت ذره‌ی اصلاح شده قید را برآورده خواهد

[1] Janikow

ساخت. این مثال نشان می‌دهد که هر قید خطی را می‌توان به سادگی و با استفاده از الگوریتم اصلاح یا عملگر ویژه، برآورده ساخت.

ژنوکوپ کارآمد است اما طراحی آن به مسئله بستگی دارد. همان‌طور که در بالا نیز نشان داده شد، ژنوکوپ به قیدهای خطی و شکل‌های خاص از قیود غیرخطی، که در آن‌ها بتوان یک متغیر را نسبت به متغیرهای دیگر حل نمود، محدود می‌باشد. این از دید تلاش کاربر یک نکته‌ی منفی و از دید کارآمدی الگوریتم تکاملی، یک نکته‌ی مثبت است.

۱۹–۳–۴ ژنوکوپ II

ژنوکوپ II [مایکلویکز و آتیا[1]، ۱۹۹٤] الگوریتم ژنوکوپ که در بالا توضیح داده شد را با یک مجازات پویا مانند آنچه که در بخش ۱۹–۲–۵ توضیح داده شد، ترکیب می‌نماید. ژنوکوپ II از عملگرهای ویژه برای بیشینه ساختن امکان‌پذیری جمعیت الگوریتم تکاملی استفاده می‌نماید. ابتدا، تمام قیود خطی را با اصلاح ذرات امکان‌ناپذیر به روش مطرح شده در ژنوکوپ، برآورده می‌سازیم. سپس، با توجه به اینکه تمام قیود خطی را برآورده ساخته‌ایم، قیود غیر خطی را با مینیمم‌سازی $\phi(x)$ در معادله‌ی (۱۹–۸)، که در آن همه‌ی قیود غیر خطی هستند، اداره می‌نماییم. وزن r_i در معادله‌ی (۱۹–۸) برابر $1/\tau$ است. ژنوکوپ II مقداری ثابت از τ را برای چند نسل نگه می‌دارد. پس از مدتی (برای مثال، پس از گذشت تعداد معینی نسل، یا پس از آنکه جزء مشخصی از جمعیت امکان‌پذیر گشت)، ژنوکوپ II، τ را کاهش می‌دهد. این کار فضار قید را افزایش داده و بدین ترتیب تدریجا باعث جذب ذرات بیشتر به مجموعه‌ی امکان‌پذیر می‌شود. مقاله‌های اولیه در زمینه‌ی ژنوکوپ II پیشنهاد می‌کنند که τ هر بار با فاکتور ۱۰ کاهش یابد [مایکلویکز و آتیا، ۱۹۹٤]، [مایکلویکز و شئوناور، ۱۹۹٦].

۱۹–۳–۵ ژنوکوپ III

ژنوکوپ III [مایکلویکز و ناژیاث[2]، ۱۹۹۵] اصلاحی دیگر بر روی ژنوکوپ می‌باشد. در این روش، یک الگوریتم هم‌تکاملی جمعیتی $P_r = \{x_r\}$ از نقاط مرجع، که همه‌ی قیود را برآورده می‌سازند، و جمعیت $P_s = \{x_s\}$ از نقاط جستجو که تمام قیود خطی را، بنا بر رویکرد استفاده شده در ژنوکوپ، برآورده می‌سازند، نگه می‌دارد. اندازه‌های P_r و P_s ممکن است با یکدیگر متفاوت باشند. ما مقادیر تابع هزینه‌ی هر ذره x_s از P_s را با استفاده از نسخه‌ی اصلاحی، تخصیص می‌دهیم. این بدین معنی است که از اطلاعات P_r برای اصلاح

[1] Attia
[2] Nazhiyath

x_s و برای به دست آوردن ذره‌ی x'_s که تمامی قیود را برآورده می‌سازد، استفاده می‌نماییم. سپس تخصیص $f(x_s) \leftarrow f(x'_s)$ را انجام می‌دهیم. x'_s را با تولید یک سری از نقاط $x = ax_s + (1-a)x_r$ برای مجموعه‌ای از اعداد اتفاقی $a \in [0,1]$ و برای $x_r \in P_r$ انتخاب شده به‌صورت اتفاقی، ایجاد می‌نماییم. این بدین معنی است که به جستجو میان مجموعه‌ای اتفاقی از نقاط که بر روی خط مستقیم میان x_s و x_r واقع شده‌اند می‌پردازیم. پس از آنکه یک x امکان‌پذیر از این جستجو به دست آمد، تخصیص $x \leftarrow x'_s$ و همچنین تخصیص $f(x_s) \leftarrow f(x'_s)$ را انجام می‌دهیم. همچنین اگر $f(x'_s) < f(x_r)$، x_r را با x'_s در P_r جایگزین می‌نماییم. در آخر، x_s را با یک احتمال مشخص ρ (که توسط کاربر تعیین می‌گردد)، با x'_s در P_s جایگزین می‌نماییم.

این سؤال که آیا باید x_s را با نسخه‌ی اصلاح شده‌ی آن x'_s جایگزین نماییم، به وراثت لامارکی[1] مربوط می‌شود: بدین معنی که، آیا یک ارگانیسم می‌تواند خصلت‌هایی را که در طول زندگی خود به دست می‌آورد به فرزندان و نسل‌های بعد منتقل نماید؟ برخی محققین ذرات را هیچ گاه با نسخه‌های اصلاح شده‌شان جایگزین نمی‌نمایند ($\rho = 0$)، در حالی که برخی دیگر ذرات را همواره با نسخه‌ی اصلاح شده‌شان جایگزین می‌نمایند ($\rho = 1$) و برخی دیگر نیز توصیه می‌کنند که اگر ρ میان ۵٪ و ۲۰٪ انتخاب شود، نتایج خوبی به دست خواهد آمد [مایکلویکز و شئوناور، ۱۹۹۶]، [اوروش[2] و دیویس، ۱۹۹۳].

شکل ۱۹-۶ طرح کلی الگوریتم ژنوکوپ III را نشان می‌دهد. اولین گام در این الگوریتم مقداردهی اولیه‌ی اتفاقی P_s می‌باشد. این جمعیت بدون توجه امکان‌پذیری و تنها با توجه به برآورده شدن قیود خطی، آنچنان که در ژنوکوپ بیان گردید، ایجاد می‌گردد. دومین گام، ارزیابی $f(x_s)$ برای هر $x_s \in P_s$ می‌باشد. این کار در پایین شکل ۱۹-۶ و با عبارت *"Evaluate f(x)"* صورت می‌پذیرد. گام سوم، مقداردهی اولیه‌ی اتفاقی P_r می‌باشد. این جمعیت را به‌گونه‌ای تولید می‌نماییم که هر ذره تمامی قیود را برآورده سازد.[3] گام چهارم ارزیابی $f(x_r)$ برای هر $x_r \in P_r$ می‌باشد. در آخر الگوریتم تکاملی را برای رشد دادن جمعیت‌های P_s و P_r اجرا می‌نماییم.

به نظر می‌رسد که الگوریتم ژنوکوپ III نتایج خوبی را به دست می‌دهد و به همین دلیل الگوریتمی جذاب برای کاربردها و تحقیقات آتی به شمار می‌آید.برای مثال، می‌توان از انواع مختلفی از الگوریتم‌های تکاملی برای تکامل جمعیت‌های P_s و P_r استفاده نمود و الزاماً نیازی به استفاده از یک الگوریتم تکاملی واحد برای هر دو جمعیت نیست. همچنین می‌توان این الگوریتم را با یک ρ خودانطباق، آزمایش نمود. توجه داشته

[1] Lamarckian Inheritance
[2] Orvosh
[3] پیدا کردن ذراتی که تمام قیود را برآورده سازند کار چالش برانگیزی است، اما در ژنوکوپ III فرض بر آن است که روشی برای این کار وجود دارد. بحث برنامه‌نویسی مقید که در بخش ۱۹,۷ مطرح خواهد گردید به این موضوع مرتبط است.

باشید که [مایکلویکز و شئوناور، ۱۹۹۶] نشان می‌دهد که نیازی نیست در هر نسل یک P_r جدید ایجاد شود؛ برای مثال، ممکن است مجبور باشیم گام *"generate a new population P_r"* از شکل ۱۹–۶ را هر چند دفعه یک بار در حلقه اجرا نماییم. این کار تلاش محاسباتی را کاهش خواهد داد. در آخر، ترکیبات ژنوکوپ III با سایر الگوریتم‌های تکاملی مقید ممکن است عملکرد را بهبود ببخشد.

جمعیتی اتفاقی از نقاط جستجو $\leftarrow P_s$

$f(x_s)$ را برای هر $x_s \in P_s$ محاسبه کن

جمعیتی اتفاقی از نقاط امکان‌پذیر $\leftarrow P_r$

$f(x_r)$ را برای هر $x_r \in P_r$ محاسبه کن

تا زمانی که شرایط توقف برآورده نشده است

  از یک الگوریتم تکاملی با مقادیر $f(x_s)$ برای ایجاد جمعیتی جدید از P_s استفاده کن

  $f(x_s)$ را برای هر $x_s \in P_s$ در الگوریتم زیر محاسبه کن

  از یک الگوریتم تکاملی با مقادیر $f(x_r)$ برای ایجاد جمعیتی جدید از P_r استفاده کن

  $f(x_r)$ را برای هر $x_r \in P_r$ محاسبه کن

نسل بعدی

ارزیابی $f(x_s)$

اگر $x_s \in \mathcal{F}$

  $f(x_s)$ را با استفاده از تابع هزینه محاسبه کن

در غیر این صورت

  $x'_s \leftarrow x_s$

  تا زمانی که $x'_s \notin \mathcal{F}$

    یک x_r را به‌صورت اتفاقی از P_r انتخاب کن

    $a \sim U[0,1]$ را به‌صورت اتفاقی تولید کن

    $x'_s \leftarrow a x_s + (1-a) x_r$

  $f(x_s) \leftarrow f(x'_s)$

  اگر $f(x'_s) < f(x_r)$ آنگاه $x_r \leftarrow x'_s$

  $a \sim U[0,1]$ را به‌صورت اتفاقی تولید کن

  اگر $a < \rho$ آنگاه $x_s \leftarrow x'_s$

پایان اگر

شکل ۱۹–۶ طرح کلی الگوریتم ژنوکوپ III برای مینیمم‌سازی $f(x)$ که دارای چند قید است.

۱۹–۴ رویکردهای دیگر برای بهینه‌سازی مقید

این بخش به‌طور خلاصه به بحث در مورد چند رویکررد دیگر به بهینه‌سازی مقید می‌پردازد. این رویکردها، رویکردهای تابع مجازات نیستند، و همچنین شامل نمایش‌ها و عملگرهای ویژه نمی‌شوند. به همین دلیل این رویکردها را در بخشی مجزا مورد بحث قرار داده‌ایم. بخش ۱۹–٤–۱ به بحث در مورد الگوریتم‌های فرهنگی، و بهش ۱۹–٤–۲ به بحث در مورد بهینه‌سازی چندهدفه برای مسائل مقید، می‌پردازد.

۱۹–۴–۱ الگوریتم‌های فرهنگی

الگوریتم‌های فرهنگی (CA) الگوریتم‌های تکاملی هستند که از فضای عقیده برای پیش بردن تکامل استفاده می‌نمایند، بدین معنی که مادامی که یک CA در تلاش برای حل یک مسئله‌ی بهینه‌سازی است، جستجوی آن دارای گرایشاتی در جهت‌های مشخصی است. CAهای مقید روش‌های مجازات نیستند، چرا که روش‌های مجازات تابع هزینه‌ی ارزیابی شده‌ی راه‌حل‌های امکان‌ناپذیر را افزایش می‌دهند، در حالی که CAهای مقید جستجو را به‌گونه‌ای جهت‌دهی می‌کنند که راه‌حل‌های امکان‌ناپذیر دارای شانس کمتری برای وجود در جمعیت باشند. با این حال، استفاده از فضای عقیده در یک CA گستره‌ی وسیعی از رویکردها و پیاده‌سازی‌های ممکن را، به دلیل بنیاد فرهنگی CAها، به وجود می‌آورد. CAها را در فصل ۱۵ مورد بحث قرار دادیم بنابراین در اینجا بیش از این به آن‌ها نمی‌پردازیم و تحقیق در مورد استفاده از CAها در بهینه‌سازی مقید را به خوانندگان واگذار می‌نماییم [بکرا و کوئلو کوئلو، ۲۰۰٤]، [کوئلو کوئلو و بکرا، ۲۰۰۲].

۱۹–۴–۲ بهینه‌سازی چندهدفه

فصل ۲۰ به بحث در مورد بهینه‌سازی چندهدفه (MOP[1]) خواهد پرداخت. یک MOP مسئله‌ای است که در آن می‌خواهیم به‌طور همزمان M تابع هزینه را مینیمم نماییم:

$$\min_x [f_1(x), \dots, f_M(x)] \qquad (٤٤–١٩)$$

[1] Multi-Objective Optimization

یک مسئله‌ی بهینه‌سازی مقید را می‌توان با تعریف اولین هدف به‌عنوان هزینه و تعریف سایر اهداف به‌عنوان قیود، به‌صورت یک MOP در نظر گرفت. یک مسئله‌ی بهینه‌سازی مقید را که به‌صورت معادله‌ی (۱۹–۱) نوشته شده است را در نظر بگیرید:

$$\min_x f(x) \quad \text{به‌طوری که } g_i(x) \le 0 \text{ برای } i \in [1,m] \\ \text{و } h_j(x) = 0 \text{ برای } j \in [1,p] \qquad (۱۹–۴۵)$$

این مسئله معادل MOP از معادله‌ی (۱۹–۴۴) می‌باشد اگر:

$$\begin{aligned} f_1(x) &= f(x) \\ f_2(x) &= G_1(x) \\ &\cdot \\ &\cdot \\ &\cdot \\ f_M(x) &= G_{m+p}(x) \end{aligned} \qquad (۱۹–۴۶)$$

که در آن $G_i(x)$ در معادله‌ی (۱۹–۸) داده شده است. بنابراین، می‌توان از هر الگوریتم MOP برای حل یک مسئله‌ی بهینه‌سازی مقیید استفاده نمود. فصل ۲۰ الگوریتم‌های تکاملی برای MOPها را معرفی خواهد نمود. تحقیقات در مورد استفاده از الگوریتم‌های MOP برای بهینه‌سازی مقید را می‌توان در [آگوییر[1] و همکاران، ۲۰۰۴]، [کای[2] و وانگ[3]، ۲۰۰۶]، [کوئلو کوئلو، ۲۰۰۰a]، [کوئلو کوئلو، ۲۰۰۲] و [مزورا–مونتس و کوئلو کوئلو، ۲۰۰۸] و بسیاری مراجع دیگر یافت.

۱۹–۵ رتبه‌بندی راه‌حل‌های نامزد

بخش‌های قبل، چند روش برای رتبه‌بندی راه‌حل‌های نامزد برای مسائل بهینه‌سازی مقید را معرفی نمودند. این بخش رویکردهایی که قبلا معرفی شده‌اند را خلاصه کرده و چند رویکرد دیگر را نیز معرفی می‌نماید. ابتدا خلاصه‌ای از رویکردهای قبلی را بیان می‌کنیم.

- معادله‌ی (۱۹–۸) تابع هزینه را با تابعی از میزان بزرگی نقض قید، مجازات می‌نماید.
- معادله‌ی (۱۹–۱۰)، معادله‌ی (۱۹–۸) را به‌گونه‌ای اصلاح می‌نماید که تمام ذرات امکان‌پذیر دارای رتبه‌ی بهتری از تمامی ذرات امکان‌ناپذیر باشند. در این رویکرد، ذرات امکان‌ناپذیر بر اساس میزان نقض قید رتبه‌بندی می‌گردند. بخش ۱۹–۲–۱۳ نیز از این رویکرد استفاده می‌کند.

[1] Aguirre
[2] Cai
[3] Wang

- معادله‌ی (۱۹-۱۳) تمامی ذرات امکان‌پذیر را از تمامی ذرات امکان‌ناپذیر بهتر رتبه‌بندی کرده به‌گونه‌ای که ذرات امکان‌ناپذیر به جای میزان بزرگی نقض قید، بر اساس تعداد نقض قیدهایشان رتبه‌بندی می‌گردند.
- معادله‌ی (۱۹-۱٤) تابع هزینه را هم با میزان بزرگی نقض قید و هم با تعداد نقض قید مجازات می‌نماید.
- معادله‌ی (۱۹-۱۹) تابع هزینه را با مجازات نقض قیدی که با افزایش شماره‌ی نسل افزایش می‌یابد، مجازات می‌نماید.
- معادله‌ی (۱۹-۲۳) و (۱۹-۳٤) از مجازات نقض قیدی استفاده می‌نمایند که تابع تعداد ذرات امکان‌پذیر در جمعیت می‌باشد.
- بخش ۱۹-۲-۷ و ۱۹-۲-۸ مجازات هزینه را بر اساس ترکیبی از تعداد ذرات امکان‌پذیر و هزینه‌یی نسبی ذرات مختلف، تنظیم می‌نمایند.
- بخش ۱۹-۲-۱۲ از یک فرایند اتفاقی برای تعیین رتبه‌ی راه‌حل‌های نامزد استفاده می‌نماید.

تا به اینجا چندین رویکرد رتبه‌بندی برای بهینه‌سازی مقید را مورد بحث قرار داده‌ایم. حال به معرفی سه رویکرد رتبه‌بندی دیگر می‌پردازیم.

۱۹-۵-۱ رتبه‌بندی بیشترین نقض قید

به جای استفاده از مجموع میزان بزرگی نقض قید و یا تعداد نقض قیدها، می‌توان ذرات را با استفاده از بیشترین بزرگی نقض قید رتبه‌بندی نمود [تاکاهاما[1] و ساکای[2]، ۲۰۰۹]. در این صورت، تابع هزینه‌ی مجازات شده از معادله‌ی (۱۹-۸) را با معادله‌ی زیر جایگزین می‌نماییم

$$\phi(x) = f(x) + \max G_i(x) \qquad (٤٧-١٩)$$

همچنین می‌توان راه‌حل‌های نامزد را با استفاده از ترکیبی از مجموع بزرگی نقض قید، تعداد نقض قیدها و بیشترین بزرگی نقض قید، رتبه‌بندی نمود.

[1] Takahama
[2] Sakai

۱۹–۵–۲ رتبه‌بندی ترتیب قید

[ری و همکاران، ۲۰۰۹b]، راهی را برای ترکیب بزرگی نقض قید و تعداد نقض قیدها مطرح می‌کند. قرض کنید x_k، kاُمین ذره در جمعیتی N ذره‌ای است. همچنین فرض کنید یک مسئله‌ی بهینه‌سازی مقید با $(m+p)$ قید در اختیار داریم. ما از نماد $G_i(x_k)$ برای نشان دادن میزان بزرگی نقض iاُمین قید توسط x_k استفاده می‌نماییم (توجه داشته باشید که $G_i(x_k) \geq 0$). ما سپس از نماد $c_i(x_k)$ برای نشان دادن رتبه‌ی iاُمین نقض قید توسط x_k (که رتبه‌ی کمتر به منزله‌ی نقض قید کمتر می‌باشد)، استفاده می‌نماییم. اگر $G_i(x_k) = 0$ باشد، آنگاه $c_i(x_k) = 0$ خواهد بود. توجه داشته باشید که $c_i(x_k) \in [0, N]$ می‌باشد. مثالی ساده از یک جمعیت با اندازه‌ی پنج به‌صورت زیر است:

$$\left.\begin{matrix} G_1(x_1) = 3.5 \\ G_1(x_2) = 5.7 \\ G_1(x_3) = 0.0 \\ G_1(x_4) = 1.3 \\ G_1(x_5) = 0.0 \end{matrix}\right\} \rightarrow \begin{cases} c_1(x_1) = 2 \\ c_1(x_2) = 3 \\ c_1(x_3) = 0 \\ c_1(x_4) = 1 \\ c_1(x_5) = 0 \end{cases} \qquad (۴۸–۱۹)$$

سپس اندازه‌ی نقض قید را به‌صورت زیر تعریف می‌نماییم

$$v(x_k) = \sum_{i=1}^{m+p} c_i(x_k) \qquad (۴۹–۱۹)$$

۱۹–۵–۳ مقایسات سطح ϵ

مقایسات سطح ϵ مشابه رویکرد مجازات ایستا از بخش ۱۹–۲–۱ می‌باشد، از این لحاظ که از وزن‌های تابع مجازات متفاوتی، بسته به سطح نقض قید، استفاده می‌شود. با این حال، مقایسات سطح ϵ تنها از دو سطح نقض قید برای رتبه‌بندی استفاده می‌نمایند [تاکاهاما و ساکای، ۲۰۰۹].

ابتدا مقدار نقض قید $M(x)$ برای هر ذره x را یا با ترکیب تمام نقض قیدها و یا با پیدا نمودن بیشترین مقدار نقض قید، اندازه می‌گیریم:

$$M(x) = \begin{cases} \sum_{i=1}^{m+p} G_i(x) & \text{روش جمع قید} \\ \max_i G_i(x) & \text{روش بیشترین قید} \end{cases} \qquad (۵۰–۱۹)$$

همان‌طور که در بش ۱۹–۵–۱ ذکر شد، می‌توان روش جمع قید و روش بیشترین قید را با یکدیگر ترکیب نمود و از این طریق $M(x)$ را به دست آورد.

سپس، ذره‌های x و y را به‌صورت زیر رتبه‌بندی می‌نماییم:

x از y بهتر است اگر:

$$\begin{cases} f(x) < f(y) \text{ و } M(x) \le \epsilon \text{ و } M(y) \le \epsilon, & \text{یا} \\ f(x) < f(y) \text{ و } M(x) = M(y), & \text{یا} \\ M(x) < M(y) \text{ و } M(y) > \epsilon \end{cases} \qquad (۵۱-۱۹)$$

که در آن $\epsilon > 0$ یک آستانه‌ی نقض قید است که توسط کاربر تعیین می‌گردد. می‌توان دید که یک نقض قید که کمتر از ϵ است، در هنگام رتبه‌بندی امکان‌پذیر محسوب می‌شود. توجه داشته باشید که اگر $\epsilon = \infty$، آنگاه ذرات تنها بر اساس هزینه رتبه‌بندی خواهند شد. اگر $\epsilon = 0$، ذرات امکان‌پذیر بر اساس هزینه و ذرات امکان‌ناپذیر تنها بر اساس نقض قیدشان رتبه‌بندی شده و ذرات امکان‌پذیر همواره دارای رتبه‌ی بهتری نسبت به ذرات امکان‌ناپذیر خواهند بود. ما معمولاً مقدار ϵ را با افزایش شماره‌ی نسل کاهش داده و بدین ترتیب میزان اهمیت برآورده ساختن قید را به تدریج افزایش می‌دهیم:

$$\epsilon(0) = M(x_p)$$
$$\epsilon(t) = \begin{cases} \epsilon(0)\left(1 - \frac{t}{T_c}\right)^c & \text{اگر } 0 < t < T_c \\ 0 & \text{اگر } t \ge T_c \end{cases} \qquad (۵۲-۱۹)$$

که در آن $\epsilon(t)$ مقدار ϵ در طول نسل tاُم می‌باشد. x_p ذره‌ای است که دارای pاُمین مقدار نقض قید به لحاظ کوچکی بوده به‌گونه‌ای که $p = N/5$ و N اندازه‌ی جمعیت است. c و T_c نیز پارامترهای میزان‌سازی هستند که اغلب دارای مقادیر $c = 100$ و $T_c = t_{max}/5$ می‌باشند [تاکاهاما و ساکای، ۲۰۰۹]. همچنین می‌توان دیگر پارامترهای میزان‌سازی را برای کاهش ϵ به‌عنوان تابعی از t امتحان نمود.

۱۹-۶ مقایسه‌ای میان روش‌های اداره کردن قید

این بخش مقایسه‌ای میان ۹ روش نامساوی اداره نمودن قید را ارائه می‌دهد. ما از معادله‌ی (۱۹-۱۶) برای اندازه‌گیری بزرگی نقض قید یک راه‌حل نامزد استفاده می‌نماییم در جایی که $G_i(x)$ با استفاده از معادله‌ی (۱۹-۸) و با $\beta = 1$ به دست می‌آید. روش‌های اداره نمودن قید که آزمایش می‌نماییم عبارتند از:

۱. EE: الگوریتم تکاملی گزینشی از بخش ۱۹-۲-۳.

۲. DP: روش مجازات پویا از معادله‌ی (۱۹-۱۶) در بخش ۱۹-۲-۵ با $c = 10$ و $\alpha = 2$.

۳. DS: روش مجازات پویا ترکیب شده با روش برتری نقاط امکان‌پذیر، که در معادله‌ی (۱۹–۱۸) و در بخش ۱۹–۲–۵ تعریف گردیده است، با $c = 10$ و $\alpha = 2$.

٤. EP: روش مجازات نمایی پویا از معادله‌ی (۱۹–۲۰) در بخش ۱۹–۲–۵–۱ با $\alpha = 10$.

۵. ES: روش مجازات نمایی پویا ترکیب شده با روش برتری نقاط امکان‌پذیر، که در معادله‌ی (۱۹–۲۲) و در بخش ۱۹–۲–۵–۱ تعریف گردیده با $\alpha = 10$.

٦. AP: روش مجازات انطباقی از معادله‌ی (۱۹–۲۳) در بخش ۱۹–۲–٦ با $\beta_1 = 4$، $\beta_2 = 3$ و $k = n$، که در آن n ابعاد مسئله است.

۷. SR: روش رتبه‌بندی اتفاقی از بخش ۱۹–۲–۱۲ با $P_f = 0.45$.

۸. NP: روش مجازات نیچه از بخش ۱۹–۲–۱۳.

۹. ϵC: روش مقایسه‌ی سطح ϵ از بخش ۱۹–۵–۳ با $c = 100$، $T_c = 200$ و $p = N/5$. که در آن N اندازه‌ی جمعیت است.

از آنجا که روش‌های قید ذکر شده در بالا تنها به چگونگی رتبه‌بندی راه‌حل‌های نامزد می‌پردازند، می‌توان از هر الگوریتم تکاملی همراه با این روش‌های اداره نمودن قید استفاده نمود. در این بخش از الگوریتم BBO شکل ۱٤–۳ در حین محاسبه‌ی برازندگی هر ذره با یکی از روش‌های اداره نمودن قید در لیست بالا، استفاده می‌نماییم.

ما روش‌های اداره نمودن قید را بر روی محک‌های کنگره‌ی محاسبات تکاملی ۲۰۱۰[1]، که در ضمیمه‌ی ج.۲ آورده شده‌اند و با اندازه‌ی بعد $n = 10$، آزمایش می‌نماییم. با این حال، تنها از محک‌هایی استفاده می‌نماییم که شامل قیدهای تساوی نمی‌شوند یعنی: C01، C07، C08، C13، C14 و C15.

مسائل با قیدهای تساوی نیاز به اداره‌ی ویژه دارند. همان‌طور که در بخش ۱۹–۱–۲–۲ ذکر شد، قیدهای تساوی بسیار سخت‌گیرانه هستند. اگر یک جمعیت ابتدایی را به‌صورت اتفاقی تولید نماییم، احتمال آنکه ذره‌ای تولید شود که قید تساوی را برآورده سازد، الزاماً صفر خواهد بود. دو روش اساسی برای تولید ذراتی که قید تساوی را برآورده سازند وجود دارد: (۱) استفاده از اطلاعات خاص مسئله به همان گونه‌ای که در بخش ۱۹–۳ گفته شد، (۲) استفاده از معادله‌ی (۱۹–۸) برای تبدیل قید تساوی به قید نامساوی، استفاده از مقادیر بزرگ ϵ در ابتدای الگوریتم تکاملی و کاهش تدریجی آن با افزایش شماره‌ی نسل. همچنین امکان فرمول‌بندی الگوریتم‌های تکاملی کلی با قید تساوی نیز وجود دارد. تمرکز این بخش بر اداره‌ی قیدهای

[1] CEC 2010

نامساوی قرار دارد. بنابراین، استفاده از انطباق‌های پویای ϵ جهت اجرای مقایسه‌های مشابه برای مسائل با قیدهای تساوی را به عهده‌ی خواننده واگذار می‌نماییم.

ما از اندازه‌ی جمعیت ۱۰۰ و حد بالای نسل برابر ۱۰۰ استفاده می‌نماییم. با این کار، به مجموع ۱۰۰۰۰ ارزیابی تابع در طول شبیه‌سازی الگوریتم تکاملی نیاز خواهیم داشت. توجه داشته باشید که بسیاری مطالعات از صدها هزار ارزیابی تابع استفاده نموده‌اند. ما معتقدیم که این مقدار برازندگی تابع برای مسائل دنیای واقعی واقع‌گرایانه نیست. ارزیابی‌های تابع در هنگام حل مسائل دنیای واقعی به لحاظ محاسباتی پرهزینه هستند و در نتیجه اجرای صدها هزار ارزیابی تابع برای آن‌ها منطقی نیست. ما محققین و دانشجویان را به جای استفاده از مقادیر بسیار بزرگ و غیرمنطقی برای تعداد ارزیابی‌های تابع، به تمرکز بیشتر بر روی دستیابی به همگرایی خوب با تعداد ارزیابی‌های تابع نسبتاً کم تشویق می‌نماییم. این کار میانبری است از نظریات به کاربردهای عملی در تحقیقات الگوریتم تکاملی. برای اطلاعات بیشتر به بخش ۲۱-۱ رجوع نمایید.

به دلیل استفاده از تعدد از ارزیابی‌های تابع کم، نتایج ارائه شده در این بخش با نتایج منتشر شده در مقالات دیگر قابل قیاس نیست. اما در اینجا هدف صرفاً دستیابی به بهترین عملکرد با تعداد غیرمعقولی از ارزیابی‌های تابع نیست. در اینجا هدف ارائه‌ی یک مقایسه میان روش‌های اداره نمودن قید در یک زمین بازی مشترک است. استفاده از ۱۰۰ ذره و ۱۰۰ نسل تعادل بسیار خوبی را به دست می‌دهد چرا که در این صورت تعداد ارزیابی‌های تابع آنقدر زیاد نخواهد بود که اجرای هر شبیه‌سازی مدت بسیار زیادی به طول بیانجامد و در عین حال نیز این تعداد ارزیابی تابع به قدری هست که الگوریتم‌های تکاملی و روش‌های اداره نمون قید فرصت تمایز از یکدیگر را بیابند.

ما جهش را با جایگزین نمودن یک خاصیت در یک ذره با مقداری اتفاقی، که دارای توزیع یکنواخت بر روی دامنه‌ی جستجو می‌باشد، پیاده‌سازی می‌نماییم. هر خاصیت موجود در هر ذره دارای احتمال ۱٪ جهش در هر نسل می‌باشد. ما همچنین از پارامتر نخبه‌گرایی برابر ۲، که به معنی نگه داشتن بهترین دو ذره از هر نسل به نسل بعد می‌باشد، استفاده می‌نماییم.

جهت نخبه‌گرایی، بهترین ذره را ذره‌ی امکان‌پذیری در نظر می‌گیریم که دارای کمترین هزینه است. اگر هیچ ذره‌ی امکان‌پذیری وجود نداشته باشد، آنگاه بهترین ذره را به‌عنوان ذره‌ای که دارای کمترین هزینه‌ی مجازات شده است در نظر می‌گیریم. توجه داشته باشید که هزینه‌ی مجازات شده از طریق یکی از روش‌های اداره نمودن قید بالا به دست می‌آید. دلیل استفاده از این رویکرد آن است که برخی از روش‌های اداره نمودن قید در بالا، به ذرات امکان‌ناپذیر رتبه‌ی بهتری نسبت به ذرات امکان‌پذیر اختصاص می‌دهند. از این رویکرد هنگامی که تعداد زیادی از ذرات وجود داشته و رتبه‌بندی به انتخاب برای بازترکیب منتهی می‌شود، می‌توان

استفاده نمود. اما اگر تنها دو ذره‌ی نخبه را از یک نسل به نسل دیگر نگه می‌داریم، باید اطمینان حاصل نماییم که ذرات امکان‌پذیر بر ذرات امکان‌ناپذیر ترجیح داده می‌شوند. این کار باعث می‌شود همین که الگوریتم تکاملی به یک ذره‌ی امکان‌پذیر دست یافت، همیشه و تا آخر شبیه‌سازی حداقل یک ذره‌ی امکان‌پذیر در اختیار داشته باشد.

جدول ۱۹-۱ مقایسه‌ای از روش‌های اداره نمودن قید بر روی شش محک بهینه‌سازی مقید CEC 2010 که شامل قیدهای تساوی نمی‌شوند، را نشان می‌دهد. نتایج ارائه شده در این جدول بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده‌اند. ما مقادیر آفست $\{o_i\}$ را برای محک‌های ضمیمه‌ی ج.۲ به‌صورت اتفاقی تولید می‌نماییم، اما از $\{o_i\}$ یکسان برای تمامی روش‌های اداره نمودن قید برای یک آزمایش مونت کارلو استفاده می‌نماییم. برخی از محک‌ها از یک ماتریس چرخش M، که به‌صورت اتفاقی تولید می‌شود، استفاده می‌نمایند.

جدول ۱۹-۱ چند ویژگی جالب را نشان می‌دهد. اول آنکه باید توجه کرد که تمامی الگوریتم‌های در مورد C01 عملکرد یکسانی دارند. این موضوع حاکی از آن است که یا C01 بسیار آسان بوده و همه‌ی روش‌ها بسیار خوب کار می‌کنند و یا آنکه بسیار سخت بوده و تمامی روش‌ها عملکرد بدی در مورد آن دارند. دوم آنکه، تمامی الگوریتم‌ها به غیر از EE در مورد C13 عملکرد نسبتاً یکسانی دارند. EE نمی‌تواند حتی بعد از ۲۰ شبیه‌سازی مونت کارلو راه‌حلی امکان‌پذیر را برای C13 بیابد و به همین دلیل مقدار تابع هزینه‌ی آن برابر ∞ قرار داده شده است. EE در مورد C13، C14 و C15 بدترین عملکرد را دارد. این موضوع از این جهت جالب است که این‌ها محک‌هایی هستند که برآورده ساختن قیدهایشان از بقیه سخت‌تر است (جدول ج.۱ در ضمیمه‌ی ج.۲ را ببینید).

همچنین بنا بر جدول ۱۹-۱، به‌طور میانگین، روش مجازات نمایی از معادله‌ی (۱۹-۲۰) در بخش ۱۹-۲-۵-۱ بهترین عملکرد را دارد. با این حال، مسائل بهینه‌سازی مقید بسیاری وجود دارند که در مقالات متعدد مورد بررسی و بحث قرار گرفته‌اند. همچنین روش‌های اداره نمودن قید که در لیست بالا آورده شده‌اند دارای تعدادی پارامتر میزان‌سازی می‌باشند. شاید اگر از مقادیر دیگری برای این پارامترهای میزان‌سازی و یا از توابع محک متفاوتی استفاده می‌نمودیم، نتایج دیگری به دست می‌آمد.

جدول ۱۹-۱ مقایسه‌ی میان بهترین مقادیر تابع هزینه‌ی امکان‌پذیر که توسط نه الگوریتم BBO همراه با اداره نمودن قید بر روی شش مسئله‌ی محک ۱۰ بعدی پیدا شده است. نایج بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است. لیستی که در ابتدای بخش ۱۹-٦ آورده شده است را جهت آگاهی از معانی اختصارات ببینید. بهترین هزینه برای هر محک با هونت درشت نمایش داده شده است.

	C01	*C07*	*C08*	*C13*	*C14*	*C15*
EE	*-0.46*	***19800***	2.39×10^5	∞	5.78×10^{13}	6.911×10^{13}
DP	*-0.46*	*31900*	1.51×10^5	*-600*	1.64×10^{13}	0.066×10^{13}
DS	*-0.45*	*24700*	1.46×10^5	*-601*	2.34×10^{13}	0.228×10^{13}
EP	*-0.44*	*22000*	$\mathbf{1.03 \times 10^5}$	*-593*	$\mathbf{0.01 \times 10^{13}}$	$\mathbf{0.001 \times 10^{13}}$
ES	***-0.47***	*56800*	1.32×10^5	***-606***	0.01×10^{13}	0.005×10^{13}
AP	*-0.46*	*21000*	1.64×10^5	*-599*	0.13×10^{13}	0.072×10^{13}
SR	*-0.46*	*50500*	1.25×10^5	*-592*	4.93×10^{13}	0.231×10^{13}
NP	*-0.46*	*30900*	1.97×10^5	*-596*	0.99×10^{13}	0.353×10^{13}
ϵC	*-0.46*	*30900*	7.68×10^5	*-604*	2.74×10^{13}	0.160×10^{13}

۱۹-۷ نتیجه‌گیری

در این فصل دیدیم که روش‌های اداره نمودن قید زیادی وجود دارند که می‌توان از آن‌ها همراه با الگوریتم‌های تکاملی استفاده نمود. بسیاری از این الگوریتم‌ها دارای سطح عملکرددی یکسانی هستند. به جای آزمایش تمامی این الگوریتم‌ها جهت پیدا نمودن بهترین روش، بهتر است چند قاعده که ممکن است هنگام حل مسائل بهینه‌سازی مهم باشند را به خاطر بسپاریم.

قواعد مهم برای بهینه‌سازی مقید

- همانند مسائل بهینه‌سازی غیرمقید، هر چه بیشتر بتوانیم از اطلاعات خاص مسئله در الگوریتم تکاملی خود استفاده نماییم، احتمال موفقیت بیشتر خواهد بود. اداره نمودن قیود با استفاده از عملگرها و نمایش‌های ویژه راه مؤثرتری نسبت به رویکردهای کلی‌تر می‌باشد. استفاده ابزارهای بهینه‌سازی جعبه سیاه آسان و گاهی لازم است، اما تقریباً همیشه می‌توان با استفاده از خبرگی دامنه‌ی دشوار عملکرد بهتری را به دست آورد.
- در یک مسئله‌ی بهینه‌سازی مقید باید میزان سختی برآورده ساختن قیود را برآورد کنیم. این میزان سختی را می‌توان با پارامتر زیر اندازه گرفت:

$$\rho = |\mathcal{F}|/|\mathcal{S}| \qquad (۵۳-۱۹)$$

که در آن $|\mathcal{F}|$ اندازه‌ی مجموعه‌ی امکان‌پذیر بوده و $|\delta|$ نیز اندازه‌ی فضایی جستجو است. می‌توان ρ را با تولید تعداد زیادی ذره به‌صورت اتفاقی در فضای جستجو و و آزمایش این که چند عدد از آن‌ها قیود را برآورده می‌سازند، تخمین زد. در این مرحله نیازی به ارزیابی تابع هزینه نیست. هدف این کار، برآورد میزان سختی قیود است. اگر درصد کمی از ذرات قیود را برآورده سازند، قیود سخت بوده و الگوریتم تکاملی مقید باید بر روی برآورده ساختن قیود تمرکز نماید. اگر درصد بالایی از ذرات اتفاقی قیود را برآورده سازند، آنگاه قیود نسبتاً ساده بوده و الگوریتم تکاملی مقید می‌تواند بیشتر بر روی مینیمم‌سازی تابع تمرکز نماید.

- ما باید میزان سختی برآورده ساختن هر قید را اندازه بگیریم. همانند بالا، این کار را می‌توان با تولید تعداد بسیار زیادی ذره در دامنه‌ی جستجو انجام داد. قیودی که توسط تعداد بسیار کمی از ذرات اتفاقی برآورده می‌گردند قیود سخت بوده و الگوریتم تکاملی مقید باید بر روی برآورده ساختن آن قیود تمرکز نماید. قیدهایی که توسط تعداد نسبتاً زیادی از ذرات برآورده می‌گردند قیود ساده بوده و الگوریتم تکاملی مقید نیازی به تمرکز زیاد بر روی آن‌ها نخواهد داشت. متناوبا، می‌توان قیود را نرمالیزه کرد تا میزان سختی برآورده ساختن تمام قیود با هم برابر باشد.
- یکی از بزرگترین چالش‌ها در هر مسئله‌ی بهینه‌سازی مقید، پیدا کردن راه‌حل‌های امکان‌پذیر است. این موضوع خصوصا در مورد مسائل با قیود تساوی صدق می‌کند. در این مورد، شاید بهتر باشد الگوریتم برآورده‌سازی قید را قبل و یا به جای الگوریتم تکاملی مقید اجرا نماییم. الگوریتم‌های برآورده‌سازی قید در حوزه‌ای مطالعاتی به نام برنامه‌نویسی قید قرار دارند. برنامه‌نویسی قید در خارج از محدوده‌ی این کتاب است اما زمینه‌ای بسیار مهم در ارتباط با بهینه‌سازی مقید می‌باشد. هر کس که به‌طور جدی به بهینه‌سازی مقید علاقه‌مند باشد باید برنامه‌نویسی قید را مورد مطالعه قرار دهد. برخی مقدمات مناسب در این زمینه را می‌توان در [دچتر[1]، ۲۰۰۳]، [موریوت[2] و استاکی[3]، ۱۹۹۸] و [روسی[4] و همکاران، ۲۰۰۶] یافت.
- با وجود تمام نکات بالا، میزان سختی برآورده‌سازی قید الزاماً نشانی از میزان سختی مسئله‌ی بهینه‌سازی مقید نیست. برخی مسائل با ناحیه‌ی امکان‌پذیری نسبتاً کوچک برای الگوریتم‌های تکاملی مقید سخت نیستند. برای مثال، در [مایکلویکز و شئوناور، ۱۹۹۶] دو مسئله با

[1] Dechter
[2] Morriott
[3] Stuckey
[4] Rossi

$\rho = 0.0111\%$ و $\rho = 0.0003\%$ گزارش شده‌اند که حل آن‌ها برای الگوریتم‌های تکاملی مقید نسبتاً ساده بوده است.

- هنگام اجرای یک الگوریتم تکاملی مقید، باید حساب تعداد ذراتی که قیدها را برآورده می‌سازد از یک نسل به نسل دیگر داشته باشیم. جمعیت‌هایی که در آن‌ها همه‌ی ذرات امکان‌پذیرند معمولاً ناکارآمد هستند. بنابراین باید حواسمان باشد که هم ذرات امکان‌پذیر و هم ذرات امکان‌ناپذیر در جستجوی ما برای رسیدن به بهینه‌ی مقید وجود داشته باشند.

تحقیقات کنونی و آتی در زمینه‌ی الگوریتم‌های تکاملی مقید

بهینه‌سازی تکاملی مقید یک زمینه تحقیقاتی فعال است چرا که: (۱) این زمینه یک زمینه‌ی نسبتاً جدید است، (۲) در نظریات دارای کمبودهایی است و (۳) مسائل بهینه‌سازی دنیای واقعی تقریباً همیشه مقید هستند. فصل حاضر را با ذکر برخی جهت‌های محبوب و مهم در تحقیقات حاضر به پایان می‌بریم.

- بیشتر تحقیقات کنونی شامل شرکت دادن روش‌های اداره‌ی قید استاندارد، مانند آنچه که در ایین فصل مورد بحث قرار گرفت، در الگوریتم‌های تکاملی جدیدتر می‌شود. هر روز الگوریتم‌های تکاملی جدید معرفی می‌گردند. این الگوریتم‌های تکاملی جدید معمولاً چیزی نیستند جز اصلاحاتی بر روی الگوریتم‌های تکاملی قدیمی‌تر، اما گاهاً دارای ویژگی‌ها و قابلیت‌های جدید و متمایزی هستند (فصل ۱۷ را ببینید). اطلاع از نحوه‌ی عملکرد روش‌های اداره‌ی قید کنونی هنگامی که در انواع مختلف الگوریتم‌های تکاملی به کار می‌روند، حائز اهمیت است. عملکرد نسبی الگوریتم‌های تکاملی مختلف بر روی مسائل غیرمقید الزاماً دارای همبستگی خاصی با عملکرد نسبی آن‌ها هنگامی که برای مسائل مقید به کار می‌روند، نیست.
- این فصل به بحث در مورد مسائل بهینه‌سازی مقید پرداخت و فصل ۲۰ مسائل بهینه‌سازی چندهدفه را مورد بحث قرار خواهد داد. تحقیقات کنونی در حال ترکیب نمودن ان دو زمینه جهت پیدا نمودن الگوریتم‌هایی برای دستیابی به راه‌حل مسائل بهینه‌سازی چندهدفه‌ی مقید می‌باشند [ین، ۲۰۰۹].
- نتایج نظری می‌توانند زمینه‌ای بسیار مفید برای تحقیقات آتی در بهینه‌سازی مقید باشند. این کتاب مدل‌های مارکوف، مدل‌های سیستم پویا و نظریه‌ی شما را برای GAها و GP مورد بحث قرار داده است. شاید این ابزارها را بتوان برای تحلیل الگوریتم‌های تکاملی مقید به کار برد.
- یکی از موضوعاتی که به برنامه‌نویسی قید (که در بالا ذکر شد) مرتبط است، جستجوی مرز قید جهت حل مسائل بهینه‌سازی مقید می‌باشد [لگوئیزامون و کوئلو کوئلو، ۲۰۰۹]. جستجوی مرز به

برنامه‌نویسی قید مرتبط است اما برنامه‌نویسی قید بر یافتن راه‌حل‌های امکان‌پذیر تمرکز کرده در حالی که جستجوی مرز به رشد دادن جمعیتی که در مرز قید واقع شده است، می‌پردازد.

- همان‌طور که پیش از این نیز ذکر شد، برخی مسائل دارای قیودی هستند که برآورده ساختنشان سخت بوده و به همین دلیل یافتن نواحی امکان‌پذیر در فضای جستجو چالش‌برانگیز می‌باشد. با این حال، در ورای یافتن نواحی امکان‌پذیر، چالش طراحی یک الگوریتم تکاملی، که بتواند به‌طور مؤثر بین نواحی امکان‌پذیر و نواحی امکان‌ناپذیر نوسان نماید، قرار دارد. این نوع رفتار معمولاً برای مسائلی که جوابشان در مرز قید واقع می‌شود، مفید است [شئوناور و مایکلویکز، ۱۹۹۶].
- همان‌طور که می‌توان الگوریتم‌های تکاملی را به طرق گوناگون ترکیب نمود، روش‌ها اداره‌ی قید را نیز می‌توان ترکیب نمود. برای مثال، یک گروه از روش‌های اداره‌ی قید می‌توانند از نتایج یک تابع هزینه‌ی یکسان استفاده کرده و بهترین روش در هر نسل، بر نسل بعد غالب خواهد بود [مالیپدی[1] و سوگانثان، ۲۰۱۰]. این همانند برخی الگوریتم‌های چندهدفه از فصل ۲۰، که در آن‌ها از توابع هزینه‌ی متفاوت در مراحل مختلف فرآیند بهینه‌سازی استفاده می‌شود، می‌باشد.
- فرااکتشافات نیز می‌توانند الگوریتم‌های تکاملی و روش‌های اداره نمودن متعددی را در یک الگوریتم واحد ترکیب نمایند. به یاد آورید که یک اکتشاف در واقع خانواده‌ای از الگوریتم‌ها است (برای مثال، یک خانواده از انواع ACO یا یک خانواده از انواع DE). یک فرااکتشاف نیز خانواده‌ای از خانواده‌های الگوریتم‌هاست (برای مثال، خانواده‌ای شامل اکتشاف ACO و اکتشاف DE و سایر اکتشافات). فرااکتشافات را می‌توان برای هر نوع مسئله‌ی بهینه‌سازی به کار برد اما در اینجا به دلیل نویدی که برای مسائل مقید می‌دهند، آن‌ها را دوباره ذکر نمودیم [تینوکو[2] و کوئلو کوئلو، ۲۰۱۳].

مطالعات جامع بهینه‌سازی مقید را می‌توان در [ایبن، ۲۰۰۱]، [کوئلو کوئلو، ۲۰۰۲] و [کوئلو کوئلو و مزورا–مونتس، ۲۰۱۱] یافت. خوانندگانی که به تحقیق بیشتر علاقه دارند باید توجه داشته باشند که کوئلو کوئلو دارای مقالات بسیاری در زمینه‌های مرتبط با بهینه‌سازی تکاملی مقید بوده، که تعداد این مقالات تا تاریخ آگوست ۲۰۱۲ برابر ۱۰۳۶ مقاله می‌باشد [کوئلو کوئلو، ۲۰۱۲a].

[1] Mallipeddi
[2] Tinoco

مسائل

تمارین نوشتاری

۱۹–۱ بسیاری از محک‌ها با قید تساوی از $\epsilon \approx 0.0001$ در معادله‌ی (۱۹–۷) استفاده می‌نمایند. احتمال برآورده ساختن قید اسکالر $|x| \leq \epsilon$ با این مقدار از ϵ و تولید اتفاقی x چه قدر است؟

۱۹–۲ این مسئله نشان می‌دهد چگونه روش برتری نقاط امکان‌پذیر، که از معادله‌ی (۱۹–۱۱) استفاده می‌نماید، در صورتی که برای تمامی xها $f(x) \geq 0$ باشد موفق بوده و چگونه ناکام خواهد بود اگر این فرض برآورده نشود. از معادلات (۱۹–۱۰) و (۱۹–۱۱) جهت پیدا نمودن $\phi'(x)$ برای یک جمعیت دو عنصری با مشخصات زیر استفاده نمایید.

الف)

$$f(x_1) = 0, \quad \sum_i r_i G_i(x_1) = 1$$
$$f(x_2) = 10, \quad \sum_i r_i G_i(x_2) = 0$$

ب)

$$f(x_1) = -10, \quad \sum_i r_i G_i(x_1) = 1$$
$$f(x_2) = 0, \quad \sum_i r_i G_i(x_2) = 0$$

۱۹–۳ این مسئله نحوه‌ی عملکرد روش برتری نقاط امکان‌پذیر که از معادله‌ی (۱۹–۱۲) استفاده می‌نماید، را نشان می‌دهد. از معادلات (۱۹–۱۰) و (۱۹–۱۲) برای پیدا کردن $\phi'(x)$ برای جمعیت دو عنصره که در مسئله‌ی ۱۹–۲ نشان داده شده است، استفاده نمایید.

۱۹–٤ یک عبارت تحلیلی برای کوچکترین مقدار K در الگوریتم تکاملی گزینشی از معادله‌ی (۱۹–۱۳) بیابید که $\phi(\bar{x}) > \phi(x)$ را برای تمامی $\bar{x} \notin \mathcal{F}$ و $x \in \mathcal{F}$ تضمین نماید.

۱۹–۵ فرض کنید چهار ذره در یک جمعیت الگوریتم تکاملی با مقادیر هزینه و میزان نقض قید زیر در اختیار دارید:

$$\begin{array}{lll} f(x_1) = 3, & G_1(x_1) = 0, & G_2(x_1) = 0 \\ f(x_2) = 2, & G_1(x_2) = 1, & G_2(x_2) = 0 \\ f(x_3) = 1, & G_1(x_3) = 1, & G_2(x_3) = 1 \\ f(x_4) = 4, & G_1(x_4) = 0, & G_2(x_4) = 0 \end{array}$$

از فرمول‌بندی برازندگی خود–انطباق از بخش ۱۹–۲–۸ جهت پیدا کردن مقادیر هزینه‌ی مجازات شده برای این ذرات استفاده نمایید. جواب خود را توضیح دهید.

۱۹–٦ فرض کنید چهار ذره در یک جمعیت الگوریتم تکاملی با مقادیر هزینه و میزان نقض قید همانند مسئله‌ی ۱۹–۵ در اختیار دارید. از روش تابع مجازات خود–انطباق از بخش ۱۹–۲–۹ جهت پیدا نمودن مقادیر هزینه‌ی مجازات شده‌ی این ذرات استفاده نمایید. جواب خود را توضیح دهید.

۱۹–۷ این مسئله با الگوریتم اداره نمودن تبعیضی و انطباقی قید از بخش ۱۹–۲–۱۰ سر و کار دارد.

الف) توضیح دهید که چگونه الگوریتم به‌روزرسانی r_i از معادله‌ی (۱۹–۳٤) به نسبت مطلوب ذرات امکان‌پذیر به ذرات امکان‌ناپذیر دست می‌یابد.

ب) نحوه‌ی تأثیر افزایش γ در معادله‌ی (۱۹–۳٤) را توضیح دهید.

۱۹–۸ برخی الگوریتم‌های تکاملی مقید از جمله الگوریتم رتبه‌بندی اتفاقی از بخش ۱۹–۲–۱۲ و رویکرد مجازات نیچه از بخش ۱۹–۲–۱۳ شامل مقایسه‌ی ذرات، جهت تعیین این که کدام ذره بیشترین میزان نقض قید را دارد، می‌شوند. سه راه برای اندازه‌گیری میزان نقض قید پیشنهاد نمایید.

۱۹–۹ مسئله‌ی فروشنده‌ی دوره‌گرد (TSP) یک مسئله‌ی مقید است: یک راه‌حل نامزد باید از هر شهر دقیقاً یکبار رد شود تا یک مسیر مجاز به شمار آید. عملگرهای برش برای نمایش مسیر از ذرات TSP در بخش ۱۸–۳–۱ بررسی شده‌اند. کدام یک از این عملگرها قید TSP را رعایت کرده و کدام یک رعایت نمی‌کنند؟

۱۹–۱۰ فرض کنید چهار ذره در یک جمعیت الگوریتم تکاملی با مقادیر هزینه و میزان نقض قید همانند مسئله‌ی ۱۹–۵ در اختیار دارید. فرض کنید که از مقایسه‌ی سطح ϵ از معادله‌ی (۱۹–۵۱) همراه با روش جمع قید استفاده نماییم.

الف) برای چه مقادیری از ϵ، رتبه‌ی x_2 بهتر از x_1 خواهد بود؟

ب) برای چه مقادیری از ϵ، رتبه‌ی x_3 بهتر از x_1 خواهد بود؟

ج) برای چه مقادیری از ϵ، رتبه‌ی x_2 بهتر از x_3 خواهد بود؟

تمارین کامپیوتری

۱۹–۱۲ شکل ۱۹–۱ را با $\alpha = 0.5$ بازتولید نمایید. چه تفاوتی میان شکل خود و شکل ۱۹–۱ می‌بینید؟

۱۹–۱۳ فرض کنید یک فضای جستجوی دایروی با شعاع ۱ واحد دارید که مرکز آن در مبدأ واقع شده است. فرض کنید ذرات باید حتماً در درون یک دایره با شعاع $\rho_c = 0.1$ و با مرکز مبدا قرار بگیرند.

الف) از الگوریتم ژنوکوپ III برای تولید یک x_r اتفاقی امکان‌پذیر، یک x_s اتفاقی امکان‌ناپذیر و یک پارامتر اتفاقی $a \in [0,1]$ جهت تولید کی ذره‌ی اصلاح‌شده‌ی بالقوه مانند x'_s، استفاده نمایید.این آزمایش را چندین بار تکرار نمایید تا احتمال اینکه x'_s امکان‌پذیر باشد به دست آید. این آزمایش را برای $\rho_c = 0.5$ نیز تکرار نمایید.

ب) بخش (الف) را برای یک دامنه‌ی کروی نیز تکرار نمایید.

۱۹-۱٤ بخش ۱۹-٦، نه روش اداره‌ی محدودیت را همراه با BBO مقایسه نمود. برخی از روش‌های اداره نمودن محدودیت، که در این فصل در موردشان بحث شده است، را بر روی یک یا تعداد بیشتری از مسائل بهینه‌سازی مقید و همراه با الگوریتم‌های تکاملی دیگری به غیر از BBO، اجرا نمایید.

فصل بیستم

بهینه‌سازی چندهدفه

هدف‌های چندگانه و اغلب متناقض، در بیشتر مسائل دنیای واقعی وجود دارند.

اکارت زیتزلر[1] [زیتزلر و همکاران، ۲۰۰٤]

تمام مسائل بهینه‌سازی دنیای واقعی چندهدفه هستند. این فصل به بحث در مورد نحوه‌ی اصلاح الگوریتم‌های تکاملی برای استفاده در مسائل بهینه‌سازی چندهدفه (MOP[2]) می‌پردازد. همان‌طور که در نقل قول بالا خواندید، مسائل بهینه‌سازی دنیای واقعی غالباً (و یا شاید همیشه)، دارای چند هدف هستند و این اهداف معمولاً با هم تناقض دارند. برای مثال:

- هنگام طراحی یک پل ممکن است بخواهیم هزینه‌ی آن را مینیمم و قدرتش را ماکزیمم نماییم. جهت کم کردن هزینه باید از استایروفوم[3] استفاده شود که در این صورت پل ضعیف خواهد بود. اگر بخواهیم پل بیشترین قدرت را داشته باشد باید از تیتانیوم استفاده نماییم که در این صورت پل بسیار گران خواهد بود. در این صورت، بهترین تعادل میان قدرت و هزینه کدام است؟
- هنگام خرید یک خودرو، ممکن است بخواهیم هزینه را مینیمم و راحتی خودرو را ماکزیمم نماییم. یک خودرو با بیشترین میزان راحتی بسیار گران و یک خودرو با کمترین هزینه، بسیار ناراحت خواهد بود.
- هنگام طراحی یک محصول مصرفی، می‌خواهیم سود و سهم بازار را ماکزیمم نماییم. یک محصول با بیشترین میزان سود نفوذ زیادی را در بازار به ارمغان نخواهد آورد تا بتوان محصولات آتی شرکت را نیز در بازار جای داد. اما از سوی دیگر نیز، در صورتی که بخواهیم بیشترین سهم بازار را داشته باشیم، سود کافی نخواهیم داشت.
- هنگام طراحی یک سیستم کنترل، ممکن است بخواهیم زمان صعود و فرارفت (فراجهش) را مینیمم نماییم. یک کنترل‌کننده که دارای کمترین زمان صعود است دارای مقدار زیادی فراجهش بوده و یک کنترل‌کننده‌ی زیرمیرا (فراجهش صفر) نیز دارای زمان صعود کندی خواهد بود.
- هنگام طراحی یک سیستم کنترل، ممکن است بخواهیم حساسیت ورودی را ماکزیمم و حساسیت آشوب را مینیمم نماییم. یک کنترل‌کننده با بیشترین حساسیت ورودی نسبت به نویز بسیار حساس بوده و یک کنترل‌کننده با کمترین حساسیت آشوب نیز دارای واکنش‌های ضعیفی نسبت به ورودی‌های کنترل خواهد بود.

---

[1] Eckart Zitzler
[2] Multi-Objective Optimization Problem
[3] Styrofoam

بهینه‌سازی چندهدفه با نام‌های بهینه‌سازی چند معیاری، بهینه‌سازی چند عملکردی و بهینه‌سازی برداری نیز شناخته می‌شود. در این فصل فرض بر آن است که متغیر مستقل x، n بعدی بوده و MOP نیز یک مسئله‌ی مینیمم‌سازی است. یک MOP را می‌توان به‌صورت زیر نوشت:

$$\min_x f(x) = \min_x [f_1(x), f_2(x), \dots, f_k(x)] \qquad (۱-۲۰)$$

در معادله‌ی بالا به دنبال مینیمم‌سازی یک بردار $f(x)$ از توابع هستیم. صدالبته که نمی‌توان یک بردار را به معنای واقعی کلمه "مینیمم" کرد. با این حال، هدف ما در یک MOP مینیمم‌سازی همزمان همه‌ی k تابع $f_i(x)$ می‌باشد. می‌بینید که برای MOPها به تعریفی دوباره از بهینگی نیاز داریم.

بهینه‌سازی چندهدفه سالیان بسیاری توسط جامعه‌ی تحقیقات در عملیات مورد مطالعه واقع شده است [اهرگوت[1]، ۲۰۰۵]. به نظر می‌رسد که [روزنبرگ[2]، ۱۹۶۷] اولین جایی باشد که در آن استفاده از الگوریتم‌های تکاملی برای MOPها پیشنهاد شده باشد. همچنین [ایتو[3] و همکاران، ۱۹۸۳] نیز اولین پیاده‌سازی در این موضوع را ارائه داده و [شافر[4]، ۱۹۸۵] نیز اولین مقاله‌ی شاخته شده در این مورد می‌باشد.

MOPها غالباً شامل محدودیت می‌شوند، اما در این فصل با MOPهای مقید سروکار نخواهیم داشت. قیود را می‌توان همان‌گونه که در الگوریتم‌های تکاملی تک هدفه اعمال کردیم (فصل ۱۹ را ببینید)، در الگوریتم‌های تکاملی چندهدفه (MOEA[5]) نیز اعمال نماییم. برخی محققین تکنیک‌های اداره‌ی قیدی طرح نموده‌اند که مخصوص MOPها می‌باشند، هر چند در این فصل به این تکنیک‌ها نخواهیم پرداخت.

مروری بر فصل

بخش ۲۰-۱ به بحث در مورد بهینگی پرتو[6]، که تعمیمی از بهینگی MOPهایی است که به فرم معادله‌ی (۲۰-۱) می‌باشند، می‌پردازد. از آنجا که یک MOP دارای چند هدف است، راه‌های زیادی برای اندازه‌گیری عملکرد یک MOEA وجود دارد و ما برخی از این روش‌ها را در بخش ۲۰-۲ مورد بحث قرار خواهیم داد و این بحث را با ارائه‌ی برخی از MOEAهای محبوب دنبال خواهیم نمود. بخش ۲۰-۳ به بحث در مورد MOEAهایی می‌پردازد که صریحا از مفهوم بهینگی پرتو استفاده نمی‌کنند. بخش ۲۰-٤ نیز MOEAهایی را مورد بررسی قرار می‌دهد که صریحا از این مفهوم استفاده می‌نمایند. بخش ۲۰-۵ به ما نشان خواهد داد که

[1] Ehrgott
[2] Rosenberg
[3] Ito
[4] Schaffer
[5] Multi-Objective Evolutionary Algorithms
[6] Pareto Optimality

چگونه می‌توان بهینه‌سازی زیست‌جغرافی–محور (BBO، فصل ١٤ را ببینید) را با برخی رویکردهای MOEA ترکیب نمود. این فصل همچنین یک مطالعه‌ی مقایسه‌ای از محک‌های چندهدفه ارائه خواهد داد. بخش پایانی این فصل مرجعی برای منابع اضافی بوده و چند موضوع مهم در مورد تحقیقات کنونی و آتی MOEA را مطرح خواهد نمود.

۲۰–۱ بهینگی پرتو

این بخش برخی مفاهیم و مثال‌های اساسی که به MOPها مرتبط می‌باشند را به طرح می‌کشد. ابتدا برخی تعاریفی را که معمولاً در بهینه‌سازی چندهدفه به کار می‌بریم، معرفی می‌نماییم.

۱. *غلبگی*: یک نقطه مانند x^* بر x غالب خواهد بود اگر دو شرط زیر برقرار باشد:

- برای تمامی $i \in [1,k]$، $f_i(x^*) \leq f_i(x)$
- وجود داشته باشد $j \in [1,k]$ که $f_j(x^*) < f_j(x)$

دو شرط بالا به این معنی هستند که x^* برای تمامی مقادیر تابع هدف حداقل به خوبی x بوده و حداقل برای یک مقدار تابع هزینه، از x بهتر است. ما از نمادگذاری زیر برای نشان دادن غلبه‌ی x^* بر x استفاده می‌نماییم:

$$x^* \succ x \qquad (۲–۲۰)$$

این نمادگذاری ممکن است کمی گیج‌کننده باشد چرا که علامت $\succ$ شبیه علامت "بزرگتر یا مساوی" می‌باشد، اما از آنجا که در این فصل بیشتر با مسائل مینیمم‌سازی سروکار داریم، علامت $\succ$ به معنی آن است که مقادیر تابع x^* کمتر یا مساوی مقادیر تابع x می‌باشند. با این حال، این علامت استانداردی است که در ادبیات مربوط به MOP به کار می‌رود و ما نیز از همین نمادگذاری استفاده خواهیم نمود. عبارت "x^* بر x برتری دارد" با عبارت "x^* بر x غالب است" یکی است.

۲. *غلبگی ضعیف*: یک نقطه مانند x^* بر x غلبگی ضعیف خواهد داشت اگر برای تمامی $i \in [1,k]$، $f_i(x^*) \leq f_i(x)$ باشد. این بدین معنی است که برای همه‌ی مقادیر تابع هدف، x^* به خوبی x می‌باشد. توجه داشته باشید که اگر x^* بر x غالب باشد، آنگاه حتما غلبگی ضعیف نیز خواهد داشت. همچنین توجه داشته باشید که اگر برای تمامی $i \in [1,k]$ $f_i(x^*) = f_i(x)$ باشد، آنگاه x^* و x دارای غلبگی ضعیف بر یکدیگر خواهند بود. ما از نمادگذاری زیر برای نشان دادن غلبگی ضعیف x^* بر x استفاده می‌نماییم:

$$x^* \succcurlyeq x \qquad (۳-۲۰)$$

۳. *نامغلوب*: گوییم یک نقطه مانند x^* نامغلوب است اگر هیچ نقطه‌ای مانند x وجود نداشته باشد که بر آن غالب باشد. ناسرخورده، قابل قبول و کارامد، کلمات هم معنی نامغلوب می‌باشند.

٤. *نقاط بهینه‌ی پرتو*: یک نقطه‌ی بهینه‌ی پرتو مانند x^*، که با نام نقطه‌ی پرتو نیز شناخته می‌شود، نقطه‌ای است که در فضای جستجو توسط هیچ x دیگر مغلوب نشده است، بدین معنی که

$$\begin{aligned} &x^* \text{ بهینه‌ی پرتو است اگر و فقط اگر} \\ &\nexists x: f_i(x) \le f_i(x^*) \text{ برای تمامی } i \in [1,k] \text{ و } f_j(x) \le f_j(x^*) \text{ برای برخی } j \in [1,k] \end{aligned} \qquad (٤-۲۰)$$

۵. *مجموعه‌ی بهینه‌ی پرتو*: مجموعه‌ی بهینه‌ی پرتو، که با نام مجموعه‌ی پرتو نیز شناخته شده و با P_s نشان داده می‌شود، مجموعه‌ای است از همه‌ی x^* که نامغلوب‌اند.

$$P_s = \{x^* : [\nexists x: f_i(x) \le f_i(x^*)\} \text{ برای تمامی } i \in [1,k] \text{ و } f_j(x) \le f_j(x^*) \text{ برای برخی } j \in [1,k] \qquad (۵-۲۰)$$

مجموعه‌ی پرتو، مجموعه‌ی کارامد یا مجموعه‌ی قابل قبول نیز نامیده می‌شود. هر چند این عبارت بیشتر به برآورده‌سازی قید اشاره دارد تا بهینگی پرتو.

٦. *مرز پرتو*[1]: مرز پرتو، که با نام‌های مجموعه‌ی نامغلوب نیز شناخته شده و با P_f نشان داده می‌شود، مجموعه‌ای از تمامی بردارهای تابع $f(x)$ متناظر با مجموعه‌ی پرتو می‌باشد.

$$P_f = \{f(x^* : x^* \in P_s\} \qquad (٦-۲۰)$$

گاهاً پیش آمده است که در مقالات از دو عبارت مجموعه‌ی پرتو و مرز پرتو به‌صورت اشتباه و به جای یکدیگر استفاده شده است. با این حال، فهرست بالا تعاریف فنی درستی را ارائه می‌دهند. توجه داشته باشید که این که گفته می‌شود x^* نامغلوب است، الزاما به معنی آن نیست که x^* بر تمامی xهایی که مساوی x^* نیستند غالب است، اما ممکن است که برای تمامی $i \in [1,k]$ ، $f_i(x^*) = f_i(x)$ باشد. در این صورت، هر دوی x^* و x نسبت به یکدیگر نامغلوب خواهند بود و هیچ یک دیگری را مغلوب نخواهد ساخت. همچنین ممکن است که برای مثال در یک مسئله‌ی دو هدفه، $f_1(x) < f_1(x^*)$ و $f_2(x^*) < f_2(x)$ باشد. در این مسئله نیز x^* و x نسبت به یکدیگر نامغلوب خواهند بود و هیچ یک دیگری را مغلوب نخواهد ساخت.

[1] Pareto Front

این ایده‌ی بهینگی پرتو برای MOPها غالباً به فرانسیس اجورث[1] که آن را در سال ۱۸۸۱ معرفی نمود [اجورث، ۱۸۸۱] و ویلفردو پرتو[2] که کار اجورث را در سال ۱۹۸٦ بسط داد [پرتو، ۱۸۹٦]، نسبت داده می‌شود. با این حال، ایده‌ی مصالحه و سبک و سنگین نمودن برای هر کسی که سعی کرده باشد تعادلی میان اهداف متناقض ایجاد نماید، ملموس است.

مثال ۲۰-۱

فرض کنید یک MOP که در آن متغیر مستقل x دوبعدی است $(n = 2)$ در اختیار داریم. همچنین فرض کنید که x می‌تواند شش مقدار گسسته $x^{(i)}$ به ازای $i \in [1,6]$ را بپذیرد. حال فرض کنید دو هدف $(k = 2)$ با مقادیر تابع زیر در اختیار داریم

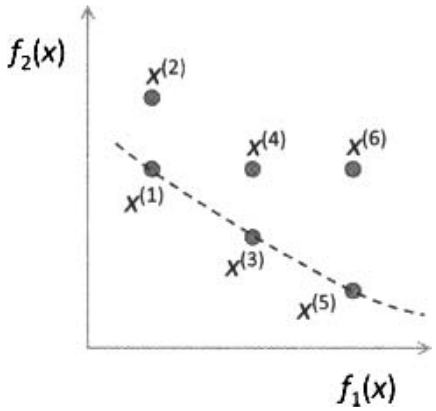
$$\begin{aligned} &f_1(x^{(1)}) = 1, \quad f_2(x^{(1)}) = 3 \\ &f_1(x^{(2)}) = 1, \quad f_2(x^{(2)}) = 4 \\ &f_1(x^{(3)}) = 2, \quad f_2(x^{(3)}) = 2 \\ &f_1(x^{(4)}) = 2, \quad f_2(x^{(4)}) = 3 \\ &f_1(x^{(5)}) = 3, \quad f_2(x^{(5)}) = 1 \\ &f_1(x^{(6)}) = 3, \quad f_2(x^{(6)}) = 3 \end{aligned} \qquad (۲۰-۷)$$

اگر $x = x^{(1)}$ یا $x = x^{(2)}$ آنگاه $f_1(x)$ مینیمم خواهد شد. اگر $x = x^{(5)}$ آنگاه $f_2(x)$ مینیمم خواهد شد. حتی یک مقدار از x وجود ندارد که هم $f_1(x)$ و هم $f_2(x)$ را مینیمم نماید. بهینه‌ترین مقدار x مقداری است که بهترین تعادل را میان چند هدف فراهم آورد. بدین ترتیب، *بهترین* به قضاوت خاص مسئله‌ی ما بستگی خواهد داشت. یک نقطه‌ی جالب دیگر در معادله‌ی (۲۰-۷)، $x^{(3)}$ می‌باشد چرا که هر نقطه‌ی $x \neq x^{(3)}$ یا $f_1(x) > f_1(x^{(3)})$ و یا $f_2(x) > f_2(x^{(3)})$ را نتیجه خواهد داد.

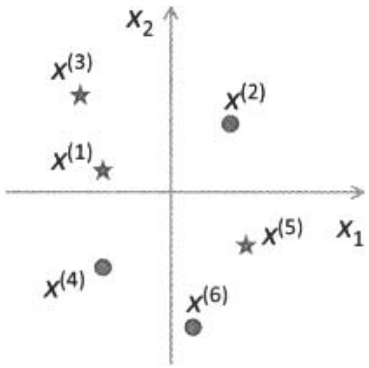
یک راه خوب برای به تصویر کشیدن این مسئله شکل ۲۰-۱ است که نموداری از $f_1(x)$ در مقابل $f_2(x)$ را نشان می‌دهد. این شکل به خوبی نشان می‌دهد که برای $x \in \{x^{(1)}, x^{(3)}, x^{(5)}\}$، x دیگری وجود ندارد که به‌صورت همزمان مقادیر تمامی توابع هدف را کاهش دهد. بدین ترتیب، $x^{(1)}$، $x^{(3)}$ و $x^{(5)}$ تعادل خوبی را برای این MOP به دست خواهند داد و به همین دلیل در مجموعه‌ی پرتو قرار خواهند گرفت. اگر تمامی نقاط بهینه در صفحه‌ی f_1/f_2 از شکل ۲۰-۱ را به یکدیگر متصل نماییم، منحنی‌ای به دست خواهد آمد که حد پایینی را برای تمام نقاط دیگر در این صفحه تشکیل خواهد داد.

[1] Francis Edgeworth
[2] Vilferdo Pareto

یک راه دیگر برای به تصویر کشیدن این مسئله، ترسیم نموداری از فضای جستجو که در آن مجموعه‌ی پرتو با نمادگذاری‌های خاصی نشان داده شده است. شکل ۲۰-۲ یکی از روش‌های ممکن برای نشان دادن فضای جستجو در این مال را نشان می‌دهد. در این شکل نقاط پرتو با ستاره نمایش داده شده‌اند. این شکل ناحیه‌ای از فضای جستجو را نشان می‌دهد که متناظر مرز پرتو از شکل ۲۰-۱ می‌باشد.



شکل ۲۰-۱ مثال ۲۰-۱: $\{x^{(1)}, x^{(3)}, x^{(5)}\}$ از مجموعه‌ی پرتو برای این مسئله‌ی مینیمم‌سازی چندهدفه. بردارهای تابع که متناظر مجموعه‌ی پرتو می‌باشند، مرز پرتو را شکل می‌دهند.



شکل ۲۰-۲ مثال ۲۰-۱: این شکل یک فضای جستجوی دو بعدی را برای مثال ۲۰-۱ نشان می‌دهد. فضای جستجو شامل شش بردار دو بعدی می‌شود. ستاره‌ها نشانگر مجموعه‌ی پرتو می‌باشند.

مثال ۲۰-۲

MOP زیر را در نظر بگیرید

$$\min f(x) = \min[f_1(x), f_2(x)] = \min[x_1^2 + x_2^2, (x_1 - 2)^2 + (x_2 - 2)^2 \qquad (۸-۲۰)$$

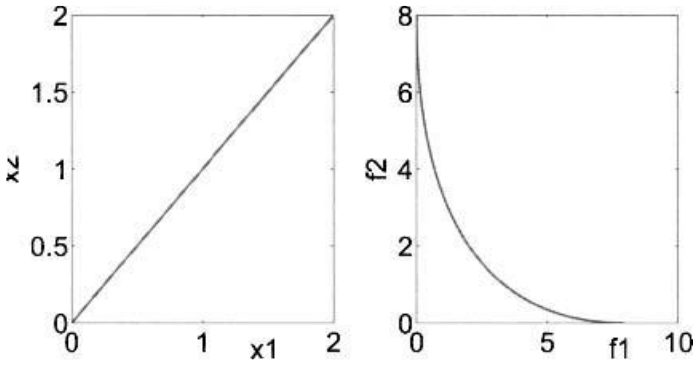
که در آن $x_1 \in [0,2]$ و $x_2 \in [0,2]$ می‌باشد. این یک MOP دو بعدی $(n = 2)$ می‌باشد چرا که هر x در فضای جستجو دارای دو عنصر است. همچنین، این MOP دارای دو هدف $(k = 2)$ است. نقطه‌ی $x^{(1)} = (0,0)$ تابع $f_1(x)$ را مینیمم می‌نماید. بنابراین این نقطه یکی از نقاط پرتو است. نقطه‌ی $x^{(2)} = (2,2)$ نیز $f_2(x)$ را مینیمم می‌نماید و به همین دلیل این نقطه نیز یکی از نقاط پرتو محسوب می‌شود. اگر از جستجوی brute-force برای پیدا نمودن نقاط پرتو استفاده نماییم خواهیم دید که مجموعه‌ی پرتو به‌صورت زیر خواهد بود

$$P_s = \left\{x: x_1 = x_2, \text{ که در آن } x_1 \in [0,2]\right\} \qquad (۹-۲۰)$$

بنابراین، مجموعه‌ی پرتو یک خط مستقیم را در فضای جستجو ایجاد می‌نماید. می‌توان مرز پرتو را با جایگزینی نقاط پرتو در معادله‌ی (۲۰-۸) به دست آورد:

$$P_f = \left\{(f_1, f_2): f_1 = 2x_1^2, f_2 = 2(x_1 - 2)^2, \quad \text{که در آن } x_1 \in [0,2]\right\} \qquad (۱۰-۲۰)$$

شکل ۲۰-۳ مجموعه‌ی پرتو و مرز پرتو را برای این مثال نشان می‌دهد. هر نقطه‌ای غیر از نقاط پرتو متناظر بردار تابعی است که در سمت بالا و راست مرز پرتو واقع می‌شود.



شکل ۲۰-۳ مثال ۲۰-۲: شکل سمت چپ مجموعه‌ی پرتو را نشان می‌دهد. شکل سمت راست نیز مرز پرتوی متناظر مجموعه‌ی پرتو را نشان می‌دهد. هر نقطه‌ای که در مجموعه‌ی پرتو قرار دارد تعادل معقولی را برای مسئله‌ی بهینه‌سازی چندهدفه فراهم می‌آورد.

غلبگی ϵ[1]

یکی از محدودیت‌های غلبگی پرتو آن است که دارای طبیعتی سیاه و سفید است. برای مثال، سه مجموعه‌ی مقادیر تابع هزینه‌ی زیر را در نظر بگیرید:

[1] ϵ Dominance

$$\begin{aligned} f_1(x) &= 200, \quad f_2(x) = 300 \\ f_1(y) &= 201, \quad f_2(y) = 301 \\ f_1(z) &= 500, \quad f_2\left(x^{(3)}\right) = 600 \end{aligned} \qquad (۱۱-۲۰)$$

x هم بر y و هم بر z غالب است. اما مفهوم غلبگی پرتو هیچ‌گونه تمایزی برای میزان غلبگی قائل نیست و به همین دلیل قادر به تشخیص دو راه‌حل نامزد که در فضای تابع هدف بسیار به هم نزدیک می‌باشند، نیست. در معادله‌ی (۲۰-۱۱)، x بر y غالب است اما x و y بسیار شبیه هم هستند و می‌توان گفت که تقریباً نسبت به هم نامغلوب‌اند. در حقیقت، تقریباً می‌توان گفت که y بر x غالب است. این موضوع به نوعی مفهوم غلبگی ϵ می‌باشد.

۱. غلبگی ϵ افزایشی[1]: گوییم نقطه‌ای مانند x^* دارای غلبه‌ی افزایشی ϵ بر x است هرگاه برای تمامی $i \in [1,k]$، وجود داشته باشد یک $\epsilon \geq 0$ به‌گونه‌ای که $f_i(x^*) \leq f_i(x) + \epsilon$. بدین معنی که x^* به غلبه بر x نزدیک است و مقدار این نزدیکی به‌صورت افزایشی و با پارامتر ϵ اندازه‌گیری می‌شود.

۲. غلبگی ϵ ضربی[2]: گوییم نقطه‌ای مانند x^* دارای غلبه‌ی ضربی ϵ بر x است هرگاه برای تمامی $i \in [1,k]$، وجود داشته باشد یک $\epsilon \geq 0$ به‌گونه‌ای که $f_i(x^*) \leq f_i(x)(1+\epsilon)$. بدین معنی که x^* به غلبه بر x نزدیک است و مقدار این نزدیکی به‌صورت ضربی و با پارامتر ϵ اندازه‌گیری می‌شود.

ما از نمادگذاری زیر برای نشان دادن غلبه‌ی ϵ نقطه‌ی x^* بر x استفاده می‌نماییم.

$$x^* \succ_\epsilon x \qquad (۱۲-۲۰)$$

نوع ϵ (افزایشی یا ضربی) از متن روشن خواهد بود. توجه داشته باشید که اگر $\epsilon = 0$ باشد، غلبگی ϵ با غلبگی ضعیف معادل خواهد بود اگر:

$$(x^* \succcurlyeq x) \Leftrightarrow \epsilon = 0 \text{ برای } (x^* \succ_\epsilon x) \qquad (۱۳-۲۰)$$

همچنین توجه داشته باشید که غلبگی ϵ برای مقادیر $\epsilon > 0$ حتی از غلبگی ضعیف هم ضعیف‌تر است. این بدین معنی است که اگر x^* بر x غلبگی ضعیف داشته باشد، آنگاه برای همه‌ی مقادیر $\epsilon \geq 0$ بر x غلبگی ϵ خواهد داشت. برعکس، اگر x^* به ازای یک مقدار $\epsilon > 0$ بر x غلبگی ϵ داشته باشد، آنگاه ممکن است بر x غلبگی ضعیف نیز داشته باشد

[1] Additive ϵ Dominance
[2] Multiplicative ϵ Dominance

$$(x^* \succcurlyeq x) \Longrightarrow \epsilon > 0 \text{ برای } (x^* \succ_\epsilon x) \qquad (۱٤-۲۰)$$

رابطه‌ی غلبگی ϵ بین دو ذره به مقدار ϵ استفاده شده در تعریف ما بستگی دارد. در معادله‌ی (۲۰-۱۱)، برای تمامی $\epsilon \geq 0$، $x \succ_\epsilon y$ می‌باشد. همچنین، $y \succ_\epsilon x$ از نوع افزایشی برای $\epsilon \geq 1$ و از نوع ضربی برای $\epsilon \geq 0.005$، برقرار می‌باشد.

۲۰-۲ اهداف بهینه‌سازی چندهدفه

هدف یک الگوریتم بهینه‌سازی تک هدفه معمولاً سرراست است: کمترین مقدار تابع هزینه و بردار تصمیم متناظر با آن را پیدا کن. با این حال، حتی در یک بهینه‌سازی تک هدفه نیز ممکن است به چند عملکرد متفاوت از الگوریتم تکاملی علاقه‌مند باشیم. برای مثال، ممکن است نه تنها به پیدا کردن کمترین مقدار تابع هزینه، بلکه به پیدا کردن سریع یک راه‌حل "خوب" که الزاما بهترین راه‌حل نیست، علاقه‌مند باشیم. همچنین ممکن است به دنبال پیدا کردن تعداد زیادی راه‌حل خوب در نواحی مختلف فضای جستجو باشیم. بنابراین، حتی در مسائل به ظاهر سرراست بهینه‌سازی تک هدفه نیز ممکن است عملکردهای مختلفی مد نظرمان باشد. این پیچیدگی با بهینه‌سازی چندهدفه بیشتر هم می‌شود. برخی اهداف بالقوه‌ی یک MOEA به‌صورت زیر می‌باشند:

۱. ماکزیمم کردن تعداد ذراتی که در فاصله‌ای مشخص از مجموعه‌ی پرتوی واقعی می‌یابیم.
۲. مینیمم کردن میانگین فاصله‌ی میان مجموعه‌ی پرتوی تخمین زده شده توسط MOEA و مجموعه‌ی پرتوی واقعی.
۳. ماکزیمم کردن تنوع ذراتی که در مجموعه‌ی پرتوی تخمینی می‌یابیم.
٤. مینیمم کردن فاصله‌ی یک راه‌حل نامزد در فضای تابع هدف تا یک نقطه‌ی ایده‌آل.

اهداف ۱ و ۲ به دنبال یافتن "بهترین" تخمین از مجموعه‌ی پرتوی حقیقی می‌باشند. هدف سوم، یافتن یک مجموعه‌ی متنوع از راه‌حل‌ها است تا کاربر اطلاعات کافی برای تصمیم‌گیری صحیح از میان مصالحات موجود را داشته باشد. بر خلاف سایر اهداف، هدف ٤ به دنبال یافتن راه‌حلی است که تا جای ممکن به راه‌حل ایده‌آل تصمیم‌گیرنده نزدیک باشد؛ راه‌حلی که ممکن است وجود نداشته باشد. با این حال، هدف اکثر MOEAهای حاضر یافتن بهترین تخمین از مجموعه‌ی پرتو می‌باشد.

در اهداف ۱ و ۲ فرض بر آن است که ما از مجموعه‌ی حقیقی پرتو اطلاع داریم. بنابراین، این شروط هنگام آزمایش نمودن MOEA بر روی محک‌ها بسیار مفید خواهد بود، اما هنگام به کار بردن MOEA در مورد مسائل دنیای واقعی، این شروط بی‌فایده خواهند بود. اما اگر مجموعه‌ی پرتوی واقعی P_s را بدانیم و

MOEA تخمینی از مجموعه‌ی پرتو $\hat{P}_s$ را به دست بدهد، فاصله‌ی میانگین بین آن‌ها را می‌توان به‌صورت زیر محاسبه نمود:

$$M_1(P_s, \hat{P}_s) = \frac{1}{|\hat{P}_s|} \sum_{x \in \hat{P}_s} \min_{x^* \in P_s} ||x^* - x|| \qquad (۱۵-۲۰)$$

که در آن $||.||$ یک متریک فاصله‌ی تعریف شده توسط کاربر است.

هدف ۳ از فهرست بالا را می‌توان به چند طریق اندازه گرفت. راه اول آن است که فاصله‌ی هر ذره تا نزدیکترین همسایه‌اش در مجموعه‌ی تخمینی پرتو را اندازه بگیریم. راه دوم آن است که فاصله‌ی میان دو ذره با بیشترین فاصله از یکدیگر در مجموعه‌ی تخمینی پرتو، را اندازه بگیریم. یک راه دیگر نیز آن است که میانگین تعداد ذراتی، که دارای فاصله‌ای بیشتر از یک حد مشخص از تک تک عناصر مجموعه‌ی تخمینی پرتو می‌باشند، را محاسبه نماییم [زیتزلر و همکاران، ۲۰۰۰]:

$$M_1(\hat{P}_s) = \frac{1}{|\hat{P}_s|} \sum_{x \in \hat{P}_s} |x' \in \hat{P}_s : ||x' - x|| > \sigma| \qquad (۱۶-۲۰)$$

که در آن σ یک حد فاصله است که توسط کاربر تعیین می‌شود. در کل، M_2 با افزایش تعداد عناصر در $\hat{P}_s$، افزایش یافته و همچنین تنوع عناصر در $\hat{P}_s$ نیز افزایش می‌یابد. [خار[1] و همکاران، ۲۰۰۳] چندین متریک تنوع دیگر برای MOPها را به بحث می‌گذارد.

هدف چهارم ار فهرست بالا با نام‌های بهینه‌سازی بردار هدف[2] [وینک[3] و همکاران، ۱۹۹۲]، دستیابی به هدف[4] [ویلسون و مکلئود[5]، ۱۹۹۳] و برنامه‌نویسی هدف[6] شناخته می‌شود. در این هدف فرض بر آن است که کاربر یک نقطه‌ی ایده‌آل در فضای تابع هدف را در ذهن دارد. این هدف همچنین به تعریفی از "فاصله" نیاز دارد. ما معمولاً از فاصله‌ی اقلیدسی D_2، که گاهاً با نام فاصله‌ی نُرم ۲ نیز شناخته می‌شود، میان بردار تابع هدف f و نقطه‌ی ایده‌آل f^* استفاده می‌نماییم. فاصله‌ی میان f و f^* به‌صورت زیر تعریف می‌گردد:

$$D_2^2\big(f^*(x), f(x)\big) = \big||f^*(x) - f(x)|\big|_2^2 = \sum_{i=1}^{k} \big(f_i^*(x) - f_i(x)\big)^2 \qquad (۱۷-۲۰)$$

[1] Khare
[2] Target Vector Optimization
[3] Wienke
[4] Goal Attainment
[5] Macleod
[6] Goal Programming

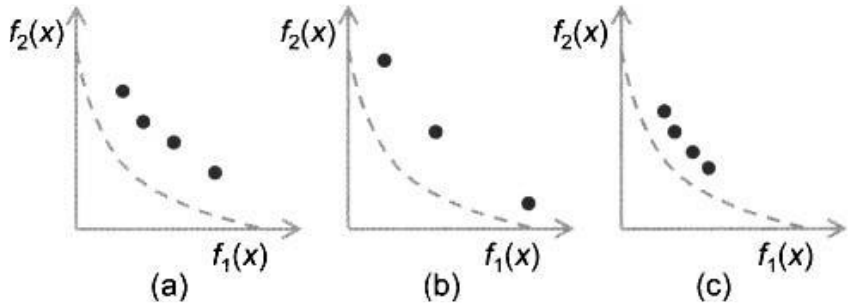
با این حال، می‌توان از تعریف دیگری از فاصله مانند نُرم ۲ وزن‌دهی شده و یا نُرم ۱ یا نرم بینهایت، استفاده نمود.

مثال ۲۰-۲ را به یاد آورید. کاربر ممکن است تصور نماید که دستیابی به هر دوی $f_1(x) = 0$ و $f_2(x) = 0$ ایده‌آل است. پس از به دست آوردن مرز پرتو از شکل ۲۰-۳، می‌توان دید که نزدیکترین نقطه به نقطه‌ی ایده‌آل به لحاظ فاصله‌ی اقلیدسی، $x_1 = x_2 = 1$ است و این نقاط، $f_1(x) = 2$ و $f_2(x) = 2$ را به دست می‌دهند. از سوی دیگر، اگر راه‌حل ایده‌آل کاربر $f_1(x) = 2$ و $f_2(x) = 0$ باشد، آنگاه نزدیکترین نقطه به نقطه‌ی ایده‌آل، $x_1 = x_2 = 1.2$ خواهد بود که در این صورت مقادیر $f_1(x) = 2.87$ و $f_2(x) = 1.29$ به دست خواهند آمد. استفاده از هدف نوع چهارم از فهرست بالا جهت اندازه‌گیری عملکرد MOEA، ترجیح کاربر را در راه‌حل نهایی MOP دخیل می‌نماید.

توجه داشته باشید که می‌توان اهداف ۱-۳ را بر اساس مرز پرتو و یا مجموعه‌ی پرتو دنبال نمود. برای مثال، در هدف اول، به جای ماکزیمم نمودن ذراتی که در فاصله‌ی معینی از مجموعه‌ی پرتو قرار دارند، می‌توان تعداد ذراتی را که بردارهای توابعشان در فاصله‌ی خاصی از مرز حقیقی پرتو قرار دارند، ماکزیمم نمود. به طور خلاصه، می‌توان دید که معیارهای عملکردی بالقوه‌ی بسیاری برای یک MOEA وجود دارد. به بیان دیگر، بهینه ساختن عملکرد MOEA خود یک MOP است. این موضوع ارزیابی MOEAها را پیچیده می‌سازد.

مثال ۲۰-۳

شکل ۲۰-٤ عملکرد سه الگوریتم تکاملی مختلف بر روی یک MOP را نشان می‌دهد. شکل ۲۰-٤ (الف) راه‌حلی را نشان می‌دهد که به میزان معقولی متنوع بوده و به مرز پرتوی حقیقی نزدیک می‌باشد. شکل ۲۰-٤ (ب) راه‌حلی را نشان می‌دهد که از شکل ۲۰-٤ (الف) متنوع‌تر بوده اما شامل تنها سه راه‌حل می‌شود در حالی که شکل ۲۰-٤ (الف) دارای چهار راه‌حل است. شکل ۲۰-٤ (ج) راه‌حل‌هایی را نشان می‌دهد که نسبت به راه‌حل‌های شکل‌های ۲۰-٤ (الف) و ۲۰-٤ (ب) به مرز پرتوی حقیقی نزدیکتر بوده اما دارای میزان تنوع نامطلوبی می‌باشد. کدام یک از این سه راه بهترین است؟ جواب این سؤال به اولویت‌های تصمیم‌گیرنده بستگی دارد.



شکل ۲۰-٤ مثال ۲۰-۳: این شکل سه راه‌حل الگوریتم تکاملی بالقوه برای یک MOP دو هدفه را نشان می‌دهد. مرز حقیقی پرتو با خط‌چین نمایش داده شده است و نقاط دایروی تخمین‌هایی هستند که توسط هر الگوریتم تکاملی پیدا شده‌اند. کدام راه‌حل بهترین است؟ این موضوع به اولویت‌های تصمیم‌گیرنده نسبت به میزان تنوع و میزان نزدیکی به مرز پرتوی حقیقی بستگی دارد.

۲۰-۲-۱ فراتوده[1]

یک متریک دیگر که محققان اغلب از آن برای اندازه‌گیری کیفیت مرز پرتو استفاده می‌نمایند، فراتوده‌ی آن می‌باشد. فرض کنید یک MOEA، M نقطه‌ی تخمینی را در یک مرز پرتو مانند $\hat{P}f_f = \{f(x_i)\}$ برای $j \in [1, M]$ یافته است. $f(x_i)$ یک تابع k بعدی است. فراتوده را می‌توان به‌صورت زیر محاسبه نمود

$$S(\hat{P}_f) = \sum_{j=1}^{M} \prod_{i=1}^{k} f_i(x_j) \qquad (۱۸-۲۰)$$

اگر برای یک MOP دو MOEA با دو تخمین متفاوت از مرز پرتو در اختیار داشته باشیم، می‌توانیم از فراتوده برای اندازه‌گیری کیفیت دو تخمین و مقایسه‌ی آن‌ها با یکدیگر استفاده نماییم. برای یک مسئله‌ی مینیمم‌سازی، فراتوده‌ی کوچکتر به معنای تخمین مرز پرتوی بهتر می‌باشد.

مثال ۲۰-٤

فرض کنید دو MOEA در اختیار داریم. هر یک از این دو MOEA جهت تخمین مرز پرتوی یک مسئله‌ی بهینه‌سازی دو هدفه طراحی شده‌اند. شکل ۲۰-۵ این تخمین‌ها را نشان می‌دهد. شکل ۲۰-۵ (الف) دارای دو نقطه‌ی تخمینی از مرز پرتو می‌باشد

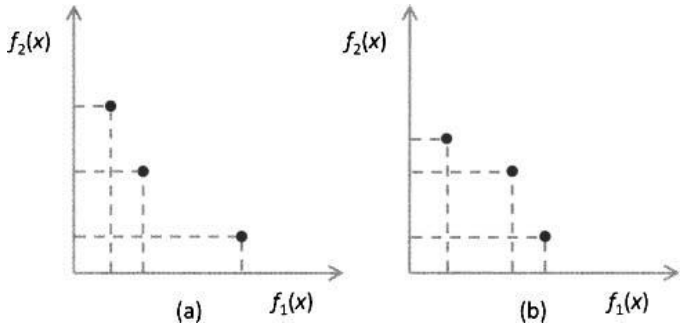
$$\hat{P}_f(1) = \{[f_1(x_j), f_2(x_j)]\} = \{[1,5], [2,3], [5,1]\} \qquad (۱۹-۲۰)$$

[1] Hypervolume

بدین ترتیب فراتوده‌ی این تخمین برابر $16 = 5 + 6 + 5$ خواهد بود. نقاط تخمینی مرز پرتوی شکل ۲۰–۵ (ب) به‌صورت زیر می‌باشند

$$\hat{P}_f(2) = \{[f_1(x_j), f_2(x_j)]\} = \{[1,4], [3,3], [4,1]\} \qquad (۲۰–۲۰)$$

بدین ترتیب فراتوده‌ی این تخمین برابر $17 = 4 + 9 + 4$ خواهد بود. بنابر اندازه‌ی فراتوده از معادله‌ی (۲۰–۱۸)، تخمین شکل ۲۰–۵ (الف) کمی بهتر از تخمین ۲۰–۵ (ب) می‌باشد.



شکل ۲۰–۵ مثال ۲۰–٤: این شکل دو تخمین متفاوت از یک MOP دو هدفه را نشان می‌دهد. از مقدار فراتوده برای سنجش میزان مطلوبیت تخمین‌ها استفاده می‌شود. تخمین سمت چپ دارای فراتوده‌ای برابر ١٦ و تخمین سمت راست دارای فراتوده‌ای برابر ۱۷ می‌باشد.

نمی‌توان از فراتوده‌ی کورکورانه به‌عنوان نشانگری از کیفیت مرز پرتو استفاده نمود. معادله‌ی (۲۰–۱۸) نشان می‌دهد که یک تخمین مرز پرتوی تهی ($M = 0$) کمترین مقدار ممکن S را نتیجه می‌دهد. بنابراین، برای دستیابی به یک اندازه‌ی دقیق‌تر باید از فراتوده‌ی نرمالیزه شده $S_n(\hat{p}_f) = S(\hat{p}_f)/M$ استفاده نمود. با این حال، حتی این مقدار هم ممکن است متریک مناسبی برای تخمین مرز پرتو نباشد. این موضوع را می‌توان با در نظر گرفتن احتمال اینکه یک تخمین مرز پرتوی خاص دارای مقدار فراتوده‌ی نرمالیزه شده‌ی $S_n(\hat{p}_f(1))$ باشد، مشاهده نمود. حال فرض کنید یک نقطه‌ی جدید را به $\hat{p}_f(1)$ اضافه کنیم تا $\hat{p}_f(2)$ به دست آید. بدین ترتیب، با این که تفاوت میان $\hat{p}_f(1)$ و $\hat{p}_f(2)$ تنها در این است که $\hat{p}_f(2)$ یک نقطه بیشتر از $\hat{p}_f(1)$ دارد، ممکن است $S_n(\hat{p}_f(2)) > S_n(\hat{p}_f(1))$ شود. واضح است که $\hat{p}_f(2)$ از $\hat{p}_f(1)$ بهتر است اما $S_n(\hat{p}_f(2))$ بزرگتر از $S_n(\hat{p}_f(1))$ خواهد شد که این منطقی نیست.

این موضوع باعث می‌شود تا فراتوده را نه نسبت به مبدأ فضای تابع هدف، بلکه نسبت به نقطه‌ی مرجعی که در خارج از مرز پرتو قرار دارد، محاسبه نماییم. فرض کنید که می‌خواهیم تعداد Q تخمین مرز پرتو

$\hat{P}_f(q)$ را برای $q \in [1, Q]$، مقایسه نماییم. ابتدا، بردار نقطه‌ی مرجع $r = [r_1, \dots, r_k]$ را با شرط زیر محاسبه می‌نماییم

$$r_i > \max_q [\max_{x \in \hat{P}_S(q)} f_i(x)] \qquad (۲۱-۲۰)$$

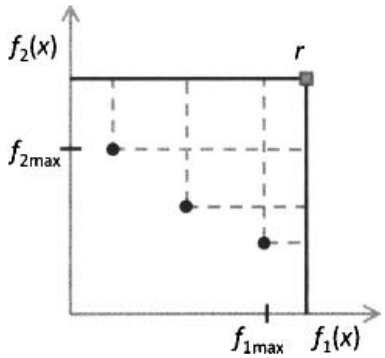
سپس مقدار فراتوده S' را نسبت به نقطه‌ی مرجع محاسبه می‌نماییم

$$S'\left(\hat{P}_f(q)\right) = \sum_{j=1}^{M(q)} \prod_{i=1}^{k} \left(r_i - f_i\left(x_j(q)\right)\right) \qquad (۲۲-۲۰)$$

در معادله‌ی بالا، $M(q)$ تعداد نقاط در qاُمین تخمین مرز پرتو را نشان می‌دهد. هر چه مقدار فراتوده‌ی نقطه‌ی مرجع بیشتر باشد، تخمین مرز پرتو برای مسئله‌ی مینیمم‌سازی بهتر خواهد بود. همچنین می‌توان از مقدار فراتوده‌ی نقطه‌ی مرجع نرمالیزه شده استفاده نمود:

$$S'_n\left(\hat{P}_f(q)\right) = S'\left(\hat{P}_f(q)\right)/M(q) \qquad (۲۳-۲۰)$$

اگر بخواهیم متریک مورد استفاده تعداد نقاط پرتو را نیز در نظر بگیرد، می‌توان از $S'(\hat{P}_f(q))$ استفاده نمود. شکل ۲۰-۶ فراتوده‌ی نقطه‌ی مرجع را در دو بعد نشان می‌دهد.



شکل ۲۰-۶ محاسبه‌ی فراتوده‌ی نقطه‌ی مرجع $S'(\hat{P}_f(q))$ از معادله‌ی (۲۰-۲۲). نقطه‌ی مرجع r یک نقطه‌ی دلخواه بوده که عنصر iاُمش از هر یک از نقاط در تخمین مرز پرتو بزرگتر می‌باشد. هر چه $S'(\hat{P}_f(q))$ بزرگتر باشد، تخمین مرز پرتو برای مسئله‌ی مینیمم‌سازی بهتر خواهد بود.

در بسیاری از مقالات MOEA، مسائل MOP به مسائل ماکزیمم‌سازی تبدیل شده که در این صورت مقادیر بزرگتر در معادله‌ی (۲۰-۱۸) مطلوب‌تر خواهند بود. همچنین تعداد نقاط بیشتر در $\hat{P}_f$ (M بزرگتر) نیز مطلوب‌تر خواهد بود. معادلات (۲۰-۱۸) و (۲۰-۲۲)، دو ایده‌ی اساسی محاسبه‌ی فراتوده را به دست می‌دهند. با این حال، در مقالات و کتب دیگر روش‌ها، تعاریف و الگوریتم‌های دیگری برای محاسبه‌ی

فراتوده معرفی شده‌اند [آئوگر[1] و همکاران، ۲۰۱۲]، [برینگمن[2] و فرایدریش[3]، ۲۰۱۰]، [زیتزلر و همکاران، ۲۰۰۳]. بیشتر مقالات از اتحاد M فرابسته[4] از معادلات (۲۰–۱۸) و (۲۰–۲۲) جهت محاسبه‌ی فراتوده استفاده می‌نمایند. برای مثال، اگر فراتوده‌ی اتحاد فرابسته‌ها از شکل ۲۰–۵ را محاسبه نماییم، هر دو تخمین $\hat{P}_f$ دارای فراتوده‌ای برابر ۱۱ خواهند شد. روش‌های دیگر محاسبه‌ی فراتوده ممکن است به نتایج مختلفی در مورد دو تخمین $\hat{P}_f$ منجر شود. با این حال، پیاده‌سازی معادلات (۲۰–۱۸) و (۲۰–۲۲) ساده‌تر از محاسبه‌ی فراتوده‌ی اتحاد فرابسته‌ها می‌باشد. واضح است که با همبستگی قوی میان جمع M فراتوده و فراتوده‌ی اتحاد M فرابسته وجود دارد، هرچند که این همبستگی کاملاً خطی نیست (مسئله‌ی ۲۰–۷ را ببینید).

۲۰–۲–۲ پوشش نسبی[5]

یک راه دیگر برای مقایسه‌ی تخمین‌های مرز پرتو، محاسبه‌ی میانگین تعداد ذراتی از یک تخمین است که تحت غلبه‌ی ضعیف حداقل یک ذره از تخمین دیگر می‌باشند [زیتزلر و ثیل، ۱۹۹۹]. فرض کنید دو تخمین $\hat{P}_f(1)$ و $\hat{P}_f(2)$ را در اختیار داریم. در این صورت، پوشش $\hat{P}_f(1)$ نسبت به $\hat{P}_f(2)$ عبارت است از میانگین تعداد ذراتی از $\hat{P}_f(2)$ که تحت غلبه‌ی ضعیف حداقل یکی از ذرات $\hat{P}_f(1)$ قرار دارند:

$$C\left(\hat{P}_f(1), \hat{P}_f(2)\right) = \frac{\left|a_2 \in \hat{P}_f(2) \text{ به طوری که } \exists \left[a_1 \in \hat{P}_f(1) \text{ به طوری که } a_1 \succcurlyeq a_2\right]\right|}{\left|\hat{P}_f(2)\right|} \qquad (۲۰–۲۴)$$

توجه داشته باشید که $C\left(\hat{P}_f(1), \hat{P}_f(2)\right) \in [0,1]$. اگر $C\left(\hat{P}_f(1), \hat{P}_f(2)\right) = 0$، آنگاه برای هر ذره مانند $a_2 \in \hat{P}_f(2)$، هیچ ذره‌ای در $\hat{P}_f(1)$ وجود نخواهد داشت که دارای غلبگی ضعیف بر a_2 باشد. اگر $C\left(\hat{P}_f(1), \hat{P}_f(2)\right) = 1$، آنگاه برای هر ذره مانند $a_2 \in \hat{P}_f(2)$، حداقل یک ذره در $\hat{P}_f(1)$ وجود خواهد داشت که بر a_2 غلبگی ضعیف داشته باشد. با این که نمی‌توان از معادله‌ی پوشش برای به دست آوردن اندازه‌ی مطلق میزان مطلوبیت تخمین مرز پرتو استفاده نمود، اما این معادله در مقایسه‌ی چند تخمین مفید خواهد بود.

[1] Auger
[2] Bringmann
[3] Friedrich
[4] Union of M hyperboxes
[5] Relative Coverage

۲۰–۳ الگوریتم‌های تکاملی غیر پرتو محور

این بخش چند MOEA را که صریحاً از ایده‌ی غلبگی پرتو استفاده نمی‌کنند مورد بحث قرار می‌دهد. بخش ۲۰–۳–۱ روش تجمع، بخش ۲۰–۳–۲ الگوریتم ژنتیک برداری سنجیده شده (VEGA[1])، بخش ۲۰–۳–۳ رویکردهای مرتب‌سازی وابسته به واژه‌نگاری[2]، بخش ۲۰–۳–٤ روش قید ϵ و بخش ۲۰–۳–۵ رویکردهای جنسیت–محور را مورد بحث قرار خواهد داد.

۲۰–۳–۱ روش‌های تجمعی

روش‌های تجمعی، بردار تابع هدف یک MOP را در یک تابع هدف اسکالر ترکیب می‌نمایند. برای مثال، می‌توان MOP با k هدف از معادله‌ی (۲۰–۱) را به‌صورت مسئله‌ی زیر در آورد

$$\min_x f(x) \Rightarrow \min_x \sum_{i=1}^{k} w_i f_i(x) \quad \text{که در آن } \sum_{i=1}^{k} w_i = 1 \qquad (۲۵–۲۰)$$

$\{w_i\}$ مجموعه‌ای از وزن‌های مثبت است که جمع عناصرش برابر با ۱ است. معادله‌ی (۲۰–۲۵) با نام رویکرد مجموع وزن شناخته می‌شود. با این حال می‌توان از روش‌های تجمعی دیگری نیز استفاده نمود. برای مثال، می‌توان اهداف را به‌صورت ضربی ترکیب نمود:

$$\min_x f(x) \Rightarrow \min_x \prod_{i=1}^{k} f_i(x) \qquad (۲۶–۲۰)$$

اگر از روش تجمعی ضربی استفاده نماییم، باید از مثبت بودن تمامی توابع هدف برای تمامی xها اطمینان حاصل نماییم. اما از هر روش تجمعی که استفاده نماییم، نکته‌ی اصلی در تبدیل نمودن MOP به یک مسئله‌ی بهینه‌سازی تک هدفه می‌باشد.

مثال ۲۰–۵

MOP مثال ۲۰–۲ را در نظر بگیرید. اکر از معادله‌ی (۲۰–۲۵) برای تبدیل نمودن این MOP به یک مسئله‌ی تک هدفه استفاده نماییم، عبارت زیر به دست خواهد آمد:

$$\begin{aligned}\min_x \{w_1 f_1(x) + w_2 f_2(x)\} \\ = \min_x \{w_1(x_1^2 + x_2^2) \\ + (1 - w_1)[(x_1 - 2)^2 + (x_2 - 2)^2]\}\end{aligned} \qquad (۲۷–۲۰)$$

[1] Vector Evaluated Genetic Algorithm
[2] Lexicographic Ordering

می‌توان این معادله را با گرفتن مشتق جزیی نسبت به x_1 و x_2 مینیمم نمود

$$\frac{\partial[w_1 f_1(x) + w_2 f_2(x)]}{\partial x_1} = 2x_1 + 4(w_1 - 1)$$
$$\frac{\partial[w_1 f_1(x) + w_2 f_2(x)]}{\partial x_2} = 2x_2 + 4(w_1 - 1) \qquad (۲۸-۲۰)$$

با قرار دادن این معادلات برابر صفر و حل آن‌ها، مجموعه‌ی پرتو به دست خواهد آمد

$$x_1^* = x_2^* = 2(1 - w_1) \qquad (۲۹-۲۰)$$

جایگذاری مجموعه‌ی پرتو در معادلات $f_1(x)$ و $f_2(x)$، مرز پرتو را به دست خواهد داد

$$f_1(x^*) = 8(1 - w_1)^2$$
$$f_2(x^*) = 2w_1^2 \qquad (۳۰-۲۰)$$

رسم نمودار معادلات (۲۰-۲۹) و (۲۰-۳۰) هنگامی که w_1 از ۰ تا ۱ تغییر می‌کند، نمودارهای شکل ۲۰-۳ را به دست خواهد داد.

مثال ۲۰-۵ نشان می‌دهد که روش تجمعی می‌تواند حداقل برای برخی MOPها مجموعه‌ی پرتو و مرز پرتو بیابد. در حقیقت، حل معادله‌ی (۲۰-۲۵) برای هر مجموعه‌ای از وزن‌ها به یک نقطه‌ی بهینه‌ی پرتو منجر خواهد شد. با این حال، اگر مرز پرتو مقعر باشد، آنگاه دیگر روش تجمعی قادر به یافتن مجموعه و مرز کامل پرتو نخواهد بود. مثال بعدی این موضوع را نشان می‌دهد.

مثال ۲۰-۶

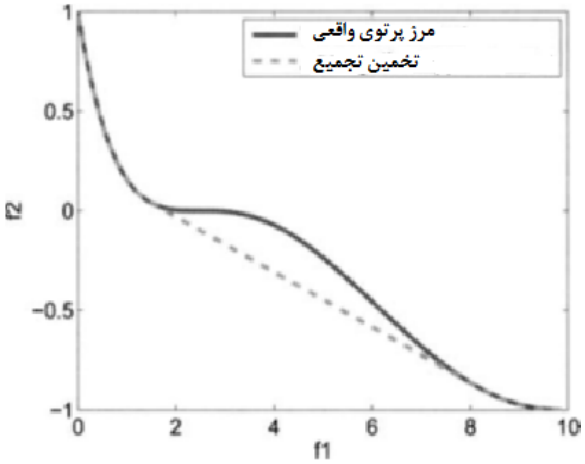
مسئله‌ی زیر را در نظر بگیرید

$$\min_x f(x) = \min_x [x^2, cos^3 x] \qquad (۳۱-۲۰)$$

در معادله‌ی بالا $x \in [0,4]$ می‌باشد. این یک MOP تک بعدی با دو هدف می‌باشد. می‌توان دو هدف را در یک هدف اسکالر ترکیب نمود

$$\min f(x) = \min[wx^2 + (1 - w)cos^3 x] \qquad (۳۲-۲۰)$$

که در آن $w \in [0,1]$ بوده و می‌توان معادلات (۲۰-۳۱) و (۲۰-۳۲) را با جستجوی جامع حل نمود. با این حال، راه‌حل‌های این دو معادله یکی نیستند. شکل ۲۰-۷ راه‌حل‌های دو مسئله را نشان می‌دهد. می‌توان دید که مرز حقیقی واقعی، مقعر است. روش تجمعی قسمت محدب مرز پرتو را به درستی به دست می‌دهد، اما قسمت مقعر را به درستی به دست نمی‌دهد.



شکل ۲۰-۷ مثال ۲۰-۶: روش تجمعی از معادله‌ی (۲۰-۳۲) قسمت محدب مرز پرتو را به درستی به دست می‌دهد، اما قسمت مقعر را به درستی به دست نمی‌دهد.

دستیابی به هدف برای مرزهای پرتوی مقعر

روش تجمعی تنها در یافتن مرز پرتوی مسئله‌ی موجود در مثال ۲۰-۶ دچار مشکل نمی‌شود. در حقیقت، یافتن مرز پرتوی مقعر با استفاده از روش تجمعی ناممکن است [فلمینگ[1] و همکاران، ۲۰۰۵]. با این حال، مرزهای پرتوی مقعر را می‌توان با استفاده از تعمیمی از روش دستیابی به هدف از بخش ۲۰-۲، پیدا نمود. دستیابی به هدف را گاهاً می‌توان به‌صورت حل مسئله‌ی زیر بیان نمود:

$$\min \alpha \quad \text{به طوری که} \quad f_i(x) \le f_i^* + w_i\alpha \quad \text{برای تمامی} \quad i \in [1,k] \quad \text{و برای برخی} \quad x \qquad (۲۰-۳۳)$$

در معادله‌ی بالا، f_i^* مقدار ایده‌آل iاُمین هدف بوده و $\{w_i\}$ نیز مجموعه‌ای از وزن‌های مثبت است که میزان اهمیت نسبی هر هدف را نشان می‌دهند. معادله‌ی (۲۰-۳۳) به راه‌حل‌هایی که بهتر از ایده‌آل کاربر باشند نیز اجازه‌ی وجود می‌دهد. اگر $\alpha > 0$، آنگاه نقطه‌ی ایده‌آل قابل دستیابی نخواهد بود اما راه‌حل معادله‌ی (۲۰-۳۳) بردار راه‌حلی را خواهد یافت که به اندازه‌ی ممکن به نقطه‌ی ایده‌آل نزدیک خواهد بود. اگر $\alpha < 0$، آنگاه می‌توان راه‌حلی را یافت که از نقطه‌ی ایده‌آل هر هدف بهتر باشد. می‌توان نشان داد [چن و لیو، ۱۹۹٤]، [کوئلو کوئلو، ۱۹۹۹] که حل معادله‌ی (۲۰-۳۳) برای تمامی ترکیبات وزن $\{w_i\}$ به‌گونه‌ای که $\sum_i w_i = 1$، مرز پرتوی MOP اصلی را به دست خواهد داد حتی اگر مرز پرتو مقعر باشد.

[1] Fleming

توجه داشته باشید که معادله‌ی (۲۰-۳۳) یک مسئله‌ی بهینه‌سازی تک هدفه با k قید و $(n+1)$ متغیر مستقل (اسکالر α و بردار تصمیم اصلی x) می‌باشد. بنابراین می‌توان از الگوریتم‌های بهینه‌سازی مقید (فصل ۱۹ را ببینید) برای حل معادله‌ی (۲۰-۳۳) استفاده نمود.

۲۰-۳-۲ الگوریتم ژنتیک برداری سنجیده شده (VEGA)

VEGA در واقع MOEA اصلی می‌باشد [شافر، ۱۹۸۵]. در VEGA عمل انتخاب بر روی جمعیت با استفاده از یک هدف در هر زمان صورت می‌پذیرد. این کار، مجموعه‌ای از زیرجمعیت‌ها را به دست می‌دهد. پس از آن، ذرات را از زیرجمعیت‌ها انتخاب می‌نماییم تا والدین برای نسل بعد به دست آیند. سپس والدین را با استفاده از روش‌های استاندارد بازترکیب الگوریتم تکاملی ترکیب کرده تا فرزندان به دست آیند. شکل ۲۰-۸ طرح کلی VEGA را نشان می‌دهد.

جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن: $P = \{x_j\}$ برای $j \in [1, N]$

$M \leftarrow \left\lceil \frac{N}{k} \right\rceil$

تا زمانی که شرایط توقف برآورده نشده است

هزینه‌ی $f_i(x_j)$ را برای هر ذره‌ی $x_j \in P$ محاسبه کن

برای $i = 1$ تا k

M ذره‌ی انتخاب شده از P به صورت احتمالاتی و با استفاده از $P_i \leftarrow f_i(.)$

i بعدی

N ذره‌ی انتخاب شده از $P \leftarrow \{P_1, \dots, P_k\}$

N فرزند تولید شده از بازترکیب ذرات موجود در $C \leftarrow P$

فرزندان موجود در C را به‌صورت احتمالاتی دچار جهش کن

$P \leftarrow C$

نسل بعد

عناصر غیرمغلوب $\hat{P}_s \leftarrow P$

شکل ۲۰-۸ طرح کلی الگوریتم ژنتیک برداری ارزیابی شده (VEGA) برای حل یک مسئله‌ی بهینه‌سازی با k هدف.

شکل ۲۰-۸ نشان می‌دهد که VEGA با جمعیتی N ذره‌ای، که معمولاً به‌صورت اتفاقی تولید می‌گردد، کار خود را آغاز می‌نماید. در هر نسل، مقادیر همه‌ی k تابع هزینه را برای تمامی N ذره محاسبه می‌نماییم.

سپس از یک روش انتخاب دلخواه (بخش ۸-۷ را ببینید)، برای انتخاب M ذره استفاده می‌نماییم. توجه داشته باشید که $M = \lceil N/k \rceil$ کوچکترین عدد صحیحی است که بزرگتر یا مساوی N/k می‌باشد. ما این انتخاب را به‌صورت احتمالاتی انجام می‌دهیم، بدین صورت که ابتدا از $f_1(.)$ برای تولید جمعیت P_1، از $f_2(.)$ برای تولید جمعیت P_2 و ... استفاده می‌نماییم. پس از آنکه P_i زیرجمعیت تولید نمودیم، آن‌ها را جهت ایجاد جمعیت والد P ترکیب می‌نماییم. سپس ذرات موجود در P را بازترکیب کرده تا یک مجموعه‌ی فرزند مانند C به دست آید. جهت عمل بازترکیب می‌توان از هر یک از الگوریتم‌های تکاملی بحث شده در این کتاب مانند الگوریتم‌های ژنتیک، تکامل دیفرانسیلی، بهینه‌سازی زیست‌جغرافی-محور و غیره، استفاده نمود. می‌توان دید که نام VEGA کمی نابه‌جا انتخاب شده است چرا که بسته به روش بازترکیبی که استفاده می‌نماییم، می‌توان آن را VEDE و یا VEBBO و یا هر نام خلاصه‌ی دیگری که مناسب روش بازترکیب مورد استفاده باشد، نامید. این نکته در مورد بسیاری MOEAهای مشهور دیگری که در این فصل در مورد آن‌ها بحث خواهیم نمود، نیز صادق است.

همان‌طور که پیش از این در الگوریتم‌های تکاملی تک هدفه مشاهده نموده‌ایم، نخبه‌گرایی می‌تواند عملکرد بهینه‌سازی را به مقدار قابل توجهی بهبود بخشد. این موضوع در مورد MOEAها نیز صادق است. برای پیاده‌سازی نخبه‌گرایی در شکل ۲۰-۸ چند راه وجود دارد. برای مثال، می‌توان در هر نسل بهترین ذرات نسبت به هر تابع هدف را پیدا کرده و آن‌ها را برای نسل بعد حفظ نمود. راه دیگر آن است که ذرات غیرمغلوب را در هر نسل یافته و حداقل تعدادی از آن‌ها را برای نسل بعد حفظ نمود. با اینکه VEGA عمدتا به‌عنوان الگوریتمی نخبه‌گرا شناخته نمی‌شود، اضافه نمودن نخبه‌گرایی به شکل ۲۰-۸ اساس آن را تغییر نداده و احتمالاً باعث بهبود عملکردش هم خواهد شد.

یک الگوریتم مشابه VEGA، که عمدتا با نام الگوریتم ژنتیک حاجلا-لین (HLGA[1]) شناخته می‌شود، از رویکرد جمع وزن‌ها استفاده می‌نماید. در این الگوریتم بردار وزن معادله‌ی (۲۰-۲۵) قسمتی از متغیر تصمیم هر ذره می‌باشد [حاجلا و لین، ۱۹۹۷]. این رویکرد از بهینه‌سازی تک هدفه بر اساس روش جمع وزن‌ها از معادله‌ی (۲۰-۲۵) استفاده می‌نماید. HLGA از به اشتراک‌گذاری برازندگی برای دستیابی به وزن‌های متنوع استفاده می‌نماید (بخش ۸-۶-۳-۱ را ببینید).

[1] Hajela-Lin Genetic Algorithm

۲۰-۳-۳ مرتب‌سازی واژه‌نگاری

مرتب‌سازی واژه‌نگاری مانند VEGA است با این تفاوت که به کاربر اجازه می‌دهد ترتیب اهمیت اهداف را تعیین نماید [فورمن[1]، ۱۹۸۵]. در این روش از انتخاب مسابقه‌ای با مقایسه‌ی ذرات بر اساس اهداف اولویت‌بندی شده، استفاده می‌شود. همچنین می‌توان به جای استفاده از اهداف اولویت‌بندی شده، از اهداف انتخاب شده به‌صورت اتفاقی برای هر مسابقه استفاده نمود [کورساو[2]، ۱۹۹۱]. مرتب‌سازی واژه‌نگاری از لحاظ اداره‌ی ترتیبی اهداف مانند رویکرد حافظه‌ی رفتاری در بهینه‌سازی مقید (بخش ۱۹-۲-۱۱ را ببینید)، می‌باشد.

شکل ۲۰-۹ طرح کلی روش مرتب‌سازی واژه‌نگاری را نشان می‌دهد. در روش مرتب‌سازی واژه‌نگاری اصلی، حلقه‌ی بیرونی در شکل ۲۰-۹ برای $i \in [1,k]$ و بر اساس اولویت توابع هدف اجرا می‌شود. در تنوع اتفاقی، حلقه‌ی بیرونی آن قدر اجرا می‌شود تا یک معیار معین، که توسط کاربر تعیین می‌گردد، برآورده شود. در این صورت، اندیس i در هر نسل به‌صورت اتفاقی تغییر می‌کند. مانند VEGA در اینجا نیز می‌توان تنوعات بسیار زیادی را در شکل ۲۰-۹ پیاده‌سازی نمود. برای مثال می‌توان نخبه‌گرایی و یا هر الگوریتم تکاملی که در این کتاب مورد بحث واقع شده است را با این روش ترکیب نمود.

```
جمعیت تولید شده به صورت اتفاقی ← P
تا زمانی که شرایط توقف برآورده نشده است
    اندیس تابع هدف را تعیین کن: i
    یک الگوریتم تکاملی را با جمعیت P آغاز کن
    از الگوریتم تکاملی جهت مینیمم نمودن f_i(x) استفاده کن و جمعیت نهایی را با P نشان بده
الگوریتم تکاملی بعدی
```

شکل ۲۰-۹ مرتب‌سازی واژه‌نگاری برای یک مسئله‌ی بهینه‌سازی با k هدف. تابع هدف $f_i(x)$ در هر دوره به اولویت‌بندی کاربر بستگی داشته و یا می‌توان به‌صورت اتفاقی انتخاب گردد. همچنین، تابع هدف $f_i(x)$ ممکن است از یک نسل به نسل دیگر الگوریتم تکاملی تغییر نماید.

[1] Fourman
[2] Kursaw

۲۰–۳–۴ روش قید ϵ

روش قید ϵ [ریتزل[1] و همکاران، ۱۹۹۵] در هر زمان یک تابع هدف را مینیمم کرده و مقادیر سایر توابع هدف را به یک آستانه‌ی معین محدود می‌نماید. ابتدا مقدار مینیمم هر تابع هدف $f_i(x)$ را با استفاده از الگوریتم تکاملی تک هدفه می‌یابیم ($i \in [1,k]$). این کار حد پایین قید تابع هدف ϵ_i را به دست خواهد داد:

$$\epsilon_i > \min_i f_i(x) \quad \text{برای } i \in [1,k] \qquad (۳٤–۲۰)$$

سپس اولین هدف را، در حالی که سایر اهداف را محدود به کوچکتر بودن از یک آستانه‌ی معین کرده‌ایم، مینیمم می‌نماییم:

$$\min_x f_1(x) \quad \text{به طوری که } f_i(x) < \epsilon_i \text{ برای } i \in [1,k]،\ i \neq 1 \qquad (۳۵–۲۰)$$

سپس هدف بعدی را با استفاده از آخرین جمعیت منتج از معادله‌ی (۲۰–۳۵) به‌عنوان جمعیت آغازین نسل بعد، مینیمم می‌نماییم:

$$\min_x f_2(x) \quad \text{به طوری که } f_i(x) < \epsilon_i \text{ برای } i \in [1,k]،\ i \neq 2 \qquad (۳٦–۲۰)$$

این فرایند را برای تمامی k هدف تکرار می‌نماییم. سپس مقادیر ϵ_iها را و فرایند مینیمم‌سازی متوالی را تکرار می‌نماییم. این رویکرد متوالی مانند مرتب‌سازی واژه‌نگاری (بخش ۲۰–۳–۳ را ببینید) بوده و از دید بهینه‌سازی مقید نیز مشابه رویکرد رفتار حافظه‌ای در بهینه‌سازی مقید می‌باشد (بخش ۱۹–۲–۱۱ را ببینید). شکل ۲۰–۱۰ طرح کلی MOEA قید ϵ را نشان می‌دهد. چالش‌برانگیزترین قسمت این الگوریتم، چگونگی قرار دادن ϵ_i برابر با مقداری بزرگتر از $f_i^*(x)$ و همچنین چگونگی کاهش آن برای $i \in [1,k]$ در شکل ۲۰–۱۰ می‌باشد. همانند سایر MOEAها می‌توان تنوعات بسیاری مانند نخبه‌گرایی و ترکیب سایر الگوریتم‌های تکاملی، را در شکل ۲۰–۱۰ در نظر گرفت.

برای هر تابع هدف $f_i(x)$ که در آن $i \in [1,k]$
از یک الگوریتم تکاملی برای پیدا کردن $f_i^*(x) = \min_x f_i(x)$ استفاده کن
ϵ_i را برابر عددی بزرگتر از $f_i^*(x)$ قرار بده
تابع هدف بعدی
الگوریتم تکاملی را با جمعیت P برای MOP مقداردهی کن

[1] Ritzel

تا زمانی که شرایط توقف برآورده نشده است
برای هر تابع هدف $f_i(x)$ که در آن $i \in [1,k]$
یک الگوریتم تکاملی را با نتیجه‌ی الگوریتم تکاملی قبلی مقداردهی اولیه کن
از یک الگوریتم تکاملی جهت مینیمم ساختن $f_i(x)$ به طوری که $f_r(x) < \epsilon_r$ برای $r \in [1,k]$ و $r \neq i$، استفاده کن
جمعیت نهایی الگوریتم تکاملی $\leftarrow P$
تابع هدف بعدی
ϵ_i را برای $i \in [1,k]$ کاهش بده
دوره‌ی بعدی

شکل ۲۰-۱۰ طرح کلی روش قید ϵ برای حل یک مسئله‌ی بهینه‌سازی با k هدف.

۲۰-۳-۵ رویکردهای جنسیت-محور

رویکرد جنسیت-محور، به هر ذره بر اساس تابع هدفی که قرار است با آن سنجیده شود، یک جنس اختصاص می‌دهد [آلنسون[1]، ۱۹۹۲]، [لیس[2] و ایبن، ۱۹۹۷]. رویکردهای جنسیت-محور همچنین از یک جمعیت دوم با نام آرشیو[3] استفاده می‌نمایند. آرشیو اجتماعی از ذرات نامغلوب بوده و مانند نخبه‌گرایی مانند از بین رفتن ذرات می‌شود. بسیاری انواع دیگر MOEA نیز از آرشیو استفاده می‌نمایند که در زیر به بحث در مورد برخی از آن‌ها پرداخته‌ایم.

در یک رویکرد جنسیت-محور برای یک MOP با k هدف، k جنس مختلف داریم که هر یک متناظر با یکی از اهداف می‌باشند. ما یک جمعیت ابتدایی با تعداد مساوی ذرات برای هر جنس تولید می‌نماییم. سپس بر اساس $f_i(x)$ از هر جنس i ذره‌ای را انتخاب می‌نماییم. سپس از ذرات انتخاب شده جهت عمل بازترکیب و تولید فرزندان استفاده می‌نماییم. می‌توان جنس فرزند را بر اساس هدفی که فرزند در آن بهترین عملکرد را دارد، تعیین نماییم. می‌توان برای عمل بازترکیب از هر یک از روش‌هایی استفاده شده در این کتاب استفاده نمود. برای برش استاندارد GA تک-نقطه‌ای می‌توان از تنها دو ذره برای بازترکیب استفاده نمود. برای برش چند-والدی (بخش ۸-۸-۵ را ببینید)، می‌توان از یک یا چند ذره از هر جنس استفاده نمود. در انتهای هر

[1] Allenson
[2] Lis
[3] Archive

نسل، معمولاً جمعیت را با آرشیو مقایسه کرده و ذرات نامغلوب را در آن ذخیره و ذرات مغلوب را از آن حذف می‌نماییم. شکل ۲۰-۱۱ طرح کلی رویکرد جنسیت-محور را برای بهینه‌سازی چندهدفه نشان می‌دهد.

```
اندازه جمعیت مطلوب برای هر جنس = N_g
k جمعیت الگوریتم تکاملی (P_i)، را مقداردهی اولیه کن به‌طوری که |P_i| = N_g باشد (i ∈ [1, k])
اندازه جمعیت الگوریتم تکاملی: N ← kN_g
تا زمانی که شرایط توقف برآورده نشده است
    برای m = 1 تا N
        برای i = 1 تا k
            از f_i(x) جهت انتخاب یک والد به‌صورت احتمالاتی از P_i استفاده کن
        i بعدی
        از k ذره‌ی انتخاب شده جهت تولید ذره‌ی فرزند c_m استفاده کن
    m بعدی
    جنسیت فرزندان را تعیین کن {c_m}
    فرزندان را به‌صورت اتفاقی دچار جهش کن
    فرزندان غیرمغلوب را در آرشیو ذخیره کن
    ذرات مغلوب را از آرشیو حذف کن
    جمعیت‌های الگوریتم تکاملی را با فرزندانی از جنس مناسب جایگزین کن
نسل بعد
```

شکل ۲۰-۱۱ طرح کلی یک الگوریتم جنسیت-محور برای حل یک مسئله‌ی بهینه‌سازی با k هدف.

جای زیادی برای اعمال اصلاحات به الگوریتم شکل ۲۰-۱۱ وجود دارد. برای مثال، بسته به بعد x ممکن است بخواهیم بیش از یک والد از هر جنس برای عمل بازترکیب انتخاب نماییم. همچنین، خط "*جنسیت فرزندان را تعیین کن*" جزییات تعیین نشده‌ی بسیاری را به جای می‌گذارد. آیا باید هر فرزند را به تنها یک جنس محدود نماییم؟ آیا باید تعداد ذرات هر جنس در جمعیت برابر باشد؟ رویکرد نشان داده شده در شکل ۲۰-۱۱ N فرزند تولید می‌نماید، اما ممکن است تعداد بیشتری فرزند بخواهیم تا بتوانیم جمعیتی از فرزندان با فرزندان بسیار برازنده برای هر جنس به دست آوریم.

عبارت "*فرزندان غیر غالب را آرشیو کن*" در شکل ۲۰-۱۱ جزییات زیادی را تعیین نشده به جای می‌گذارد. آییا باید اجازه دهیم تا آرشیو بدون هیچ محدودیتی بزرگ شود؟ آیا باید یک حد بالا برای آرشیو در نظر بگیریم؟ آیا باید ذراتی که نسبت به جمعیت حاضر نامغلوب هستند را به آرشیو اضافه کنیم یا اینکه باید ذراتی را که نسبت به آرشیو نامغلوب هستند را به آن اضافه نماییم؟ برخی از این موارد مرتبط با آرشیو را به‌صورت کلی‌تر در بخش ۲۰-٤ مورد بحث قرار خواهیم داد.

۲۰-۴ الگوریتم‌های تکاملی پرتو-محور

رویکردهای MOEA بخش‌های قبل، سعی در یافتن یک مجموعه‌ی پرتوی متنوع از راه‌حل‌ها برای یک MOP دارند. با این حال، هیچ یک از این رویکردها به‌صورت مستقیم از مفهوم بهینگی پرتو برای محاسبه‌ی غلبگی نسبی میان ذرات و یا گروه‌های ذرات، استفاده نمی‌کنند (به غیر از زمانی که ذرات در شکل ۲۰-۱۱ به آرشیو اضافه می‌شوند). بخش‌های در پیش رو، رویکردهایی را معرفی می‌نمایند که به‌صورت مستقیم از غلبگی پرتو استفاده می‌نمایند.

- بخش ۲۰-٤-۱ بهینه‌ساز تکاملی چندهدفه ساده (SEMO[1]) و بهینه‌سازی تکاملی چندهدفه‌ی تنوع (DEMO[2]) را مورد بحث قرار می‌دهد.
- بخش ۲۰-٤-۲ ϵ-MOEA محور را مورد بحث قرار می‌دهد.
- بخش ۲۰-٤-۳ الگوریتم ژنتیک مرتب‌سازی بر حسب ذرات غیرمغلوب (NSGA[3]) و نسخه‌ی به‌روزرسانی شده‌ی آن (NSGA-II) را مورد بحث قرار می‌دهد.
- بخش ۲۰-٤-٤ الگوریتم ژنتیک چندهدفه (MOGA[4]) را مورد بحث قرار می‌دهد.
- بخش ۲۰-٤-٥ الگوریتم ژنتیک پرتوی نیچه (NPGA[5]) را مورد بحث قرار می‌دهد.
- بخش ۲۰-٤-٦ الگوریتم تکاملی استحکام پرتو (SPEA[6]) و نسخه‌ی به‌روزرسانی شده‌ی آن (SPEA2) را مورد بحث قرار می‌دهد.
- بخش ۲۰-٤-۷ استراتژی تکاملی پرتوی آرشیو شده (PAES[7]) را مورد بحث قرار می‌دهد.

[1] Simple Evolutionary Multi-objective Optimizer
[2] Diversity Evolutionary Multi-objective Optimizer
[3] Non-dominated Sorting Genetic Algorithm
[4] Multi-Objective Genetic Algorithm
[5] Niched Pareto Genetic Algorithm
[6] Strength Pareto Evolutionary Algorithm
[7] Pareto Archived Evolutionary Strategy

۲۰-۴-۱ بهینه‌سازهای تکاملی چندهدفه

این بخش به بحث در مورد دو بهینه‌ساز تکاملی چندهدفه می‌پردازد: بهینه‌ساز تکاملی چندهدفه‌ی ساده (SEMO) و بهینه‌ساز تکاملی چندهدفه‌ی تنوع (DEMO). در این بخش خواهید دید که ایده‌ی اصلی این دو الگوریتم از EP و ES نشات می‌گیرد.

بهینه‌ساز تکاملی چندهدفه‌ی ساده (SEMO)

SEMO اولین بار به‌عنوان یک بهینه‌سازی دودویی ارائه گردید [لائومنز[1] و همکاران، ۲۰۰۳]، اما می‌توان آن را به راحتی به بهینه‌سازی پیوسته نیز بسط داد. در SEMO کار با یک جمعیت اتفاقی از ذرات آغاز می‌گردد. اندازه‌ی جمعیت آغازین الگوریتم اصلی SEMO برابر با ۱ می‌باشد. اندازه‌ی جمعیت با پیدا شدن ذرات غیرمغلوب افزایش می‌یابد. در هر نسل، یک ذره که به‌صورت تصادفی انتخاب شده است دچار جهش شده تا فرزند تولید گردد. اگر ذره‌ی فرزند نسبت به جمعیت نامغلوب باشد، آن را به جمعیت اضافه کرده و هر ذره‌ای که مغلوب باشد را از جمعیت حذف می‌نماییم. شکل ۲۰-۱۲ الگوریتم SEMO را نشان می‌دهد. جهش اتفاقی در شکل ۲۰-۱۲ می‌تواند هر یک از روش‌های جهشی که در فصل‌های ۵، ٦ و بخش ۸-۹ مورد بحث قرار گرفته‌اند، باشد.

الگوریتم SEMO یک نقطه‌ی شروع مناسب برای بهینه‌سازی چندهدفه را در اختیار ما قرار می‌دهد. می‌توان این الگوریتم را با استفاده از ایده‌های MOEAهای دیگر اصلاح نمود. برای مثال، به جای استفاده از "randomly selected individual from P" به‌عنوان والد y، می‌توان از انتخاب متناسب با برازندگی استفاده نمود. همچنین، می‌توان y را با استفاده از چندین والد تولید نمود. همچنین می‌توان مانند آنچه که در SPEA2 اتفاق می‌افتد، جمعیت را به‌صورت متناوب هرس نمود (بخش ۲۰-٤-٦ را ببینید).

[1] Laumanns

جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن: $P = \{x_j\}$
هزینه‌ی $f_i(x_j)$ را برای هر ذره‌ی $x_j \in P$ و هر هدف محاسبه کن
تا زمانی که شرایط توقف برآورده نشده است
  جهش ذراتی که به صورت اتفاقی از P انتخاب شده‌اند $\leftarrow y$
  اگر y توسط هیچ یک از ذرات موجود در P مغلوب نشده است، آنگاه
    $P \leftarrow \{P, y\}$
    تمام ذراتی که توسط y مغلوب شده‌اند را از P حذف کن
  پایان اگر
نسل بعد

شکل ۲۰-۱۲ طرح کلی بهینه‌ساز تکاملی چندهدفه‌ی ساده (SEMO).

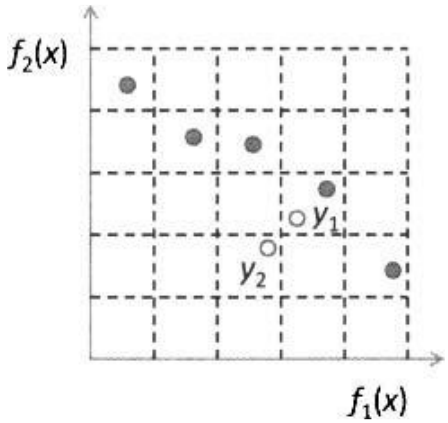
بهینه‌ساز تکاملی چندهدفه‌ی تنوع

یکی از مشکلات SEMO رشد نامحدود جمعیت آن می‌باشد. برای رفع این مشکل می‌توان هنگام تست y جهت قرار گرفتن در جمعیت P، از غلبگی ϵ به جای غلبگی ساده استفاده نمود. با این کار بهینه‌ساز تکاملی چندهدفه‌ی تنوع (DEMO) به دست خواهد آمد. DEMO از همان الگوریتم شکل ۲۰-۱۲ استفاده می‌نماید با این تفاوت که از غلبگی ϵ به‌عنوان معیار قرار دادن y در جمعیت P استفاده می‌شود [هوربا[1] و نیومن، ۲۰۱۰]. بدین صورت، یک استاندارد برای قرار دادن فرزند y در جمعیت به دست می‌آید.؛ ذره‌ی y را در جمعیت قرار می‌دهیم اگر تحت غلبه‌ی ϵ هیچ ذره‌ی دیگری از جمعیت P قرار نداشته باشد. از آنجا که غلبگی ϵ نوع ضعیف‌تر از غلبگی پرتو است (بخش ۲۰-۱ را ببینید)، معیار قرار دادن y در جمعیت در DEMO از SEMO سختگیرانه‌تر است. DEMO فضای هدف را به چند فرابسته تقسیم کرده و به جمعیت اجازه نمی‌دهد بیش از یک ذره را در هر فرابسته قرار دهد. ما معمولاً از غلبگی ϵ افزایشی در DEMO استفاده می‌نماییم.

شکل ۲۰-۱۳ مفهوم DEMO را برای یک مسئله‌ی بهینه‌سازی دو هدفه نشان می‌دهد. فرزند y_1 در جمعیت قرار نگرفته است چرا که تحت غلبه‌ی ϵ ذره‌ای است که در همان فرابسته واقع شده است. این در حالی است که فرزند y_2 در جمعیت واقع شده است چرا که تحت غلبه‌ی ϵ هیچ یک از اعضای فعلی جمعیت قرار ندارد. توجه داشته باشید که شکل ۲۰-۱۳ دقیقا درست نیست چرا که فرابسته‌های DEMO نسبت به

[1] Horoba

جمعیت حاضر تعریف می‌گردند. با این حال، این شکل مفهومی تصویری از استفاده از غلبگی ϵ در DEMO نشان می‌دهد. با این که در شکل ۲۰-۱۳ $\epsilon_1 = \epsilon_2$ نشان داده شده است (به معنای این که بسته‌ها مربعی هستند)، کاربر می‌تواند مقادیر متفاوتی برای هر اندیس هدف $i \in [1, k]$ انتخاب نماید.



شکل ۲۰-۱۳ غلبگی ϵ در بهینه‌ساز تکاملی چندهدفه‌ی تنوع (DEMO). اعضای فعلی جمعیت با دایره‌های توپر و فرزندان نیز با دایره‌های توخالی نمایش داده شده‌اند. یک فرزند تنها در صورتی به جمعیت اضافه می‌گردد که تحت غلبه‌ی هیچ یک از اعضای فعلی جمعیت نباشد. در این شکل، فرزند y_2 به جمعیت اضافه شده و فرزند y_1 به جمعیت اضافه نخواهد شد.

۲۰-۴-۲ الگوریتم تکاملی چندهدفه‌ی ϵ-محور (ϵ-MOEA)

ϵ-MOEA از مفهوم غلبگی ϵ به همان ترتیبی که در DEMO استفاده گردید، استفاده می‌نماید [دب و همکاران، ۲۰۰۵]. ϵ-MOEA شامل یک جمعیت و یک آرشیو می‌شود. در هر نسل یک ذره از جمعیت و یک ذره از آرشیو انتخاب کرده و از آن‌ها در عمل بازترکیب جهت تولید فرزند استفاده می‌نماییم.

اگر فرزند بر ذره‌ای در جمعیت غلبه داشته باشد، ذره‌ی مغلوب را با فرزند جایگزین می‌نماییم. اگر فرزند بیش از یک ذره را مغلوب نماید، آنگاه یکی از ذرات مغلوب را به‌صورت اتفاقی با فرزند جایگزین می‌نماییم.

سپس فرزند را با آرشیو مقایسه می‌نماییم. چهار حالت ممکن است پیش بیاید: (۱) اگر فرزند توسط هر یک از ذرات آرشیو مغلوب شده باشد، آنگاه فرزند در آرشیو قرار نخواهد گرفت، (۲) اگر ذره فرزند هر یک از ذرات آرشیو را مغلوب ساخت، آنگاه فرزند به آرشیو اضافه شده و ذره‌ی مغلوب شده از آرشیو حذف خواهد شد.

اگر هیچ یک از این دو حالت برقرار نباشد، آنگاه جعبه‌ی ϵ، که با $B(x)$ نمایش داده می‌شود را برای ذره‌ی x محاسبه می‌نماییم:

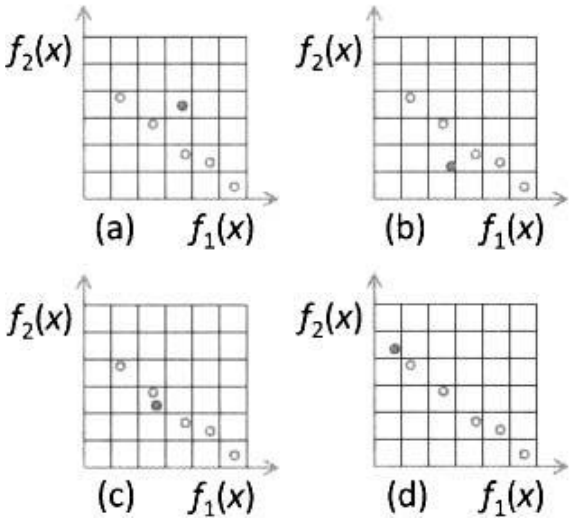
$$B_j(x) = \left\lfloor \frac{f_j(x)}{\epsilon_j} \right\rfloor \qquad (۳۷-۲۰)$$

k تعداد اهداف، ϵ_j دقت مطلوب jاُمین هدف، $\lfloor . \rfloor$ عملگری است که بزرگترین عدد صحیح کوچکتر یا مساوی آرگومانش را برگردانده و $j \in [1, k]$ می‌باشد. این کار ما را به سوی حالت سوم راهنمایی می‌کند. (۳) اگر فرزند x در فرابسته‌ی یک ذره‌ی آرشیو شده مانند a قرار داشته باشد، آنگاه فرزند جایگزین a خواهد شد اگر به مبدأ فضای تابع هدف نزدیک‌تر باشد:

$$\left[B(x) = B(a),\ \sum_{j=1}^{k} f_j^2(x) < \sum_{j=1}^{k} f_j^2(a)\right] \text{ جایگزین } a \text{ خواهد شد اگر } x \qquad (۳۸-۲۰)$$

در شرایط بالا فرض بر آن است که مقادیر توابع هدف نرمالیزه بوده و با یکدیگر متناسب هستند. معادله‌ی (۲۰-۳۸) از فاصله‌ی اقلیدسی برای اندازه‌گیری فاصله استفاده می‌نماید. با این حال می‌توان از هر نُرم دیگری نیز استفاده نمود. (٤) اگر هیچ یک از سه حالت قبل اتفاق نیافتد، فرزند را به آرشیو اضافه کرده و بدین ترتیب اندازه‌ی آرشیو به اندازه‌ی یک ذره اضافه خواهد شد.

شکل ۲۰-۱٤ این چهار حالت ممکن را نشان می‌دهد. توجع داشته باشید که منطق به کار رفته در اینجا اجازه نمی‌دهد در هر جعبه‌ی ϵ بیش از یک ذره حضور داشته باشد. با این که آرشیو می‌تواند از یک نسل به نسل دیگر رشد کرده و بزرگ شود، اندازه‌ی آن با استفاده از این منطق محدود می‌شود. شکل ۲۰-۱٤(الف) حالتی را نشان می‌دهد که فرزند مغلوب یک یا چند ذره‌ی آرشیو شده می‌باشد. در این حال فرزند به آرشیو اضافه نخواهد شد. شکل ۲۰-۱٤(ب) حالتی را نشان می‌دهد که در آن فرزند یک یا چند ذره از ذرات آرشیو را مغلوب ساخته است. در این حالت فرزند جایگزین ذرات مغلوب شده خواهد شد. شکل ۲۰-۱٤(ج) حالتی را نشان می‌دهد که فرزند و ذرات آرشیو شده نسبت به هم نامغلوب بوده و فرزند در بسته‌ی ϵ یکی از ذرات آرشیو شده قرار دارد. در این حالت، فرزند در صورتی که به مبدأ فضای تابع هدف نزدیک‌تر باشد، جایگزین ذره‌ای که در همان بسته‌ی ϵ قرار دارد، خواهد شد. شکل ۲۰-۱٤(د) حالتی را نشان می‌دهد که فرزند و ذرات آرشیو شده نسبت به هم نامغلوب بوده و فرزند در بسته‌ی ϵ هیچ یک از ذرات آرشیو شده قرار ندارد. در این صورت فرزند به آرشیو اضافه شده و اندازه‌ی آرشیو به اندازه‌ی یک ذره افزایش می‌یابد.



شکل ۲۰-۱٤ منطق ϵ-MOEA برای اضافه نمودن یک فرزند به آرشیو. دایره‌های توخالی ذرات آرشیو شده و دایره‌ی پر ذره‌ی فرزند است و خانه‌های مربعی بر روی فضای تابع هدف معرف بسته‌های ϵ هستند. در حالت (الف)، فرزند به آرشیو اضافه نمی‌شود. در حالت (ب)، فرزند جایگزین ذراتی که تحت غلبه‌اش می‌باشند شده که این کار در این شکل اندازه‌ی آرشیو را به اندازه‌ی یک ذره کاهش می‌دهد. در حالت (ج) فرزند جایگزین ذره‌ی آرشیو شده‌ای می‌شود که در بسته‌ی ϵ آن قرار دارد. این بدین دلیل است که فرزند به مبدأ صفحه‌ی f_1/f_2 نزدیک‌تر است. در حالت (د)، فرزند به آرشیو اضافه شده و اندازه‌ی آرشیو به اندازه‌ی یک ذره افزایش می‌یابد.

شکل ۲۰-۱۵ طرح کلی ϵ-MOEA را نشان می‌دهد. می‌توان برخی تنوعات را در این شکل به کار برد. برای مثال، معمولاً از انتخاب مسابقه‌ای با اندازه‌ی مسابقه‌ی دو برای انتخاب والد x از P استفاده می‌شود. با این حال، می‌توان از هر روش انتخاب دیگری نیز استفاده نمود. معمولاً والد a نیز به‌صورت اتفاقی از آرشیو A انتخاب می‌شود. در اینجا نیز می‌توان از هر روش انتخابی استفاده نمود. روش بازترکیبی که برای تولید فرزند استفاده می‌نماییم می‌تواند هر یک از روش‌های بخش ۸-۸ باشد. اگر از بازترکیب چند والدی استفاده نماییم، باید در مورد تعداد والدین انتخاب شده از P و همچنین تعداد والدین انتخاب شده از A تصمیم‌گیری نماییم.

جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن: $P = \{x_j\}$ برای $j \in [1, N]$
ذرات نامغلوب را از P به A کپی کن
تا زمانی که شرایط توقف برآورده نشده است
  یک والد مانند x را از P و یک والد مانند a را از A انتخاب کن
  x و a را بازترکیب کرده تا فرزند c به وجود آید
  $D_P \leftarrow \{x \in P : x \text{ بر } c \text{ غلبه دارد}\}$
  اگر $D_P \neq \emptyset$، آنگاه
    یک $x \in D_P$ را به‌صورت اتفاقی جایگزین c کن
  پایان اگر
  $D_A \leftarrow \{a \in A : a \text{ بر } c \text{ غلبه دارد}\}$
  اگر $D_A \neq \emptyset$، آنگاه
    c را به A اضافه کن
    D_A را از A حذف کن
  در غیر این صورت اگر معادله‌ی (۲۰–۳۸) برآورده می‌شود
    c را به A اضافه کن
    a را از A حذف کن
  در غیر این صورت اگر (c نسبت به A نامغلوب است) و ($B(c) \neq B(a)$ برای تمامی aها)، آنگاه
    c را به A اضافه کن
  پایان اگر
نسل بعد

شکل ۲۰–۱۵ شبه کد بالا، طرح کلی ϵ–MOEA را برای حل یک مسئله‌ی بهینه‌سازی با k هدف را نشان می‌دهد.

۲۰–۴–۳ الگوریتم ژنتیک مرتب‌سازی بر حسب ذرات غیرمغلوب (NSGA)

NSGA در [اسرینیواس[1] و دب، ۱۹۹٤] مطرح گردیده و بر اساس ایده‌های طرح شده در [گلدبرگ، ۱۹۸۹a] می‌باشد. NSGA هزینه‌ی هر ذره را بر اساس میزان غلبگی آن تعیین می‌نماید. ابتدا تمامی ذرات را

[1] Srinivas

به درون یک جمعیت موقتی T کپی می‌نماییم. سپس تمامی ذرات غیرمغلوب موجود در T را می‌یابیم. سپس به این ذرات، که آن‌ها را با مجموعه‌ی B نشان می‌دهیم، کمترین مقدار هزینه را اختصاص می‌دهیم. سپس B را از T حذف کرده تمامی ذرات نامغلوب مجموعه‌ی کاهش یافته‌ی T را می‌یابیم. به این ذرات دومین کمترین مقدار هزینه را اختصاص می‌دهیم. این فرایند را آنقدر ادامه می‌دهیم تا به تمامی ذرات یک مقدار هزینه اختصاص داده شود. بدین صورت مقادیر هزینه‌ی ذرات بر اساس میزان نامغلوب بودنشان خواهد بود. شکل ۲۰-۱۶ طرح کلی NSGA را نشان می‌دهد.

شکل ۲۰-۱۶ نشان می‌دهد که NSGA با یک جمعیت N ذره‌ای که معمولاً به‌صورت اتفاقی تولید می‌گردد، آغاز می‌شود. در هر نسل مقادیر تمامی k تابع هزینه را برای تمامی ذرات محاسبه می‌نماییم. سپس ذرات را به یک جمعیت موقت T کپی می‌نماییم. سپس به تمامی ذراتی که غیرمغلوب هستند مقدار هزینه‌ی ۱ را اختصاص می‌دهیم. حال این ذرات را از T حذف کرده و ذرات غیرمغلوب از مجموعه‌ی کاهش یافته‌ی T را می‌یابیم و به این ذرات مقدار هزینه‌ی ۲ را اختصاص می‌دهیم. این فرایند را آنقدر ادامه می‌دهیم تا تمامی ذرات دارای مقدار هزینه‌ای بر اساس میزان غلبه‌شان شوند. سپس از مقادیر هزینه‌ی $\phi(.)$ در شکل ۲۰-۱۶ برای انجام فرایند انتخاب استفاده کرده و ذرات موجود در P را با استفاده از یکی از الگوریتم‌های تکاملی و روش‌های بازترکیب دلخواه بازترکیب می‌نماییم. در آخر، جمعیت فرزندان را دچار جهش کرده، والدین را با فرزندان جایگذاری نموده و به نسل بعد می‌رویم.

گاهاً NSGA برای ناکارامدیش به دلیل حلقه‌ی داخلی، که ار آن برای محاسبه‌ی هزینه‌ی $\phi(.)$ در شکل ۲۰-۱۶ استفاده می‌شود، مورد نقد قرار می‌گیرد. اما برای مسائل دنیای واقعی، قسمت اعظمی از تلاش محاسباتی به ارزیابی‌های توابع $f_i(.)$ اختصاص یافته و از این رو حلقه‌ی داخلی NSGA بار محاسباتی ناچیزی را به کل این فرایند اضافه می‌نماید.

جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن: $P = \{x_j\}$ برای $j \in [1, N]$
تا زمانی که شرایط توقف برآورده نشده است
  $T \leftarrow P$ جمعیت موقتی
  $c \leftarrow 1$ سطح نامغلوبی
  تا زمانی که $|T| > 0$
    ذرات نامغلوب در $T \leftarrow B$
    $c \leftarrow$ هزینه $\phi(x)$ برای تمامی $x \in B$

B را از T حذف کن
$c \leftarrow c + 1$
سطح نامغلوبی بعدی
N فرزند تولید شده از ترکیب ذرات درون $P \leftarrow C$
فرزندان درون C را به‌صورت احتمالاتی دچار جهش کن
$P \leftarrow C$
نسل بعد

شکل ۲۰-۱۶ طرح کلی الگوریتم ژنتیک مرتب‌سازی بر حسب ذرات غیرمغلوب (NSGA) برای حل یک مسئله‌ی بهینه‌سازی با k هدف. از مقادیر هزینه‌ی $\phi(x_j)$ برای انتخاب ذرات جهت بازترکیب، استفاده می‌نماییم.

NSGA-II

NSGA-II نسخه‌ی اصلاح شده‌ی NSGA می‌باشد [دب و همکاران، ۲۰۰۲a]. NSGA-II در هنگام محاسبه‌ی مقدار هزینه‌ی ذره‌ی x نه تنها ذراتی که آن را مغلوب ساخته‌اند، بلکه ذراتی که توسط آن مغلوب می‌شوند را به حساب می‌آورد. همچنین برای هر ذره، یک فاصله‌ی ازدحام را نیز محاسبه می‌نماییم. این مقدار با پیدا نمودن فاصله تا نزدیکترین ذرات در راستای هر یک از ابعاد تابع هدف به دست می‌آید. از فاصله‌ی ازدحام جهت اصلاح برازندگی هر ذره استفاده می‌شود. در NSGA-II مجموعه‌ی آرشیو وجود ندارد، بلکه در آن از استراتژی تکاملی $\mu + \lambda$ جهت پیاده‌سازی نخبه‌گرایی استفاده می‌شود (فصل ٦ را ببینید).

در NSGA فاصله‌ی ازدحام هر ذره برابر میانگین فاصله‌ی آن از ذرات همسایه در راستای هر یک از ابعاد تابع هدف، قرار داده می‌شود. برای مثال، فرض کنید در NSGA جمعیتی با N ذره در اختیار داریم. همچنین فرض کنید بردار تابع هدف x به‌صورت زیر است

$$f(x) = [f_1(x), \dots, f_k(x)] \qquad (۳۹-۲۰)$$

برای هر بعد تابع هدف، نزدیکترین مقدار بزرگتر و نزدیکترین مقدار کوچکتر در جمعیت را به‌صورت زیر می‌یابیم:

$$f_i^-(x) = \max_y [f_i(y) \text{ به طوری که } f_i(y) < f_i(x)]$$
$$f_i^+(x) = \max_y [f_i(y) \text{ به طوری که } f_i(y) > f_i(x)] \qquad (۴۰-۲۰)$$

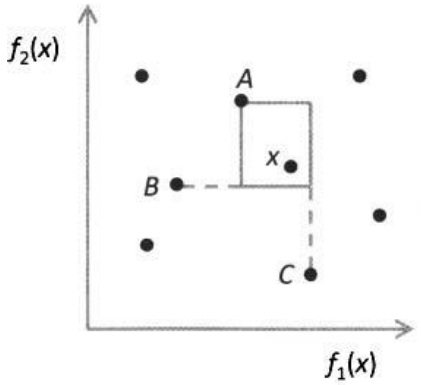
سپس فاصله‌ی ازدحام x را به‌صورت زیر محاسبه می‌نماییم:

$$d(x) = \sum_{i=1}^{k} \left(f_i^+(x) - f_i^-(x) \right) \qquad (۴۱-۲۰)$$

ذراتی که در نقاط پرازدحام‌تر فضای تابع هدف قرار دارند، دارای فاصله‌ی ازدحام کوچکتری خواهند بود. ذراتی که در اکسترمم‌های فضای تابع هدف قرار دارند دارای فاصله‌ی ازدحام بی‌نهایت می‌باشند:

$$x \in \left\{ arg\min_y f_i(y) \cup arg\max_y f_i(y) \right\} \text{ برای } d(x) = \infty \text{ برای تمامی } i \in [1, k] \qquad (۴۲-۲۰)$$

فاصله‌ی ازدحام متناظر نصف محیط بزرگترین فرامکعب، که با نام کیوبوید[1] شناخته می‌شود، می‌باشد [دب و همکاران، ۲۰۰۰]. مرز این کیوبوید از مختصات نزدیکترین همسایگان x در طول هر بعد از فضای تابع هدف، فراتر نمی‌رود. شکل ۲۰-۱۷ فرامکعب را در یک فضای تابع هدف دو بعدی نشان می‌دهد. این فرامکعب در واقع یک مستطیل است[2]. در شکل ۲۰-۱۷ نزدیکترین همسایگان x در جهت f_1، نقاط A و C بوده و نزدیکترین همسایه‌ی آن در جهت f_2، نقاط A و B می‌باشند.



شکل ۲۰-۱۷ محاسبه‌ی فاصله‌ی ازدحام در NSGA-II. فاصله‌ی ازدحام x در واقع نصف محیط بزرگترین فرامکعب است (این فرامکعب در فضای دو بعدی یک مستطیل است). مرز این فرامکعب در فضای تابع هدف از مختصات نزدیکترین همسایگان x فراتر نمی‌رود.

حال که تمامی ذرات موجود در جمعیت دارای فاصله‌ی ازدحام می‌باشند، از آن به‌عنوان یک پارامتر مرتب‌سازی ثانویه جهت رتبه‌بندی هر ذره استفاده می‌نماییم. همانند الگوریتم NSGA در اینجا نیز هر ذره را بر اساس میزان نامغلوبیت آن رتبه‌بندی می‌نماییم. اما در NSGA-II این رتبه‌بندی را با استفاده از فاصله‌ی

[1] Cuboid

[2] بر خلاف آنچه که در [دب و همکاران، ۲۰۰۰] مطرح گردیده است، فرامکعب بزرگترین فضای بسته‌ی شامل x که شامل ذره‌ی دیگری نیست، نمی‌باشد. همان‌طور که از شکل ۲۰،۱۷ بر می‌آید، در صورتی که این تعریف درست باشد، فرامکعب باید شکل دیگری به خود می‌گرفت.

ازدحام کمی ظریف‌تر و بهتر انجام می‌دهیم. بنابراین، x دارای رتبه‌ی بهتری نسبت به y خواهد بود اگر $\phi(x) < \phi(y)$ یا $\phi(x) = \phi(y)$ بوده و $d(x) > d(y)$ باشد.

۲۰-۴-۴ الگوریتم ژنتیک چندهدفه (MOGA)

MOGA در [فونسکا[1] و فلمینگ، ۱۹۹۳] معرفی گردید. در MOGA نیز همانند NSGA هزینه‌ی ذرات بر اساس غلبگی آن‌ها تعیین می‌گردد، اما رویکرد MOGA به این موضوع از سوی مخالف است. در NSGA هزینه‌ی x بر اساس تعداد ذراتی که نیاز است از جمعیت حذف شده تا ذره‌ی x نامغلوب گردد تعیین شده، در حالی که در MOGA هزینه‌ی x بر اساس ذراتی که آن را مغلوب می‌سازند تعیین می‌گردد. بنابراین، برای هر ذره‌ی مغلوب مانند x هزینه بر اساس تعداد ذراتی که آن را مغلوب ساخته و همچنین بر اساس تعداد ذراتی که نزدیک آن هستند، تعیین می‌گردد. منظور نمودن میزان نزدیکی باعث تقویت تنوع در جمعیت می‌گردد.

شدر MOGA، x دارای رتبه‌ی بهتری نسبت به y خواهد بود ($\phi(x) < \phi(y)$) اگر توسط تعداد کمتری ذرات در جمعیت مغلوب شده ($d(x) < d(y)$) و یا توسط تعداد برابری ذرات مغلوب گشته شده و تعداد ذرات کمتری در فضای تابع هدف در نزدیکی آن واقع شده باشند ($s(x) < s(y)$). رویکرد رتبه‌بندی را می‌توان به‌صورت زیر مطرح نمود:

$$\begin{aligned} &d(x) = \left|x' \in P : x' \text{ بر } x \text{ غالب است}\right| \\ &s(x) = |x' \in P \,||f(x) - f(x')|| < \sigma| \\ &\left\{d(x) < d(y), \text{ یا } d(x) = d(y) \text{ و } s(x) < s(y)\right\} \text{ اگر } \phi(x) < \phi(y) \end{aligned} \qquad (۴۳-۲۰)$$

که در آن σ یک پارامتر به اشتراک‌گذاری است که توسط کاربر تعیین می‌گردد و $||.||$ نیز نوعی متریک فاصله است. اشتراک‌گذاری را نیز می‌توان به‌صورت خودکار پیاده‌سازی نمود تا نیازی به تعریف پارامتر توسط کاربر نباشد [آهن[2] و راماکریشنا[3]، ۲۰۰۷]. شکل ۲۰-۱۸ طرح کلی MOGA را نشان می‌دهد. همانند سایر MOEAها که در این فصل مطرح گردیدند، می‌توان تنوعات زیادی را در شکل ۲۰-۱۸ در نظر گرفت.

[1] Fonseca
[2] Ahn
[3] Ramakrishna

جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن: $P = \{x_j\}$ برای $j \in [1, N]$
تا زمانی که شرایط توقف برآورده نشده است
از معادله‌ی (۲۰-۴۳) برای پیدا کردن $\phi(x_j)$ برای هر $x_j \in P$ استفاده کن
N فرزند تولید شده از ترکیب ذرات درون $C \leftarrow P$
فرزندان درون C را به‌صورت احتمالاتی دچار جهش کن
$P \leftarrow C$
نسل بعد

شکل ۲۰-۱۸ طرح کلی الگوریتم ژنتیک چندهدفه (MOGA) برای حل یک مسئله‌ی بهینه‌سازی با k هدف. ما از رتبه‌ی $\phi(x_j)$ برای انتخاب والد جهت عمل بازترکیب استفاده می‌نماییم.

۲۰-۴-۵ الگوریتم ژنتیک پرتوی نیچه (NPGA)

NPGA در [هورن[1] و همکاران، ۱۹۹٤] مطرح گردید. این الگوریتم از حیث تعیین هزینه‌ی ذرات مانند NSGA و MOGA عمل کرده و هزینه را بر اساس غلبگی تعیین می‌نماید. NPGA در واقع تلاشی است برای کاهش تلاش محاسباتی NSGA و MOGA. ما دو ذره‌ی x_1 و x_2 را به‌صورت اتفاقی از جمعیت انتخاب می‌نماییم. سپس یک زیرمجموعه مانند S از جمعیت را، که معمولاً حدود ۱۰٪ آن می‌باشد، به‌صورت اتفاقی انتخاب می‌نماییم. اگر یکی از ذرات x_1 و x_2 توسط حداقل یکی از ذرات موجود در S مغلوب شده و دیگری نشده باشد، آنگاه ذره‌ی غیرمغلوب، که آن را با r نمایش می‌دهیم، برنده‌ی مسابقه شده و برای بازترکیب انتخاب می‌شود. اگر هر دو ذره توسط حداقل یکی از ذرات S مغلوب شده و با هیچ کدام توسط هیچ یک از ذرات S مغلوب نشده باشد، آنگاه از اشتراک برازندگی برای تعیین برنده استفاده می‌نماییم. بدین صورت که، ذره‌ای که در کم‌ازدحام‌ترین ناحیه از فضای تابع هدف واقع شده باشد برنده‌ی مسابقه خواهد بود. این فرایند انتخاب را می‌توان به‌صورت زیر توصیف نمود:

$$d_i = |y \in S : y \succ x_i| \quad \text{برای } i \in [1,2] \qquad (۲۰-٤٤)$$

$s_i = x_i$ فاصله ازدحام برای $i \in [1,2]$

[1] Horn

$$r = \begin{cases} x_1 & \text{اگر} \begin{cases} (d_1 = 0) \text{ و } (d_2 > 0), & \text{یا} \\ (d_1 > 0) \text{ و } (d_2 > 0) \text{ و } (s_1 < s_2), & \text{یا} \\ (d_1 = 0) \text{ و } (d_2 = 0) \text{ و } (s_1 < s_2) \end{cases} \\ x_2 & \text{در غیر صورت} \end{cases}$$

d_i نشانگر تعداد ذراتی است که ذره‌ی x_i را مغلوب ساخته، s_i فاصله‌ی ازدحام x_i بوده و r ذره‌ای است (x_1 یا x_2) که در نهایت برای عمل بازترکیب انتخاب می‌نماییم. فاصله‌ی ازدحام s_i را می‌توان به یکی از روش‌های بحث شده در بخش ۸-۶-۳-۱ و یا با استفاده از معادله‌ی (۲۰-۴۳) و یا هر محاسبه‌ی دیگری که مقدار ازدحام x_1 و x_2 را اندازه می‌گیرد، محاسبه نمود. فاصله‌ی ازدحام برای ذراتی که در نواحی پرازدحام‌تر فضای جستجو یا فضای تابع هدف واقع شده‌اند، کوچکتر است. استفاده از فاصله‌ی ازدحام در NPGA باعث تقویت تنوع شده و مانند هر الگوریتم دیگری از این دست، باعث مناسب شدن آن برای مسائل چند پیمانه‌ای و یا مسائلی که در آن‌ها کاربر به دنبال یافتن راه‌حل‌هایی خوب در نواحی گسترده‌ای از فضای تابع یا فضای جستجو می‌باشد، می‌شود. توجه داشته باشید که فاصله‌ی ازدحام از معادله‌ی (۲۰-۴۴) را می‌توان یا در فضای تابع و یا در فضای جستجو محاسبه نمود. این موضوع به اولویت‌های کاربر بستگی دارد.

شکل ۲۰-۱۹ طرحی کلی از NPGA را نشان می‌دهد. NPGA با داشتن زیرمجموعه‌ای اتفاقی مانند S در معادله‌ی (۲۰-۴۴)، اولین MOEA است که با کاهش تعداد ذرات درگیر در فرایند رتبه‌بندی، باعث صرفه‌جویی در تلاش محاسباتی شده است [کوئلو کوئلو، ۲۰۰۹]. مانند سایر MOEAهای بحث شده در این فصل می‌توان شکل ۲۰-۱۹ را به‌گونه‌ای اصلاح نمود که شامل نخبه‌گرایی، آرشیو، استراتژی‌های مختلف بازترکیب و یا استراتژی‌های متنوع انتخاب شود.

جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن: $P = \{x_j\}$ برای $j \in [1, N]$
تا زمانی که شرایط توقف برآورده نشده است
  $R \leftarrow \emptyset$
  تا زمانی که $|R| < N$
    ذرات x_1 و x_2 را به‌صورت اتفاقی از P انتخاب کن
    زیرمجموعه‌ی S را به‌صورت اتفاقی از P انتخاب کن
    از معادله‌ی (۲۰-۴۴) برای انتخاب r از $\{x_1, x_2\}$ استفاده کن
    $R \leftarrow \{R, r\}$

ذرات درون R را جهت به دست آوردن N فرزند بازترکیب کن
فرزندان را به‌صورت اتفاقی دچار جهش کن
فرزندان $\leftarrow P$
نسل بعد

شکل ۲۰-۱۹ طرح کلی الگوریتم ژنتیک پرتوی نیچه (NPGA) برای حل یک مسئله‌ی بهینه‌سازی با k هدف.

۲۰-۴-۶ الگوریتم تکاملی استحکام پرتو (SPEA)

SPEA اولین MOEA است که صریحا از نخبه‌گرایی استفاده نموده است [زیتزلر و ثیل، ۱۹۹۹]، [زیتزلر و همکاران، ۲۰۰٤]. صدالبته که هر یک از MOEAهایی که پیش از این معرفی گردیدند را نیز می‌توان با نخبه‌گرایی پیاده‌سازی نمود، اما بنا به دلایلی اکثر آن‌ها در ابتدای معرفی‌شان شامل نخبه‌گرایی نبوده‌اند. استفاده از نخبه‌گرایی هم در الگوریتم‌های تکاملی تک هدفه و هم در الگوریتم‌های تکاملی چندهدفه، یک حس مشترک است. همچنین، استفاده از نخبه‌گرایی جهت تضمین همگرایی در MOEAها ضروری است [رودالف و آگاپی[1]، ۲۰۰۰]. با این حال، اگر در یک MOEA روش‌های مطابق با اولویت‌ها و ترجیح کاربر به کار رفته باشند و این اولویت‌ها و ترجیح‌ها در طول زمان تغییر یابند، آنگاه نخبه‌گرایی ممکن است باعث زوال عملکرد شود [زیتزلر و همکاران، ۲۰۰۰]. این موضوع مشابه اشکالات استفاده از نخبه‌گرایی در مسائل بهینه‌سازی پویا، که در آن‌ها تابع برازندگی متغیر با زمان است، می‌باشد (فصل ۲۱ را ببینید).

در SPEA تمامی ذرات نامغلوبی که در طول فرایند یادگیری پیاده شده‌اند در یک آرشیو قرار می‌گیرند. هرگاه یک ذره‌ی نامغلوب می‌یابیم آن را به آرشیو کپی می‌نماییم. به هر ذره‌ی موجود در آرشیو مانند α یک مقدار توان $S(\alpha)$ نسبت داده می‌شود که این مقدار با تعداد ذراتی از جمعیت که تحت غلبه‌ی α هستند متناسب است:

$$\text{برای تمامی } \alpha \in A \quad S(\alpha) = \frac{\left|x \in \{P\} \text{ که به طوری } \alpha \succ x\right|}{N+1} \qquad (٤٥-٢٠)$$

در معادله‌ی بالا P مجموعه‌ی راه‌حل‌های نامزد، N اندازه‌ی P و A مجموعه‌ی آرشیو است. توجه داشته باشید که $S(\alpha) \in [0,1)$ می‌باشد. برای هر ذره مانند x در P، مجموعه‌ای $\alpha(x)$ از تمام ذرات آرشیو شده‌ای

[1] Agapie

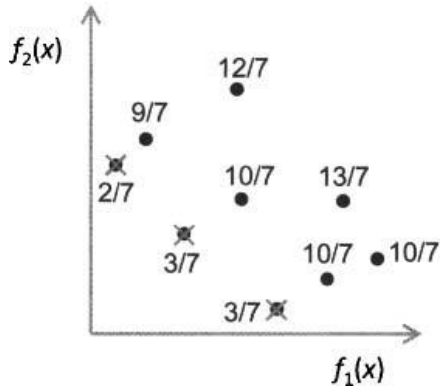
که بر آن غلبه دارند را می‌یابیم. سپس هزینه‌ی خام x را، که با $R(x)$ نشان داده می‌شود، به‌صورت مجموع توان ذرات موجود در $\alpha(x)$ محاسبه می‌نماییم:

$$R(x) = 1 + \sum_{y \in \alpha(x)} S(y) \quad \text{برای تمامی } x \in P$$
$$\alpha(x) = \{y \in A : y \succ x\} \quad \text{که در آن} \qquad (۴۶-۲۰)$$

اضافه نمودن ۱ در معادله‌ی بالا صرفاً جهت اطمینان از $R(x) \geq 1$ بوده که این خود باعث اطمینان از $R(x) > S(\alpha)$ برای تمامی $x \in P$ و تمامی $\alpha \in A$ می‌شود. توجه داشته باشید که اگر x دارای هزینه‌ی خام کمی باشد، آنگاه x یک ذره با عملکرد خوب تلقی خواهد شد[1].

شکل ۲۰-۲۰ محاسبات توان و هزینه‌ی خام را برای آرشیو با اندازه‌ی $|A| = 3$ و جمعیت با اندازه‌ی $|P| = 6$ نشان می‌دهد. در این شکل مقادیر توان نقاط مرز پرتو به‌صورت تعداد نرمالیزه‌ی ذراتی که مغلوب آن‌ها هستند، نشان داده شده است. همچنین، در این شکل هزینه‌ی خام هر ذره به‌صورت جمع عدد ۱ و تعداد نقاط مرز پرتوی غالب بر آن‌ها نشان داده شده است. توجه داشته باشید که هر چه ذره‌ای از مرز پرتو دورتر باشد، مقدار هزینه‌ی خام آن بزرگتر است (بدین معنی که ذره توسط تعداد بیشتری از نقاط مرز پرتو مغلوب شده است). همچنین، به ذره‌ی مغلوب شده در گوشه‌ی بالا-چپ با هزینه‌ی خام 9/7 توجه نمایید و آن را با دو ذره‌ی مغلوب در گوشه‌ی پایین-راست با هزینه‌ی خام 10/7 مقایسه نمایید. ذرات موجود در گوشه‌ی پایین-راست به دلیل این که در ناحیه‌ی پرازدحام‌تری از فضای تابع واقع شده‌اند، دارای هزینه‌ی خام بیشتری هستند. از آنجا که این نقاط در ناحیه‌ی پرازدحامی واقع شده‌اند، توان نقاط مرز پرتویی که آن‌ها را مغلوب می‌سازند بیشتر بوده و به همین دلیل هزینه‌ی خام آن‌ها افزایش می‌یابد.

[1] در مقالات مربوط به *SPEA* از $R(x)$ با عنوان برازندگی خام یاد می‌شود، اما ما در اینجا آن را با نام هزینه‌ی خام معرفی کردیم تا نام آن با این حقیقت که $R(x)$ کمتر بهتر و $R(x)$ بیشتر بدتر است، در یک راستا باشد.



شکل ۲۰-۲۰ نحوه‌ی محاسبه‌ی توان‌ها و هزینه‌های خام در SPEA برای یک مسئله‌ی بهینه‌سازی دو هدفه. نقاط مرز پرتو با سمبل × نشان داده شده و مقدار توانشان در کنار هر یک نوشته شده است. ذراتی که از نقاط مرز پرتو نیستند نیز با دایره‌های توپر نمایش داده شده و مقدار هزینه‌ی خام هر یک در کنارش نوشته شده است.

همان‌طور که پیش از این نیز به آن اشاره شد، در هر نسل تمامی ذرات موجود در $\{P, A\}$ که نامغلوب هستند به آرشیو A اضافه می‌گردند. با این حال، این موضوع ممکن است باعث افزایش نامحدود آرشیو شود. در SPEA این مشکل بالقوه با یک روش خوشه‌بندی برطرف می‌گردد [زیتزلر و ثیل، ۱۹۹۹]. همان‌طور که می‌دانید آرشیو دارای $|A|$ ذره است. ابتدا هر ذره به‌صورت یک خوشه تعریف می‌گردد. سپس، دو خوشه که از همه به هم نزدیک‌ترند با یکدیگر ادغام شده تا تعداد خوشه‌های A یک واحد کاهش یابد. این کار را آنقدر ادامه می‌دهیم تا تعداد خوشه‌های آرشیو به N_A برسد. N_A اندازه‌ی مطلوب آرشیو است. در آخر، تنها یک نقطه از هر خوشه حفظ می‌شود. این نقطه معمولاً نقطه‌ای است که به مرکز خوشه نزدیکتر است.

SPEA2

SPEA2 نسخه‌ای بهبود یافته از SPEA است [زیتزلر و همکاران، ۲۰۰۱]. ابتدا، نه تنها به ذرات موجود در آرشیو، بلکه به تمام ذرات موجود در جمعیت یک مقدار توان $S(\alpha)$ اختصاص می‌دهیم:

$$\alpha \in \{P, A\} \text{ برای تمامی } S(\alpha) = \left|x \in \{P, A\} \text{ به طوری که } \alpha \succ x\right| \qquad (۴۷-۲۰)$$

با مقایسه‌ی معادله‌ی بالا با معادله‌ی (۲۰-۴۵) می‌توان دریافت که در اینجا مقادیر توان را نرمالیزه نمی‌کنیم.

در گام بعد، هزینه‌ی خام هر تمامی ذرات موجود در P را با کمی تغییر و با محاسبه‌ی مجموع توان‌های ذرات مغلوب‌کننده، هم در جمعیت و هم در آرشیو، به دست می‌آوریم:

$$x \in P \text{ برای تمامی } R(x) = \sum_{y \in \alpha(x)} S(y) \qquad (۴۸-۲۰)$$

که در آن $\alpha(x) = \{y \in \{P, A\} : y \succ x\}$

با مقایسه‌ی معادله‌ی بالا با معادله‌ی (۲۰-٤٦) نیز می‌توان دید که در معادله‌ی بالا عدد ۱ را به هزینه‌ی خام اضافه نمی‌نماییم.

سپس، هزینه‌ی خام هر $x \in P$ را بر اساس این که چند ذره در نزدیک آن قرار دارد، اصلاح می‌نماییم. این بدین معنی است که هزینه‌ی ذراتی را که نزدیک تعداد زیادی ذره‌ی دیگر در فضای تابع قرار دارند، مجازات می‌نماییم. این کار را با پیدا کردن فاصله‌ی بین $f(x)$ و $f(y)$ برای تمامی $x \in P$ و $y \in \{P, A\}$ $(x \neq y)$، انجام می‌دهیم. این فاصله می‌تواند از هر نُرم برداری که کاربر آن را مناسب می‌یابد، به دست آید. اما معمولاً برای این منظور از فاصله‌ی اقلیدسی استفاده می‌شود. برای هر $x \in P$ فاصله‌ی بین آن و هر $y \in \{P, A\}$ را به‌صورت افزایشی مرتب می‌نماییم تا برای هر x یک لیست فاصله‌ی مرتب شده با $(|P| + |A|)$ عنصر به دست آید. سپس، jاُمین عنصر لیست فاصله را، که فاصله‌ی بین x و jاُمین همسایه‌ی آن (به لحاظ نزدیکی) نشان می‌دهد، انتخاب کرده و با $\sigma_j(x)$ نمایش می‌دهیم. برای انتخاب j می‌توان از استراتژی‌های مختلفی استاده نمود. برای مثال، برخی محققین با $j = \sqrt{|P| + |A|}$ به نتایج خوبی دست یافته‌اند اما برخی دیگر به سادگی از $j = 1$ استفاده نموده‌اند [زیتزلر و همکاران، ۲۰۰٤]. چگالی x را به‌صورت زیر تعریف می‌نماییم

$$D(x) = \frac{1}{\sigma_j(x) + \gamma} \qquad (۲۰-٤۹)$$

در معادله‌ی بالا مقدار γ در مخرج کسر را به‌گونه‌ای انتخاب می‌نماییم که $D(x) < 1$ شود. در مقاله‌ی اصلی SPEA 2 مقدار γ برابر ۲ پیشنهاد شده است. در آخر، مقدار هزینه‌ی اصلاح شده‌ی x با اضافه نمودن هزینه‌ی خام به چگالی به دست می‌آید:

$$C(x) = R(x) + D(x) \qquad (۲۰-۵۰)$$

از آنجایی که تمامی ذرات نامغلوب دارای هزینه‌ی خام ۰ هستند، و از آنجا که برای تمامی xها، $D(x) < 1$ است، می‌توان دید که تمامی ذرات نامغلوب دارای هزینه‌ی $C(x) < 1$ هستند.

یکی دیگر از اصلاحات اعمال شده در SPEA2 شامل کنترل نمودن اندازه‌ی آرشیو می‌شود. در SPEA برای اندازه‌ی آرشیو حد پایینی وجود ندارد، اما در SPEA2 اندازه‌ی آرشیو در یک مقدار معین نگه داشته می‌شود. اگر در طول فرایند SPEA2 اندازه‌ی آرشیو زیادی کوچک شود، ذرات کم هزینه از جمعیت را، هر چند که مغلوب باشند، به آن اضافه کرده تا اندازه‌ی آن به مقدار مطلوب برسد. در SPEA از روش خوشه‌بندی جهت کنترل حد بالای اندازه‌ی آرشیو استفاده می‌شود. در SPEA2 اگر اندازه‌ی آرشیو از میزان معینی بزرگتر

شود، ذراتی را با پیدا کردن فاصله‌ی هر $x \in A$ تا نزدیکترین همسایه‌اش در فضای تابع هدف، حذف می‌نماییم:

$$x \in A \text{ برای } D_{min}(x) = \min_{y}[\sum_{i=1}^{k}\left(f_i(x) - f_i(y)\right)^2 \text{ که در آن } y \in A \text{ و } y \neq x] \qquad (۵۱-۲۰)$$

با این کار تعداد $|A|$ مقدار از $D_{min}(x)$ به دست خواهد آمد. سپس از D برای نشان دادن مجموعه‌ای از ذرات که دارای کمترین مقدار $D_{min}(x)$ هستند استفاده می‌نماییم:

$$D = \left\{x : D_{min}(x) \leq D_{min}(y) \text{ برای } y \in A\right\} \qquad (۵۲-۲۰)$$

از آنجا که فاصله‌ی از ذره‌ی x تا ذره‌ی y با فاصله از ذره‌ی y تا x برابر است، بنابراین حداقل دو ذره در D وجود خواهد داشت. از میان تمام ذراتی که در D وجود دارند، آنی را که به هر ذره از آرشیو (که در D وجود ندارد) نزدیکتر است، انتخاب کرده و با x_{min} نمایش می‌دهیم:

$$x_{min} = arg\min_{x}[\min_{y}\sum_{i=1}^{k}\left(f_i(x) - f_i(y)\right)^2 \text{ که در آن } y \in A \text{ و } y \notin D] \qquad (۵۳-۲۰)$$

x_{min} را از آرشیو حذف کرده و بدین ترتیب اندازه‌ی آرشیو به اندازه‌ی یک واحد کاهش خواهد یافت. اگر $|A|$ خیلی بزرگ باشد، معادلات (۲۰-۵۱) تا (۲۰-۵۳) را تکرار کرده و هر بار یک ذره را حذف می‌نماییم تا به اندازه‌ی مطلوب $|A|$ برسیم.

آخرین اصلاح اعمال شده در SPEA2 آن است که تنها اعضای A جهت فرایند انتخاب و بازترکیب برای تولید جمعیت در نسل بعد در نظر گرفته می‌شوند. مقالات بسیاری تنوعات و اصلاحات بسیاری را برای SPEA و SPEA2 ارائه نموده‌اند اما شکل ۲۰-۲۱ الگوریتم بنیادین را نشان می‌دهد.

شکل ۲۰-۲۱ شامل اصول SPEA است اما جزییات بسیاری را به خلاقیت محقق واگذار می‌نماید. در اینجا به برخی از نکات مربوط به اصلاحات ممکن اشاره می‌نماییم.

- خواننده باید اندازه‌ی جمعیت و اندازه‌ی N_A را انتخاب نماید. معمولاً $N_A < N$ است.
- عبارت "*ذرات غیرمغلوب را از* A *به* P *کپی کن*" نشان‌دهنده‌ی آن است که تمامی ذرات $x \in P$ باید با تمامی ذرات $y \in \{P, A\}$ مقایسه شوند. هر ذره مانند x که توسط هیچ یک از yها مغلوب نشده باشد به A کپی خواهد شد. با این حال، این عبارت به این سؤال که آیا ذرات نامغلوب باید از P حذف شوند یا خیر پاسخی نمی‌دهد. از آنجایی که ذرات نامغلوب در A قرار دارند، شاید منطقی نباشد که کپی آن‌ها در P را نگه داشت. اما این خود سؤال دیگری را در مورد اندازه‌ی جمعیت P به

وجود می‌آورد. اگر ذرات نامغلوب را از P حذف نماییم، آیا باید ذرات دیگری را جایگزین آن‌ها نماییم؟ می‌توان P را در یک حالت کاهش یافته رها نمود و همیشه و بی‌توجه به اندازه‌ی P، N فرزند ایجاد نمود. همچنین می‌توان ذراتی که از P حذف می‌شوند را با ذرات اتفاقی جایگزین نمود.

اندازه جمعیت $= N$

بیشترین اندازه آرشیو $= N_A$

جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن: $P = \{x_j\}$ برای $j \in [1, N]$

مجموعه‌ی آرشیو A را با مجموعه‌ی تهی مقداردهی اولیه کن

تا زمانی که شرایط توقف برآورده نشده است

ذرات نامغلوب را از P به A کپی کن

$A \leftarrow \{A \cup \{x \in P : \nexists(y \in \{P, A\} : y \succ x)\}$

ذرات مغلوب را از A حذف کن

تا زمانی که $|A| > N_A$

از یک روش خوشه‌بندی (SPEA) و یا معادلات (۲۰-۵۱)-(۲۰-۵۳) (SPEA2) برای حذف کردن ذرات از A استفاده کن

اگر از SPEA2 استفاده شد، آنگاه

تا زمانی که $|A| < N_A$

ذره‌ی غیرتکراری با کمترین هزینه را از P به A اضافه کن:

$A \leftarrow \{A, (arg\min_x C(x)$ به طوری که $x \in P, x \notin A)\}$

پایان اگر

از معادله‌ی (۲۰-٤٦) (SPEA)یا معادله‌ی (۲۰-۵۰) (SPEA2) جهت محاسبه‌ی هزینه‌ی ذرات موجود در P استفاده کن

والدین را از $\{P, A\}$ (SPEA) و یا از S (SPEA2) انتخاب کن

از یک روش بازترکیب برای تولید فرزند C از والدین استفاده کن

فرزند C را به‌صورت احتمالاتی دچار جهش کن

از یک روش جایگزینی جهت جایگزین نمودن ذرات درون P با ذرات C استفاده کن

نسل بعد

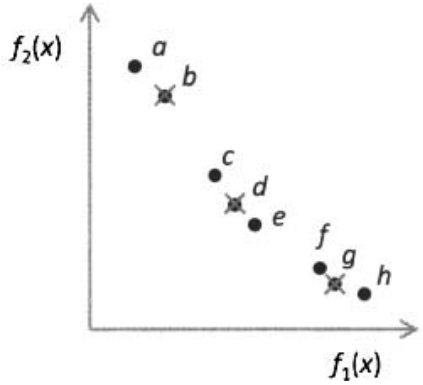
شکل ۲۰-۲۱ طرح کلی الگوریتم تکاملی استحکام پرتو (SPEA و SPEA2).

- حلقه‌ی "while $|A|>N_A$" در صورت بزرگ شدن بیش از حد آرشیو، ذرات را از نواحی پرازدحام فضای تابع هدف حذف می‌نماید. SPEA و SPEA2 هر یک این کار را به روش خود انجام می‌دهند. بی‌شک خواننده می‌تواند روش‌های دیگری را نیز برای خود پیدا کرده و آزمایش کند.
- حلقه‌ی "while $|A|<N_A$" ذرات کم هزینه را در صورت کوچک بودن بیش از حد آرشیو، به آن اضافه می‌نماید. این کار تنها برای SPEA2 صورت می‌پذیرد. شدر SPEA این گام حذف شده و $|A|$ دارای حد پایین نخواهد بود.
- عبارت "*والدین را از* $\{P, A\}$ *(*SPEA*)، یا از* A *(*SPEA2*) انتخاب کن*" جای زیادی را برای انعطاف باقی می‌گذارد. می‌توان از هر نوع روش انتخابی برای این گام استفاده نمود (بخش ۸–۷ را ببینید). موفقیت SPEA و SPEA2 به میزان زیادی به پیاده‌سازی این گام بستگی دارد.
- عبارت "*از یک روش بازترکیب برای تولید فرزندان از والدین استفاده کن*" نیز جای زیادی را برای انعطاف‌پذیری باقی می‌گذارد. می‌توان از هر یک از الگوریتم‌های تکاملی بحث شده در این کتاب و هر نوع روش بازترکیبی جهت تولید فرزندان استفاده نمود (بخش ۸–۸ را ببینید). همانند انتخاب، موفقیت SPEA و SPEA2 قویا به پیاده‌سازی بازترکیب وابسته است.
- عبارت "*از یک روش جایگزینی برای جایگزین نمودن ذرات در* P *با ذرات در* C *استفاده کن*" را می‌توان به چند روش متفاوت پیاده‌سازی نمود. اگر $|C| = |P|$، می‌توان به سادگی P را با C جایگزین نمود. اگر $|C| < |P|$ می‌توان P را با بهترین N ذره از مجموعه‌ی $C \cup P$ جایگزین نمود. همچنین می‌توان برای انتخاب بهترین N ذره، از نوعی الگوریتم متناسب با برازندگی استفاده نمود. اگر $|C| > |P|$، می‌توان بهترین N ذره را از C، یا از $C \cup P$ و از هر دوی این مجموعه‌ها و با استفاده از یک الگوریتم متناسب با برازندگی، انتخاب نمود.

مثال ۲۰–۷

این مثال نحوه‌ی هرس شدن آرشیو در SPEA2، مانند آنچه که در معادلات (۲۰–۵۱)–(۲۰–۵۳) نشان داده شد، را نمایش می‌دهد. شکل ۲۰–۲۲ آرشیوی از ذرات نامغلوب در یک فضای دو بعدی تابع هدف نشان می‌دهد. فرض کنید می‌خواهیم اندازه‌ی آرشیو را از اندازه‌ی هشت (که در شکل نشان داده شده است) به پنج تقلیل دهیم. ابتدا ذراتی را که از همه به هم نزدیکتر هستند را می‌یابیم. این ذرات، دو ذره‌ی f و g می‌باشند. از بین این دو ذره، g را حذف می‌نماییم چرا که فاصله‌ی g تا نزدیکترین همسایه‌اش بعد از f (ذره‌ی h)، کمتر از فاصله‌ی f تا نزدیکترین همسایه‌اش بعد از g (ذره‌ی e) است. حال که g را حذف

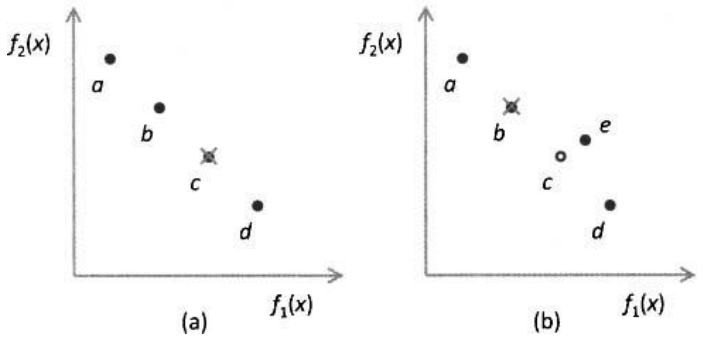
نموده‌ایم، بار دیگر دو ذره‌ی که از بقیه به یکدیگر نزدیکتر هستند را می‌یابیم. این دو ذره، ذرات d و e خواهند بود. همانگونه که g را حذف نمودیم، در اینجا نیز با همان منطق ذره‌ی d را حذف می‌نماییم. پس از حذف d بار دیگر دو ذره‌ای را که از بقیه به یکدیگر نزدیکتر هستند (ذرات a و b) را یافته و در اینجا نیز به همان روال پیشین، ذره‌ی b را حذف می‌نماییم. حال اندازه‌ی آرشیو به مقدار مطلوب یعنی پنج رسیده است. توجه داشته باشید که این روش همواره ذرات اکسترمم را در آرشیو نگه می‌دارد (در این مثال ذرات اکسترمم a و h می‌باشند).



شکل ۲۰-۲۲ مثال ۲۰-۷: این شکل نحوه‌ی کاهش اندازه‌ی آرشیو ذرات نامغلوب در SPEA2 را نشان می‌دهد. در این شکل، که از [زیتزلر و همکاران، ۲۰۰٤] اقتباس شده است، اندازه‌ی آرشیو با حذف ذرات g، d و b از هشت به پنج تقلیل یافته است.

مثال ۲۰-۸

رویکرد SPEA2 برای کاهش اندازه‌ی آرشیو ممکن است باعث بدتر شدن تخمین مرز پرتو شود. شکل ۲۰-۲۳ این موضوع را نشان می‌دهد. شکل ۲۰-۲۳(الف) یک تخمین مرز پرتو شامل چهار نقطه را نشان می‌دهد. اگر بخواهیم اندازه‌ی آرشیو را برابر سه نگه داریم، ذره‌ی c را حذف خواهیم نمود چرا که این ذره در پرازدحام‌ترین ناحیه واقع شده است. با این حال، چند نسل بعد، الگوریتم تکاملی ممکن است راه‌حل نامغلوب e را یافته و آن را به آرشیو اضافه نماید (شکل ۲۰-۲۳(ب)). حال، چون ذره‌ی b در پرازدحام‌ترین ناحیه‌ی شکل ۲۰-۲۳(ب) واقع شده است، SPEA2 ذره‌ی b را از آرشیو حذف کرده و ذره‌ی e را در آرشیو نگه خواهد داشت. به‌صورت بصری می‌توان دید از آنجا که ذره‌ی c بر ذره‌ی آرشیوشده‌ی e غلبه دارد، باید آن را در جمعیت نگه می‌داشتیم. این موضوع به این نکته اشاره دارد که اگرچه روش فاصله‌ی SPEA2 روشی مناسب برای هرس نمودن مجموعه‌ای از ذرات می‌باشد، شاید بهتر باشد هیچگاه هیچ ذره‌ی نامغلوبی را حذف ننماید [زیتزلر و همکاران، ۲۰۰٤].



شکل ۲۰-۲۳ مثال ۲۰-۸: این مثال نشان می‌دهد که چگونه ممکن است فرایند هرس نمودن آرشیو، سهواً به تخمین مرز پرتو آسیب برساند. ذره‌ی c به دلیل واقع شدن در ناحیه‌ی پرازدحام از آرشیو حذف شده و ذره‌ی e به آرشیو اضافه شده است. این در صورتی است که اگر ذره‌ی c حفظ می‌شد، بر ذره‌ی e غلبه می‌نمود.

در SPEA، SPEA2 و هر MOEA دیگری که شامل یک آرشیو باشد، می‌توان به چند طریق از آرشیو استفاده نمود. روش اول، استفاده از آرشیو برای ذخیره نمودن راه‌حل‌های نامغلوب است. این رویکرد بالاترین سطح جداسازی میان جمعیت MOEA و آرشیو آن را فراهم می‌آورد. روش دوم، کپی نمودن ذرات آرشیو به جمعیت فرزندان در انتهای هر نسل است. این رویکرد سطحی از تعامل میان جمعیت و آرشیو را فراهم می‌آورد. در روش سوم می‌توان به آرشیو اجازه‌ی شرکت در فرایند انتخاب را داد تا بدین ترتیب، نه تنها جمعیت، بلکه آرشیو نیز در فرایند بازترکیب مشارکت نماید. این رویکرد که توسط SPEA و SPEA2 مورد استفاده واقع می‌شود، بیشترین میزان تعامل میان آرشیو و جمعیت را فراهم می‌آورد.

۲۰-۴-۷ استراتژی تکاملی پرتو آرشیو شده (PAES)

PAES در [نولز[1] و کورن[2]، ۲۰۰۱] معرفی گردید. این الگوریتم از استراتژی تکاملی $(1 + 1)$ (فصل ٦ را ببینید) الهام گرفته است. در هر نسل، یک ذره جهش یافته و یک فرزند را به وجود می‌آورد. هر بار که یک والد یک فرزند را ایجاد می‌کند، در صورتی که فرزند مغلوب هیچ یک از ذرات موجود در آرشیو نباشد، فرزند را به آرشیو اضافه می‌نماییم. اگر اندازه‌ی آرشیو از یک آستانه‌ی معین تجاوز کند، آرشیو را با حذف ذراتی که دارای کوچکترین فاصله‌ی ازدحام هستند، هرس می‌نماییم (این بدین معنی است که ذراتی که در پرازدحام‌ترین نواحی از فضای تابع هدف واقع شده‌اند را حذف می‌نماییم). PAES اولیه از صفحه‌ی مشبک در فضای تابع هدف برای محاسبه‌ی فاصله‌ی ازدحام استفاده می‌نماید. این ایده مانند بسته‌های ϵ در

[1] Knowles
[2] Corne

ϵ–MOEA می‌باشد (بخش ۲۰–٤–۲ را ببینید). این روش همچنین مشابه رویکرد [پارکس[1] و میلر[2]، ۱۹۹۸]، که در آن ذرات به آرشیو اضافه نمی‌شوند مگر به میزان کافی متفاوت از ذرات کنونی آرشیو باشند، می‌باشد. شکل ۲۰–۲٤ طرح کلی الگوریتم اصلی PAES را نشان می‌دهد.

```
حد بالای اندازه‌ی آرشیو = N_A
جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن: P = {x_j} برای j ∈ [1, N]
مجموعه‌ی آرشیو A را با مجموعه‌ی تهی مقداردهی اولیه کن
تا زمانی که شرایط توقف برآورده نشده است
    یک والد مانند x را از P انتخاب کن
    x را دچار جهش کن
    اگر x توسط هیچ یک از ذرات درون A مغلوب نشده است، آنگاه
        x را به A اضافه کن
    پایان اگر
    اگر |A| > N_A، آنگاه
        فاصله‌ی ازدحام s(α) را برای تمامی α ∈ A محاسبه کن
        α_min ← arg min_α s(α)
        α_min را از A حذف کن
    پایان اگر
نسل بعد
```

شکل ۲۰–۲٤ طرح کلی استراتژی تکاملی پرتو آرشیو شده (PAES). $s(\alpha)$ فاصله‌ی ازدحام α بوده و برای ذراتی که در نواحی پرازدحام فضای تابع می‌باشند، کوچک می‌باشد.

[1] Parks
[2] Miller

۲۰–۵ بهینه‌سازی چندهدفه زیست‌جغرافی–محور

این بخش نشان می‌دهد که چگونه می‌توان بهینه‌سازی زیست‌جغرافی–محور، که در فصل ١٤ مورد بحث قرار گرفت، را با برخی از MOEAهای بحث شده در این فصل ترکیب نمود. ترکیب رویکرد BBO با رویکردهای متنوع MOEA چندین الگوریتم بهینه‌سازی چندهدفه‌ی زیست‌جغرافی–محور (MOBBO[1]) را به وجود خواهد آورد. سپس مطالعه‌ای مقایسه‌ای را میان الگوریتم‌های MOBBO بر روی برخی مسائل محک چندهدفه ارائه خواهیم نمود. از این بخش می‌توان به‌عنوان الگویی برای تعمیم سایر الگوریتم‌های تکاملی به بهینه‌سازی چندهدفه استفاده نمود.

۲۰–۵–۱ BBO برداری سنجیده شده

در این بخش به بحث در مورد ترکیب BBO با VEGA (بخش ۲۰–۳–۲ را ببینید)، می‌پردازیم. به یاد آورید که شکل ۲۰–۸ الگوریتم VEGA برای یک مسئله‌ی بهینه‌سازی با k هدف را نشان می‌دهد. از آنجا که BBO بر پایه‌ی مهاجرت قرار دارد، مبنای مهاجرت (به) BBO چندهدفه را بر مقدار تابع هدف k_iاُم هر ذره قرار می‌دهیم. k_i یک اندیس تابع هدف اتفاقی در iاُمین مهاجرت می‌باشد. سپس، مبنای مهاجرت (از) را بر مقدار تابع هدف k_eاُم قرار می‌دهیم. k_e نیز یک اندیس تابع هدف اتفاقی در iاُمین مهاجرت می‌باشد. با این کار الگوریتم بهینه‌سازی زیست‌جغرافی–محور برداری سنجیده (VEBBO[2]) به دست خواهد آمد. این الگوریتم در شکل ۲۰–۲۵ نشان داده شده است.

جمعیتی از راه‌حل‌های نامزد را مقداردهی اولیه کن: $P = \{x_j\}$ برای $j \in [1, N]$
تا زمانی که شرایط توقف برآورده نشده است
  هزینه‌ی $f_i(x_j)$ را برای هر هدف و برای هر ذره‌ی $x_j \in P$ محاسبه کن
  رتبه‌ی x_j نسبت به تابع هدف $r_{ji} \leftarrow i$، $j \in [1, N]$ و $i \in [1, k]$
  $\lambda_{ji} \leftarrow \frac{r_{ji}}{\sum_{q=1}^{N} r_{qi}}$ برای $j \in [1, N]$ و $i \in [1, k]$
  $\mu_{ji} \leftarrow 1 - \lambda_{ji}$، برای $j \in [1, N]$ و $i \in [1, k]$
  برای هر ذره‌ی x_j، $j \in [1, N]$
    برای هر متغیر مستقل $s \in [1, n]$

[1] Multi-Objective Biogeography-Based Optimization
[2] Vector Evaluated Biogeography-Based Optimization

عدد صحیح اتفاقی با توزیع یکنواخت $= k_i \leftarrow rand(1,k)$
$r \leftarrow rand(0,1)$
اگر $r < \lambda_{j,k_i}$ آنگاه عملیات مهاجرت (به) را انجام بده
$k_e \leftarrow rand(1,k)$
ذره‌ی مهاجرت کننده x_e را به‌صورت احتمالاتی انتخاب کن که در آن
$\Pr(x_e = x_m) = \mu_{m,k_e} / \sum_{q=1}^{N} \mu_{q,k_e}$ برای $m \in [1,N]$
$x_j(s) \leftarrow x_e(s)$
پایان مهاجرت (به)
متغیر مستقل بعدی
ذره‌ی x_j بعدی
جمعیت را به‌صورت احتمالاتی دچار جهش کن
نسل بعد
عناصر نامغلوب $P \leftarrow \hat{P}_s$

شکل ۲۰–۲۵ طرح کلی بهینه‌سازی زیست‌جغرافی–محور برداری سنجیده شده (VEBBO) برای حل یک مسئله‌ی بهینه‌سازی n بعدی با k هدف. در هر نسل، بهترین ذره x_b نسبت به iاُمین مقدار هدف دارای رتبه‌ی $r_{bi} = 1$ بوده و بدترین ذره x_w دارای رتبه‌ی $r_{wi} = N$ خواهد بود.

۲۰–۵–۲ BBO مرتب‌کننده‌ی نامغلوب

حال به بحث در مورد ترکیب نمودن BBO با NSGA (بخش ۲۰–٤–۳ را ببینید) می‌پردازیم. به یاد آورید که شکل ۲۰–۱٦ الگوریتم NSGA را ارائه می‌نماید. برای اصلاح شکل ۲۰–۱٦ برای ترکیب با BBO تنها باید عبارت بازترکیب را تغییر دهیم و آن را به یک عملیات مهاجرت تبدیل نماییم. با این کار الگوریتم بهینه‌سازی زیست‌جغرافی–محور مرتب‌کننده‌ی نامغلوب (NSBBO[1]) به دست خواهد آمد که در شکل ۲۰–۲٦ نشان داده شده است.

[1] Nondominated Sorting Biogeography-Based Optimization

نرخ مهاجرت (به) $\lambda_j \leftarrow \frac{\phi(x_j)}{\sum_{q=1}^{N}\phi(x_q)}$ برای $j \in [1,N]$
نرخ مهاجرت (از) $\mu_j \leftarrow 1-\lambda_j$ برای $j \in [1,N]$
برای هر ذره‌ی x_j برای $j \in [1,N]$
برای هر متغیر مستقل $s \in [1,n]$
$r \leftarrow rand(0,1)$
اگر $r < \lambda_j$، آنگاه
ذره‌ی مهاجرت کننده‌ی x_e را به‌صورت احتمالاتی انتخاب کن به طوری که
$\Pr(x_e = x_m) = \frac{1}{N}$ برای $m \in [1,N]$
$x_j(s) \leftarrow x_e(s)$
پایان مهاجرت
متغیر مستقل بعدی
ذره‌ی بعدی
جمعیت فرزندان $\leftarrow \{x_j\}$

شکل ۲۰-۲۶ طرح کلی بهینه‌سازی زیست‌جغرافی-محور مرتب‌کننده‌ی نامغلوب (NSBBO) برای حل یک مسئله‌ی بهینه‌سازی nبعدی با k هدف و با اندازه‌ی جمعیت N.

۲۰-۵-۳ BBO پرتو نیچه

در این بخش روشی ساده برای ترکیب BBO با NPGA (بخش ۲۰-٤-۵ را ببینید)، ارائه خواهد شد. به یاد آورید که شکل ۲۰-۱۹ الگوریتم NPGA را نشان می‌دهد. همانند الگوریتم NSBBO از بخش قبل، جهت اصلاح شکل ۲۰-۱۹ برای BBO تنها باید عبارت بازترکیب "*ذرات درون R را بازترکیب کن*" را با عملیات مهاجرت BBO جایگزین نماییم. از آنجایی که در NPGA ذرات موجود در R بر اساس میزان نامغلوب بودنشان انتخاب می‌شوند، می‌توان به سادگی از نرخ‌های مهاجرت یکسان برای هر ذره‌ی موجود در R استفاده نمود. با این کار، الگوریتم بهینه‌سازی زیست‌جغرافی-محور نیچه (NPBBO[1]) به دست خواهد آمد. این الگوریتم در شکل ۲۰-۲۷ نشان داده شده است.

در اینجا لازم به ذکر است که می‌توان الگوریتم‌های MOBBO از این بخش را با تمام انواع ممکن BBO از فصل ١٤ ترکیب نمود. برای مثال، می‌توان از نوع مهاجرت (از) BBO به جای مهاجرت (به) استفاده نمود.

[1] Niched Pareto BBO

همچنین می‌توان از منحنی‌های مهاجرت غیرخطی و یا از مهاجرت‌های مخلوط استفاده نمود. از سویی، می‌توان به جای انتقال یک متغیر مستقل در هر زمان، گروهی از متغیرهای مستقل را در هر زمان مهاجرت داد (بخش ١٤–٥ را ببینید). می‌توان از جمعیتی موقتی برای مهاجرت استفاده نمود تا نیازی به تغییر ذرات مهاجرت کننده پیش از تکمیل تمام مهاجرت‌ها وجود نداشته باشد. به طور کلی می‌توان تمامی انواع، تعامیم و ترکیبات BBO، که در مقالات و کتاب‌ها معرفی شده‌اند، را با رویکردهای MOBBO معرفی شده در این بخش ترکیب نمود تا الگوریتم‌های MOBBO متنوع و بسیاری به دست آیند. این کار را با استفاده هر یک از الگوریتم‌های تکاملی دیگری که در این کتاب مورد بحث واقع شده‌اند نیز می‌توان انجام داد. یک زمینه‌ی بسیار مفید برای تحقیقات آتی، تعمیم الگوریتم‌های تکاملی جدید، که به تازگی معرفی شده‌اند از جمله آن‌هایی که در فصل ۱۷ معرفی گردیدند، به بهینه‌سازی چندهدفه می‌باشد.

برای هر ذره‌ی $x_j \in R$ که در آن $j \in [1, N]$

برای هر متغیر مستقل $s \in [1, n]$

$r \leftarrow rand(0,1)$

اگر $r < 1/N$، آنگاه

ذره‌ی مهاجرت کننده‌ی x_e را به‌صورت احتمالاتی انتخاب کن به طوری که

$\Pr(x_e = x_m) = 1/N$ برای $m \in [1, N]$

$x_j(s) \leftarrow x_e(s)$

پایان مهاجرت

متغیر مستقل بعدی

ذره‌ی بعدی

$\{x_j\}$ ← جمعیت فرزندان

شکل ۲۰–۲۷ طرح کلی الگوریتم بهینه‌سازی زیست‌جغرافی–محور پرتو نیچه (NPBBO) برای حل یک مسئله‌ی بهینه‌سازی n بعدی با k هدف و با اندازه‌ی جمعیت N. این شبه کد جایگزین عبارت بازتولید در شکل ۲۰–۱۹ می‌شود.

۲۰–۵–۴ BBO استحکام پرتو

این بخش روشی را برای ترکیب BBO با SPEA یا SPEA2 (بخش ۲۰–٤–٦ را ببینید)، ارائه می‌دهد. به یاد آورید که شکل ۲۰–۲۱ الگوریتم SPEA و SPEA2 را نشان می‌دهد. جهت اصلاح شکل ۲۰–۲۱ برای BBO، باید عبارات "*والدین را انتخاب کن*" و "*از یک روش بازترکیب استفاده کن*" را تغییر دهیم. این کار

را می‌توان با محاسبه‌ی نرخ‌های مهاجرت با استفاده از هزینه‌ی خام از معادله‌ی (۲۰-٤٦) برای SPEA و هزینه‌ی اصلاح شده از معادله‌ی (۲۰-۵۰) برای SPEA2، انجام داد. سپس می‌توان مهاجرت BBO را با استفاده از این نرخ‌ها پیاده‌سازی نمود. در این بخش، از رویکرد SPEA، که در آن والدین می‌توانند هم از مجموعه‌ی جمعیت P و هم از مجموعه‌ی آرشیو A انتخاب شوند، استفاده می‌نماییم. با این کار الگوریتم بهینه‌سازی زیست‌جغرافی-محور استحکام پرتو (SPBBO[1]) به دست خواهد آمد. این الگوریتم در شکل ۲۰-۲۸ نشان داده شده است.

از معادله‌ی (۲۰-٤۵) جهت محاسبه‌ی قدرت $S(\alpha)$ برای هر ذره‌ی $\alpha \in A$ استفاده کن
هزینه‌ی $R(\alpha) \leftarrow 1 - S(\alpha)$ را برای هر $\alpha \in A$ محاسبه کن
از معادله‌ی (۲۰-٤٦) جهت محاسبه‌ی هزینه‌ی $R(x)$ برای هر ذره‌ی $x \in P$ استفاده کن
نرخ مهاجرت (به) $\lambda_j \leftarrow \frac{R(x_j)}{\sum_{q=1}^{|P|} R(x_q)}$ برای تمامی $x_j \in P$
نرخ مهاجرت (از) $\mu_j \leftarrow 1 - R(x_j) / \sum_{q=1}^{|P|+|A|} R(x_q)$ برای تمامی $x_j \in \{P, A\}$
برای هر $x_j \in P$
  برای هر متغیر مستقل $s \in [1, n]$
    $r \leftarrow rand(0,1)$
    اگر $r < \lambda_j$، آنگاه
      ذره‌ی مهاجرت کننده‌ی x_e را به‌صورت احتمالاتی انتخاب کن به طوری که
      $\Pr(x_e = x_m) = \mu_m / \sum_{q=1}^{|P|+|A|} \mu_q$ برای $x_m \in \{P, A\}$
      $x_j(s) \leftarrow x_e(s)$
    پایان مهاجرت
  متغیر مستقل بعدی
ذره‌ی بعدی

شکل ۲۰-۲۸ طرح کلی قسمت مهاجرت از الگوریتم بهینه‌سازی زیست‌جغرافی-محور استحکام پرتو (SPBBO) برای حل یک مسئله‌ی بهینه‌سازی با n متغیر مستقل. این شبه کد جایگزین شش خط از شکل ۲۰-۲۱ که با عبارت "*از معادله‌ی (۲۰-٤٦) استفاده کن*" آغاز شده و با عبارت "*از یک روش جایگزینی استفاده کن*" پایان می‌یابد، می‌شود. توجه داشته باشید که مهاجرت (به) در ذرات موجود در P اتفاق افتاده در حالی که مهاجرت (از)، از ذرات موجود در $P \cup A$ اتفاقی می‌افتد.

[1] Strength Pareto BBO

۲۰–۵–۵ شبیه‌سازی‌های BBO چندهدفه

در این زیربخش، نتایج شبیه‌سازی الگوریتم‌های BBO معرفی شده در زیربخش‌های قبل را ارائه می‌دهیم. برای الگوریتم از اندازه‌ی جمعیت ۱۰۰ و محدودیت نسل ۱۰۰۰ استفاده می‌نماییم. همچنین، از جهش یکنواخت متمرکز در مرکز دامنه‌ی جستجو با نرخ ۱٪ در نسل در متغیر مستقل استفاده می‌نماییم (بخش ۸–۹ را ببینید). هر ۱۰۰ نسل یکبار، جمعیت را جهت وجود ذرات تکراری بررسی کرده و ذرات تکراری را با ذرات تولید شده به‌صورت اتفاقی جایگزین می‌نماییم (بخش ۸–٦–۱ را ببینید). در VEBBO، NSBBO و NPBBO نخبه‌گرایی را با بررسی جمعیت برای ذرات نامغلوب پیاده‌سازی می‌نماییم. اگر ذره‌ی نامغلوب را بیابیم، بدترین ذرات موجود در جمعیت از لحاظ نامغلوب بودن را با دو ذره‌ی نامغلوب انتخاب شده به‌صورت اتفاقی از نسل قبل، جایگزین می‌نماییم.

در SPBBO از نخبه‌گرایی استفاده نمی‌کنیم چرا که SPBBO ذرات نامغلوب را در آرشیو ذخیره می‌نماید (شکل ۲۰–۲۸ را ببینید). برای SPBBO، وقتی ذرات نامغلوب را از جمعیت P به آرشیو A انتقال می‌دهیم، آن ذرات P را با تولید شده به‌صورت اتفاقی جایگزین می‌نماییم تا مقدار $|P|$ در مقدار ۱۰۰ باقی بماند. برای $|A|$ حد پایینی وجود ندارد اما مقدار بیشینه‌ی آن را با استفاده از یک الگوریتم ساده‌ی خوشه‌بندی، مانند آنچه که در بخش ۲۰–٤–٦ توضیح داده شد، به مقدار ۱۰۰ محدود می‌نماییم.

چهار الگوریتم MOBBO را بر روی برخی محک‌های چندهدفه‌ی غیرمقید از ضمیمه‌ی ج.۳ آزمایش می‌نماییم. هر یک از این محک‌ها دارای ۱۰ بعد می‌باشند. از محک U01 به دلیل محدب بودن مرز پرتوی آن، از محک U04 به دلیل مقعر بودن مرز پرتوی آن، از U06 به دلیل مرز پرتوی ناپیوسته‌ی آن و از محک U10 به دلیل سه هدفه بودن آن استفاده می‌نماییم (سایر محک‌هایی که آزمایش می‌نماییم دارای دو هدف هستند).

عملکرد الگوریتم‌ها را با دو متریک اندازه‌گیری می‌نماییم: فراتوده نقطه‌ی مرجع S' از معادله‌ی (۲۰–۲۲) و فراتوده نقطه‌ی مرجع نرمالیزه شده S'_n از معادله‌ی (۲۰–۲۳). این متریک‌ها تنوع را لحاظ نمی‌کنند، اما میزان نزدیکی مرز پرتوی تخمینی به مرز پرتوی حقیقی را به لحاظ می‌نمایند. فراتوده نقطه‌ی مرجع S' تعداد نقاط موجود در مرز پرتوی تخمینی را نیز لحاظ می‌نماید.

جدول ۲۰–۱ نتایج را نشان می‌دهد. این نتایج بر روی ۱۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده‌اند. می‌توان دید که در کل SPBBO دارای بهترین عملکرد است. با این حال، برای تابع ناپیوسته‌ی U06، VEBBO دارای بهترین عملکرد به لحاظ مقدار کل فراتوده بوده در حالی که NSBBO دارای بهترین عملکرد به لحاظ فراتوده‌ی نرمالیزه می‌باشد. این بدین معنی است که، VEBBO بهترین ترکیب کمیت و کیفیت مرز پرتو را

می‌یابد، در حالی که NSBBO بهترین کیفیت را می‌یابد. در مورد تابع پیچیده‌تر U10، اگرچه SPBBO بهترین مقدار کل فراتوده را می‌یابد، اما NSBBO کیفیت بهتری از نقاط مرز پرتو را به دست می‌دهد. به‌طور خلاصه، می‌توان گفت که در کل SPBBO به دلیل آرشیوش دارای بهترین عملکرد است، اما تضمینی وجود ندارد که برای هر مسئله‌ی خاص بهترین عملکرد را داشته باشد.

جدول ۲۰-۱ نتایج BBO چندهدفه بر روی توابع هدف ۱۰ بعدی. جدول فراتوده‌ی نسبی و فراتوده‌ی نسبی نرمالیزه شده را برای مهاجرت BBO خطی (اولین عدد در هر زوج مرتب) و مهاجرت سینوسی (دومین عدد در هر زوج مرتب)، نشان می‌دهد. بهترین عملکرد برای هر تابع محک نسبت به فراتوده‌ی نسبی و فراتوده‌ی نسبی نرمالیزه شده به‌صورت اعداد درشت نمایش داده شده است. برای یادآوری بحث مربوط به مقایسه‌ی میان مهاجرت سینوسی و مهاجرت خطی می‌توانید به بخش ۱۴-۴-۱ مراجعه نمایید.

		U01	U04	U06	U10
VEBBO	Hyper	(8.02, 8.93)	(2.73, 2.51)	(**63.53**, 59.99)	(299.68, 444.24)
	Norm	(1.28, 1.98)	(0.19, 0.18)	(21.18, 19.95)	(76.72, 103.07)
NSBBO	Hyper	(5.76, 8.01)	(3.27, 3.51)	(56.51, 48.44)	(373.51, 599.38)
	Norm	(1.26, 1.69)	(0.21, **0.22**)	(**21.94**, 20.09)	(101.40, **118.89**)
NPBBO	Hyper	(9.23, 18.78)	(2.95, 3.05)	(57.92, 48.31)	(419.96, 577.28)
	Norm	(1.18, 2.02)	(0.15, 0.15)	(20.01, 20.09)	(58.65, 57.33)
SPBBO	Hyper	(13.87, **33.10**)	(4.48, **4.60**)	(14.82, 18.33)	(934.29, **3884.32**)
	Norm	(0.90, **2.24**)	(**0.22**, **0.22**)	(3.47, 4.08)	(90.65, 108.08)

۲۰-۶ نتیجه‌گیری

هدف این فصل فراهم آوردن شرحی مفصل از MOEAها نبوده و تنها چند MOEA بسیار مشهور و ایده‌های مرتبط با آن را مورد پوشش قرار داده است. MOEAها بسیار دیگری نیز مطرح شده‌اند و هر روزه نیز MOEAهای جدیدی ارائه می‌گردند. کتاب‌های مرتبط با MOEA عبارتند از [ساکاوا[1]، ۲۰۰۲]، [کولت[2] و سیاری[3]، ۲۰۰۴]، [کوئلو کوئلو و همکاران، ۲۰۰۷]، [دب، ۲۰۰۹] و [تن[4] و همکاران، ۲۰۱۰]. همچنین، رویکردهای تجمع-محور مانند بهینه‌سازی تجمع ذرات (فصل ۱۱ را ببینید)، در حال کسب محبوبیت در زمینه‌ی MOPها می‌باشند [بنکز[5] و همکاران، ۲۰۰۸].

[1] Sakawa
[2] Collette
[3] Siarry
[4] Tan
[5] Banks

[کوئلو کوئلو، ۲۰۰۶] دید تاریخی جالبی از MOEAها به دست می‌دهد. اولین مطالعه‌ی جامع فنی بر روی MOEAها در [فونسسکا و فلمینگ، ۱۹۹۵] انجام شده است. سایر مطالعات جامع را می‌توان در [کوئلو کوئلو، ۱۹۹۹]، [وان ولژویزن و لامونت، ۲۰۰۰]، [زیتزلر و همکاران، ۲۰۰٤] و [کوناک[1] و همکاران، ۲۰۰٦] یافت. اگرچه این مقالات با رشد انفجاری تحقیقاتی که در زمینه‌ی MOEAها انجام می‌گیرد، به سرعت در حال قدیمی شدن هستند، اما هنوز مقالات بسیار مفیدی بوده و دید مناسبی از موارد بنیادین مرتبط با MOEAها به دست می‌دهند.

در این فصل مشاهده نمودیم که تنوع موضوع بسیار مهمی در MOEAها می‌باشد. برخی رویکردها جهت افزایش تنوع شامل به اشتراک‌گذاری برازندگی، شبکه‌ها، خوشه‌بندی، ازدحام کردن، آنتروپی و محدودیت‌های جفت شدن، می‌شود [فونسسکا و فلمینگ، ۱۹۹۵]. همان‌طور که در بخش ۸-٦ نیز بحث شد، تنوع می‌تواند برای بهینه‌سازی تک هدفه نیز بسیار حایز اهمیت باشد. تمام مکانیزم‌های تنوعی که در آن بخش مورد بحث قرار گرفت را می‌توان به MOEAهای این فصل نیز اعمال نمود.

برخی موضوعات مهم جهت تحقیقات آتی در زمینه‌ی MOEA شامل موارد زیر می‌شود.

- تطبیق خودکار آنلاین پارامترهای میزان‌سازی MOEA.
- ترکیب MOEAها با استراتژی‌های جستجوی محلی.
- MOEAهایی که می‌توانند با تعداد ارزیابی‌های تابع برازندگی کم عملکرد مناسبی از خود نشان دهند.
- MOEAها برای چندی هدف (بیش از دو یا سه هدف).
- در نظر گرفتن ترجیح کاربر در MOEAها.
- رویکردهای مفهومی جدید در طراحی MOEAها که به روش‌های رتبه‌بندی پرتو استاندارد بستگی ندارند.
- نظریه‌ی MOEA و مدل‌های ریاضی.

برخی از این موضوعات را در پاراگراف‌های زیر مورد بحث قرار می‌دهیم.

موضوع دوم از لیست بالا که ترکیب استراتژی‌های جستجوی محلی با MOEAها می‌باشد، موضوع بسیار مهمی است. به طور خاص، MOEAها را می‌توان با الگوریتم‌های مشتق-محور (یا دیگر روش‌های جستجوی محلی) ترکیب نمود و بدین ترتیب نتایج بهینه‌سازی را به‌گونه‌ای مناسب میزان نمود. این گونه الگوریتم‌ها را الگوریتم‌های ممتیک[2] می‌گویند، چرا که شامل استفاده از اطلاعات خاص مسئله در الگوریتم ترکیبی می‌شوند.

[1] Konak
[2] Memetic Algorithms

به نظر می‌رسد که از استراتژی‌های ممتیک در بهینه‌سازی تک هدفه بسیار استفاده می‌شود [اونگ[1] و همکاران، ۲۰۰۷]، اما از آن‌ها در MOPها هنوز استفاده نشده است. با این حال استثناهایی نیز وجود دارد [ژاسکیویکز[2] و زیلنیویکز[3]، ۲۰۰۶].

موضوع ارزیابی‌های تابع برازندگی پرهزینه برای MOPها از این جهت حایز اهمیت است که این نوع توابع برازندگی معمولاً در مسائل دنیای واقعی وجود دارند و همچنین به این دلیل که معمولاً MOPها به تعداد ارزیابی‌های تابع برازندگی بسیار بیشتری نسبت به مسائل تک هدفه نیاز دارند. بخش ۲۱-۱ توابع برازندگی پرهزینه را از یک دید کلی مورد بحث قرار می‌دهد، اما تحقیقاتی در مورد الگوریتم‌های تکاملی که مخصوص MOPها با تابع برازندگی پرهزینه طراحی شده‌اند نیز وجود دارند [چافکار[4] و همکاران، ۲۰۰۵]، [اسکندری و گیگر[5]، ۲۰۰۸]، [نولز، ۲۰۰۵]، [سانتانا-کوئینترو[6] و همکاران، ۲۰۱۰]. [گوه[7] و تن، ۲۰۰۷] MOEAها برای مسائل با ارزیابی‌های تابع برازندگی نویزی را مورد بررسی قرار می‌دهد.

طراحی MOEAها برای چندین هدف (بیش از ۱۰ هدف) نیز یک زمینه‌ی تحقیقاتی بسیار مهم برای تحقیقات آتی می‌باشد. برخی نتایج در این زمینه منتشر شده است، اما مسئله‌ی چالش برانگیزتر الزاما تخمین مجموعه‌ی پرتو نبوده، بلکه چگونگی کمک به کاربر (تصمیم‌گیرنده) جهت انتخاب یک راه‌حل از تخمین مجموعه‌ی پرتوی MOEA می‌باشد. برخی تحقیقات در زمینه‌ی مسائل چندهدفه بر چالش‌های خاص تاکید دارند [فلمینگ و همکاران، ۲۰۰۵]، اما تحقیق دیگری نشان می‌دهد که یافتن تخمین مجموعه‌ی پرتو برای مسائل با چندین هدف ساده‌تر است [شوتز[8] و همکاران، ۲۰۱۱]. پس از کمی تفکر می‌توان به‌صورت حسی این موضوع را تایید کرد چرا که هر چه تعداد اهداف در تناقض بیشتر باشد، احتمال این که یک راه‌حل نامزد اتفاقی عملکرد خوبی را برای حداقل یکی از این اهداف به دست دهد، بیشتر خواهد شد. [وان ولژویزن[9] و لامونت، ۲۰۰۰] نشان داده است که هر چه تعداد اهداف یک MOP بیشتر شود، مجموعه‌ی پرتو بزرگتر خواهد شد.

با این حال، اگر چه ممکن است پیدا کردن تخمین مجموعه‌ی پرتو با تعداد اهداف بیشتر ساده‌تر به نظر برسد، اما به راه‌حل‌های نامزد بیشتری احتیاج خواهد داشت. برای مثال، اگر فرض کنیم که ۱۰ راه‌حل نامزد

[1] Ong
[2] Jaszkiewicz
[3] Zielniewicz
[4] Chafekar
[5] Geiger
[6] Santana-Quintero
[7] Goh
[8] Schutze
[9] Van Veldhuizen

می‌تواند تخمین مجموعه پرتوی خوبی برای یک تابع دو هدفه به دست دهد، آنگاه احتمالا به ۱۰۰ ذره در MOEA دو هدفه نیاز خواهیم داشت. این بدین معنی است که ممکن است به 10^k ذره برای یک MOP با k هدف نیاز داشته باشیم. این خود بدین معنی است که برای مثال به ۱۰۰۰۰۰ ذره برای یک MOP چندهدفه‌ی نسبتا کوچک احتیاج خواهیم داشت. بنابراین، مشکل مسائل چندهدفه دشواری نظری تخمین مجموعه‌ی پرتو نبوده و دشوار بودن عملی تلاش محاسباتی و تخمین سطوح با ابعاد بالا و با در اختیار داشتن تنها چند نقطه می‌باشد. [شوتز و همکاران، ۲۰۱۱] مرور بسیار خوبی از تحقیقات جاری بر روی مسائل چندهدفه به دست می‌دهد.

بدین ترتیب به موضوع ترجیح کاربر می‌رسیم. ما معمولاً در هنگام حل یک MOP دارای رجحان‌های از پیش تعریف شده‌ای می‌باشیم. برای مثال، ممکن است برخی از اهداف، از اهداف دیگر دارای اهمیت بیشتری باشند و یا ممکن است برای ترکیبات خاصی از اهداف اهمیت بیشتری قایل باشیم. کاربر همیشه به تمام مجموعه‌ی پرتو علاقه‌مند نیست. اگر بتوانیم به‌گونه‌ای رجحان کاربر را در یک MOEA به کار ببریم، خواهیم توانست تکامل را به سمت ناحیه‌ای از فضای جستجو یا فضای هدف که مطلوب کاربر است هدایت نماییم. بدین ترتیب، چون نیازی به تخمین تمام مجموعه پرتو نخواهد بود، قادر خواهیم بود اندازه‌ی جمعیت MOEA برای مسائل چندهدفه را کاهش دهیم. از آنجا که مسائل چندهدفه معمولاً دارای اهدافی هستند که با یکدیگر دارای همبستگی می‌باشند، یک روش دیگر برای حل مسائل چندهدفه آن است که تعداد اهداف را کاهش دهیم [لوپز جیمیز[1] و همکاران، ۲۰۰۹].

تخمین مجموعه‌ی پرتو به اندازه‌ی کافی سخت است، اما حتی اگر یک الگوریتم تکاملی بتواند تخمین خوبی را به دست آورد، یک انسان چگونه می‌تواند میان مجموعه‌ی بزرگ از راه‌حل‌ها تصمیم‌گیری نماید؟ برخی MOEA که در این کتاب مورد بحث واقع شدند شامل ترجیح کاربر می‌شوند (بخش ۲۰-۳-۱ را ببینید)، اما در این فصل به‌صورت کامل در مورد این موضوع بحث نشده است. اولین تلاش برای دخالت دادن رجحان کاربر در یک MOEA در [تاناکا[2] و تانینو[3]، ۱۹۹۲] ارائه شده است. از آن زمان تا کنون، بسیار دیگری نیز ارائه شده‌اند. برای یک دستیابی به یک مرور خوب از این موضوع می‌توانید به [تیل و همکاران، ۲۰۰۹] مراجعه نمایید.

نتایج نظری برای MOEAها بسیار کم و پراکنده‌اند و فضای بسیار زیادی برای فعالیت در این زمینه وجود دارد. [رودالف و آگاپی، ۲۰۰۰] یک مدل مارکوف اولیه برای MOEAها ارائه نموده است. همچنین چند

[1] Lopez Jaimes
[2] Tanaka
[3] Tanino

محقق دیگر نظریه‌ی MOEAها را مطالعه نموده‌اند [زیتزلر و همکاران، ۲۰۱۰]، اما در مقایسه با الگوریتم‌های تکاملی تک هدفه، مطالعات نظری MOEAها بسیار کم و پراکنده می‌باشد.

در آخر، خوانندگانی که به مطالعه‌ی بیشتر در زمینه‌ی نتایج و تحقیقات MOEA علاقه‌مند باشند باید توجه داشته باشند که کارلوس کوئلو کوئلو مجموعه‌ای مفصل از مقالات و کتب مرتبط با بهینه‌سازی تکاملی چندهدفه را منتشر نموده است. این مجموعه شامل ٤٨٦١ مرجع تا مردادماه ۱۳۹۱ می‌باشد [کوئلو کوئلو، ۲۰۱۲b].

مسائل

تمارین نوشتاری

۲۰-۱ فرض کنید مجموعه‌ای از نقاط و یک MOP در اختیار دارید، آیا این جمله که یک نقطه بر نقاط دیگر غلبه دارد همواره درست است؟

۲۰-۲ آیا هر MOP دارای یک مجموعه‌ی پرتو است؟

۲۰-۳ شکل ۲۰-۱ یک مرز پرتو برای یک MOP که هر دو هدف را مینیمم می‌سازد، نشان می‌دهد، فرض کنید که می‌خواهیم: ۱) f_1 را مینیمم و f_2 را ماکزیمم نماییم، ۲) f_1 را ماکزیمم و f_2 را مینیمم نماییم، ۳) هر دو را ماکزیمم نماییم. در هر سه حالت یک مرز پرتوی محدب کشیده و آن را توضیح دهید.

۲۰-٤ نقاط و مقادیر زیر را برای یک مسئله‌ی مینیمم‌سازی چندهدفه در نظر بگیرید:

$$f_1(x^{(1)}) = 1, \quad f_2(x^{(1)}) = 1$$
$$f_1(x^{(2)}) = 1, \quad f_2(x^{(2)}) = 2$$
$$f_1(x^{(3)}) = 2, \quad f_2(x^{(3)}) = 1$$
$$f_1(x^{(4)}) = 2, \quad f_2(x^{(4)}) = 2$$

الف) کدام نقطه بر سایر نقاط غلبه دارد؟

ب) دو نقطه‌ی $x^{(2)}$ و $x^{(3)}$ کدام نقطه را مغلوب می‌سازند؟

ج) کدام نقطه نامغلوب است؟

د) کدام نقطه بهینه‌ی پرتو است؟

۲۰-۵ نقاط مسئله‌ی ۲۰-٤ را در نظر بگیرید. برای چه مقادیری از ϵ نقاط $x^{(2)}$، $x^{(3)}$ و $x^{(4)}$ بر نقطه‌ی $x^{(1)}$ غلبه‌ی ϵ افزایشی خواهند داشت؟ برای چه مقادیری از ϵ این نقاط بر $x^{(1)}$ غلبه‌ی ϵ ضربی خواهند داشت؟

۲۰–٦ مثالی از دو نقطه در یک مسئله‌ی مینیمم‌سازی دو هدفه‌ی دو بعدی ارائه دهید به‌گونه‌ای که به ازای هر مقدار از ϵ یک نقطه بر نقطه‌ی دیگر غلبه‌ی ϵ ضربی نداشته باشد.

۲۰–۷ مثالی از دو مرز پرتوی P_1 و P_2 ارائه دهید به‌گونه‌ای که هر دو دارای تعداد برابری از نقاط بوده و اجتماع فراحجم‌های P_2 بزرگتر از P_1 و اشتراک فراحجم‌های P_2 کوچکتر از P_1 باشد.

۲۰–۸ در یک مسئله‌ی مینیمم‌سازی دو هدفه، تخمین مرز پرتوی ٤ نقطه‌ای $f(x)$ و تخمین مرز پرتوی ۳ نقطه‌ای $f(y)$ را مانند زیر در نظر بگیرید:

$$f_1(x^{(1)}) = 3,\quad f_2(x^{(1)}) = 4,\quad f_1(y^{(1)}) = 1,\quad f_2(y^{(1)}) = 3$$
$$f_1(x^{(2)}) = 3,\quad f_2(x^{(2)}) = 3,\quad f_1(y^{(2)}) = 4,\quad f_2(y^{(2)}) = 3$$
$$f_1(x^{(3)}) = 2,\quad f_2(x^{(3)}) = 2,\quad f_1(y^{(3)}) = 4,\quad f_2(y^{(3)}) = 1$$
$$f_1(x^{(4)}) = 5,\quad f_2(x^{(4)}) = 2$$

پوشش x نسبت به y چیست؟ پوشش y نسبت به x چیست؟ با توجه به مقادیر پوشش، کدام تخمین پرتو بهتر است؟

۲۰–۹ چرا در روش تجمعی ضربی از معادله‌ی (۲۰–۲٦) باید فرض کنیم تمام اهداف نامنفی هستند؟

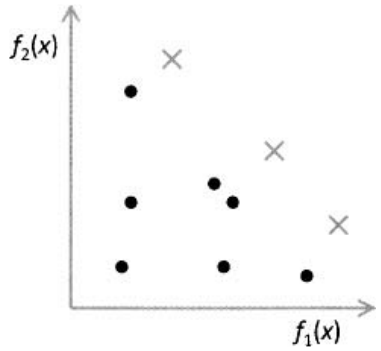
۲۰–۱۰ تفاوت میان نخبه‌گرایی و یک آرشیو را توضیح دهید.

۲۰–۱۱ بیشترین تعداد ذره‌ای که می‌تواند در آرشیو یک ϵ–MOEA دو هدفه که در آن $f_1 \in [0, f_{i,max}]$ برای $i \in [1,2]$ می‌باشد، چه قدر است؟ در مورد ϵ–MOEA سه هدفه چطور؟

۲۰–۱۲ شکل ۲۰–۱۷ یک مستطیل را نشان می‌دهد که می‌توان از آن برای محاسبه‌ی فاصله‌ی ازدحام در NSGA-II استفاده نمود. فرض کنید از مستطیلی استفاده می‌نماییم که بزرگترین مستطیلی است که شامل x بوده و شامل هیچ نقطه‌ی دیگری نمی‌شود. این مستطیل را رسم نمایید.

۲۰–۱۳ دو نقطه‌ی x و y را در یک فضای تابع هدف دو بعدی در نظر بگیرید. فرض کنید فاصله‌ی ازدحام $d_1(x)$ با استفاده از نزدیکترین همسایه‌های x در هر بعد و فاصله‌ی ازدحام $d_2(x)$ با استفاده از بزرگترین مستطیلی که شامل x بوده و شامل نقطه‌ی دیگری نمی‌شود، محاسبه گردد. آیا $d_1(x) < d_2(y)$ به معنای $d_2(x) < d_2(y)$ خواهد بود؟

۲۰–۱٤ شکل ۲۰–۲۹، که ذرات در جمعیت و آرشیو یک مسئله‌ی ماکزیمم‌سازی دو هدفه با استفاده از SPEA را نشان می‌دهد [زیتزلر و تیل، ۱۹۹۹]، را در نظر بگیرید. مقادیر هزینه‌ی خام هر ذره‌ی در جمعیت و مقادیر قدرت هر ذره‌ی در آرشیو چه قدر است؟



شکل ۲۰-۲۹ مسئله‌ی ۲۰-۱٤: دایره‌های توپر ذرات در جمعیت بوده و ضربدرها ذرات درون آرشیو هستند.

۲۰-۱۵ فرض کنید سه ذره‌ی زیر را در یک مسئله‌ی مینیمم‌سازی دو هدفه در اختیار داریم:

$$f_1(x_1) = 3, \quad f_2(x_1) = 4$$
$$f_1(x_2) = 4, \quad f_2(x_2) = 3$$
$$f_1(x_3) = 2, \quad f_2(x_3) = 2$$

رتبه‌های VEBBO r_{ji} در شکل ۲۰-۲۵ چه خواهند بود؟

تمارین کامپیوتری

۲۰-۱٦ از روش جستجوی جامع با دقت جستجوی ۰٫۰۱ برای یافتن مجموعه‌ی پرتو و مرز پرتوی مسئله‌ی زیر استفاده نمایید.

$$\min[\cos(x_1 + x_2), \quad \sin(x_1 - x_2)]$$

در مسئله‌ی بالا دامنه‌ی جستجو برای هر بعد $[0, \pi]$ می‌باشد.

۲۰-۱۷ از روش جمع وزن‌ها برای کاهش دو هدف مسئله‌ی ۲۰-۱۷ به یک هدف استفاده نمایید. برای چه مقادیری از وزن‌های w_1 و w_2 راه‌حل مسئله‌ی تک هدفه با مرز پرتوی مسئله‌ی دو هدفه برابر خواهد شد؟

۲۰-۱۸ الگوریتم تکاملی جنسیت-محور شکل ۲۰-۱۱ را بر روی یک مسئله‌ی چندهدفه اجرا نمایید. عملکرد الگوریتم را در حالت‌های مقایسه‌ای زیر اندازه بگیرید:

الف) یک والد در هر زیرجمعیت در برابر دو والد در هر زیر جمعیت

ب) عضویت هر فرزند تنها در یک زیرجمعیت در برابر عضویت هر فرزند در بیش از یک زیرمجموعه

ج) جایگزین نمودن ذرات تکراری در هر نسل در برابر جایگزین نکردن ذرات تکراری.

فصل بیست و یکم

توابع برازندگی گران، شلوغ و پویا

الگوریتم‌های تکاملی اغلب باید مسائل بهینه‌سازی را در حضور گستره‌ی وسیعی از عدم قطعیت حل نمایند.

یائوچو جین[1] و یورگن برانک[2] [جین و برانک، ۲۰۰۵]

هر کس که پیش از ۱۹۷۰ بر روی الگوریتم‌های تکاملی کار می‌کرده است از زمان خودش جلوتر بوده است. در آن سال‌های ابتدایی حدود ۱۲ نفر کارهای بسیار بنیادین و اساسی بر روی الگوریتم‌های تکاملی انجام دادند به‌طوری که می‌توان به هر یک از آن‌ها لقب "پدر الگوریتم‌های تکاملی" را نسبت داد. اما تمامی این کارهای اولیه به دلیل کمبود و ناکافی بودن منابع کامپیوتری از پای افتادند. الگوریتم‌های تکاملی که در دهه‌ی ۱۹٦۰ وجود داشتند باید بسیار کوچک و ساده می‌بودند تا اجرای آن‌ها در یک مدت زمان معقول قابل انجام می‌بود. قدرت محاسباتی در دهه‌ی ۱۹٦۰ به اندازه‌ای نبود که بتوان با آن تحقیقات الگوریتم‌های تکاملی را پیش برد.

در طول دهه‌ی ۱۹۷۰ منابع کامپیوتری قدرتمندتر شده و بیشتر در دسترس بودند به‌گونه‌ای که در دهه‌ی ۱۹۸۰ تحقیقات الگوریتم‌های تکاملی از رکود درآمده و به جای اولیه خود بازگشته و به زمینه‌ای فعال بدل گشت. اگر به INSPEC، که مرکز اطلاعات کامپیوتری مقالات در حوزه‌ی کامپیوتر و مهندسی است، سری بزنید خواهید دید که در دهه‌ی ۱۹۷۰ تنها یک مقاله در زمینه‌ی GAها منتشر شده است. این در حالی است که در دهه‌ی ۱۹۸۰ تعداد مقالات در این زمینه به ۳۷ و در دهه‌ی ۱۹۹۰ به ۷۹٤۲ و در آخر در دهه‌ی اول قرن ۲۱ به ۳۵٤٤۰ عدد رسیده است. فناوری محاسباتی اکنون به قدری قدرتمند شده است که هر کس بخواهد می‌تواند الگوریتمی تکاملی را بر روی کامپیوتر رومیزی خود نوشته و به حل مسائل چالش برانگیز بپردازد.

نهایت سرعت کامپیوترهای مرکزی در دهه‌ی ۱۹٦۰، ۱۰ مگاهرتز بوده است. این در حالی است که یک کامپیوتر عادی در قرن ۲۱ سرعتی برابر ۱۰ گیگاهرتز و چه بسا بیشتر دارد (با در نظر گرفتن پردازنده‌های چند هسته‌ای). بدین ترتیب می‌توان دید که اندازه‌ی قدرت محاسباتی از ۱۹٦۰ تا ۲۰۱۰ هزار برابر شده است. با این حال، هم اکنون نیز الگوریتم‌های تکاملی بعضاً به چند روز زمان برای کامل شدن اجرایشان نیاز دارند. این یکی از دلایل تأکیدی است که بر روی موازی‌سازی در زمینه‌ی الگوریتم‌های تکاملی وجود دارد. اما موازی‌سازی چالش‌های خاص خود را به همراه خواهد داشت.

---

[1] Yaochu Jin
[2] Jurgen Branke

مروری بر فصل

این فصل به بحث در مورد نحوه‌ی کاهش هزینه‌ی محاسباتی الگوریتم‌های تکاملی می‌پردازد. در دنیای واقعی، برازندگی توابع هزینه می‌تواند به لحاظ محاسباتی بسیار گران و پرهزینه باشد. توابع محک استفاده شده در این کتاب ساده هستند و می‌توان در عرض چند میلی ثانیه آن‌ها را محاسبه نمود. اما توابع هزینه‌ی دنیای واقعی ممکن است به چند روز زمان برای ارزیابی نیاز داشته باشند. در چنین شرایطی نمی‌توان الگوریتم‌های تکاملی که به چند هزار ارزیابی تابع نیاز دارند را اجرا نمود. بخش ۲۱-۱ نحوه‌ی رسیدگی به توابع هزینه‌ی گران را نشان خواهد داد.

موارد مرتبطی که در دنیای واقعی با آن‌ها مواجه می‌شویم شامل توابع هزینه متغیر با زمان و توابع هزینه نویزی می‌شود. توابع هزینه می‌توانند به دلیل طبیعت پویا و بعضاً غیر قابل پیش‌بینی دنیایمان دستخوش تغییر شوند. بخش ۲۱-۲ نحوه‌ی برخورد با مسائل بهینه‌سازی پویا را نشان خواهد داد. در آخر، توابع هزینه به دلیل عدم وجود دقت در بسیاری از مسائل و یا گنگی ذاتی در تعیین کیفیت راه‌حل‌های نامزد، نویزی می‌شوند. بخش ۲۱-۳ نحوه‌ی حل مسائل بهینه‌سازی نویزی را نشان می‌دهد.

۲۱-۱ توابع برازندگی گران

در بسیاری از مسائل دنیای واقعی، یک بار ارزیابی تابع برازندگی ممکن است چند دقیقه، چند ساعت، چند روز و یا حتی بیشتر طول بکشد. در این بخش به بحث در مورد نحوه‌ی کاهش زمان ارزیابی برازندگی می‌پردازیم تا بدین ترتیب میزان محاسبات الگوریتم‌های تکاملی را کاهش دهیم.

هر کس که با الگوریتم‌های تکاملی در دنیای واقعی کار کرده باشد می‌داند که ارزیابی تابع برازندگی وقت‌گیرترین قسمت الگوریتم می‌باشد. شاید این موضوع برای مسائل محک و آموزشی صادق نباشد، اما برای مسائل دنیای واقعی تقریباً همیشه صادق است. جان کوزا معتقد است که تلاش محاسباتی مورد نیاز برای محاسبه‌ی تابع برازندگی "معمولاً آنقدر زیاد است که توجه به سایر جنبه‌های اجرای الگوریتم تقریباً هیچ فایده‌ای ندارد" [کوزا، ۱۹۹۲، ضمیمه H]. در [بنژاف و همکاران، ۱۹۹۸، بخش ۱۱-۱] به گفته‌ی مشابهی بر می‌خوریم: "تقریباً تمام زمان اجرای GP صرف ارزیابی برازندگی می‌شود". مسائل دنیای واقعی اغلب شامل توابع برازندگی هستند که شامل یک یا چند خصیصه‌ی زیر می‌باشند [نولز، ۲۰۰۵].

- یک بار ارزیابی تابع برازندگی به چند دقیقه، ساعت و یا روز نیاز دارد. این ویژگی خصوصاً در مورد توابع برازندگی که باید به‌صورت تجربی (و نه از طریق شبیه‌سازی) ارزیابی شوند، صادق است.

بسیاری از الگوریتم‌های تکاملی اولیه به دلیل کمبود منابع شبیه‌سازی در سیستم‌های تجربی پیاده‌سازی شده‌اند [روچنبرگ، ۱۹۹۸]، [روچنبرگ، ۱۹۷۳].

- ارزیابی‌های تابع برازندگی قابل موازی‌سازی نیستند. این ویژگی بالأخص در مورد توابع برازندگی که باید به‌صورت تجربی و با منابع محدود ارزیابی شوند صادق است. برای مثال، برخی مسائل بهینه‌سازی نیازمند راه‌اندازی‌های آزمایشی خاصی هستند که به کنش و واکنش با کاربر انسانی نیاز دارند. گاهی نیاز است یک متخصص انسانی برازندگی راه‌حل‌های نامزد را ارزیابی نماید. برخی توابع برازندگی قابل اندازه‌گیری نیستند و در عوض به ارزیابی توصیفی و ذهنی از سوی متخصصان انسانی نیاز دارند. این موضوع برای مثال، در مورد الگوریتم‌هایی که موسیقی و یا نوعی از هنر را تولید می‌کنند، صادق است [نیرهاوس[1]، ۲۰۱۰].
- تعداد ارزیابی‌های تابع برازندگی دارای محدودیت زمانی و یا نوع دیگری از محدودیت منابع می‌باشد. این خصیصه در مورد مسائلی که باید تا زمانی خاص حل شده و یا الگوریتم‌های تکاملی که باید به‌صورت زمان–حقیقی اجرا شوند و یا توابع برازندگی که باید به‌صورت تجربی و توسط متخصصانی با مهارت‌های خاص ارزیابی شوند، صادق است.

برخی روش‌ها برای کاهش دادن تلاش محاسباتی مورد نیاز جهت ارزیابی‌های تابع برازندگی به قرار زیراند.

- عدم محاسبه‌ی هزینه‌ی ذراتی که پیش از این هزینه‌ی آن‌ها محاسبه شده است. در بسیاری از الگوریتم‌های تکاملی برخی ذرات ممکن است بدون هیچ تغییری از یک نسل به نسل دیگر منتقل شوند. ذرات یکسانی که از نسل i به نسل $i+1$ منتقل می‌شوند نیازی به ارزیابی دوباره در نسل $i+1$ نخواهند داشت. اگر مسئله پویا نباشد، هزینه‌ی این ذرات از پیش معلوم خواهد بود.

این ایده را می‌توان به دنبال نمودن بردارهای تمام راه‌حل‌های نامزد و هزینه‌های آن‌ها در طول الگوریتم تکاملی تعمیم داد. برای این کار می‌توان پس از هر نسل، هر ذره و هزینه‌اش را در یک آرشیو نگه‌داری نمود. اگر جمعیتی ثابت با N ذره در اختیار داشته باشیم، آنگاه پس از T نسل آرشیوی از NT ذره (منهای ذرات تکراری) در اختیار خواهیم داشت. آرشیو در فرایند تکاملی دخالتی نخواهد داشت و از آن تنها برای جلوگیری از ارزیابی هزینه‌ی غیرضروری استفاده خواهیم نمود. هر بار که نیاز به ارزیابی یک هزینه داشته باشیم، ابتدا درون این آرشیو را نگاه می‌کنیم تا ببینیم آن ذره‌ی خاص قبلاً ارزیابی شده است یا خیر. پس از گذشت تعداد

[1] Nierhaus

زیادی نسل، آرشیو می‌تواند بسیار بزرگ شده و جستجو میان آن می‌تواند بسیار گران و پرهزینه شود. با این حال، بسته به مسئله، این جستجو احتمالاً بسیار کم هزینه‌تر از ارزیابی هزینه خواهد بود.

- اگر مشاهده شود که ارزیابی‌های تابع برازندگی بسیار خوب و یا بسیار بد عمل می‌نمایند، می‌توان آن‌ها را کوتاه نمود [گذرکول و راس، ۱۹۹۷۷]. اگر در میانه‌ی ارزیابی تابع برازندگی یک ذره متوجه شویم که ذره عملکرد بسیار خوبی دارد، می‌توانیم قبل از موعد مقرر از روتین ارزیابی خارج شده و برازندگی تقریباً زیادی را به آن ذره اختصاص داده و بدین ترتیب نیمی از تلاش محاسباتی آن ارزیابی را ذخیره نماییم. به‌صورت مشابه، اگر در میانه‌ی ارزیابی تابع برازندگی یک ذره متوجه شویم که ذره عملکرد بسیار بدی دارد، می‌توانیم قبل از موعد مقرر از روتین ارزیابی خارج شده و بزارندگی نسبتاً کمی را به آن ذره اختصاص دهیم و نیمی از تلاش محاسباتی را حفظ نماییم.
- اگر نیاز باشد ارزیابی تابع برازندگی را برای مجموعه‌ی بزرگی از نمونه‌ها انجام دهیم، می‌توانیم هزینه را با استفاده از زیرمجموعه‌ای از نمونه‌ها تخمین بزنیم. برای مثال، فرض کنید می‌خواهیم $f(x) = \sum_{i=1}^{M} f_i(x)$ را نسبت به x مینیمم نماییم. این معادله به این معنی است که تابع برازندگی خود ترکیبی از چند زیرتابع است. این حالت اغلب مواقعی پیش می‌آید که می‌خواهیم یک تابع را در طول چندین ناحیه‌ی عملیاتی بهینه نماییم. برای مثال، ممکن است بخواهیم عملکرد ردیابی یک روبات را برای چندین حالت اولیه‌ی مختلف و برای چندین وظیفه‌ی مختلف بهینه نماییم. یک رویکرد آن است که $f_1(x)$ را در T نسل اول مینیمم کرده، سپس $f_1(x) + f_2(x)$ را در T نسل بعدی، سپس $f_1(x) + f_2(x) + f_3(x)$ را در T نسل سوم و ... مینیمم نماییم. این کار را آن قدر ادامه می‌دهیم تا در نهایت $f(x) = \sum_{i=1}^{M} f_i(x)$ را در طول نسل‌های $(M-1)T$ تا MT مینیمم کرده باشیم. یک رویکرد دیگر، مینیمم کردن ترکیبی اتفاقی از توابع $f_i(x)$ و افزایش تعداد نمونه‌های توابع با افزایش شماره‌ی نسل می‌باشد. این رویکرد، نمونه‌گیری اتفاقی[1] نام داشته [بنژاف و همکاران، ۱۹۹۸، بخش ۱۰-۱-۵] و مشابه مرتب‌سازی واژه‌نگاری (بخش ۲۰-۳-۳ را ببینید) و روش قید ϵ در بهینه‌سازی چندهدفه (بخش ۲۰-۳-٤ را ببینید)، می‌باشد.
- اگر ارزیابی‌های تابع برازندگی توسط کامپیوتر انجام شود، می‌توان از روش‌های استاندارد افزایش سرعت اجرای نرم‌افزارها استفاده نمود. این روش‌ها شامل ارائه‌های پیش تخصیص[2]، استفاده از ویژگی‌های بهینه شده‌ی زبان‌های برنامه‌نویسی (برای مثال، عملیات ارائه‌ای در MATLAB)، کاهش

[1] Stochastic Sampling
[2] Pre-Allocating Arrays

دقت کامپیوتر، استفاده از جدول‌های جست‌وجو برای توابع پیچیده و غیرفعال نمودن گرافیک می‌باشد.

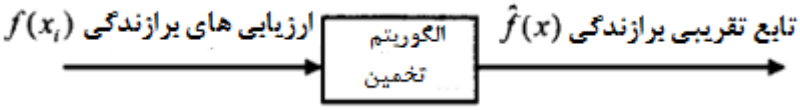
در ادامه‌ی این بخش به بحث در مورد برخی روش‌های تخمین توابع برازندگی (بخش ۲۰-۱-۱ را ببینید)، چگونگی بهبود عملکرد تخمین تابع با استفاده از تبدیل تابع برازندگی (بخش ۲۰-۱-۲ را ببینید)، چگونگی استفاده از تخمین تابع برازندگی در یک الگوریتم تکاملی (بخش ۲۰-۱-۳ را ببینید)، زمان استفاده از چندین تخمین تابع برازندگی در یک الگوریتم تکاملی (بخش ۲۰-۱-٤ را ببینید)، خطر بیش برازش[1] تخمین یک تابع برازندگی (بخش ۲۰-۱-۵ را ببینید) و نحوه‌ی ارزیابی تخمین یک تابع برازندگی (بخش ۲۰-۱-٦ را ببینید)، خواهیم پرداخت.

۲۱-۱-۱ تخمین تابع برازندگی

ما می‌توانیم با ایجاد مدل تابع برازندگی، از تلاش مورد نیاز برای ارزیابی تابع برازندگی بکاهیم. همچنین می‌توان از مدل‌های تابع برازندگی برای بهبود عملکرد الگوریتم تکاملی استفاده نمود. از این مدل‌ها با عنوان جانشین[2]، سطوح پاسخ[3] و یا متامدل‌ها یاد می‌شود [شی و رشید[4]، ۲۰۱۰]. عنوان جانشین از آن رو انتخاب شده است که می‌توان به مدل برازندگی به‌عنوان جانشینی موقتی برای ارزیابی دقیق برازندگی نگاه کرد. همچنین از عنون متامدل نیز به این دلیل استفاده می‌شود که ارزیابی برازندگی اغلب خود تنها یک تخمین است (برای مثال، یک شبیه‌سازی که یک فرایند فیزیکی را مدل می‌نماید). بنابراین، مدل تابع برازندگی در واقع یک مدل با مرتبه‌ی پایین‌تر نسبت به یک مدل مرتبه‌ی بالاتر است.

فرض کنید تابع برازندگی $f(x)$ را در اختیار داریم و آن را بر روی M ذره‌ی $\{x_i\}$ ارزیابی نموده‌ایم. می‌توان از این M مقدار تابع برازندگی برای تخمین مقدار تابع در هر نقطه‌ای از فضای جستجو استفاده نمود. واضح است که باید انتظار داشت تخمین تابع با خطایی همراه باشد چرا که اگر تخمین‌ها بدون نقص بودند دیگر نیازی به استفاده از ارزیابی‌های اضافی نبود. با این حال، حتی اگر تخمین برازندگی دارای خطا باشد، ممکن است این خطاها به قدری کوچک باشند که بتوان از تخمین به طرز مفیدی استفاده نمود. شکل ۲۱-۱ ایده‌ی اصلی تخمین برازندگی را نشان می‌دهد. یک تخمین $\hat{f}(x)$ بر اساس مقادیر معلوم تابع برازندگی $f(x_i)$ ایجاد می‌شود.

[1] Overfitting
[2] Surrogate
[3] Response Surfaces
[4] Rasheed

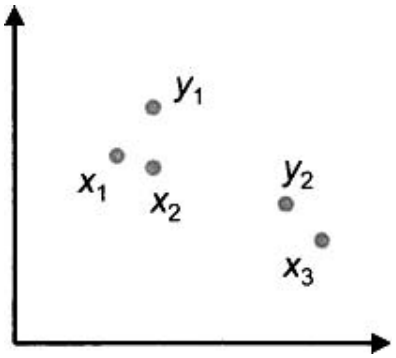


شکل ۲۱-۱ تخمین تابع برازندگی. از مقادیر دقیق تابع برازندگی $\{f(x_i)\}$ جهت تخمین $f(x)$ برای $x \notin \{x_i\}$ استفاده می‌شود.

سابقه‌ی استفاده از تخمین تابع برازندگی برای کاهش تلاش محاسباتی الگوریتم تکاملی به دهه‌ی ۱۹۶۰ بر می‌گردد [دانهم[1] و همکاران، ۱۹۶۳]. در آن زمان منابع محاسباتی بسیار محدودتر از امروز بوده‌اند. از آن زمان تا کنون، محققان الگوریتم‌های بسیاری را جهت انجام عمل تخمین در شکل ۲۱-۱ به کار برده‌اند. در حقیقت، می‌توان از هر الگوریتم تخمین با درون‌یابی استفاده نمود. رویکردهای مختلف شامل الگوریتم k همسایه نزدیک، توابع شعاع-محور[2]، شبکه‌های عصبی، منطق فازی، خوشه‌بندی، درخت‌های تصمیم‌گیری، مدل‌های چندجمله‌ای، مدل‌های کریجینگ[3]، سری‌های تیلور، سری‌های فوریه، مدل‌های NK، مدل‌های فرایندی گاوسی و ماشین‌های بردار پشتیبان[4]، می‌شود [جین، ۲۰۰۵]، [شی و رشید، ۲۰۱۰]. جزییات این رویکردها در این کتاب مورد بحث قرار نخواهد گرفت اما کافی است که بگوییم که تقریباً هر الگوریتم تخمینی را می‌توان برای تخمین برازندگی در الگوریتم‌های تکاملی به کار برد.

یکی از ساده‌ترین الگوریتم‌های تخمینی که می‌توان در شکل ۲۱-۱ به کار برد، تخمین برازندگی یک ذره با استفاده از برازندگی نزدیکترین همسایه آن که ارزیابی شده است، می‌باشد. این روش تقلید برازندگی نام داشته و به یک تخمین تکه‌ای ثابت از فضای برازندگی تقلیل می‌یابد. شکل ۲۱-۲ تقلید برازندگی را نشان می‌دهد.

[1] Dunham
[2] Radial-Basis Functions
[3] Kriging Models
[4] Supported Vector Machines



شکل ۲۱-۲ تقلید برازندگی در یک فضای جستجوی دو بعدی. برازندگی ذرات y_1 و y_2 با استفاده از روتین تابع برازندگی و یا به‌صورت تجربی، به‌صورت دقیق، تعیین شده است. ذرات x_1، x_2 و x_3 ارزیابی نشده‌اند. می‌توان مقادیر برازندگی این سه ذره را با استفاده از نزدیکترین همسایه‌شان تعیین نمود: $\hat{f}(x_1) = f(y_1)$، $\hat{f}(x_2) = f(y_2)$ و $\hat{f}(x_3) = f(y_3)$.

یکی از موارد مورد اهمیت در شکل ۲۱-۱ آن است که چگونه با در دسترس قرار گرفتن اطلاعات جدید، تخمین $\hat{f}(.)$ را به روزرسانی نماییم. به‌صورت کلی برای ما مطلوب است الگوریتم تخمین برازندگی فرم بازگشتی داشته باشد تا بتوانیم به محض در دسترس قرار گرفتن اطلاعات و داده‌های جدید، $\hat{f}(.)$ را با کمترین تلاش، به روز رسانی نماییم. با این حال، اگر فضای برازندگی پویا باشد، بهتر است الگوریتم تخمین، مقادیر قدیمی داده‌های برازندگی را از فرایند تولید $\hat{f}(.)$ بکاهد. به روزرسانی یک تخمین برازندگی با استفاده از داده‌ی جدید، به روز رسانی جانشین[1] آنلاین نام دارد.

یک رویکرد دیگر در تخمین برازندگی، تعیین برازندگی فرزندان در یک الگوریتم تکاملی بر حسب برازندگی والدینشان است. این روش، وراثت برازندگی[2] نام دارد [اسمیت و همکاران، ۱۹۹۵]. برازندگی یک فرزند را می‌توان با میانگین و یا میانگین وزنی برازندگی والدینش تخمین زد. وزن‌ها را می‌توان بر اساس اینکه فرزند به کدام یک از والدینش شبیه‌تر است تعیین نمود. می‌توان این ایده را به الگوریتم‌های تکاملی که در آن‌ها هر فرزند ممکن است هر تعداد والدی داشته باشد، تعمیم داد. حتی می‌توان این ایده را به‌گونه‌ای تعمیم داد که برازندگی هر فرزند با میانگین وزنی برازندگی کل جمعیت محاسبه گردد. در این صورت نیز، وزن‌ها بر اساس این که فرزند چه قدر به هر ذره‌ی ارزیابی شده در جمعیت شبیه است، تعیین می‌شوند [ساستری و همکاران، ۲۰۰۱]. همچنین می‌توان از ایده‌های پیچیده‌تری از وراثت برازندگی استفاده نمود. برای مثال، می‌توان همبستگی میان متغیرهای مستقل و مقادیر برازندگی را لحاظ نمود [پلیکان و ساستری،

[1] Online Surrogate Updating
[2] Fitness Inheritance

۲۰۰٤]. در [دوچین[1] و همکاران، ۲۰۰۳] نتیجه‌گیری شده است که وراثت برازندگی تنها برای مسائل نسبتاً ساده مؤثر است. به‌طور خاص، در مورد مسائل چندهدفه وراثت برازندگی تنها در صورتی مؤثر خواهد بود که مرز پرتو پیوسته و محدب باشد.

۲۱-۱-۱-۱ مدل‌های چند جمله‌ای

تخمین ثابت تکه‌ای در روش تقلید برازندگی نقطه‌ی شروع خوبی است چرا که به ما نشان می‌دهد چگونه می‌توان تخمین برازندگی را به چند جمله‌ای‌هایی از درجه‌ی بالاتر تعمیم داد. برای مثال، می‌توان برازندگی را به‌عنوان تابعی خطی تخمین زد

$$\hat{f}(x) = a(0) + \sum_{k=1}^{n} a(k)x(k) \qquad (۱-۲۱)$$

در معادله‌ی بالا n ابعاد مسئله بوده و $x(k)$، kاُمین عنصر ذره‌ی x می‌باشد. این یک مثال ساده از مدل چند جمله‌ای است و با نام سطح پاسخ نیز شناخته می‌شود. مقادیر $a(k)$ را می‌توان با حل نمودن مسئله‌ی زیر به دست آورد:

$$\min \sum_{i=1}^{M} \left(f(x_i) - \left[a_0 + \sum_{k=1}^{n} a(k)x_i(k) \right] \right) \qquad (۲-۲۱)$$

در معادله‌ی بالا M تعداد ذراتی است که مقادیر برازندگیشان را به‌صورت دقیق می‌دانیم، x_i، iاُمین ذره‌ای است که مقدار برازندگیش را به‌صورت دقیق می‌دانیم و $x_i(k)$ نیز kاُمین عنصر x_i می‌باشد. مینیمم‌سازی معادله‌ی (۲-۲۱) بر روی $(n+1)$ پارامتر $a(k)$ و برای $k \in [0,n]$ صورت می‌پذیرد. معادله‌ی (۲-۲۱) را می‌توان با استفاده از یک الگوریتم بازگشتی حداقل مربعات[2] حل نمود [سایمون، ۲۰۰٦، فصل ۳]. در این صورت، با به دست آمدن مقادیر جدید برازندگی ($M=1$، $M=2$ و غیره)، تلاش بسیار اندکی برای به روزرسانی راه‌حل معادله‌ی (۲-۲۱) مورد نیاز خواهد بود.

به جای مدل خطی معادله‌ی (۱-۲۱) می‌توان از مدل دقیق‌تری استفاده نمود:

$$\hat{f}(x) = a(0) + \sum_{k=1}^{n} a(k)x(k) + \sum_{j,k=1}^{n} a(j,k)x(j)x(k) \qquad (۳-۲۱)$$

[1] Ducheyne
[2] Recursive Least Squares Algorithm

این یک مدل مرتبه‌ی دوم با (n^2+n+1) پارامتر است. این معادله را نیز می‌توان با استفاده از الگوریتم بازگشتی حداقل مربعات حل نمود چرا که این مدل نیز نسبت به پارامترهای $a(k)$ و $a(j,k)$، خطی است. پس از آنکه مفهوم مدل‌سازی چند جمله‌ای را درک نمودیم، می‌توانیم شکل‌های مختلفی از این مدل را امتحان نماییم:

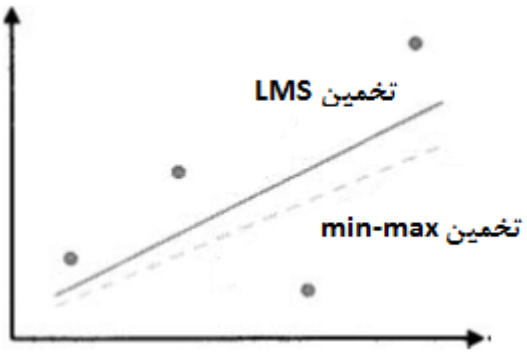
$$\hat{f}(x) = a(0) + \sum_{k=1}^{n} a(k)g\big(x(k)\big) + \sum_{j,k=1}^{n} a(j,k)h\big(x(j),x(k)\big) \qquad (۲۱-٤)$$

در معادله‌ی بالا $g(.)$ و $h(.)$ می‌توانند هر تابع خطی یا غیرخطی باشند. برای نمونه، اگر بدانیم که در مسئله‌ی بهینه‌سازی، تابع برازندگی را می‌توان با توابع مثلثاتی ارائه نمود، می‌توان از عبارت‌های سینوسی و کسینوسی برای $g(.)$ استفاده کرد.

ممکن است بخواهیم از روش‌های دیگری غیر از روش حداقل مربعات برای تخمین تابع برازندگی استفاده نماییم. برای نمونه، شاید به جای پیدا کردن مدلی که معادله‌ی (۲۱-۱) را حل نماید بخواهیم مدلی را پیدا کنیم که معادله‌ی زیر را حل نماید

$$\min_{\{a(k)\}} \max_{i} \left| f(x_i) - \left[a_0 + \sum_{k=1}^{n} a(k)x_i(k) \right] \right| \qquad (۲۱-۵)$$

در معادله‌ی بالا نیز مینیمم‌سازی بر روی $(n+1)$ پارامتر $a(k)$ انجام می‌شود. شکل ۲۱-۳ تفاوت میان مینیمم‌سازی جمع مربعات خطاهای تخمین و مینیمم‌سازی بزرگترین خطا را نشان می‌دهد. شرط حداقل مربعات در معادله‌ی (۲۱-۲) جذاب است چرا که می‌توان آن را به‌صورت تحلیلی حل نمود، اما شرط min-max ممکن است مطلوب‌تر باشد چرا که باعث می‌شود خطای بدترین حالت، مینیمم شود. تخمین min-max در نواحی از فضای جستجو که تطبیق دادن مدل آسان باشد، خطای تخمین را قربانی خواهد کرد تا خطای تخمین را در نواحی چالش برانگیزتر فضای جستجو کاهش دهد.



شکل ۲۱-۳ مقایسه‌ی میان تخمین خط راست حداقل میانگین مربعات و تخمین min-max. تخمین حداقل مربعات را می‌توان به‌صورت تحلیلی حل نمود اما تخمین min-max ممکن است نتیجه‌ی مطلوب‌تری را به دست دهد.

۲۱-۱-۱-۲ طراحی و تحلیل آزمایش‌های کامپیوتر

طراحی و تحلیل آزمایش‌های کامپیوتری (DACE[1]) یک روش تخمینی اتفاقی است که در آن میزان مطلوب بودن تخمین با استفاده از آزمایش‌های عیب‌یابی مشخص می‌شود [جونز[2] و همکاران، ۱۹۹۸]. با فرض در اختیار داشتن M ارزیابی تابع برازندگی $f(x_i)$ برای بردارهای n بعدی x_i، تابع برازندگی با استفاده از معادله‌ی زیر تخمین زده می‌شود

$$f(x) = \mu + \epsilon(x) \qquad (۶-۲۱)$$

در معادله‌ی بالا μ یک ثابت بوده و $\epsilon(x)$ یک عبارت تصحیح است. در DACE فرض بر آن است که عبارت تصحیح $\epsilon(x)$، یک عبارت گاوسی با میانگین μ و واریانس σ^2 برای تمامی xها می‌باشد. این بدین معنی است که تابع چگالی احتمال (PDF) $f(x)$ به‌صورت زیر است

$$PDF\big(f(x)\big) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[\frac{-(f(x)-\mu)^2}{2\sigma^2}\right] \qquad (۷-۲۱)$$

با این حال، در DACE یک فرض بسیار مهم وجود دارد و آن این است که عبارت‌های $\epsilon(x)$ برای مقادیر مختلف x مستقل نیستند. این بدین معنی است که برای مقادیری از x که شبیه هم هستند، عبارت‌های تصحیح نیز مشابه می‌باشند. در DACE همچنین فرض بر این است که ضرایب همبستگی ρ_{ij} بین $f(x_i)$ و $f(x_j)$ را می‌توان به‌صورت زیر محاسبه نمود:

[1] Design and Analysis of Computer Experiments
[2] Jones

$$d_{ij} = \sum_{k=1}^{n} \theta_k \left| x_i(k) - x_j(k) \right|^{p_k}$$
$$\rho_{ij} = Corr\left(f(x_i), f(x_j)\right) = \exp(-d_{ij}) \qquad (۸-۲۱)$$

در معادله‌ی بالا $x_i(k)$، kاُمین عنصر iاُمین راه‌حل نامزد بوده، $p_k \in [1,2]$ و $\theta_k \geq 0$ پارامترهای مدل بوده و $d_{ij} \geq 0$ نیز نوعی متریک فاصله است. می‌توان دید که برای مقادیر کوچک d_{ij}، x_i و x_j دارای همبستگی نزدیک به ۱ می‌باشند. برای مقادیر بزرگ d_{ij}، همبستگی x_i و x_j به ۰ نزدیک خواهد بود. با فرض در اختیار داشتن M ارزیابی تابع برازندگی، آن‌ها را در یک بردار قرار داده و آن‌ها را به‌صورت زیر پارامتری می‌نماییم:

$$f(x) = [f(x_1) \dots f(x_M)]^T$$
$$= \mu 1_M + [\epsilon(x_1) \dots \epsilon(x_M)]^T \qquad (۹-۲۱)$$

در معادله‌ی بالا، 1_M ماتریسی ستونی با اندازه‌ی M است که تمام درایه‌های آن برابر ۱ است. توجه داشته باشید که از $f(x)$ هم برای نشان دادن برازندگی یک راه‌حل نامزد x و هم برای نشان دادن بردار M عنصری شامل M برازندگی $\{x_i\}$ استفاده می‌نماییم. PDF گاوسی M تابع برازندگی از معادله‌ی (۲۱-۹) را می‌توان از رابطه‌ی زیر محاسبه نمود

$$PDF\left(f(x)\right) = \frac{1}{(2\pi)^{M/2}|C|^{1/2}} \exp\left[-\frac{(f(x) - \mu 1_M)^T C^{-1} (f(x) - \mu 1_M)}{2}\right] \qquad (۱۰-۲۱)$$

در معادله‌ی بالا C ماتریس کوواریانس $f(x)$ می‌باشد. به یاد آورید که کوواریانس C_{ij} میان دو متغیر اتفاقی $f(x_i)$ و $f(x_j)$ که دارای واریانس یکسان σ^2 می‌باشند را می‌توان از رابطه‌ی زیر محاسبه نمود [سایمون، ۲۰۰۶، فصل ۲]:

$$C_{ij} = \rho_{ij} \sigma^2 \qquad (۱۱-۲۱)$$

بنابراین، معادله‌ی (۲۱-۱۰) را می‌توان به‌صورت زیر نوشت

$$PDF\left(f(x)\right) = \frac{1}{(2\pi)^{M/2} \sigma^M |R|^{1/2}} \exp\left[-\frac{(f(x) - \mu 1_M)^T R^{-1} (f(x) - \mu 1_M)}{2\sigma^2}\right] \qquad (۱۲-۲۱)$$

که در آن R ماتریس همبستگی بوده و درایه‌ای که در سطر i و ستون jاُم این ماتریس واقع شده باشد، ρ_{ij} خواهد بود.

با فرض در اختیار داشتن مجموعه‌ای از راه‌حل‌های نامزد مانند $\{x_i\}$ و برداری از ارزیابی‌های برازندگی مانند $f(x)$، می‌توان مقادیری از μ و σ را، که بهترین تطبیق میان مقادیر اندازه‌گیری شده‌ی $f(x)$ و مقادیر

فرضی آن را به دست می‌دهند، تعیین نمود. معادله‌ی (۲۱-۱۲) PDF تابع $f(x)$ را، که با احتمال به دست آمدن مقدار خاصی از $f(x)$ متناسب است، به دست می‌دهد. بنابراین، جهت یافتن بهترین تطبیق میان مقادیر اندازه‌گیری شده‌ی $f(x)$ و مقادیر پارامتری فرضی آن، باید مقادیری از μ و σ را بیابیم که مقدار $PDF(f(x))$ در معادله‌ی (۲۱-۱۲) را ماکزیمم می‌نماید. ابتدا ماکزیمم‌سازی نسبت به μ را در نظر می‌گیریم. با گرفتن مشتق جزیی از جزء نمایی نسبت به μ و قرار دادن آن برابر صفر خواهیم داشت:

$$\frac{\partial(f(x)-\mu 1_M)^T R^{-1}(f(x)-\mu 1_m)}{\partial\mu}=0 \qquad (۲۱-۱۳)$$

در معادله‌ی بالا از عبارت $2\sigma^2$ در مخرج صرف نظر شده است چرا که این عبارت مستقل از μ است. با حل معادله‌ی (۲۱-۱۳) خواهیم داشت:

$$-2f^T(x)R^{-1}1_M+2\mu 1_M^T R^{-1}1_M=0$$
$$\mu=\frac{f^T(x)R^{-1}1_M}{1_M^T R^{-1}1_M} \qquad (۲۱-۱٤)$$

با گرفتن مشتق جزیی از معادله‌ی (۲۱-۱۲) نسبت به σ^2 خواهیم داشت:

$$\frac{\partial PDF\big(f(x)\big)}{\partial\sigma^2}=\frac{1}{2\sigma^2}\left[\frac{(f(x)-\mu 1_M)^T R^{-1}(f(x)-\mu 1_m)}{\sigma^2}-M\right]PDF\big(f(x)\big) \qquad (۲۱-۱٥)$$

با قرار دادن معادله‌ی بالا برابر صفر خواهیم داشت:

$$\sigma^2=\frac{(f(x)-\mu 1_M)^T R^{-1}(f(x)-\mu 1_m)}{M} \qquad (۲۱-۱٦)$$

معادلات (۲۱-۱٤) و (۲۱-۱٤) مقادیر بهینه‌ی μ و σ را برای تخمین تابع برازندگی با استفاده از DACE به دست می‌دهند.

حال M ارزیابی تابع برازندگی $f(x)$ از $\{x_i\}$ در نظر بگیرید. همچنین فرض کنید توابع برازندگی مانند آنچه که در معادله‌ی (۲۱-۸) نشان داده شده است دارای همبستگی می‌باشند. فرض کنید یک ارزیابی تابع برازندگی دیگر مانند $f(x^*)$ برای یک راه‌حل نامزد دیگر مانند x^* را به دست آوریم. در این صورت بردار تابع برازندگی را بزرگ می‌نماییم تا برداری با اندازه‌ی $(M+1)$ به دست آید.

$$\tilde{f}(x)=[f^T(x)\quad f(x^*)]^T \qquad (۲۱-۱۷)$$

حال می‌توان ماتریس همبستگی را به‌صورت زیر بازنویسی نمود

$$\tilde{R}=\begin{bmatrix}R & r\\ r^T & 1\end{bmatrix} \qquad (۲۱-۱۸)$$

در معادله‌ی بالا r بردار همبستگی میان M ارزیابی تابع برازندگی $f(x)$ و ارزیابی تابع برازندگی اضافی $f(x^*)$ می‌باشد. حال می‌خواهیم PDF معادله‌ی (۲۱-۱۲) را نسبت به $f(x^*)$ ماکزیمم نماییم. با این کار تخمینی به فرم معادله‌ی (۲۱-٦) به دست خواهد آمد که بهترین تطبیق را با داده‌ی جدید $f(x^*)$ خواهد داشت. این روش، برآورد درست‌نمایی بیشینه[1] نام دارد. می‌توان PDF معادله‌ی (۲۱-۱۲) را با ماکزیمم نمودن عبارت زیر نسبت به $f(x^*)$، ماکزیمم کرد

$$\begin{aligned}&\left(\tilde{f}(x)-\mu 1_M\right)^T \tilde{R}^{-1}\left(\tilde{f}(x)-\mu 1_m\right)\\&=\begin{bmatrix}f(x)-\mu 1_m\\ f(x^*)\end{bmatrix}^T\begin{bmatrix}R & r\\ r^T & 1\end{bmatrix}^{-1}\begin{bmatrix}f(x)-\mu 1_m\\ f(x^*)\end{bmatrix}\end{aligned} \qquad (۱۹-۲۱)$$

می‌توان از نتایج لمِ معکوس نمودن ماتریس [سایمون، ۲۰۰٦، فصل ۱] استفاده نمود و نشان داد که

$$\tilde{R}^{-1}=\begin{bmatrix}R & r\\ r^T & 1\end{bmatrix}^{-1}=\frac{1}{1-r^TR^{-1}r}\begin{bmatrix}R^{-1}+R^{-1}rr^TR^{-1} & -R^{-1}r\\ -r^TR^{-1} & 1\end{bmatrix} \qquad (۲۰-۲۱)$$

با جایگزین کردن معادله‌ی بالا در معادله‌ی (۲۱-۱۹) خواهیم داشت

$$\frac{(f(x^*)-\mu)^2-2r^TR^{-1}(f(x)-\mu 1_m)(f(x^*)-\mu)}{1-r^TR^{-1}r}+f(x^*) \qquad (۲۱-۲۱)$$ عبارت‌های بدون

از آنجایی که می‌خواهیم این عبارت را نسبت به $f(x^*)$ ماکزیمم نماییم، از آن نسبت به $f(x^*)$ مشتق گرفته و برابر صفر قرار می‌دهیم

$$\frac{2(f(x^*)-\mu)-2r^TR^{-1}(f(x)-\mu 1_m)}{1-r^TR^{-1}r}=0 \qquad (۲۲-۲۱)$$

با حل معادله‌ی بالا برای $f(x^*)$ خواهیم داشت

$$f(x^*)=\mu+r^TR^{-1}(f(x)-\mu 1_m) \qquad (۲۳-۲۱)$$

این معادله نشان می‌دهد چگونه می‌توان از یک مدل حاضر برای تخمین برازندگی یک نقطه‌ی جدید مانند x^* استفاده نمود. میانگین مربعات خطای تخمین به‌صورت زیر به دست می‌آید [جونز و همکاران، ۱۹۹۸]

$$s^2(x^*)=\sigma^2\left[1-r^TR^{-1}r+\frac{(1-1_M^TR^{-1}r)^2}{1_M^TR^{-1}1_M}\right] \qquad (۲٤-۲۱)$$

[1] Maximum Likelihood Estimate

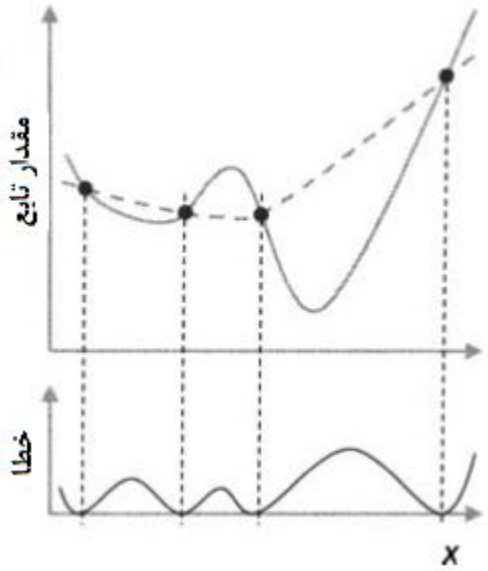
با استفاده از مقداری جبر می‌توان به سادگی نشان داد که در نقاط نمونه‌گیری شده $s(x) = 0$ می‌باشد (مسئله‌ی ۲۱-۳ را ببینید) [جونز و همکاران، ۱۹۹۸].

می‌توان از میانگین مربعات خطا جهت تعیین نقاط مناسب نمونه‌گیری برای ارزیابی‌های برازندگی جدید، استفاده نمود. در فضای جستجو دو ناحیه‌ی جذاب بالقوه برای به دست آوردن ارزیابی‌های برازندگی جدید وجود دارد. اولین ناحیه، ناحیه‌ی نزدیک یه مینیمم است به امید این که بتوان راه‌حل بهتری برای مسئله‌ی بهینه‌سازی پیدا نمود. ناحیه‌ی دوم در واقع نواحی هستند که در آن‌ها $s(x)$ بزرگ است چرا که در این نواحی عدم قطعیت بسیاری وجود دارد. شکل ۲۱-٤ این ایده را نشان می‌دهد. نمونه‌گیری از مقادیر برازندگی اضافی نزدیک مینیمم تخمین یک استراتژی انتفاع است چرا که این کار شامل جستجو در نواحی می‌شود که قبلاً دارای نتایج خوبی می‌باشند. نمونه‌گیری در نواحی با میانگین مربعات خطای بزرگ نیز یک استراتژی کاوش است چرا که این کار شامل جستجو در نواحی است که شامل اطلاعات کمی در مورد تابع برازندگی می‌باشند.

انتخاب نقاط نمونه جهت افزایش دقت مدل‌سازی، یادگیری فعال[1] نام دارد. یادگیری فعال معمولاً به معنی انتخاب نقاط نمونه در یک الگوریتم یادگیری جهت بهینه نمودن یک تابع هزینه می‌باشد. در مورد DACE که در بالا توضیح دادیم، می‌توان با استفاده از نقاط نمونه‌ی ماکزیمم، میانگین مربع خطا را کاهش داد. روش‌های شبکه عصبی اغلب شامل یادگیری فعال می‌باشند [ستلز[2]، ۲۰۱۰].

دقت اضافی تابع برازندگی را می‌توان با تخمین مقادیر بهینه‌ی $\{p_k\}$ و $\{\theta_k\}$ از معادله‌ی (۲۱-۸) به دست آورد. برای این کار می‌توان معادلات (۲۱-۱٤) و (۲۱-۱٦) را در معادله‌ی (۲۱-۱۲) جایگذاری نمود. در این صورت عبارتی از $PDF(f(X))$ به دست خواهد آمد که تنها به $\{p_k\}$ و $\{\theta_k\}$ بستگی خواهد داشت. سپس این عبارت را نسبت به $\{p_k\}$ و $\{\theta_k\}$ ماکزیمم کرده و تخمینی از مقدار بهینه‌ی ضرایب همبستگی ρ_{ij} به دست می‌آوریم. این ضرایب درایه‌های ماتریس R هستند. سپس از این مقادیر R در معادلات (۲۱-۱٤) و (۲۱-۱٦) استفاده کرده و بهترین تخمین μ و σ را به دست می‌آوریم. هر گاه که راه‌حل نامزد جدیدی مانند x^* به دست آید، از معادله‌ی (۲۱-۲۳) برای تخمین برازندگی آن استفاده می‌نماییم.

[1] Active Learning
[2] Settles



شکل ۲۱-٤ خط توپر در شکل بالا یک تابع و خط‌چین نیز تخمین تابع را نشان می‌دهد. منحنی شکل پایین، میانگین مربع خطا از معادله‌ی (۲۱-۲٤) را نشان می‌دهد. ممکن است بخواهیم با نمونه‌گیری اضافی از مقادیر تابع در نزدیکی مینیمم تخمین، تخمین خود را بهبود ببخشیم. اما در این مورد باید از جایی که خطا در بیشترین مقدار خود است نمونه‌گیری کنیم چرا که آنجا ناحیه‌ای است که در آن مینیمم تابع رخ می‌دهد.

مثال ۲۱-۱

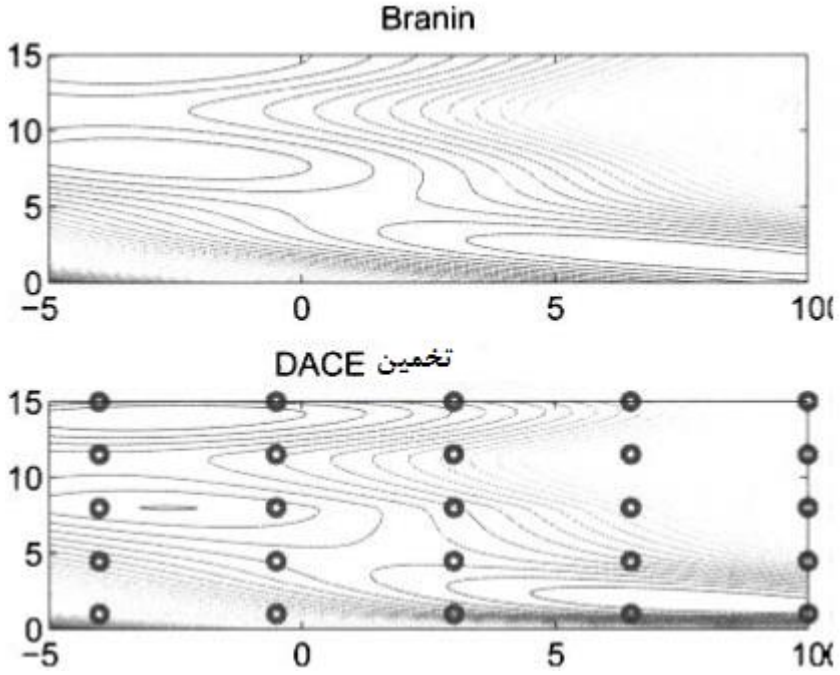
در این مثال از DACE برای تخمین تابع محک دو بعدی Branin استفاده می‌نماییم

$$f(x) = (x(2) - \left(\frac{5}{4\pi^2}\right)x(1)^2 + \frac{5x(1)}{\pi} - 6)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x(1)) + 10 \quad (۲۱-۲۵)$$

که در آن $x(1)$ و $x(2)$ دو عنصر یک راه‌حل نامزد هستند $(n = 2)$. دامنه‌ی تابع برابر $x(1) \in [-5.10]$ و $x(2) \in [0.15]$ می‌باشد. ابتدا باید تصمیم بگیریم از کدام نقاط نمونه استفاده نماییم. در این مثال به‌طور دلخواه از ۲۵ نقطه‌ی نمونه که بر روی دامنه‌ی جستجوی دو بعدی به‌صورت یکنواخت گسترده شده‌اند، استفاده می‌نماییم $(M = 25)$. سپس از تابع fmincon در MATLAB برای ماکزیمم کردن معادله‌ی (۲۱-۱۲) نسبت به $\{p_k\}$ و $\{\theta_k\}$ استفاده می‌نماییم. در این صورت خواهیم داشت

$$p_1 = 1.6194, \quad p_2 = 2 \\ \theta_1 = 0.020819, \quad \theta_2 = 0.00018011 \quad (۲۱-۲٦)$$

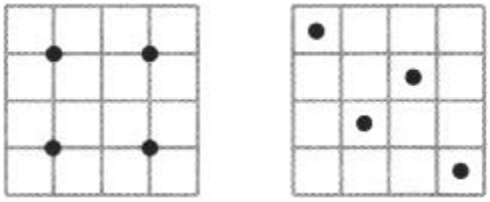
سپس از معادله‌ی (۲۱-۲۳) برای تخمین $f(x)$ استفاده می‌نماییم. شکل ۲۱-۵ نتایج را نشان می‌دهد. می‌توان دید که تخمین شکل اصلی تابع *Branin* را به خوبی به خود می‌گیرد. مهم‌تر از آن این است که تخمین شاخصه‌های چندپیمانه‌ای بودن تابع را نیز به خود می‌گیرد.



شکل ۲۱-۵ نتایج مثال ۲۱-۱. شکل بالا نمودار کنتوری تابع Branin را نشان می‌دهد. شکل پایین، ۲۵ نقطه‌ی نمونه‌ی گسترده شده به‌صورت یکنواخت و تخمین تابع Branin با استفاده از DACE را نشان می‌دهد.

در مثال ۲۱-۱ از نمونه‌گیری یکنواخت استفاده شده است. با این حال، دیگر روش‌های نمونه‌گیری ممکن است نتایج تخمین بهتری را به دست بدهند. یک روش مشهور، نمونه‌گیری فرامکعب لاتین[1] می‌باشد. در این روش یک دامنه به تعداد بازه در هر بعد تقسیم شده و سپس نقاط نمونه‌گیری را به‌گونه‌ای جایگذاری می‌نماید که هر بازه در هر بعد دارای تنها یک نقطه باشد. این روش نمونه‌گیری می‌تواند طبیعت غیر قابل پیش‌بینی و ناشناخته‌ی یک تابع را بهتر از نمونه‌گیری یکنواخت به خود بگیرد. شکل ۲۱-٦ تفاوت میان نمونه‌گیری یکنواخت و نمونه‌گیری فرامکعب لاتین را نشان می‌دهد.

[1] Latin Hypercube Sampling



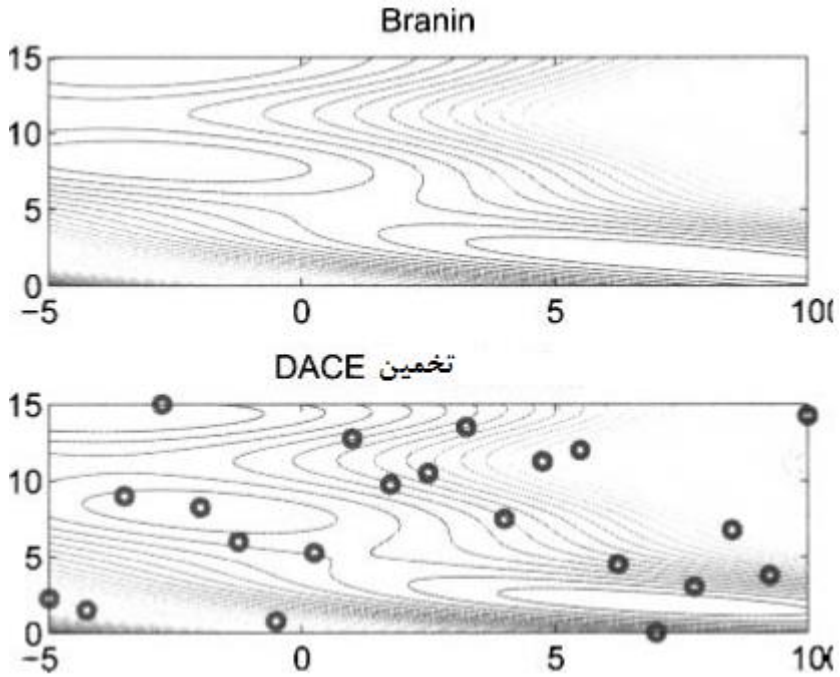
شکل ۲۱-٦ شکل سمت چپ نمونه‌گیری یکنواخت برای چهار نقطه در دامنه‌ی جستجو را نشان می‌دهد. شکل سمت راست نمونه‌گیری فرامکعب لاتین را نشان می‌دهد. توجه داشته باشید که در شکل سمت راست هر ردیف و هر ستون دارای تنها یک نقطه می‌باشد. چندین چیدمان مختلف می‌توانند این خاصیت را داشته باشند و به همین دلیل نمونه‌گیری فرامکعب لاتین یکتا نیست.

مثال ۲۱-۲

در این مثال از DACE همراه با نمونه‌گیری فرامکعب لاتین برای تخمین تابع دو بعدی Branin از مثال ۲۱-۱ استفاده می‌نماییم. در این اینجا به‌صورت دلخواه از ۲۱ نقطه‌ی نمونه ($M = 21$) استفاده می‌نماییم. سپس از تابع fmincon در MATLAB برای ماکزیمم کردن معادله‌ی (۲۱-۱۲) نسبت به به $\{p_k\}$ و $\{\theta_k\}$ استفاده می‌نماییم. در این صورت خواهیم داشت

$$\begin{matrix} p_1 = 1, & p_2 = 2 \\ \theta_1 = 0.028227, & \theta_2 = 0.0013912 \end{matrix} \qquad (۲۱-۲۷)$$

سپس از معادله‌ی (۲۱-۲۳) برای تخمین $f(x)$ استفاده می‌نماییم. شکل ۲۱-۷ نتایج را نشان می‌دهد. با مقایسه‌ی دو شکل ۲۱-۵ و ۲۱-۷ می‌توان دید که نمونه‌گیری فرامکعب لاتین تخمین بهتری را نسبت به نمونه‌گیری یکنواخت به دست می‌دهد. در حقیقت، خطای تخمین RMS در مثال ۲۱-۱ که از نمونه‌گیری یکنواخت استفاده می‌نماید برابر ۲٤-۹ بوده، در حالی که این مقدار با استفاده از نمونه‌گیری فرامکعب لاتین برابر ۱٤,۳ است. حتی با نقاط نمونه‌ی کمتر نیز نمونه‌گیری فرامکعب لاتین خطای تخمینی را به دست می‌دهد که تقریباً ۵۰٪ بهتر از نمونه‌گیری یکنواخت است. بنابراین، می‌توان دید که روش نمونه‌گیری می‌تواند تأثیر بسیاری بر نتایج تخمین DACE داشته باشد. همچنین، روشی که برای ماکزیمم کردن معادله‌ی (۲۱-۱۲) و پیدا نمودن مقادیر بهینه‌ی $\{p_k\}$ و $\{\theta_k\}$ استفاده می‌کنیم نیز می‌تواند تأثیر بسیاری بر تخمین DACE داشته باشد.



شکل ۲۱–۷ نتایج مثال ۲۱–۲. شکل بالا نمودار کنتور تابع *Branin* و شکل پایین ۲۱ نقطه‌ی نمونه، که از نمونه‌گیری فرامکعب لاتین به دست آمده‌اند و تخمین مبتنی بر DACE از تابع Branin را نشان می‌دهد.

DACE تعمیمی از الگوریتم کریجینگ است. الگوریتم کریجینگ یک روش تخمین است که به افتخار دانیل کریج، به این اسم نام‌گذاری شده است. این الگوریتم در ابتدا برای کاربردهای زمین‌شناسی به وجود آمد [کریج، ۱۹۵۱]. کریجینگ مانند DACE است تنها با این تفاوت که در کریجینگ معادله‌ی (۲۱–۸) به‌صورت زیر جایگزین می‌شود

$$d_{ij} = \sum_{k=1}^{n} \theta_k \left| x_i(k) - x_j(k) \right|^2$$
$$\rho_{ij} = Corr\left(f(x_i), f(x_j)\right) = \exp(-d_{ij}) \qquad (۲۱–۲۸)$$

این بدین معنی است که p_k از معادله‌ی (۲۱–۸) با عدد ثابت ۲ جایگزین می‌شود [چونگ و همکاران، ۲۰۰۳].

۲-۱-۲۱ تخمین توابع تبدیل یافته

گاهاً روش‌های تخمین برازندگی خوب کار نمی‌کنند. برای مثال، روش DACE از معادله‌ی (۲۱-۲۳) نیاز به معکوس نمودن ماتریس دارد. معکوس یک ماتریس ممکن است وجود نداشته باشد. اگر معکوس ماتریس وجود نداشته باشد، می‌توان از شبه معکوس استفاده نمود [گولان[1]، ۲۰۰۷]. با این حال، نکته‌ی اصلی در اینجا آن است که توابع اساسی که برای تخمین تابع برازندگی از آن‌ها استفاده می‌نماییم ممکن است برای شکل تابع برازندگی مناسب نباشند. برای مثال، اگر از یک سری فوریه برای تخمین یک تابع با رفتارهای غیرعادی و لبه‌های تیز استفاده نماییم، نمی‌توان انتظار تخمین خوب در همه‌ی نقاط دامنه‌ی تابع را داشت. در چنین مواردی می‌توان تابع برازندگی را تبدیل کرده و سپس تخمینی برای تابع تبدیل یافته پیدا نمود. برای نمونه، فرض کنید یک تابع برازندگی را در M نقطه‌ی نمونه‌ی $\{x_i\}$ ارزیابی نموده‌ایم. اگر عملکرد تخمین خوب نباشد، می‌توان نمونه‌های تابع برازندگی را با گرفتن لگاریتم طبیعیشان، تبدیل نماییم:

$$L(x_i) = \log\big(f(x_i)\big) \qquad (۲۱-۲۹)$$

سپس می‌توان با استفاده از نقاط نمونه‌ی $L(x_i)$ برای $L(x)$ تخمینی پیدا نمود. تخمین را با $\hat{L}(x)$ نشان می‌دهیم. سپس تبدیل را معکوس نموده تا تخمین تابع اصلی به دست آید:

$$f(x) = \exp\big(L(x)\big) \qquad (۲۱-۳۰)$$

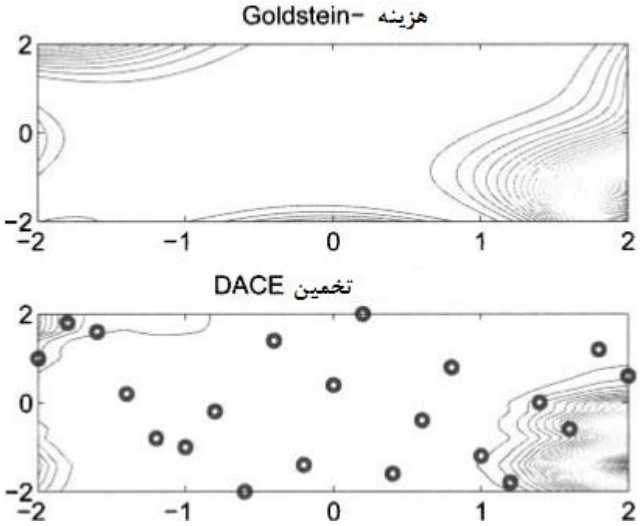
مثال ۲۱-۳

از روش DACE از بخش ۲۱-۱-۱-۲ بر روی تابع هزینه‌ی Goldstein[2] استفاده می‌نماییم [فلوداس[3] و پاردالوس، ۱۹۹۰]:

$$\begin{aligned} a &= 1 + (x(1) + x(2) + 1)^2 \\ &\quad \times (19 - 14x(1) + 3x(1)^2 - 14x(2) + 6x(1)x(2) \\ &\quad + 3x(2)^2) \\ b &= 30 + \big(2x(1) - 3x(2)\big)^2 \\ &\quad \times (18 - 32x(1) + 12x(1)^2 + 48x(2) - 36x(1)x(2) \\ &\quad + 27x(2)^2) \\ f(x) &= ab \end{aligned} \qquad (۲۱-۳۱)$$

[1] Golan
[2] Goldstein-Price Function
[3] Floudas

در معادله‌ی بالا $x(1)$ و $x(2)$ در دامنه‌ی $[-2.2]$ قرار دارند. این تابع در نزدیکی مینیمم‌اش بسیار تخت است. مینیمم در نقطه‌ی $x^*(1) = 0$ و $x^*(2) = 1$ قرار دارد. مینیمم تابع نیز $f^* = 3$ می‌باشد. ناحیه‌ی تخت باعث از بین رفتن تخمین DACE می‌شود چرا که در این ناحیه نقاط نمونه همبستگی شدیدی با یکدیگر دارند. این بدین معناست که برخی ستون‌ها در ماتریس R تقریباً کاملاً از عدد ۱ تشکیل شده‌اند و این بدین معنی است که R تقریباً مفرد[1] است. ممکن است بتوان این مسئله را با استفاده از شبه معکوس R به جای معکوس آن، حل نمود. اما در این مثال به جای این کار، از نقاط نمونه لگاریتم طبیعی می‌گیریم. با این کار شکل تابع کاملاً عوض می‌شود. گرفتن لگاریتم باعث می‌شود مقادیری از تابع که کوچک و نزدیک به هم بوده از هم جدا شده و مقادیری از تابع که بزرگ می‌باشند نیز به یکدیگر نزدیک شوند. بدین ترتیب برد کلی تابع فشرده شده و در عین حال مقادیر مشابه تابع از یکدیگر جدا شده و متمایزتر می‌شوند. سپس از تخمین DACE برای تخمین $L(x)$ استفاده کرده و بعد از آن تخمین تابع اصلی را مانند آنچه که در معادله‌ی (۲۱–۳۰) نشان داده شده است، محاسبه می‌نماییم. این اصلاح ساده در فرایند تخمین باعث می‌شود تخمین بسیار مناسبی را به دست آوریم. این تخمین در شکل ۲۱–۸ نشان داده شده است. این تخمین چندان عالی به نظر نمی‌رسد اما حداقل باعث می‌شود ماتریس R معکوس‌پذیر شود. همچنین این تخمین توانسته شکل ناحیه تخت در میانه‌ی دامنه را به خود بگیرد.



شکل ۲۱–۸ نتایج مثال ۲۱–۳. شکل بالا منحنی کنتور تابع هزینه‌ی Goldstein را نشان می‌دهد. شکل پایین نیز ۲۱ نقطه‌ی نمونه، که از نمونه‌گیری فرامکعب لاتین به دست آمده‌اند و همچنین تخمین DACE تابع را نشان می‌دهد.

[1] Singular

۲۱-۱-۳ چگونگی استفاده از تخمین برازندگی در الگوریتم‌های تکاملی

حال که الگوریتم تخمین برازندگی را در اختیار داریم، چند روش برای چگونگی استفاده از آن در یک الگوریتم تکاملی در اختیار داریم [جین، ۲۰۰۵]. ابتدا، می‌توان به سادگی جزیی دلخواه مانند r عدد از ارزیابی‌های برازندگی را با تخمین‌های برازندگی جایگزین نمود. با فرض اینکه ارزیابی تابع برازندگی حجم غالبی از تلاش محاسباتی را به خود اختصاص می‌دهد، این کار می‌تواند تلاش محاسباتی را از E به $(1-r)E$ کاهش دهد. با این حال، در استفاده از این ایده نباید زیاده‌روی کرد. اگر تعداد زیادی از ارزیابی‌های برازندگی را با تخمین‌ها جایگزین نماییم، آنگاه همگرایی الگوریتم تکاملی بسیار دیرتر صورت گرفته و تلاش ما برای کاهش محاسبات ممکن است به جای اثر سازنده، اثر مخرب داشته باشد. در حالت‌های حاد، اگر تمامی ارزیابی‌های برازندگی را با تخمین‌ها جایگزین نماییم، $r = 1$ خواهد بود. در این صورت تلاش محاسباتی تقریباً صفر خواهد بود، اما الگوریتم تکاملی هیچگاه به نتیجه‌ی مفیدی همگرا نخواهد شد.

یک راه دیگر آن است که در هر نسل تعدادی فرزند اضافی تولید نماییم و از مقادیر برازندگی تخمینی آن‌ها جهت تصمیم‌گیری در مورد اینکه کدام یک از آن‌ها را برای نسل بعد نگه داریم، استفاده نماییم. این ایده کنترل تکامل و یا مدیریت مدل نام دارد. اگر برازندگی برخی ذرات به‌صورت دقیق و برخی دیگر به‌صورت تخمینی برآورد شود، به این فرایند کنترل تکامل ذرات گفته می‌شود [شی و رشید، ۲۰۱۰]. می‌توان از روش‌های مختلفی برای تصمیم‌گیری در مورد اینکه کدام ذرات به‌صورت دقیق و کدام ذرات به‌صورت تخمینی ارزیابی شوند، استفاده نمود. برای نمونه، می‌توان این کار را به‌صورت اتفاقی انجام داد. همچنین می‌توان ارزیابی دقیق را تنها در مورد ذراتی که در هر نسل دارای برازندگی تخمینی خوبی می‌باشند، استفاده نمود. ذراتی که برازندگیشان به‌صورت دقیق ارزیابی می‌شود، ذرات کنترل شده نام دارند.

اگر در یک نسل تمامی ذرات با تابع برازندگی دقیق و در نسل دیگر تمامی ذرات به‌صورت تخمینی ارزیابی شوند، کنترل تکامل نسل-محور به دست خواهد آمد. در اینجا نیز می‌توان در مورد اینکه ارزیابی دقیق و تخمینی در کدام نسل‌ها به کار روند، از روش‌های مختلفی استفاده نمود. برای نمونه، می‌توان از ارزیابی دقیق تنها هر k نسل یکبار استفاده نمود. در این صورت k پارامتری خواهد بود که توسط کاربر تعیین خواهد شد. همچنین می‌توان تا زمانی که همگرایی تشخیص داده شود از ارزیابی تخمینی استفاده نمود (برای مثال بهترین ذره برای چند نسل مشخص بهبودی نداشته و یا انحراف استاندارد جمعیت کمتر از یک میزان مشخصی شده است)، و سپس در نسل بعد از ارزیابی دقیق استفاده نمود. پس از آن، می‌توان دوباره به ارزیابی تخمینی بازگشت. نسل‌هایی که در آن‌ها از برازندگی دقیق برای ارزیابی تمام ذرات استفاده می‌نماییم، نسل‌های کنترل شده نام دارند.

محققین انواع مختلفی از کنترل تکامل از جمله الگوریتم تکاملی ترکیبی مبتنی بر تخمین پویای برازندگی (DAFHEA[1]) [بهاتاچاریا، ۲۰۰۸] معرفی نموده‌اند. این الگوریتم در شکل ۲۱-۹ نشان داده شده است.

اندازه جمعیت $= N$
$N_c \leftarrow 5N$
N_c دره‌ی اتفاقی را تولید کن
برازندگی N_c ذره را محاسبه کن
از N_c برازندگی به دست آمده برای ایجاد تخمین برازندگی $\hat{f}(.)$ استفاده کن
بهترین N ذره‌ی جمعیت را نگه دار
تا زمانی که شرایط توقف برآورده نشده است
  از یک الگوریتم تکاملی برای ایجاد N_c فرزند استفاده کن
  از $\hat{f}(.)$ برای تخمین برازندگی فرزندان استفاده کن
  بهترین N فرزند را برای نسل بعد نگه دار
  اگر زمان تخمین جدید فرا رسیده است
    برازندگی $f(x_i)$ را برای هر ذره‌ی x_i محاسبه کن
    از مقادیر برازندگی برای به‌روزرسانی $\hat{f}(.)$ استفاده کن
  پایان اگر
نسل بعدی

شکل ۲۱-۹ طرح کلی الگوریتم تکاملی ترکیبی مبتنی بر تخمین پویای برازندگی (DAFHEA).

با این که در شکل ۲۱-۹ نشان داده نشده است، اما DAFHEA معمولاً شامل نخبه‌گرایی هم می‌شود. می‌توان تنوعات زیادی را برای شکل ۲۱-۹ در نظر گرفت. برای نمونه، می‌توان مقادیری غیر از $5N$ را برای N_c در نظر گرفت. همچنین می‌توان از الگوریتم‌های متنوعی برای تخمین برازندگی استفاده نمود. اگر چه در نسخه‌ی اصلی DAFHEA از ماشین‌های بردار پشتیبان برای این منظور استفاده می‌گردد، می‌توان از روش‌های مختلفی برای تصمیم‌گیری در مورد زمان استفاده از تخمین جدید استفاده نمود. برخی شروط معمول برای این تصمیم‌گیری شامل تعداد ثابتی نسل و یا شرایط خاص همگرایی می‌باشد.

[1] Dynamic Approximate Fitness Hybrid EA

همچنین می‌توان از نواحی اعتماد[1] [بتز[2]، ۲۰۰۹] جهت تصمیم‌گیری در مورد زمان استفاده از یک تخمین جدید، استفاده نمود. روش نواحی اعتماد بر پایه‌ی مقایسه‌ی میان مقادیر تخمینی برازندگی و مقادیر واقعی آن قرار دارد. اگر مقادیر تخمینی به مقادیر واقعی نزدیک باشند، آنگاه می‌توان نتیجه گرفت که تخمین خوب است و در این صورت می‌توان زمان میان تخمین‌ها را کاهش داد. با این حال، اگر مقادیر تخمینی به مقادیر واقعی نزدیک نباشند، آنگاه می‌توان تصمیم گرفت که تخمین ضعیف است و در این صورت باید زمان میان تخمین‌ها را کاهش داد. فرض کنید G تعداد نسل‌های میان محاسبات یک تخمین برازندگی جدید را نشان دهد. در شکل ۲۱-۹ در هر نسل N_c فرزند خواهیم داشت. در این صورت مقدار دقیق برازندگی N_e فرزند را محاسبه کرده و آن را با مقدار برازندگی تخمینیشان مقایسه می‌نماییم. اگر اختلاف RMS از یک مقدار مشخص T^+ بیشتر شود، آنگاه G را کاهش می‌دهیم. اگر این اختلاف از یک حد مشخص T^- کمتر شود، آنگاه G را افزایش می‌دهیم. مقادیر T^+ و T^- باید میزان شوند تا تعادل خوبی میان تلاش محاسباتی کاهش یافته (G بزرگ) و تخمین برازندگی دقیق (G کوچک) برقرار شود. در این رویکرد N_e نیز باید به‌گونه‌ای مناسب میزان شود.

شکل ۲۱-۹ یک شبه کد کلی است. به‌طور مشخص‌تر، می‌توان از تخمین برازندگی برای تصمیم‌گیری در مورد این که چه تعداد از N_c ذره‌ی اولیه را در جمعیت نگه داریم، استفاده نماییم. این کار، مقداردهی اولیه‌ی آگاهانه نام دارد. در این روش می‌خواهیم پایه‌ی تخمین برازندگی را بر چیزی غیر از ارزیابی‌های برازندگی N_c ذره‌ی اولیه قرار دهیم. برای مثال، می‌توان یک الگوریتم تخمین برازندگی را به‌صورت آفلاین و قبل از اجرای الگوریتم تکاملی بسازیم.

همچنین می‌توان از تخمین برازندگی در شکل ۲۱-۹ تنها برای عمل برش (در صورتی که الگوریتم تکاملی GA باشد) و یا تنها برای مهاجرت (اگر الگوریتم تکاملی BBO باشد) و یا تنها برای الگوریتم بازترکیب، بسته به نوع الگوریتم تکاملی، استفاده نمود. در این صورت از بازترکیب (برش، مهاجرت، و یا هر نوع بازترکیب دیگری) برای ایجاد تعداد زیادی فرزند (بیش از N) استفاده کرده اما تنها N فرزند برتر را، بسته به مقادیر برازندگی تخمینی آن‌ها، نگه خواهیم داشت. این کار، برای مثال، برش آگاهانه یا مهاجرت آگاهانه نام خواهد داشت.

همچنین می‌توان از تخمین برازندگی در شکل ۲۱-۹ تنها برای عمل جهش در الگوریتم تکاملی استفاده نمود. در این صورت، تعداد زیادی نسخه‌های جهش یافته از فرزندان (بیش از N) تولید خواهد شد، اما تنها

[1] Trust Regions
[2] Betts

N نسخه‌ی برتر، بسته به مقادیر برازندگی تخمینی آن‌ها برای نسل بعد نگه داشته خواهند شد. این کار جهش آگاهانه نام داشته و مشابه ایده‌ای است که در بخش ١٦-٤ در پیاده‌سازی یادگیری مقابله-محور به کار رفت.

بسته به این که می‌خواهیم چه مقدار اطلاعات از تخمین قبل را نگه داریم، می‌توان از الگوریتم‌های مختلفی برای به روزرسانی $\hat{f}(.)$ در انتهای DAFHEA استفاده نمود. این کار به میزان پویایی تابع برازندگی بستگی دارد. می‌توان از چندین مدل تخمین برازندگی استفاده کرده و و بسته به دقت آن‌ها بین مدل‌ها سوئیچ نماییم. این مشابه تخمین چند مدلی است که در بخش بعد مورد بحث قرار خواهد گرفت. با این حال، در فصل بعد بیشتر بر ترکیب مدل‌های با دقت زیاد با مدل‌های با دقت کم تمرکز خواهیم کرد.

در آخر، می‌توان شکل ٢١-٩ را با مقدار کمی تغییر به‌گونه‌ای در آورد که در آن از تخمین برازندگی تنها برای تصمیم‌گیری در مورد اینکه کدام فرزندان برای نسل بعد نگه داشته شوند، استفاده شود. این روش، رویکرد اپراتور آگاه[1] نام دارد [رشید و هیرش[2]، ٢٠٠٠] و در شکل ٢١-١٠ نشان داده شده است.

اندازه جمعیت $= N$
$N_c \leftarrow 5N$
N_c دره‌ی اتفاقی را تولید کن
برازندگی N_c ذره را محاسبه کن
از N_c برازندگی به دست آمده برای ایجاد تخمین برازندگی $\hat{f}(.)$ استفاده کن
بهترین N ذره‌ی جمعیت را نگه دار
تا زمانی که شرایط توقف برآورده نشده است
  از یک الگوریتم تکاملی برای ایجاد N_c فرزند استفاده کن
  از $\hat{f}(.)$ برای تخمین برازندگی فرزندان استفاده کن
  بهترین N فرزند را برای نسل بعد نگه دار
  برازندگی $f(x_i)$ را برای هر ذره‌ی x_i محاسبه کن
  از مقادیر برازندگی برای به‌روزرسانی $\hat{f}(.)$ استفاده کن
نسل بعدی

شکل ٢١-١٠ طرح کلی الگوریتم اپراتور آگاه.

[1] Informed Operator
[2] Hirsh

۲۱-۱-۴ مدل‌های چندگانه

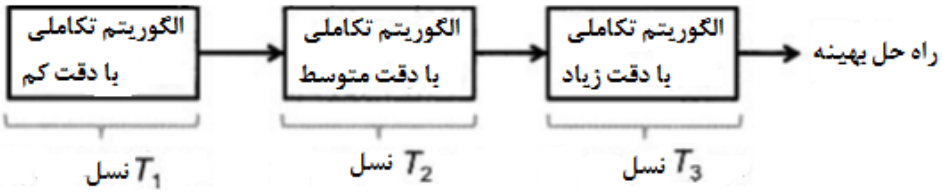
می‌توان از ارزیابی تخمینی تابع برازندگی در نسل‌های ابتدایی و از ارزیابی دقیق تابع برازندگی در نسل‌های آخر استفاده نمود. این ایده مشابه استفاده از زیرمجموعه‌ای از نمونه‌های آزمایشی برای ارزیابی تابع برازندگی، که پیش از این در مورد آن بحث نمودیم، می‌باشد. در اینجا نیز از همان نمونه‌ها استفاده می‌کنیم با این تفاوت که در نسل‌های ابتدایی الگوریتم تکاملی از ارزیابی‌های با دقت کمتر استفاده می‌نماییم. به‌عنوان یک مثال خاص، فرض کنید که ارزیابی تابع برازندگی شامل حل نمودن معادله‌ی Riccati می‌شود:

$$P = FPF^T - FPH^T(HPH^T + R)^{-1}HPF^T + Q \qquad (۳۲-۲۱)$$

این نوع معادله معمولاً در مسائل کنترل و تخمین وجود داشته و به همین دلیل بیشتر ممکن است در الگوریتم‌های تکاملی که قصد بهینه‌سازی کنترلرها و تخمین‌زن‌ها را دارند، ظاهر شود [سایمون، ۲۰۰۶]. با داشتن ماتریس‌های مربعی F، Q و R و ماتریس H، که احتمالاً مربعی نخواهد بود، باید معادله‌ی بالا را برای ماتریس مربعی P حل نماییم. عملکرد یک الگوریتم تخمین یا یک ماتریس کنترل با اثر ماتریس P متناسب است. حل معادله‌ی Ricatti می‌تواند به لحاظ محاسباتی پرهزینه باشد اما می‌توان از روش‌های تقریبی برای تخمین زدن راه‌حل‌ها استفاده نمود [امره[1] و نولز، ۱۹۸۷]. در نسل‌های ابتدایی الگوریتم تکاملی می‌توان از تقریب برای راه‌حل معادله‌ی (۲۱-۳۲) استفاده نمود. در نسل‌های انتهایی نیز می‌توان از تقریب‌های دقیق‌تر استفاده نمود.

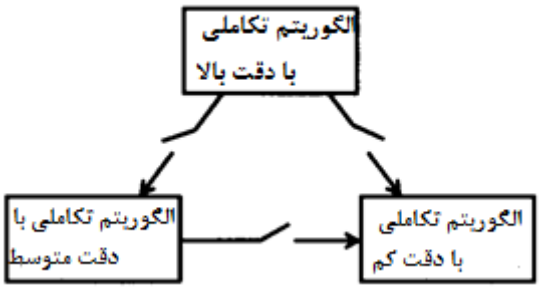
شکل ۲۱-۱۱ این فرایند را نشان می‌دهد. در ابتدای الگوریتم تکاملی و برای T_1 نسل، یک مدل برازندگی با دقت کم اجرا می‌شود. پس از T_1 نسل از جمعیت الگوریتم تکاملی برای مقداردهی اولیه‌ی الگوریتم تکاملی بعدی، که از مدلی با دقت متوسط برای T_2 نسل بهره خواهد برد، استفاده می‌شود. پس از این T_2 نسل نیز از جمعیت نهایی الگوریتم تکاملی برای مقداردهی اولیه‌ی جمعیت آخرین الگوریتم تکاملی استفاده خواهد شد. آخرین الگوریتم تکاملی از مدلی با دقت بالا برای T_3 نسل استفاده خواهد کرد. این ایده را می‌توان به هر تعداد سطح دقت تعمیم داد. در این رویکرد باید مراقب باشیم که هر الگوریتم تکاملی با جمعیتی متنوع مقداردهی اولیه شود. برای این کار می‌توان ابتدا با اندازه‌گیری از تنوع کافی جمعیت الگوریتم تکاملی اطمینان حاصل نمود و سپس از تخمین تابع برازندگی در سطح بعد استفاده نمود. همچنین می‌توان این کار را با انتقال تنها تعدادی از ذرات الگوریتم تکاملی با دقت کم به الگوریتم تکاملی با دقت بالا انجام داد. در این صورت، باقی جمعیت الگوریتم تکاملی با دقت بالا به‌گونه‌ای مقداردهی خواهد شد که تنوع در جمعیت حفظ شود.

[1] Emre



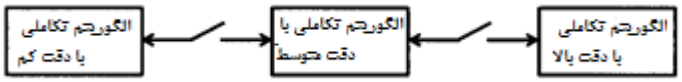
شکل ۲۱-۱۱ تخمین برازندگی مدل‌های چندگانه. الگوریتم‌های تکاملی با سطوح مختلف تخمین تابع برازندگی به‌صورت سریالی اجرا می‌شوند.

رویکرد یکپارچه‌تری از بهینه‌سازی مدل‌های چندگانه وجود دارد که در آن سطوح مختلف تخمین‌های تابع برازندگی به‌صورت موازی اجرا می‌شوند. در این رویکرد، ذرات با فرکانس‌های مشخص میان الگوریتم‌های تکاملی موازی جابه‌جا می‌شوند [سفریوی[1] و پریو[2]، ۲۰۰۰]. این رویکرد، که محاسبات تکاملی سلسله‌مراتبی نام دارد، شامل چند گزینه‌ی مختلف می‌شود. در گزینه‌ی اول، می‌توان ذرات را از الگوریتم‌های تکاملی با دقت بیشتر به الگوریتم‌های تکاملی با دقت کمتر مهاجرت داد. این روش در شکل ۲۱-۱۲ نشان داده شده است. حالت دیگر آن است که مانند شکل ۲۱-۱۳ ذرات را به‌طور متناوب میان الگوریتم‌های تکاملی با سطوح یکسان تخمین برازندگی حرکت داد. توجه داشته باشید که در الگوریتم‌های تکاملی سلسله‌مراتبی نیز می‌توان تعداد دلخواهی از سطوح دقت مدل را تعریف نمود.



شکل ۲۱-۱۲ الگوریتم تکاملی سلسله‌مراتبی. در این مدل ذرات را از الگوریتم‌های تکاملی با دقت تخمین برازندگی بیشتر به الگوریتم‌های تکاملی با دقت تخمین برازندگی کمتر مهاجرت می‌دهد. فرکانس این جابه‌جایی توسط کاربر تعیین می‌شود.

[1] Sefrioui
[2] Pariaux



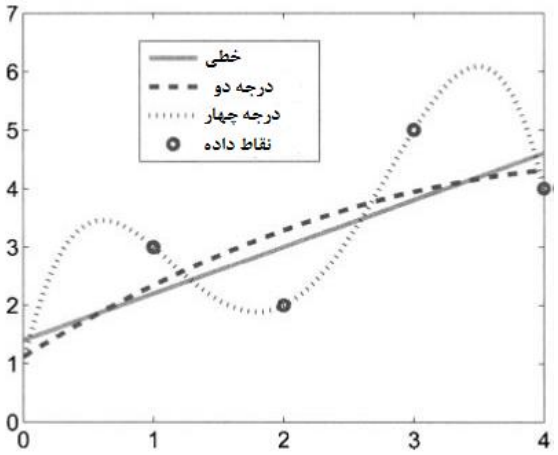
شکل ۲۱-۱۳ الگوریتم تکاملی سلسله‌مراتبی. در این مدل ذرات میان الگوریتم‌های تکاملی با سطوح تخمین برازندگی مشابه جابه‌جا می‌شوند. فرکانس این جابه‌جایی توسط کاربر تعیین می‌گردد.

یک راه دیگر برای استفاده از مدل‌های چندگانه، تولید مدل‌های چندگانه‌ای است که بتوان ترکیبات مختلفی از آن‌ها را برای هر ذره‌ی x مورد استفاده قرار داد. برای مثال، فرض کنید برازندگی M ذره را $\{x_i\}$ را ارزیابی نموده‌ایم. می‌توان از یک الگوریتم خوشه‌بندی استفاده نمود و $\{x_i\}$ را به C خوشه تقسیم نمود. با این کار $\hat{f}_k(x)$ برای $k \in [1.C]$ به دست خواهد آمد. حال برای تخمین برازندگی یک ذره مانند x می‌توان از یکی از چندین رویکرد ممکن استفاده نمود. برای مثال، می‌توان $f(x)$ را با $\hat{f}_k(x)$ تخمین زد به‌طوری که اندیس k اندیس خوشه‌ای است که به x نزدیکتر است [چانگ و آلونسو[1]، ۲۰۰٤]. متناوباً می‌توان $f(x)$ را با ترکیبی وزنی از مقادیر $\hat{f}_k(x)$ تخمین زد به‌گونه‌ای که جمع وزن‌ها برابر ۱ بوده و هر وزن تابعی از فاصله‌ی x تا خوشه‌ی مربوطه خواهد بود.

۲۱-۱-۵ بیش برازش

بیش برازش ممکن است در برخی رویکردهای تخمین برازندگی مشکل‌زا باشد. طراح الگوریتم تکاملی باید همیشه در هنگام استفاده از روش‌های تخمین برازندگی، نسبت به بیش برازش بدگمان باشد. بیش برازش اغلب در شبکه‌های عصبی یک مشکل است مگر آنکه مهندس خود عمداً از آن اجتناب نماید [کرو[2]، ۲۰۰۸]. شکل ۲۱-۱٤ مثالی از بیش برازش در هنگام تطبیق دادن یک منحنی به تعدادی نقاط را نشان می‌دهد. اگرچه در این شکل چندجمله‌ای با درجه‌ی بالاتر بهتر از چندجمله‌ای با درجه‌ی پایین‌تر بر داده‌ها منطبق می‌گردد، اما چندجمله‌ای با درجه‌ی بالاتر به خوبی تعمیم نمی‌یابد. به عبارتی می‌توان گفت که نقاط داده را به خاطر می‌سپارد اما عملکرد خوبی را در نقاط میان نقاط داده از خود نشان نمی‌دهد. برای دستیابی به عملکرد کلی بهتر باید به خطای تطبیق بیشتر در محل نقاط داده تن بدهیم.

[1] Alonso
[2] Krogh



شکل ۲۱-۱٤ این شکل مثالی از بیش برازش را نشان می‌دهد. یک تابع خطی و یک تابع درجه دوم بسیار خوب بر داده منطبق شده‌اند. تابع درجه چهارم بر داده بسیار خوب منطبق شده است اما شامل نوسانات بزرگی است و این نشان می‌دهد که این مدل نمی‌تواند به خوبی تعمیم بیابد.

بیش برازش را می‌توان با تکنیک‌های آنسامبل کاهش داد. آنسامبل گروهی از تخمین‌های برازندگی آموزش داده شده‌ی به‌صورت فردی هستند که پیش‌بینی‌هایشان هنگام تخمین مقادیر برازندگی برای نقاطی که قبلاً با آن‌ها برخورد نداشته‌ایم، ترکیب می‌شود [اپیتز[1] و ماکلین[2]، ۱۹۹۹]، [لیم[3] و همکاران، ۲۰۱۰].

۲۱-۱-۶ ارزیابی روش‌های تخمین

پس از آنکه تخمینی از یک تابع را به دست آوردیم باید مطمئن شویم نتایج خوبی را به دست می‌دهد و سپس آن را در یک الگوریتم تکاملی به کار ببریم. قدم اول همیشه آن است که مقادیر تخمین را در نقاط نمونه بررسی نماییم (نقاط نمونه نقاطی هستند که از آن‌ها در ایجاد تخمین استفاده نموده‌ایم). توابع خروجی بسیاری از روش‌های تخمین به‌صورت خودکار و کاملاً دقیق بر تابع دقیق $f(x)$ در M نقطه‌ی نمونه‌ی $\{x_i\}$ منطبق می‌شوند. با این حال، این موضوع باید بررسی شود تا مطمئن شویم الگوریتم تخمین به درستی پیاده‌سازی شده است.

یک روش برای ارزیابی دقت روش تخمین آن است که از چند نقطه‌ی دیگر به غیر از نقاط مورد استفاده در هنگام ایجاد تخمین استفاده نماییم. فرض کنید Q نقطه‌ی نمونه‌ی $\{x_i\}$ اضافی انتخاب نماییم و آن‌ها را

[1] Opitz
[2] Maclin
[3] Lim

نقاط آزمایش بنامیم. در این صورت $i \in [M+1.M+Q]$ خواهد بود. سپس تابع و تخمین را در محل نقاط آزمایش ارزیابی کرده و نحوه‌ی عمکرد تخمین را برآورد می‌نماییم. خطای RMS تخمین به‌صورت زیر به دست می‌آید

$$E_{RMS}^2 = \frac{1}{Q}\sum_{i=M+1}^{M+Q}\left(f(x_i) - \hat{f}(x_i)\right)^2 \qquad (۳۳-۲۱)$$

خطای بدترین حالت تخمین نیز به‌صورت زیر به دست می‌آید

$$E_{max} = \max_{i\in[M+1,M+Q]}\left|f(x_i) - \hat{f}(x_i)\right| \qquad (۳٤-۲۱)$$

می‌توان از هر متریکی برای ارزیابی کیفیت یک روش تخمین استفاده نمود. این متریک به اولویت‌های ما و همچنین به نوع مسئله بستگی دارد.

در یک روش دیگر، که تایید متقابل[1] یا تخمین چرخشی نام دارد [گیسر[2]، ۱۹۹۳]، جهت ارزیابی کیفیت تخمین به نقاط نمونه‌ی اضافی نیازی نیست. مانند بالا فرض کنید M نقطه‌ی نمونه و M مقدار تابع $f(x_i)$ در اختیار داریم. در روش تایید متقابل، از تمام نقاط نمونه به غیر از نقطه‌ی kاُم جهت محاسبه‌ی تخمین استفاده شده و این تخمین با $\hat{f}_k(x)$ نشان داده می‌شود. بدین ترتیب، M تخمین مختلف را حساب کرده و هر بار یکی از نقاط نمونه را جا می‌اندازیم. در این صورت M تخمین $\hat{f}_k(x)$ برای $k \in [1.M]$ خواهیم داشت به‌گونه‌ای که هر تخمین از مجموعه‌ای یکتا از $(M-1)$ نقطه‌ی نمونه استفاده خواهد نمود. سپس، هر تخمین را در نقطه‌ای که هنگام محاسبه‌ی تخمین جا انداخته بودیم ارزیابی می‌شود. به عبارت دیگر $\hat{f}_k(x_k)$ را برای $k \in [1.M]$ ارزیابی می‌نماییم. همانند بالا، خطای تخمین RMS و خطای بدترین حالت به‌صورت زیر محاسبه خواهد شد

$$E_{RMS}^2 = \frac{1}{M}\sum_{i=1}^{M}\left(f(x_i) - \hat{f}_i(x_i)\right)^2$$
$$E_{max} = \max_{i\in[1,M]}\left|f(x_i) - \hat{f}(x_i)\right| \qquad (۳۵-۲۱)$$

پس از آنکه با استفاده از تأئید متقابل مطمئن شدیم که رویکرد تخمین مورد استفاده درست است، از تمام M نقطه برای محاسبه‌ی تابع تخمین $\hat{f}(x)$ استفاده خواهیم نمود.

[1] Cross Validation
[2] Geisser

اگر منابع مورد نیاز برای مقایسه‌ی مقادیر تخمینی با مقادیر دقیق تابع را در اختیار داشته باشیم، می‌توانیم از متریک‌های دیگری برای ارزیابی کیفیت تخمین استفاده نماییم. یکی از این متریک‌ها، مقایسه‌ی تعداد ذرات استفاده شده برای بازترکیب در هنگام استفاده از مقادیر واقعی تابع برازندگی با ذراتی که هنگام استفاده از مقادیر تخمینی از آن‌ها بهره برده شده است، می‌شود. همچنین می‌توان از همبستگی میان مقادیر واقعی و تخمینی برازندگی، و یا همبستگی میان رتبه‌های تخمینی و واقعی برازندگی برای این منظور استفاده نمود [جین و همکاران، ۲۰۰۳].

قسمت‌های قبل به بحث در مورد چند روش مختلف برای تخمین تابع برازندگی پرداختند. همچنین روش‌های بسیار دیگری نیز وجود دارند که در مورد آن‌ها صحبت ننمودیم. هر یک از این روش‌ها دارای تنوعات و پارامترهای میزان‌سازی مختلفی می‌باشند. نمی‌توان به سادگی "بهترین" روش تخمین تابع برازندگی را تعیین نمود. علاوه بر خطای تخمین RMS و بیشترین خطا، که در معادله‌ی (۲۱-۳۵) تعریف نمودیم، شاخص‌های دیگری نیز هستند که هنگام ارزیابی روش‌های تخمین تابع برازندگی باید در نظر بگیریم.

- یک روش تخمین برای یک مسئله‌ی خاص چه قدر دقیق است؟
- جدا از دقت روش تخمین، عملکرد الگوریتم تکاملی هنگام استفاده از روش تخمین چگونه خواهد بود؟ توجه داشته باشید که عملکرد نسبی الگوریتم‌های تکاملی متفاوت ممکن است با روش‌های مختلف فرق کند. برای مثال، الگوریتم تکاملی اول ممکن است با روش تخمین A بهترین عملکرد را داشته باشد اما الگوریتم تکاملی دوم با روش تخمین B بهترین عملکرد خود را نشان دهد.
- یک روش تخمین تلاش محاسباتی را چه قدر کاهش می‌دهد؟
- پیچیدگی روش تخمین چه قدر است؟ این موضوع بر تعمیم‌پذیری، نگه‌داشت‌پذیری و انتقال‌پذیری کد تأثیر می‌گذارد. اصلاح کد آسان است یا دشوار (نگه‌داشت‌پذیری)؟ اضافه نمودن ویژگی‌ها و توابع جدید به کد آسان است یا دشوار (تعمیم‌پذیری)؟ اعمال کد به سایر مسائل بهینه‌سازی آسان است یا دشوار (انتقال‌پذیری)؟

نخبه‌گرایی

در آخر، باید توجه کنیم که قاعده‌ی استفاده از نخبه‌گرایی در الگوریتم‌های تکاملی ممکن است شامل توابع برازندگی گران نشود. تاکنون توصیه نموده‌ایم از نخبه‌گرایی استفاده شود تا بهترین ذرات از یک نسل به نسل دیگر حفظ شوند. با این حال، الگوریتم‌های تکاملی که به تعداد کمی ارزیابی تابع نیاز دارند ممکن

است بدون نخبه‌گرایی عملکرد بهتری داشته باشند [تورگوسا[1] و کانوک-نوکولچای[2]، ۲۰۰۲]. این موضوع به این دلیل است که وقتی اندازه‌ی جمعیت یا تعداد ارزیابی‌ها کوچک است، کاوش نسبت به انتفاع از اهمیت بیشتری برخوردار است. الگوریتم‌های تکاملی بدون نخبه‌گرایی می‌توانند کاوش را افزایش دهند.

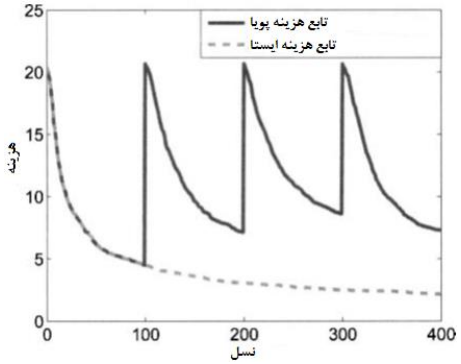
۲۱-۲ توابع برازندگی پویا

توابع برازندگی اغلب متغیر با زمان بوده و غیرایستا هستند. تغییر آن‌ها گاهی از متغیر بودن محیط آزمایش برازندگی ناشی می‌شود. برای مثال، ممکن است در هنگام میزان‌سازی کنترلر یک روبات، وظایف روبات با زمان تغییر کند.. همچنین پارامترهای خود روبات نیز ممکن است به دلیل استهلاک سنسورها و قطعات الکترودینامیکی آن، عوض شوند. گاهاً نیز نیازهای مشتری و یا کاربر با زمان تغییر می‌کند، بدین معنی که ممکن است مردم نظرشان در مورد چیزی که می‌خواهند عوض شود. در برخی دیگر از مواقع، محدودیت‌ها و قیدها به دلیل مصرف شدن منابع تغییر می‌کنند. این بخش به بحث در مورد نحوه‌ی استفاده از الگوریتم‌های تکاملی جهت دنبال نمودن بهینه‌ی پویا می‌پردازد.

مثال ۲۱-٤

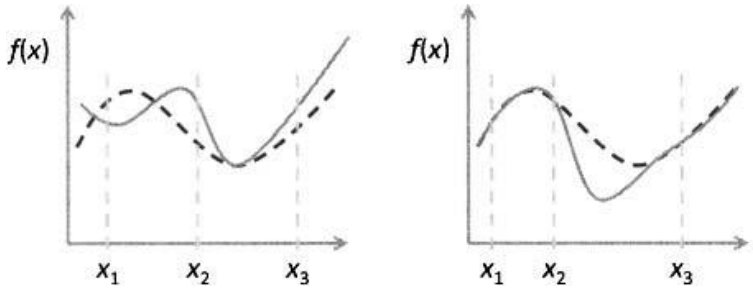
این مثال یک الگوریتم تکاملی پویا را نشان نمی‌دهد بلکه تأثیرات پویایی بر عملکرد الگوریتم تکاملی در یک تابع بهینه‌سازی محک را نشان می‌دهد. در این مثال از مولد تابع محک پویای ساده‌شده‌ی شکل ج.۲٤ و از تابع *آکلی* به‌عنوان تابع پایه استفاده می‌نماییم. هر ۱۰۰ نسل یکبار پویایی را به تابع اعمال نموده (نسل $E_{update} = 100$) و ابعاد مسئله را برابر ۱۰ فرض می‌نماییم. یک الگوریتم BBO از فصل ۱٤ را با اندازه‌ی جمعیت ۵۰ و پارامتر نخبه‌گرایی ۲ اجرا نموده و ذرات تکراری در هر نسل را با ذرات اتفاقی جایگزین می‌نماییم. شکل ۲۱-۱۵ عملکرد BBO بر روی تابع *آکلی* ایستا و تابع پویای *آکلی* را، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. عملکرد الگوریتم برای ۱۰۰ نسل ابتدایی برای هر دو حالت یکسان است اما می‌توان دید که الگوریتم BBO باید هر ۱۰۰ نسل یکبار برای تابع پویا از اول شروع کند چرا که تابع شدیدا دچار تغییر می‌شود. این مثال نیاز به روش‌های هوشمندی که بتوانند تغییرات پویا در تابع هزینه را مدیریت نمایند، به خوبی نشان می‌دهد.

[1] Torregosa
[2] Kanok-Kunulchai



شکل ۲۱-۱۵ مثال ۲۱-٤: این شکل عملکرد BBO بر روی تابع ۱۰ بعدی ایستا و پویای *آکلی* را، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. پویایی هر ۱۰۰ نسل یکبار به نسخه‌ی پویای تابع محک اعمال می‌شود. این موضوع باعث می‌شود BBO تمام پیشرفت‌هایی را که در مورد این تابع داشته است، از دست بدهد.

اولین چالش پیش رو در بهینه‌سازی پویا، تشخیص تغییر در فضای تابع برازندگی می‌باشد. این تغییر را می‌توان با استفاده از برخی ذرات نشانگر و ارزیابی مقدار برازندگیشان در هر نسل، تشخیص داد. اگر مقادیر برازندگی این ذرات از یک نسل به نسل دیگر به طرز قابل ملاحظه‌ای تغییر نماید به‌طوری که نتوان نویز را عامل آن دانست، می‌توان نتیجه گرفت که فضای برازندگی تغییر نموده است. این خود به این معنی خواهد بود که مسئله‌ی بهینه‌سازی تغییر نموده است. اما استفاده از نشانگرها روش کاملاً عاری از اشتباه نخواهد بود. برای مثال، ممکن است فضای برازندگی در نقاط نشانگر تغییر کرده در حالی که در نقطه‌ی بهینه ثابت بماند. بر عکس این حالت نیز ممکن است به این معنی که ممکن است فضای برازندگی در نقطه‌ی نشانگرها ثابت مانده و در نقطه‌ی بهینه تغییر نماید. این مشکلات در شکل ۲۱-٦ به تصویر کشیده شده‌اند.



شکل ۲۱-۱٦ تشخیص تغییرات در تابع برازندگی پویا. در هر شکل منحنی خط‌چین نشان‌دهنده‌ی تابع اصلی بوده و خط توپر نیز تابع برازندگی جدید را نشان می‌دهد. سه نقطه‌ی x_1، x_2 و x_3 نقاط نشانگر هستند که از آن‌ها برای تشخیص تغییر تابع برازندگی استفاده می‌نماییم. شکل سمت چپ نشان می‌دهد که اگرچه برازندگی نقاط نشانگر شدیداً تغییر کرده است، نقطه بهینه تغییری نداشته است. شکل سمت راست نیز عکس این موضوع را نشان می‌دهد.

پس از آنکه تغییری را در فضای برازندگی تشخیص دادیم، می‌توان از یکی از چند رویکرد ممکن برای دنبال نمودن تغییرات نقطه‌ی بهینه استفاده نماییم. یک راه ممکن آن است که جمعیت را با یک جمعیت کاملاً جدید و اتفاقی جایگزین کرده و فرایند بهینه‌سازی تکاملی را از نو آغاز نماییم. این یک روش افراطی است و از هیچ یک از اطلاعات به دست آمده از فرایند بهینه‌سازی قبلی استفاده نمی‌کند. این روش در صورتی مناسب خواهد بود که فضای برازندگی آن قدر تغییر کرده باشد که جمعیت قبلی شامل هیچ گونه اطلاعات مفیدی در مورد فضای جدید نباشد. در این صورت به اجرای یک الگوریتم تکاملی جدید بر روی یک مسئله‌ی بهینه‌سازی جدید نیاز خواهد بود.

با این حال، در بسیاری از مسائل عملی، نوعی مشابهت میان فضای برازندگی جدید و قدیمی وجود دارد. به عبارتی، فضای برازندگی به تدریج، و نه ناگهانی، تغییر می‌کند. در این صورت، می‌خواهیم فضایی جدید را با بهره بردن از نتایج الگوریتم تکاملی بر روی فضای قدیمی، بکاویم. برای نمونه، می‌توان قسمت اعظمی از جمعیت قدیمی را نگه داشت و تنها چند ذره‌ی جدید را جهت کاوش فضای جدید به جمعیت وارد نمود. همچنین می‌توان به‌صورت موقتی نرخ جهش را افزایش داد تا میزان کاوش به‌صورت موقتی افزایش یابد. این روش فراجهش[1] نام دارد [کاب[2] و گرفنستت، ۱۹۹۳]. تعداد ذرات جدیدی که به جمعیت وارد می‌کنیم، میزان افزایشی که برای نرخ جهش در نظر می‌گیریم و تعداد نسل‌هایی که در آن‌ها نرخ جهش افزایش می‌یابد، همگی بر کنترل تعادل میان انتفاع و کاوش مؤثر هستند. این تعادل خود میزان انطباق‌پذیری الگوریتم تکاملی بر فضای جدید و وابستگی آن به نتایج قدیمی را تعیین می‌نماید.

در قسمت‌های آتی چند رویکرد الگوریتم تکاملی پویا را مورد بحث قرار می‌دهیم. این رویکردها شامل الگوریتم تکاملی پیشگو[3] (بخش ۲۱-۲-۱)، الگوریتم‌های تکاملی مهاجرت-محور[4] (بخش ۲۱-۲-۲) و الگوریتم‌های تکاملی حافظه-محور[5] (بخش ۲۱-۲-۳) می‌شوند. همچنین، چالش ارزیابی عملکرد الگوریتم‌های تکاملی بر روی مسائل بهینه‌سازی پویا را نیز مورد بحث قرار می‌دهیم.

[1] Hypermutation
[2] Cobb
[3] Predictive EA
[4] Immigrant-Based EAs
[5] Memory-Based EAs

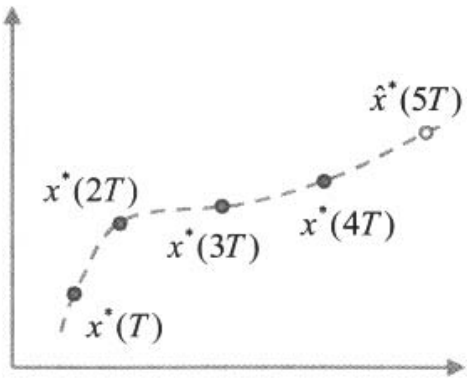
۲۱-۲-۱ الگوریتم تکاملی پیشگو

یک رویکرد به بهینه‌سازی پویا، ترکیب یک تکنیک پیشگویی با الگوریتم تکاملی است. الگوریتمی که با این کار به دست می‌آید الگوریتم تکاملی پیشگو خوانده می‌شود [هاتزاکیس[1] و والاس[2]، ۲۰۰۶]. اگر الگوریتم تکاملی در حال اجرا بوده و بهینه‌ی آن به‌صورتی قابل پیش‌بینی در حال تغییر باشد، می‌توانی مدلی از نحوه‌ی تغییر آن با زمان به وجود آورد. سپس، وقتی تغییری در فضای برازندگی شناسایی می‌شود، می‌توان از مدل مذکور برای وارد نمودن اعضای جدید به جمعیت استفاده نمود. شرط اساسی در اینجا آن است که بهینه به‌گونه‌ای "قابل پیش‌بینی" تغییر کند. اگر این شرط برآورده نشود، می‌توان جمعیت را به‌صورت اتفاقی مقداردهی کرده و الگوریتم تکاملی را مجددا از نو آغاز نمود. شکل ۲۱-۱۷ ایده‌ی اصلی الگوریتم تکاملی پیشگو و شکل ۲۱-۱۸ مثالی از تکامل پویای یک بهینه در الگوریتم تکاملی را نشان می‌دهد.

جمعیت الگوریتم تکاملی را مقداردهی اولیه کن
$X^* \leftarrow \emptyset$
تا زمانی که شرایط توقف برآورده نشده است
یک الگوریتم تکاملی را برای T نسل و یا تا زمانی که تغییری در زمینه‌ی برازندگی تسخیص داده شود، اجرا کن
بهترین ذره‌ی موجود در جمعیت را با x^* نشان بده
$X^* \leftarrow \{X^*, x^*\}$
دنباله‌ی X^* را جهت تخمین بهینه‌ی جدید $\hat{x}^*$، برون‌یابی کن
زیرجمعیتی از ذرات را که به $\hat{x}^*$ نزدیک هستند، ایجاد کن و با S نمایش بده
برخی ذرات جمعیت الگوریتم تکاملی را با S جایگزین کن
نسل بعدی

شکل ۲۱-۱۷ طرح کلی الگوریتم تکاملی پیشگو برای بهینه‌سازی پویا.

[1] Hatzakis
[2] Wallace



شکل ۲۱–۱۸ مثال از پیشرفت یک بهینه در الگوریتم تکاملی با فضای دو بعدی، به همراه مقدار پیش‌بینی شده‌ی آن. $x^*(T)$، $x^*(2T)$، $x^*(3T)$ و $x^*(4T)$ بهینه‌ی الگوریتم تکاملی پس از T، $2T$، $3T$ و $4T$ نسل می‌باشند. $\hat{x}^*(5T)$ مقدار بهینه‌ی پیش‌بینی شده است که از آن برای مقداردهی جمعیت بعدی استفاده می‌شود.

برخی جزییات پیاده‌سازی به تصمیم طراح واگذار شده است. برای نمونه، اگر قرار باشد الگوریتم برای T نسل بین برون‌یابی‌ها اجرا شود، مقدار T را چگونه باشد تعیین نمود؟ تغییرات فضای برازندگی چگونه تشخیص داده می‌شوند؟ از چه تعداد ذرات نشانگر باشد استفاده شود؟ اگر تعداد ذرات نشانگر خیلی کم باشد، ممکن است تغییری را از دست داده و تشخیص ندهیم. اما اگر از تعداد زیادی نشانگر استفاده نماییم، ممکن است باعث به هدر رفتن زمان بر روی ارزیابی‌های برازندگی غیرضروری شویم.

چگونه می‌توان در شکل ۲۱–۱۷ از مجموعه‌ی X^* برای تخمین بهینه‌ی جدید استفاده نمود؟ به بیان دیگر، از چه نوع الگوریتم برون‌یابی باید استفاده شود؟ زیرجمعیت S که نزدیک $\hat{x}^*$ می‌باشد باید چگونه تولید شود؟ یک راه آن است که S را برابر مجموعه‌ی $\{\hat{x}^*\}$ قرار دهیم. یک راه دیگر نیز آن است که S را برابر مجموعه‌ای از M نسخه‌ی جهش‌یافته‌ی $\hat{x}^*$ قرار دهیم. در این صورت M ثابتی است که توسط کاربر تعریف خواهد شد. یک راه دیگر، برابر قرار دادن S با مجموعه‌ای قطعی از M ذره‌ای است که در فرامکعب و یا فراکره‌ی دربرگیرنده‌ی $\hat{x}^*$ واقع می‌شوند. اگر میزان دقت پیش‌بینی $\hat{x}^*$ از یکبار اجرای الگوریتم تکاملی به اجرای بعدی را دنبال نماییم، می‌توانیم از آن برای تعیین اندازه‌ی فراحجم استفاده نماییم. اگر متوجه شویم که $\hat{x}^*$ به‌طور پیوسته دقیق است، آنگاه می‌توان از یک S با کاردینالیتی و فراحجم کوچک استفاده نمود. اگر متوجه شویم $\hat{x}^*$ به‌طور پیوسته نادقیق است، آنگاه باید کاردینالیتی و فراحجم S را افزایش دهیم. در آخر، باید تصمیم بگیریم کدام ذرات از جمعیت قدیمی را با S در شکل ۲۱–۱۷ جایگزین نماییم. راه‌های معمول برای این کار آن است که یا بدترین ذرات را جایگزین کرده و یا ذرات جایگزین‌شونده را به‌صورت اتفاقی انتخاب نماییم.

۲۱-۲-۲ طرح‌های مهاجر

ممکن است شرایطی به وجود آید که نتوانیم تغییرات تابع برازندگی را تشخیص دهیم و یا اینکه تابع برازندگی به‌صورت پیوسته در حال تغییر باشد. در این صورت می‌توان به‌صورت پیوسته ذرات جدیدی را وارد جمعیت نمود تا جمعیت نسبت به تغییرات فضای برازندگی مقاوم گردد چنین الگوریتم‌های طرح‌های مهاجر نام دارند [یو و همکاران، ۲۰۰۹]. دو رویکرد کلی در مورد طرح‌های مهاجر وجود دارد. یک طرح مهاجر مستقیم از ذرات موجود در جمعیت جهت تولید ذرات جدید استفاده می‌نماید. این موضوع مشابه الگوریتم‌های استاندارد بازترکیب و جهش می‌باشد. برای مثال، یک طرح مهاجر مستقیم می‌تواند ذرات نخبه از نسل حاضر و یا نسل‌های قبل را بازترکیب نموده و یا دچار جهش سازد تا ذرات جدید به وجود آیند. یک طرح مهاجر غیرمستقیم نیز ذرات جدید را بر اساس مدلی از جمعیت تولید می‌نماید. برای مثال، می‌توان از یک الگوریتم نوع PBIL (بخش ۱۳-۲-۳ را ببینید) برای مدل‌سازی جمعیت و سپس برای تولید ذرات جدید بر اساس این مدل، استفاده نمود.

پس از آنکه یکی از دو طرح مستقیم و غیرمستقیم را انتخاب نمودیم، باید در مورد موارد زیر نیز تصمیم‌گیری نماییم.

۱. ذرات جدید باید چگونه تولید شوند؟ می‌توان مجموعه‌ای از ذرات مانند X_e را بر اساس نخبه‌ها تولید نمود. همچنین می‌توان مجموعه‌ای از ذرات مانند X_r را به‌صورت اتفاقی ایجاد نمود. اگر بدانیم که فضای برازندگی به شدت تغییر خواهد نمود، می‌توان مجموعه‌ای از ذرات دوگانه مانند X_d، که مشابه ذرات مخالف در فصل ۱۶ می‌باشند، را ایجاد کرد. در آخر، می‌توان از ترکیبی از این چند گزینه استفاده نمود. اگر در طول الگوریتم تکاملی عملکرد هر یک از این سه نوع ذره را دنبال نماییم، می‌توانیم اعداد X_e، X_r و X_d را به‌صورت اقتباسی تعیین نماییم [یو و همکاران، ۲۰۰۹].

۲. چه تعداد ذره‌ی جدید باید وارد جمعیت شود؟ بیشتر محققین $0.2N$ و یا $0.3N$ ذره‌ی جدید را وارد جمعیت می‌نمایند. اگر بتوانیم فرکانس میزان تغییرات در فضای برازندگی را تشخیص دهیم، خواهیم توانست نرخ جایگزینی را نیز متناسب با آن تعیین نماییم. برای نمونه، اگر تغییر بزرگی در فضای برازندگی تشخیص دهیم، ممکن است بخواهیم ذرات جدید بیشتری را وارد جمعیت نماییم.

۳. کدام ذرات جمعیت باید با ذرات جدید جایگزین شوند؟ یک جواب معمول آن است که ذرات جایگزین شونده را به‌صورت اتفاقی انتخاب نماییم. یک جواب دیگر نیز آن است که بدترین ذرات را جایگزین نماییم. یک نکته‌ای که در اینجا باید به آن توجه نمود آن است که ذرات جدید ممکن است چندان مناسب جمعیت نباشند، اما امکان دارد برازندگیشان با گذشت زمان و با تغییر فضای

برازندگی بهبود یابد. به همین دلیل، بهتر است نوعی فاکتور زمانی داشته باشیم تا مطمئن شویم ذرات تا قبل از گذشت یک زمان خاص جایگزین نمی‌شوند [تینوس[1] و یانگ[2]، ۲۰۰۵].

اندازه جمعیت $= N$

نسبت ذرات اتفاقی که باید در هر نسل ایجاد شوند $= r_r$

نسبت ذرات نخبه محور که باید در هر نسل ایجاد شوند $= r_e$

نسبت ذرات دوگانه که باید در هر نسل ایجاد شوند $= r_d$

N ذره‌ی ابتدایی $\{x_i\}$ را ایجاد کن

برازندگی N ذره را محاسبه کن

تا زمانی که شرایط توقف برآورده نشده است

  از یک روش بازترکیب/جهش جهت رشد $\{x_i\}$ برای نسل بعد استفاده کن

  برازندگی ذرات $\{x_i\}$ را محاسبه کن

  $X \leftarrow \{$ذره‌ی تولید شده به صورت اتفاقی $Nr_r\}$

  $X \leftarrow X \cup \{$جهش نخبه‌ها $Nr_e\}$

  $X \leftarrow X \cup \{$ذرات دوگانه $Nr_d\}$

  برازندگی ذرات موجود در X را محاسبه کن

  ذرات موجود در $\{x_i\}$ را با ذرات X جایگزین کن

  r_r، r_e و r_d را طبق عملکرد ذرات جدید موجود در جمعیت تنظیم کن

نسل بعدی

شکل ۲۱-۱۹ طرح کلی یک الگوریتم تکاملی مهاجرت-محور برای بهینه‌سازی پویا.

شکل ۲۱-۱۹ مروری از یک الگوریتم تکاملی مهاجرت-محور برای بهینه‌سازی پویا را به دست می‌دهد. با این حال، تصمیمات زیادی باید توسط طراح الگوریتم تکاملی گرفته شود.

۱. از چه مقادیری باید برای r_r، r_d و r_e استفاده شود؟ همان‌طور که پیش از این نیز اشاره شد، یک مقدار معمول برای برای نسبت جزء جایگزین شونده برابر ۰٫۲ یا ۰٫۳ است. معمولاً، تعداد ذرات جایگزین نخبه، دوگانه و اتفاقی یکسان است. بنابراین: $r_r = r_d = r_e = r_T/3$.

[1] Tinos
[2] Yang

۲. آیا باید اعداد r_r، r_d و r_e به‌صورت اقتباسی انتخاب شوند؟ همانند شکل ۲۱-۱۹ می‌توان این اعداد را به‌صورت اقتباسی تعیین نمود، اما این کار باعث پیچیدگی بیشتر الگوریتم خواهد شد. در [یو و همکاران، ۲۰۰۹] روش اقتباسی برای این کار پیشنهاد شده است که بدین ترتیب عمل می‌کند: ابتدا در هر نسل برازندگی ذرات جدید ارزیابی می‌شود. سپس اگر گروه ذرات اتفاقی بهتر از گروه ذرات دوگانه و نخبه عمل نماید، آنگاه r_r، r_d و r_e به‌صورت زیر تعیین خواهند شد:

$$\begin{aligned} r_d &\leftarrow \max(r_{min}, r_d - \alpha) \\ r_e &\leftarrow \max(r_{min}, r_e - \alpha) \\ r_r &\leftarrow r_T - r_d - r_e \end{aligned} \qquad (۲۱-۳۶)$$

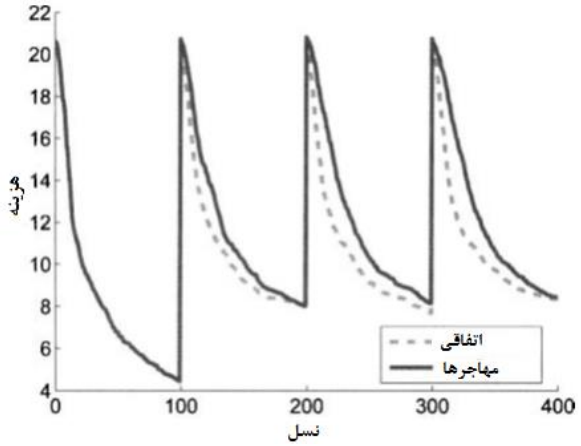
که در آن α سرعت اقتباس را کنترل کرده، r_{min} کمترین نسبت هر نوع از ذرات جدید که باید در هر نسل ایجاد شوند را تعیین کرده و ثابت r_T نیز نسبت کل ذرات جدید که باید در هر نسل تولید شود را تعیین می‌نماید. در [یو و همکاران، ۲۰۰۹] از $\alpha \approx 0.2$ و $r_{min} = 0.04$ استفاده شده است. اگر گروه ذرات دوگانه یا ذرات نخبه بهتر عمل نماید، آنگاه معادله‌ی (۲۱-۳۶) را به‌گونه‌ای بازنویسی می‌نماییم که تولید ذرات گروهی که عملکرد بهتری دارد در نسل بعد افزایش یافته و تولید ذرات انواع دیگر کاهش یابد. اما این نوع روش اقتباس باعث می‌شود این سؤال را بپرسیم که، چگونه بفهمیم کدام نوع بهترین عملکرد را دارد؟ به عبارت دیگر چگونه در مورد بهترین عملکرد میان انواع ذرات تصمیم بگیریم؟ برای این تصمیم‌گیری می‌توان معیار را بر اساس بهترین ذره‌ی موجود در هر گروه و یا میانگین عملکرد کل گروه قرار داد.

۳. کدام ذرات جمعیت را باید با ذرات جدید جایگزین نمود؟ پیش از این به این مورد اشاره‌ی مختصری نموده‌ایم. در [یو و همکاران، ۲۰۰۹] بدترین ذرات برای جایگزینی انتخاب می‌شوند اما ما می‌توانیم ذرات اتفاقی را جایگزین نماییم. همچنین می‌توان از ترکیبی از بدترین ذرات و ذرات اتفاقی برای این جایگزینی استفاده نمود. همچنین می‌توان جمعیت اصلی را با جمعیت جایگزین X در یک گروه قرار داده و سپس از یک فرایند اتفاقی برای انتخاب بهترین N ذره استفاده نمود. برای این کار می‌توان از هر یک از مکانیزم‌های انتخابی که در بخش ۸-۷-۱ مورد بحث قرار گرفتند، استفاده نمود.

٤. در آخر، باید یک روش بازترکیب و یک روش جهش برای رشد دادن ذرات $\{x_i\}$ برای نسل بعد در شکل ۲۱-۱۹ استفاده نماییم- برای این کار می‌توان از هر یک از روش‌های بازترکیب و جهشی که در بخش ۸-۹ معرفی شدند، استفاده نمود.

مثال ۲۱-۵

در این مثال عملکرد BBO را بر روی تابع *آکلی* از مثال ۲۱-٤ ارزیابی می‌نماییم. از آنجا که دو ذره‌ی نخبه را در هر نسل حفظ می‌نماییم، بهترین هزینه‌ی جمعیت باید در هر نسل کاهش یافته و به همین دلیل اگر تغییری در فضای برازندگی اتفاق بیفتد، قادر به شناسایی آن خواهیم بود. اگر بهترین هزینه افزایش یابد، آنگاه می‌توان نتیجه گرفت که تابع هزینه تغییر یافته است. در این مثال، از دو راه برای منطبق شدن با تغییرات تابع هزینه استفاده می‌نماییم: در روش اول جمعیت را با ذرات اتفاقی جدید بازتولید کرده و در روش دوم از طرح مهاجر مستقیم شکل ۲۱-۱۹ استفاده می‌نماییم. شکل ۲۱-۲۰ عملکرد BBO را برای هر دو روش نشان می‌دهد. در این شکل عملکرد BBO بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است. می‌توان دید که روش شروع دوباره‌ی اتفاقی بهتر از طرح مهاجر مستقیم عمل می‌نماید. تغییر پویای تابع هزینه به قدری اتفاقی است که جایگزین نمودن ۳۰٪ جمعیت برای بهتر بودن از شروع دوباره‌ی اتفاقی، کافی نیست.



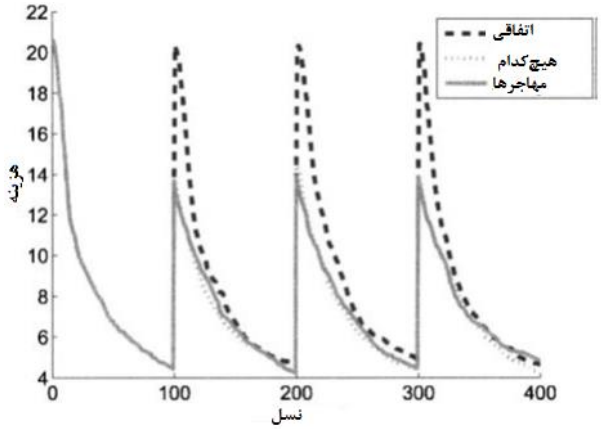
شکل ۲۱-۲۰ نتایج مثال ۲۱-۵. این شکل، عملکرد BBO بر روی تابع ۱۰ بعدی *آکلی*، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، را نشان می‌دهد. هنگامی که تابع هزینه هر ۱۰۰ نسل یکبار به دلیل چرخش بردار بایاس تغییر می‌کند، جمعیت یا دوباره به‌طور کامل و به‌صورت اتفاقی مقداردهی اولیه می‌شود و یا ۳۰٪ آن توسط طرح مهاجر مستقیم از شکل ۲۱-۲۰ جایگزین می‌شود. از آنجا که پویایی تابع هزینه بسیار اتفاقی است، شروع دوباره‌ی اتفاقی عملکرد بهتری از طرح مهاجر دارد.

مثال ۲۱–٦

در این مثال نیز دوباره عملکرد BBO بر روی تابع پویای *آکلی* را بررسی می‌نماییم، اما در این مثال تغییر پویای تابع هزینه شامل چرخش بردار بایاس نمی‌شود. به جای آن، این تغییر شامل نوعی اختلال با مقدار اتفاقی در بردار بایاس می‌شود:

$$\begin{aligned}&\theta(t) \leftarrow \theta(t-1) + 0.1(x_{max} - x_{min})\rho(t-1)\\&\theta(t) \leftarrow \min(\theta(t), x_{max} - x^*)\\&\theta(t) \leftarrow \max(\theta(t), x_{min} - x^*)\end{aligned} \qquad (۳۷–۲۱)$$

در معادله‌ی بالا $[x_{min}.x_{max}]$ فضای جستجو بوده، x^* مقدار بهینه کننده‌ی تابع هزینه‌ی بایاس نشده بوده و $\rho(t-1)$ نیز عددی اتفاقی است که از یک توزیع گاوسی با میانگین صفر و واریانس واحد گرفته شده است. برای اطلاعات بیشتر در مورد پارامترهای بالا می‌توانید به ضمیمه‌ی ج.٤ مراجعه کنید. با استفاده از معادله‌ی بالا می‌توان از قرار گرفتن بهینه‌ی $f(x-\theta(t))$ در دامنه‌ی جستجو اطمینان حاصل نمود. برای تابع *آکلی*، $x^* = 0$ خواهد بود، اما می‌توان معادله‌ی (۲۱–۳۷) را به هر تابع محک دیگری نیز اعمال نمود. معادله‌ی (۲۱–۳۷) نشان می‌دهد که بردار بایاس $\theta(t)$ بر اساس یک توزیع گاوسی با انحراف استانداردی برابر با ۱۰٪ رنج فضای جستجو تغییر می‌کند. این تغییرِ ملایم‌تر در تابع هزینه نسبت به تغییر بدون ساختار بردار بایاس در مثال ۲۱–۵، واقع‌گرایانه‌تر است. در این مثال از سه روش مختلف برای تطبیق با تغییرات تابع هزینه استفاده می‌شود: (۱) مقداردهی اولیه‌ی دوباره‌ی جمعیت با استفاده از ذرات تولید شده‌ی اتفاقی؛ (۲) استفاده از طرح مهاجر مستقیم از شکل ۲۱–۱۹. برای این منظور مقادیر r_r، r_d و r_e هر کدام برابر ۱۰٪ اندازه‌ی جمعیت انتخاب شده‌اند و از این ذرات برای جایگزین نمودن بدترین ذرات جمعیت استفاده شده است؛ (۳) نادیده گرفتن تغییرات تابع هزینه و عدم تغییر جمعیت. شکل ۲۱–۲۱ عملکرد BBO برای این سه روش را، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. می‌توان دید که مقداردهی اولیه‌ی دوباره‌ی اتفاقی بدترین عملکرد را دارد. این موضوع به این دلیل است که تغییرات تابع هزینه کمی ساخت‌یافته است و به همین دلیل جایگزین نمودن کل جمعیت باعث از دست رفتن اطلاعاتی می‌شود که در طی ۱۰۰ نسل قبل به دست آمده است. همچنین، بر اساس این شکل، طرح مهاجر و روش تادیده گرفتن هر دو عملکرد تقریباً مشابهی را دارند.



شکل ۲۱–۲۱ نتایج مثال ۲۱–٦. این شکل عملکرد BBO بر روی تابع ۱۰ بعدی *آکلی* را، که بر روی ۲۰ شبیه‌سازی مونت کارلو میانگین‌گیری شده است، نشان می‌دهد. وقتی تابع هزینه هر ۱۰۰ نسل یکبار و با بروز اختلالی ملایم در بردار بایاس تغییر می‌کند، یا کل جمعیت را جایگزین کرده، یا ۳۰٪ از آن را با طرح مهاجر جایگزین کرده و یا هیچ کاری انجام نمی‌دهیم. از آنجا که تغییرات تابع هزینه نسبتاً ساخت‌یافته است، مقداردهی اولیه‌ی دوباره‌ی اتفاقی بدترین عملکرد را دارد.

به‌طور کلی می‌توان انتظار داشت که رویکرد مهاجر عملکرد بهتری از رویکرد نادیده گرفتن داشته باشد. ممکن است بتوان با اقتباس مقادیر r_r، r_d و r_e به آن ترتیبی که در معادله‌ی (۲۱–۳٦) نشان داده شده است، عملکرد طرح مهاجر را بهبود بخشید. هر چند این کار در مثال بالا انجام نشده است. همچنین باید توجه داشت مقایسه‌ای که در شکل ۲۱–۲۱ انجام گرفته است مقایسه‌ی چندان عادلانه‌ای نیست چرا که این مقایسه میان سه الگوریتم برای تعداد یکسانی نسل انجام شده است. در الگوریتم مقداردهی اولیه‌ی اتفاقی برای هر بار تغییر تابع به N بار ارزیابی تابع برازندگی نیاز است. این در حالی است که در الگوریتم طرح مهاجر، پس از هر بار تغییر تابع به $0.3N$ ارزیابی تابع برازندگی نیاز است. واضح است که این عدد برای الگوریتم آخر که تغییرات تابع را نادیده می‌گیرد، صفر است. با این حال، تابع هر ۱۰۰ نسل یکبار تنها یکبار تغییر می‌کند. بنابراین، با اینکه مقایسه‌ی درست هنگامی اتفاق می‌افتد که سه الگوریتم بر اساس تعداد ارزیابی‌های تابع مقایسه شوند و نه تعداد نسل، اما تفاوت میان تعداد ارزیابی‌های تابع در این مثال به اندازه‌ی کم است که می‌توان از آن چشم‌پوشی نمود.

۲۱-۲-۳ رویکردهای حافظه-محور

گاهی تابع هزینه میان مجموعه‌ای متناهی از توابع تغییر می‌کند. برای مثال، فرض کنید می‌خواهیم فرایندی کنترلی در یک کارخانه را بهینه نماییم. با تغییر وظایف توسط مدیر و یا تغییر اجزاء ماشین‌ها ممکن است پارامترهای فرایند تولید تغییر کنند. این تغییرها باعث تغییر تابع هزینه می‌شود، اما این تغییرات اتفاقی نیستند. تغییرهای این چنینی معلوم و مشخص است، اما ممکن است جزییات این تغییرات برای کنترلر و فرایند بهینه‌سازی مشخص نباشد. در چنین مواردی، بهتر است راه‌حل‌های بهینه‌ی قبلی را دنبال نموده و هرگاه که تغییری در تایع هزینه تشخیص داده شد، آن‌ها را به جمعیت وارد نمود.

رویکردهای حافظه-محور صریح ذرات خوب را در یک آرشیو ذخیره می‌کنند [ولدسنبت[1] و ین، ۲۰۰۹]. هر گاه که تغییری در تابع هزینه تشخیص داده می‌شود، ذرات آرشیو به جمعیت تزریق می‌شوند. اگر تابع هزینه به تابعی که قبلاً با آن مواجه شده‌ایم تغییر کند، ذرات آرشیو راه‌حل‌های بسیار خوبی بوده و الگوریتم تکاملی به سرعت به بهینه‌ی جدید همگرا خواهد شد. شکل ۲۱-۲۲ این رویکرد را نشان می‌دهد. این شکل یک طرح کلی را نشان می‌دهد و در آن به جزییات مهمی اشاره نشده است. سؤالات زیر می‌توانند فرصت‌های بسیار خوبی برای تحقیق را در اختیار بگذارند.

۱. هنگام تشخیص تغییر چه تعداد ذره باید در آرشیو ذخیره شود؟
۲. اندازه‌ی آرشیو باید تا چه حد مجاز به رشد باشد؟ در شکل ۲۱-۲۲ حد بالایی برای اندازه‌ی آرشیو در نظر گرفته نشده است. در عمل، ممکن است بخواهیم "نقاط عملیاتی" مسئله را تشخیص دهیم. اگر نقطه‌ی عملیاتی یکی از نقاط عملیاتی باشد که قبلاً با آن مواجه شده‌ایم، نقاط نخبه پیش از این برای آن نقطه‌ی عملیاتی در آرشیو ذخیره شده‌اند. در این صورت نیازی به ذخیره کردن نخبه‌های حاضر در آرشیو نخواهد بود مگر آنکه از مقادیر از پیش ذخیره شده در آرشیو بهتر باشند. برای مثال، فرض کنید پس از آنکه تغییری در $f(.)$ تشخیص داده شد، در آرشیو جستجو کرده و متوجه شویم مجموعه‌ی حاضر ذرات نخبه مشابه ذرات از پیش ذخیره شده است. در این صورت می‌توان نتیجه گرفت مسئله‌ای که توسط الگوریتم تکاملی حل شده است مشابه مسائل از پیش حل شده است و به همین دلیل نیازی به ذخیره کردن نخبه‌های حاضر نیست مگر آنکه نخبه‌های حاضر بهتر از آرشیو باشند.
۳. تغییرات $f(.)$ را چگونه تشخیص بدهیم؟ این موضوع را در ابتدای بخش ۲۱-۲ مطرح نمودیم.

[1] Woldesenbet

٤. پس از آنکه تغییر $f(.)$ تشخیص داده شد، کدام ذرات جمعیت باید با ذرات آرشیو جایگزین شوند؟ کدام ذرات آرشیو باید به جمعیت وارد شوند؟

۲۱-۲-۴ ارزیابی عملکرد بهینه‌سازی پویا

ارزیابی عملکرد بهینه‌سازی پویا از ارزیابی عملکرد بهینه‌سازی ایستا متفاوت است. هنگام ارزیابی عملکرد بهینه‌سازی ایستا معمولاً جهت تعیین این که عملکرد الگوریتم تکاملی چه قدر مناسب بوده است، به جمعیت آخرین نسل نگاه می‌کنیم. با این حال، هنگام ارزیابی عملکرد بهینه‌سازی پویا، تابع هزینه از یک نسل به نسل دیگر تغییر می‌کند. بنابراین، نسل آخر نمی‌تواند به تنهایی نشانگر مناسبی از عملکرد الگوریتم تکاملی باشد. در عوض، باید به عملکرد در طول همه‌ی نسل‌ها نگاه کنیم [یو و همکاران، ۲۰۰۹]. دو متریک معمول برای مسائل بهینه‌سازی پویا، میانگین بهترین عملکرد $\bar{f}_b$ و میانگین عملکرد متوسط $\bar{f}_a$ می‌باشد:

$$\bar{f}_b = \frac{1}{G}\sum_{i=1}^{G} f_{i,b}$$
$$\bar{f}_a = \frac{1}{G}\sum_{i=1}^{G} f_{i,a} \qquad (۳۸-۲۱)$$

در معادله‌ی بالا G تعداد نسل‌ها، $f_{i.b}$ بهترین برازندگی در طول نسل iاُم و $f_{i.a}$ عملکرد متوسط در طول نسل iاُم می‌باشد. با استفاده از این متریک‌ها می‌توان عملکرد بهینه‌سازی پویا را میان الگوریتم‌های تکاملی مختلف مقایسه نمود. اگر چند شبیه‌سازی مونت کارلو اجرا نماییم، آنگاه یک سطح میان‌گیری دیگر نیز در اختیار خواهیم داشت: می‌توان $\bar{f}_b$ و $\bar{f}_a$ را در طول شبیه‌سازی‌های مونت کارلو نیز میانگین گرفت.

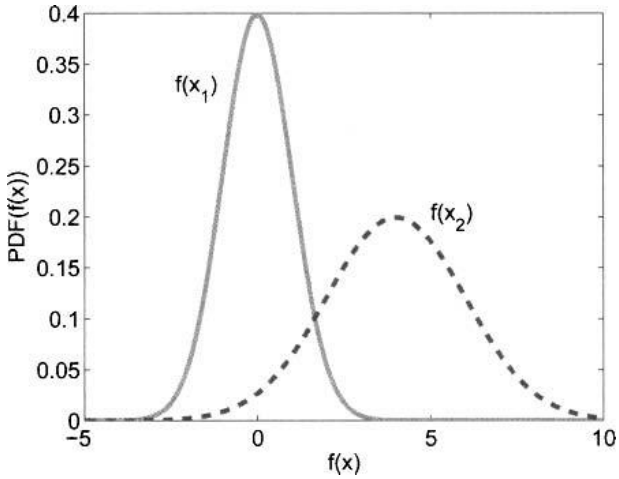
```
جمعیت اولیه {x_i} را ایجاد کن
A ← ∅ آرشیو
تا زمانی که شرایط توقف برآورده نشده است
  از یک روش بازترکیب/جهش برای رشد {x_i} جهت نسل بعدی استفاده کن
  برازندگی ذرات {x_i} را محاسبه کن
  بهترین ذرات {x_i} را در مجموعه‌ی نخبه‌ها که با E نمایش داده می‌شود ذخیره کن
  اگر تغییری در f(.) مشاهده شد
    برخی ذرات {x_i} را با ذرات آرشیو A جایگزین کن
    مجموعه‌ی نخبه‌ها (E) را در A ذخیره کن
  پایان اگر
نسل بعدی
```

شکل ۲۱-۲۲ طرح کلی یک رویکرد حافظه-محور صریح برای بهینه‌سازی پویا. این الگوریتم خصوصاً برای توابع برازندگی متغیر با زمان و یا توابعی که به مجموعه‌ای متناهی از توابع برازندگی تعلق دارد، مناسب است.

۲۱-۳ توابع برازندگی شلوغ

ارزیابی‌های تابع برازندگی در الگوریتم‌های تکاملی معمولاً با مقداری نویز همراه است. برای مثال، عدم دقت سنسورها می‌تواند باعث بروز نویز در ارزیابی‌های تابع برازندگی شود. همچنین، اگر مقادیر تابع برازندگی با استفاده از برنامه‌های شبیه‌سازی اندازه‌گیری شود، خطاهای تخمین برنامه می‌تواند باعث بروز نویز در ارزیابی‌های تابع برازندگی شود. احتمالاً اولین کسی که به مطالعه‌ی تأثیر نویز بر روی الگوریتم‌های تکاملی پرداخت، اینگو روچنبرگ، مبدع استراتژی تکاملی می‌باشد [روچنبرگ، ۱۹۷۳].

ارزیابی یک تابع برازندگی شلوغ می‌تواند به اختصاص دادن اشتباهی مقدار برازندگی زیاد به ذرات با برازندگی واقعی کم ختم شود. عکس این موضوع نیز محتمل است، بدین معنی که ممکن است به ذرات با برازندگی بالا به اشتباه مقدار برازندگی کمی اختصاص داده شود. شکل ۲۱-۲۳، PDF دو تابع برازندگی شلوغ اما بدون گرایش را نشان می‌دهد. می‌توان دید که مقادیر دقیق $f(x_1)$ و $f(x_2)$ به ترتیب برابر ۰ و ٤ می‌باشد، اما ارزیابی‌ها شامل نویز می‌باشند. بنابراین، ممکن است x_1 دارای برزاندگی ارزیابی شده‌ای بزرگتر از x_2 باشد. این موضوع می‌تواند باعث شود الگوریتم تکاملی ذره‌ی اشتباه را برای عمل بازترکیب برگزیند. به‌طور خلاصه، نویز می‌توان باعث گمراهی الگوریتم تکاملی شود.



شکل ۲۱–۲۳ این شکل PDF دو تابع مختلف را به تصویر می‌کشد. x_2 با مقدار واقعی ٤، برازنده‌تر از x_1 می‌باشد، اما بسته به نویزی که در هنگام ارزیابی تابع برازندگی وجود دارد، ممکن است الگوریتم تکاملی x_1 را برازنده‌تر از x_2 تشخیص دهد. این موضوع می‌تواند باعث انتخاب اشتباه برای نسل بعد شود.

هنگامی که ارزیابی‌های تابع برازندگی با نویز همراه باشد، نمی‌توان مطمئن بود کدام ذره بهترین است. فرض کنید دو ذره‌ی x_1 و x_2 را در اختیار داریم و مقادیر برازندگی واقعی آن‌ها نیز $f_t(x_1)$ و $f_t(x_2)$ باشد. همچنین فرض کنید مقادیر واقعی این دو برازندگی مانند آنچه که در شکل ۲۱–۲۳ نمایش داده شده است، برابر ۰ و ٤ است. به دلیل وجود نویز، نمی‌توان الزاما از $f(x_1) > f(x_2)$ نتیجه گرفت که $f_t(x_1) > f_t(x_2)$. با این حال، اگر PDFهای $f(x_1)$ و $f(x_2)$ را در اختیار داشته باشیم، می‌توان با استفاده از مقادیر مشخص $f(x_1)$ و $f(x_2)$، احتمال $f_t(x_1) > f_t(x_2)$ را محاسبه نمود. در اینجا وارد ریاضیات این موضوع نمی‌شویم، اما می‌توان محاسبات مذکور را با استفاده از روش‌های استاندارد نظریه احتمال انجام داد [گرین‌استد و اسنل، ۱۹۹۷]، [میتزنماکر و اوپفال، ۲۰۰۵]. توجه داشته باشید که در طول اجرای الگوریتم تکاملی PDF تابع برازندگی شلوغ را در اختیار نخواهیم داشت چرا که از تابع برازندگی واقعی اطلاع نداریم. با این حال، ممکن است PDF تابع برازندگی واقعی را در اختیار داشته باشیم. این وضعیت مشابه وضعیتی است که در شکل ۲۱–۲۳ نمایش داده شده است با این تفاوت که به جای برخورد با تابع برازندگی شلوغ مانند یک متغیر اتفاقی با میانگینی برابر با تابع برازندگی حقیقی، می‌توان با آن مانند یک متغیر اتفاقی با میانگینی برابر با مقدار تابع برازندگی ارزیابی شده‌ی شلوغ، برخورد نمود.

در این بخش به بحث در مورد سه روش برای برخورد با مسئله‌ی شلوغ بودن توابع برازندگی می‌پردازیم. بخش ۲۱–۳–۱ رویکرد نمونه‌گیری دوباره را مطرح نموده، بخش ۲۱–۳–۲ رویکرد تخمین برازندگی را مورد

مطالعه قرار داده و بخش ۲۱-۳-۳ نیز به بحث در مورد الگوریتم تکاملی KALMAN که از فیلتر KALMAN برای تخمین مقادیر تابع برازندگی استفاده می‌نماید، خواهد پرداخت.

۲۱-۳-۱ نمونه‌گیری دوباره

یک راه ساده برای کاهش نویز، نمونه‌گیری دوباره تابع برازندگی می‌باشد. اگر یک تابع برازندگی را برای یک ذره N بار ارزیابی نماییم و مقادیر نویز برای آن N نمونه مستقل از هم باشند، آنگاه واریانس میانگین تابع برازندگی با ضریبی از مرتبه‌ی N کاهش خواهد یافت [گرین‌استد و اسنل، ۱۹۹۷]، [میتزنماکر و اوپفال، ۲۰۰۵]. فرض کنید برازندگی ارزیابی شده یک راه‌حل نامزد مانند x به‌صورت زیر به دست آید

$$g(x) = f(x) + w \qquad (۳۹-۲۱)$$

در معادله‌ی بالا $f(x)$ برازندگی حقیقی بوده و w نویزی با میانگین صفر و واریانس σ^2 می‌باشد. این بدین معناست که برازندگی ارزیابی شده‌ی $g(x)$ دارای میانگین صفر و واریانس σ^2 است. اگر N اندازه‌گیری مختلف انجام دهیم، آنگاه هر اندازه‌گیری $g_i(x)$ دارای واریانسی برابر σ^2 خواهد بود و بهترین تخمین از برازندگی حقیقی برابرست با

$$\hat{f}(x) = \frac{1}{N}\sum_{i=1}^{N} g_i(x) \qquad (٤٠-۲۱)$$

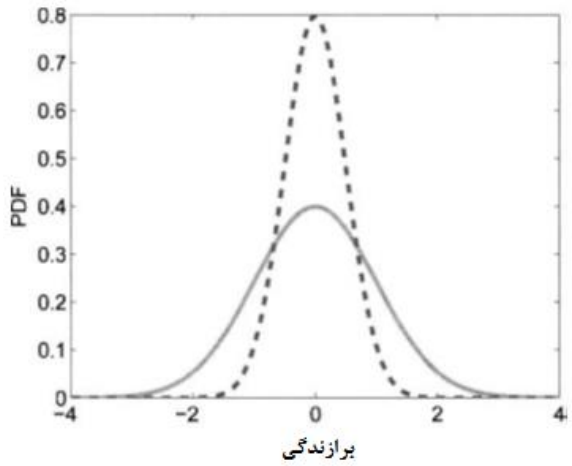
واریانس $\hat{f}(x)$ برابر σ^2/N می‌باشد. شکل ۲۱-۲٤ این ایده را نمایش می‌دهد. میانگین مجموعه‌ای از N ارزیابی تابع برازندگی، N بار دقیق‌تر از یک ارزیابی تنها می‌باشد.

با این حال، استراتژی نمونه‌گیری دوباره تنها در صورتی معتبر است که نویزهای ارزیابی‌های تابع برازندگی از یک نمونه به نمونه‌ی دیگر مستقل باشند. برای نمونه، فرض کنید برازندگی راه‌حل‌های نامزد را با ابزارهای نویزی اندازه بگیریم. حال اگر نویز ابزاری با خود دارای همبستگی زمانی باشد، آنگاه میانگین‌گیری بر روی N نمونه واریانس را با مرتبه‌ی N کاهش نخواهد داد. در این شرایط، مقدار کاهش واریانس به همبستگی نویز از یک نمونه به نمونه‌ی دیگر بستگی دارد.

اگر N ارزیابی برازندگی از راه‌حل نامزد x در اختیار داشته باشیم، آنگاه می‌توان واریانس σ^2 تخمین برازندگی را به‌صورت زیر تخمین زد:

$$\hat{f}(x) = \frac{1}{N}\sum_{i=1}^{N} g_i(x)$$
$$\hat{\sigma}^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(\hat{f}(x) - g_i(x)\right)^2 \qquad (۲۱-٤۱)$$

به‌طور حسی می‌توان گفت که در معادله‌ی $\hat{\sigma}^2$ باید عبارت N قرار گیرد، اما عبارت $(N-1)$ ترجیح داده شده است چرا که این عبارت تخمین بدون گرایشی از واریانس را به دست می‌دهد [سایمون، ۲۰۰٦، مسئله‌ی ۳-٦]. می‌توان از معادله‌ی (۲۱-٤۱) برای تعیین تعداد دفعاتی که باید از یک تابع برازندگی شلوغ جهت دستیابی به واریانس مطلوب در تخمین مقدار برزاندگی نمونه گرفت، استفاده نمود (مسئله‌ی ۲۱-۷ را ببینید). واریانس مطلوب توسط کاربر تعیین شده و به نوع مسئله بستگی دارد. با میل کردن N به سمت بی‌نهایت، واریانس به سمت صفر میل کرده و تخمین مقدار برازندگی عاری از خطا خواهد شد.



شکل ۲۱-۲٤ این شکل استراتژی نمونه‌گیری دوباره برای ارزیابی‌های تابع شلوغ را به تصویر می‌کشد. خط توپر PDF یک تابع برازندگی شلوغ را نشان داده و خط‌چین PDF متوسط چهار ارزیابی تابع برازندگی را نشان می‌دهد. هر دوی منحنی‌ها دارای میانگین صفر می‌باشند، اما متوسط ارزیابی‌ها دارای واریانسی است که ¼ یک ارزیابی تنها می‌باشد. احتمال نزدیک بودن ارزیابی متوسط به میانگین‌اش بسیار بیشتر از یک ارزیابی تنهاست.

برخی محققین تعداد ثابتی نمونه‌گیری دوباره را برای هر راه‌حل نامزد پیشنهاد می‌نمایند. اما با این کار، احتمال برابر نبودن نویز ارزیابی برازندگی برای همه‌ی راه‌حل‌های نامزد، نادیده گرفته می‌شود [دی پیترو[1] و همکاران، ۲۰۰٤]. این مورد ممکن است برای مثال، در مورد سنسورهایی که نویزشان با سیگنال اندازه‌گیری

[1] Di Pietro

شونده متناسب است، پیش بیاید [آرنولد، ۲۰۰۲]. در چنین حالتی، معادله‌ی (۲۱-۳۹) با معادله‌ی زیر جایگزین می‌شود

$$g(x) = f(x) + w(x) \qquad (۲۱-۴۲)$$

این بدین معنی است که نویز ارزیابی برازندگی به راه‌حل نامزدی که در حال ارزیابی است بستگی خواهد داشت. در چنین حالتی، نمونه‌گیری از هر x به تعداد دفعات یکسان ناکارآمد خواهد بود چرا که این کار باعث بروز دقت‌های مختلف برای راه‌حل‌های نامزد متفاوت خواهد شد. اما همچنان می‌توان از معادله‌ی (۲۱-۴۱) برای تصمیم‌گیری در مورد تعداد دفعات نمونه‌گیری از هر راه‌حل نامزد استفاده نمود.

در استراتژی نمونه‌گیری دوباره ممکن است به تعداد زیادی نمونه برای دستیابی به یک واریانس مطلوب نیاز باشد. این کار ممکن است برای توابع پرهزینه و گران امکان‌پذیر نباشد. بنابراین، شاید بهتر باشد که نمونه‌گیری دوباره را با یکی از روش‌های تخمینی که در بخش ۲۱-۱-۱ مورد بحث قرار دادیم، ترکیب نمود. همچنین می‌توان جهت کاهش تعداد نمونه‌گیری‌های دوباره، این عمل را تنها برای بهترین ذرات موجود در جمعیت انجام داد. شاید بر اساس یک ارزیابی تابع برازندگی تنها نتوان فهمید کدام ذرات بهترین هستند، اما حداقل می‌توان درکی نسبی از بهترین ذرات موجود به دست آورد و بدین ترتیب می‌توان تلاش محاسباتی مورد نیاز برای نمونه‌گیری دوباره را برای بهترین ذرات موجود در جمعیت حفظ نمود [برنک[1]، ۱۹۹۸].

یک رویکرد دیگر جهت تصمیم‌گیری در مورد تعداد دفعات نمونه‌گیری دوباره، رویکردی است که بر پایه‌ی وراثت برازندگی از بخش ۲۱-۱-۱ قرار دارد [بویی[2] و همکاران، ۲۰۰۵]. فرض کنید چند والد $\{p_i\}$ فرزندی مانند x را تولید می‌کنند. همچنین فرض کنید برازندگی تخمینی والد iاُم برابر $g(p_i)$ و انحراف استاندارد آن برابر $\sigma(p_i)$ می‌باشد. می‌توان پیش از محاسبه‌ی برازندگی فرزند، از وراثت برازندگی برای به دست آوردن تخمینی از برازندگی آن ذره و انحراف استاندارد آن، به ترتیب $\hat{f}(x)$ و $\hat{\sigma}(x)$، استفاده نمود. سپس، برازندگی فرزند را یکبار محاسبه می‌نماییم. اگر ارزیابی برازندگی ذره، که با $g(x)$ نمایش می‌دهیم، بین $\hat{f}(x) \pm 3\hat{\sigma}(x)$ از واقع شود، آنگاه $g(x)$ را به‌عنوان برازندگی معتبر می‌پذیریم. در غیر این صورت، نتیجه می‌گیریم که $g(x)$ بسیار نویزی و شلوغ بوده و از یک استراتژی نمونه‌گیری دوباره برای کاهش نویز استفاده می‌نماییم. این رویکرد کمی متناقض‌گونه به نظر می‌رسد. بنا به این رویکرد، هرچه نویز والد بیشتر باشد، احتمال قبول ارزیابی فرزند بیشتر می‌شود. ما به‌طور حسی انتظار داریم که عکس این قضیه صادق

[1] Branke
[2] Bui

باشد، یعنی هر چه مقادیر برازندگی والدین بیشتر باشد، احتمال نیاز به نمونه‌گیری دوباره‌ی برازندگی فرزند بیشتر شود [سیبرفلت[1] و همکاران، ۲۰۱۰].

یک رویکرد دیگر برای کاهش تعداد ارزیابی‌های برازندگی آن است که در ابتدای الگوریتم تکاملی تنها چند بار از نمونه‌گیری دوباره استفاده کرده و با پیشرفت الگوریتم تکاملی تعداد دفعات استفاده از این استراتژی را افزایش دهیم [سیبرفلت و همکاران، ۲۰۱۰]. این کار منطقی است چرا که معمولاً جستجوی الگوریتم تکاملی در ابتدای فرایند بهینه‌سازی نسبتاً خشن و شدید است. در نسل‌های ابتدایی الگوریتم تکاملی سعی دارد همسایگی‌های کلی راه‌حل بهینه را پیدا کند و در نسل‌های آخرین الگوریتم تکاملی سعی دارد به راه‌حل‌های دقیق‌تر برسد. بنابراین، ما تنها هنگامی که به انتهای فرایند بهینه‌سازی نزدیک می‌شویم به مقادیر دقیق تابع برازندگی نیاز پیدا می‌کنیم (مسئله‌ی ۲۱–۸ را ببینید).

در بسیاری از استراتژی‌های نمونه‌گیری دوباره فرض بر آن است که نویز برازندگی دارای توزیع نرمال است. این موضوع برای بسیاری از پدیده‌های نویزی در اندازه‌گیری‌ها و محاسبات صادق است، اما نه برای همه. بسیاری از روش‌های نمونه‌گیری دوباره که تا به حال در مقالات ارائه شده‌اند، در صورتی که نویز گاوسی نباشد، باید اصلاح شوند.

۲۱–۳–۲ تخمین برازندگی

معادله‌ی (۲۱–٤۱) یک روش میانگین‌گیری ساده برای تخمین برازندگی بر اساس نمونه‌های نویزی می‌باشد. با این حال، می‌توان از روش‌های احتمالاًتی دیگر نیز استفاده نمود. برای مثال، در [سانو[2] و کیتا[3]، ۲۰۰۲] فرض بر آن است که ذرات x_1 و x_2 که در فضای جستجو نزدیک به هم هستند، دارای مقادیر برازندگی مشابه می‌باشند:

$$f(x_1) \sim N(f(x_2), kd) \qquad (٤٣–٢١)$$

معادله‌ی بالا به این معنی است که برازندگی x_1 یک متغیر گاوسی اتفاقی با میانگین $f(x_2)$ و واریانس kd می‌باشد. k یک پارامتر مجهول و d فاصله‌ی میان x_1 و x_2 می‌باشد. اگر ارزیابی برازندگی $g(x_1)$ دارای واریانس σ^2 باشد، آنگاه

$$g(x_1) \sim N(f(x_2), kd + \sigma^2) \qquad (٤٤–٢١)$$

[1] Syberfeldt
[2] Sano
[3] Kita

با این فرضیات و با در اختیار داشتن ارزیابی‌های نویزی x_1 و x_2، می‌توان از محاسبات درست‌نمایی بیشینه برای تخمین برازندگی x_1 و x_2 استفاده نمود. این بدین معنی است که برای تخمین برازندگی یک ذره نه تنها از ارزیابی‌های نویزی خود ذره، بلکه از ارزیابی‌های نویزی همسایگانش نیز استفاده می‌نماییم [برنک و همکاران، ۲۰۰۱].

۲۱-۳-۳ الگوریتم تکاملی Kalman

الگوریتم تکاملی Kalman برای مسائل با ارزیابی‌های تابع برازندگی گران و شلوغ طراحی شده است. الگوریتم تکاملی Kalman، که در اصل اولین بار در زمینه‌ی الگوریتم‌های ژنتیک مطرح شد، رویکردی برای دنبال نمودن عدم قطعیت در مقادیر برازندگی و اختصاص دادن ارزیابی‌های برازندگی بر اساس این عدم قطعیت می‌باشد [استرود[1]، ۲۰۰۱].

یک فیلتر Kalman یک تخمین‌گر بهینه برای حالت‌های یک سیستم خطی پویا می‌باشد [سایمون، ۲۰۰۶]. در الگوریتم تکاملی Kalman فرض بر آن است که برازندگی یک ذره مانند x ثابت است. همچنین فرض بر آن است که نویز ارزیابی برازندگی تابعی از x نیست. با این فرضیات، می‌توان از فرم اسکالر و ساده‌تر فیلتر Kalman برای دنبال نمودن عدم قطعیت در هر تخمین برازندگی استفاده نمود. واریانس یک ارزیابی تابع برازندگی تنها را با R نمایش می‌دهیم. همچنین، واریانس تخمین برازندگی یک ذره مانند x بعد از k ارزیابی تابع برازندگی را با $P_k(x)$ نمایش می‌دهیم. مقدار kاُمین ارزیابی تابع برازندگی را نیز با $g_k(x)$ و تخمین برازندگی x پس از k ارزیابی تابع برازندگی را با $\hat{f}_k(x)$ نشان می‌دهیم. با این نشانه‌گذاری و با استفاده از نظریه فیلتر Kalman می‌توان برای $k = 0.1.2 \dots$ نوشت:

$$\hat{f}_{k+1}(x) = \hat{f}_k(x) + \frac{P_k(x)\left(g_{k+1}(x) - \hat{f}_k(x)\right)}{P_k(x) + R} \qquad (۴۵-۲۱)$$
$$P_{k+1}(x) = \frac{P_k(x)R}{P_k(x) + R}$$

برای تمامی xها، $P_0(x) = \infty$ قرار می‌دهیم که در این صورت خواهیم داشت:

$$\hat{f}_1(x) = g_1(x) \qquad (۴۶-۲۱)$$
$$P_1(x) = R$$

[1] Stroud

معادله‌ی بالا به این معنی است که تخمین ما از $\hat{f}_1(x)$ پس از اولین ارزیابی تخمین برازندگی $g_1(x)$ با همان اولین ارزیابی برابر است. همچنین، عدم قطعیت $P_1(x)$ در تخمین برازندگی پس از اولین ارزیابی با عدم قطعیت همان ارزیابی برابر است.

معادله‌ی (۲۱-٤٥) نشان می‌دهد هر بار که برازندگی x را ارزیابی می‌نماییم، تخمین خود از $\hat{f}(x)$ را بر اساس تخمین قبلی، تخمین آن و نتیجه‌ی آخرین ارزیابی تابع برازندگی $g(x)$ اصلاح می‌نماییم. معادله‌ی (۲۱-٤٥) نشان می‌دهد که

$$\begin{aligned}&\lim_{P_k(x)\to 0} \hat{f}_{k+1}(x) = \hat{f}_k(x)\\&\lim_{P_k(x)\to 0} \hat{f}_{k+1}(x) = g_{k+1}(x)\end{aligned} \qquad (٤٧-٢١)$$

به بیان دیگر، اگر کاملاً در مورد برازندگی x مطمئن باشیم $(P_k(x) = 0)$، آنگاه ارزیابی‌های بیشتر برازندگی x تأثیری بر روی تخمین ما از برازندگی‌اش نخواهد داشت. از سوی دیگر، اگر کاملاً در مورد برازندگی x نامطمئن باشیم $(P_k(x) \to \infty)$، آنگاه تخمین خود از برازندگی x را برابر ارزیابی بعدی تابع برازندگی قرار خواهیم داد.

معادله‌ی (۲۱-٤٥) همچنین نشان می‌دهد که برازندگی x را ارزیابی می‌نماییم، عدم قطعیت مقدار برازندگی $P(x)$ کاهش می‌یابد (به بیان دیگر تخمین برازندگی مطمئن‌تر می‌شود). معادله‌ی (۲۱-٤٥) نشان می‌دهد که

$$\begin{aligned}&\lim_{R\to 0} f_{k+1}(x) = g_{k+1}(x)\\&\lim_{R\to 0} P_{k+1}(x) = 0\\&\lim_{R\to \infty} \hat{f}_{k+1}(x) = \hat{f}_k(x)\\&\lim_{R\to \infty} P_{k+1}(x) = P_k(x)\end{aligned} \qquad (٤٨-٢١)$$

این نتایج به‌صورت حسی نیز منطقی هستند. اگر واریانس نویز تابع برازندگی (R) برابر صفر باشد، آنگاه ارزیابی تابع برازندگی بدون نقص بوده و بدین ترتیب تخمین ما برابر نتیجه‌ی ارزیابی تابع برازندگی بوده و عدم قطعیت موجود در تخمین تابع برازندگی برابر صفر خواهد بود. از سوی دیگر، اگر واریانس نویز تابع برازندگی R بی‌نهایت باشد، آنگاه نویز به قدری زیاد خواهد بود که ارزیابی‌های تابع برازندگی هیچ گونه اطلاعاتی در اختیار ما قرار نخواهد داد. در چنین حالتی، ارزیابی‌های اضافی تابع برازندگی تخمین ما از مقدار تابع برازندگی را تغییر نداده و میزان عدم قطعیت آن را نیز کاهش نخواهد داد.

الگوریتم تکاملی Kalman تخمین تابع برازندگی $\hat{f}_k(x)$ و واریانس $P_k(x)$ را برای هر ذره‌ی x از یک ارزیابی به ارزیابی دیگر دنبال می‌نماید $(k = 1.2 \dots)$. ما کسری مانند F از ارزیابی‌های در دسترس را، که

توسط کاربر تعیین می‌گردد، برای تولید و ارزیابی ذرات جدید اختصاص می‌دهیم. واریانس و تخمین برازندگی هر ذره‌ی جدید را مانند معادله‌ی (۲۱-٤٦) مقداردهی اولیه می‌نماییم. از کسر $(1-F)$ ارزیابی‌های در دسترس برای بازارزیابی ذرات موجود استفاده می‌نماییم. در این مورد، واریانس و تخمین برازندگی را بر اساس معادله‌ی (۲۱-٤٥) به‌روزرسانی می‌نماییم. هر بار که منابع کافی برای یک ارزیابی تابع برازندگی را در اختیار داشته باشیم، عددی اتفاقی مانند r که دارای توزیع یکنواخت بر روی $[0.1]$ می‌باشد، را تولید می‌نماییم. اگر $r < F$ باشد، آنگاه از فرایندهای جهش و بازترکیب برای تولید یک ذره‌ی جدید استفاده کرده و سپس برازندگی‌اش را محاسبه می‌نماییم. در غیر این صورت، ذره‌ی موجود را را بازارزیابی می‌نماییم.

وقتی زمان بازارزیابی یک ذره‌ی موجود فرا می‌رسد، از دو اصل هدایتی استفاده می‌نماییم. ابتدا، می‌توان با بازارزیابی ذراتی که واریانس تخمین برازندگیشان بزرگ است، اطلاعات بیشتری تولید نمود. دوم آنکه، می‌توان با بازارزیابی ذرات با برازندگی تخمینی بزرگ، می‌توان اطلاعات مفیدتری تولید نمود. این بدین معنی است که دستیابی به دقت بالا در تخمین ذرات با برازندگی کم برای ما اهمیت چندانی ندارد چرا که بازترکیب این ذرات جهت نسل‌های آتی الگوریتم تکاملی برای ما چندان جذاب نیست. به همین دلیل در [استرود، ۲۰۰۱] استراتژی ذیل برای انتخاب ذره‌ای مانند x_s جهت ارزیابی دوباره پیشنهاد شده است:

$$\begin{aligned} &\bar{f} \leftarrow \text{میانگین مقادیر برازندگی تخمینی جمعیت} \\ &\sigma \leftarrow \text{انحراف استاندارد مقادیر برازندگی تخمینی جمعیت} \\ &x_s \leftarrow argmax\{P(x): \hat{f}(x) > \bar{f} - \sigma\} \end{aligned} \qquad (۲۱-٤٩)$$

در معادله‌ی بالا از نوشتن اندیس k برای $\hat{f}(x)$ و $P(x)$ صرف نظر کردیم. از مقادیر به روز $\hat{f}(x)$ و $P(x)$ برای هر ذره‌ی x در معادله‌ی (۲۱-٤٩) استفاده می‌نماییم. این معادله نشان می‌دهد که در میان همه‌ی ذراتی که دارای برازندگی تخمینی با فاصله‌ی کمتر از یک انحراف استاندارد از میانگین می‌باشند، آنی را انتخاب می‌کنیم که دارای بیشترین عدم قطعیت برای ارزیابی دوباره می‌باشد. در این استراتژی فرض بر آن است که $f(x)$ برازندگی است، بنابراین $f(x)$ بزرگتر بهتر است.

هنوز فضای بسیاری برای توسعه‌ی بیشتر الگوریتم تکاملی Kalman وجود دارد. برای نمونه، چگونه می‌توان این الگوریتم را به مسائل بهینه‌سازی پویا تعمیم داد؟ کسر F را چگونه باید تعیین کرد؟ آیا راهی برای اقتباس F بر اساس عملکرد وجود دارد؟ چگونه می‌توان الگوریتم تکاملی Kalman را به الگوهای فیلتری کلی‌تر بسط داد [سایمون، ۲۰۰٦]؟

۲۱-۴ نتیجه‌گیری

مطالعات جامع در مورد تخمین برازندگی الگوریتم‌های تکاملی را می‌توان در [اونگ و همکاران، ۲۰۰٤]، [جین، ۲۰۰۵]، [نولز و ناکایاما[1]، ۲۰۰۸] و [شی و رشید، ۲۰۱۰] یافت. مجموعه‌ای از مقالات مرتبط با الگوریتم‌های تکاملی با ارزیابی‌های توابع برازندگی شلوغ را نیز می‌توان در [تن[2] و گوه، ۲۰۱۰] یافت. از جمله کتاب‌های مرتبط با الگوریتم‌های تکاملی پویا نیز می‌توان به [برنک، ۲۰۰۲]، [موریسون[3]، ۲۰۰٤]، [یانگ و همکاران، ۲۰۱۰] و [سیموئس[4]، ۲۰۱۱] اشاره نمود. همچنین یورگن برنک وبسایتی مختص الگوریتم‌های تکاملی برای بهینه‌سازی پویا راه‌اندازی نموده است [برنک، ۲۰۱۲].

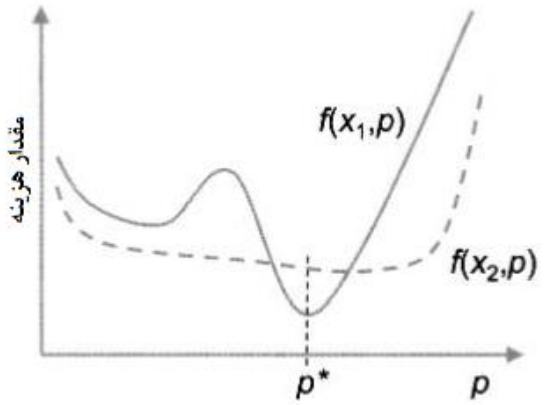
تخمین برازندگی تنها راه ممکن برای مواجه با توابع برازندگی گران نیست. می‌توان از محاسبات شبکه، که شامل استفاده از منابع محاسباتی توزیع شده که برای حل یک مسئله واحد ترکیب می‌شوند، نیز استفاده نمود [ملاب[5] و همکاران، ۲۰۰٦]، [لیم و همکاران، ۲۰۰۷]. از دیگر روش‌های صرفه جویی در وقت برای توابع برازندگی گران می‌توان به محاسبات خوشه‌ای، محاسبات ابری، و یا دیگر شکل‌های محاسبات توزیع شده اشاره نمود [توماسینی[6] و وانشی[7]، ۲۰۰۹]، [توماسینی و وانشی، ۲۰۱۰].

همچنین می‌توان تداوم و مقاومت در برابر تغییرات بردار تصمیم را مد نظر قرار داد. در این مورد، هنگامی که $f(x)$ را بهینه می‌نماییم، تداوم به معنای کیفیت $f(x+\Delta x)$ است. Δx تغییرات بردار تصمیم را نشان می‌دهد. هنگامی که یک بردار تصمیم x بهینه پیدا می‌شود، بردار تصمیمی که در راه‌حلمان پیاده‌سازی می‌کنیم ممکن است به دلایل پیاده‌سازی و عدم قطعیت‌های تولید، دچار تغییراتی شود. شکل ۲۱-۲٦ این وضعیت را نشان می‌دهد. کمترین هزینه در نقطه‌ی $x = x_1$ به دست می‌آید، اما تابع هزینه در این نقطه نسبت به تغییرات x بسیار حساس است. ما می‌توانیم یک هزینه‌ی زیربهینه را در $x = x_2$ به دست آوریم. در این صورت هزینه‌ی ما بدتر خواهد شد اما در عوض هزینه نسبت به تغییرات x مقاوم‌تر خواهد بود. بسته به مقدار تغییرات مورد انتظار، ممکن است x_2 به x_1 ترجیح داده شود. در این فصل در مورد مقاومت و تداوم بحثی ارائه نشد، اما این موضوع در مسائل دنیای واقعی از اهمیت بالایی برخوردار است. برای مروری خوب از این موضوع می‌توانید به [جین و برنک، ۲۰۰۵] و [برنک، ۲۰۰۲، فصل ۸] مراجعه کنید.

---

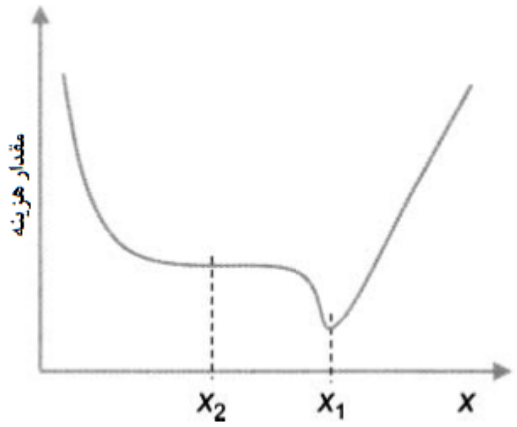
[1] Nakayama
[2] Tenne
[3] Morrison
[4] Simoes
[5] Melab
[6] Tomassini
[7] Vanneschi

هرچه تعداد مسائلی که الگوریتم‌های تکاملی به آن‌ها اعمال می‌شوند افزایش یابد، تحقیقات الگوریتم‌های تکاملی به سمت موضوعات مطرح شده در این فصل حرکت می‌کنند. از جمله برخی زمینه‌های جالب برای تحقیقات آتی می‌توان به کاربرد الگوریتم‌های تکاملی برای مسائلی اشاره نمود که شامل بیش از یکی از این ویژگی‌های می‌باشند. برای مثال، ما در مورد الگوریتم‌هایی برای مدیریت توابع پویا و یا الگوریتم‌هایی برای مدیریت توابع شلوغ بحث نمودیم. اما چگونه باید این الگوریتم‌ها را ترکیب نمود تا بتوان توابعی را که هم شلوغ و هم پویا هستند مدیریت کرد؟ یک سؤال جالب دیگر آن است که تخصیص K ارزیابی برازندگی جهت دستیابی به بهترین نتایج از یک الگوریتم تکاملی، چگونه باید صورت بپذیرد؟ توجه داشته باشید که می‌توان تعابیر مختلفی از واژه‌ی "بهترین" داشت. برای مثال، می‌توان واژه‌ی "بهترین" را بر اساس بهینه نمودن نتایج مورد انتظار الگوریتم تکاملی و یا بهینه نمودن بدترین نتیجه‌ی الگوریتم تکاملی و یا بهینه نمودن "بهترین نتیجه منهای سه سیگما"[1]، و یا هر ترکیبی از این موارد تعبیر نمود.



شکل ۲۱-۲۵ این شکل هزینه را به‌عنوان تابعی از مقدار پارامتر p نشان می‌دهد. p^* مقدار نامی پارامتر می‌باشد. $x = x_1$ بهترین هزینه را در $p = p^*$ به دست می‌دهد اما تابع در این نقطه مقاوم نیست. $x = x_2$ هزینه‌ی بدتری را به دست می‌دهد اما در این نقطه نسبت به تغییرات پارامتر مقاومت بیشتری وجود دارد.

[1] Best-case-minus-three-sigma



شکل ۲۱-۲۶ این شکل هزینه را به‌عنوان تابعی از متغیر مستقل x نشان می‌دهد. می‌توان دید که $x = x_1$ بهترین هزینه را به دست می‌دهد اما مقاومت تابع در این نقطه کم است. $x = x_2$ هزینه‌ی بدتری را ایجاد می‌کند اما تابع در این نقطه نسبت به تغییرات x مقاوم‌تر است.

مسائل

تمارین نوشتاری

۲۱-۱ یک تابع تک بعدی که از نقاط (1.3)، (2.1) و (3.4) تشکیل شده است را در نظر بگیرید.

الف) تطبیق خطی کمترین میانگین مربعات برای این تابع چیست؟ خطای RMS و min-max برای این تخمین چه قدر است؟

ب) تطبیق خطی min-max این تابع چیست؟ خطای RMS و min-max برای این تخمین چه قدر است؟

۲۱-۲ معادله‌ی (۲۱-۱۶) را ثابت کنید.

۲۱-۳ در متن کتاب بیان شده است که $s(x)$ از معادله‌ی (۲۱-۲٤) در نقاط نمونه برابر ۰ است. صحت این موضوع را ثابت کنید (راهنمایی: اگر x^* یکی از نقاط نمونه‌ی داده باشد، آنگاه r یکی از ستون‌های ماتریس R خواهد بود).

۲۱-٤ چه تعداد نمونه‌گیری فرامکعب لاتین برای یک ماتریس شبکه‌ی دو بعدی $M \times M$ وجود دارد؟

۲۱-۵ فرض کنید ارزیابی نویزی برازندگی x_1 دارای توزیع یکنواخت میان ۲- و ۲ می‌باشد. همچنین فرض کنید ارزیابی نویزی برازندگی x_2 دارای توزیع یکنواخت میان ۱- و ۳ می‌باشد.احتمال آنکه ارزیابی نویزی برازندگی x_1 از x_2 بزرگتر باشد چه قدر است؟

۲۱–۶ این جمله از بخش ۲۱–۳–۱ را اثبات کنید: "اگر تابع برازندگی را برای یک ذره N بار ارزیابی کنیم و مقادیر نویز این N نمونه از یکدیگر مستقل باشند، آنگاه واریانس میانگین تابع برازندگی با مرتبه‌ی N کاهش خواهد یافت".

۲۱–۷ فرض کنید ۱۰ ارزیابی تابع برازندگی نویزی زیر را که مربوط به یک ذره می‌باشند، در اختیار داریم:

$$[101.102.98.97.99.103.104.101.97.98]$$

از چه تعداد ارزیابی برازندگی باید میانگین‌گیری کرد تا به واریانس ۵ دست یافت؟

۲۱–۸ فرض کنید می‌خواهیم از یک الگوریتم تکاملی Kalman برای تخمین رازندگی ذره‌ی x استفاده نماییم. همچنین فرض کنید تخمین برازندگی ما $\hat{f}(x) = 10$ بوده و واریانس عدم قطعیت برابر ۱ می‌باشد. فرض کنید یک اندازه‌گیری برازندگی جدید $g(x) = 11$ با واریانس ۳ به دست آمده است. تخمین برازندگی جدید و واریانس عدم قطعیت آن چه قدر است؟

تمارین کامپیوتری

۲۱–۹ مثال ۲۱–۱ و ۲۱–۲ را برای تابع دو بعدی *آکلی* و فرض این که هر متغیر مستقل در دامنه‌ی $[-3.+3]$ واقع شده است، تکرار نمایید.

۲۱–۱۰ از معادله‌ی (۲۱–۳۸) جهت ارزیابی عملکرد سه رویکرد پویای به کار رفته در مثال ۲۱–۶ استفاده نمایید.

۲۱–۱۱ یک برنامه‌ی کامپیوتری بنویسید که به‌صورت تجربی نشان دهد میانگین N عبارت نویزی دارای واریانسی برابر با $1/N$ هر عبارت نویزی می‌باشد.

۲۱–۱۲ مثال ۲۱–۵ و ۲۱–۶ شبیه‌سازی کرده و مقادیر نسبت زیر را ردگیری نمایید:

p_r: هر چند دفعه یکبار ذرات اتفاقی R بهترین ذرات در میان ذرات جدید هستند؟

p_e: هر چند دفعه یکبار ذرات نخبه‌ی جهیده شده E بهترین ذرات در میان ذرات جدید هستند؟

p_d: هر چند دفعه یکبار ذرات دوگانه‌ی D بهترین ذرات در میان ذرات جدید هستند؟

در اینجا *بهترین* بر اساس هزینه‌ی میانگین تعریف می‌شود. این بدین معنی است که هر گاه ذرات جدید اتفاقی R، ذرات جدید نخبه‌ی جهیده E و ذرات جدید دوگانه D تولید می‌نمایید، هر چند دفعه یکبار هزینه‌ی میانگین R از هزینه‌ی میانگین E و هزینه‌ی میانگین D بهتر است و ...؟ تعداد نسل‌ها را به اندازه‌ی زیاد بگیرید که اطلاعات کافی برای نتیجه‌گیری درست و منطقی در دست داشته باشید.

ضمیمه

ضمیمه أ: چند توصیه‌ی عملی

نصیحت‌های خوب تقریباً همیشه نادیده گرفته می‌شوند، اما دلیلی برای نصیحت نکردن وجود ندارد.
آگاتا کریستی

این ضمیمه دربردارنده‌ی چند توصیه‌ی عملی برای محققین و متخصصین الگوریتم تکاملی می‌باشد. اگر الگوریتم تکاملی کار نکرد چه باید بکنیم؟ چگونه می‌توان عملکرد الگوریتم تکاملی را بهبود بخشید؟ چگونه یک محقق الگوریتم تکاملی خوب باشیم؟ در تحقیقات خود بر چه موضوعی باید تمرکز نماییم؟ کتاب *راهنمای میدانی برنامه‌نویسی ژنتیک* شامل توصیه‌های عملی بسیار خوبی می‌باشد [پولی و همکاران، ۲۰۰۸، فصل ۱۳]. توصیه‌های موجود در این کتاب بیشتر در مورد برنامه‌نویسی ژنتیک است، اما بیشتر این توصیه‌ها به قدری کلی هستند که می‌توان آن‌ها را به هر نوع الگوریتم تکاملی دیگر نیز اعمال نمود. به همین جهت برخی از این توصیه‌ها در این ضمیمه ذکر شده‌اند.

الف.۱ اشکالات را بررسی کنید

بسیاری از دانشجویان و محققین تازه‌کار انتظار دارند کدهایشان، و یا حتی کدهای دیگران همان بار اول اجرا شود. هیچ کد بدون اشکالی وجود ندارد. ما باید نرم‌افزار خود را به خوبی درک کنیم تا بتوانیم اشکالات موجود در کد و یا اشکالات ناشی از نحوه‌ی استفاده از کد را بیابیم. این بدین معنی است که باید خط به خط کد را بررسی کرده و متغیرها را با استفاده از اشکال‌یاب[1] بررسی نماییم. ما باید بفهمیم چه متغیرهایی در چه متغیرهای دیگر بارگذاری شده و علت آن را نیز دریابیم. این کار را حتی در صورت اجرا شدن کد در همان بار اول نیز باید انجام دهیم. این که کد اجرا می‌شود دلیلی بر کارکرد صحیح آن نیست.

نمی‌خواهیم بگوییم که نرم‌افزارهای آماده به اجرا کار نمی‌کنند و یا ما باید همه چیز را در مورد برنامه‌های کامپیوتری که استفاده می‌کنیم بدانیم. اما نرم‌افزارهای تولیدی از نظر کیفی با نرم‌افزارهای تحقیقاتی و مهندسی متفاوت است. اگر یک نرم‌افزار تولیدی کار نکند معمولاً می‌توان نتیجه را مشاهده کرده و می‌توان با اقدامی خلاقانه سعی در رفع مشکل نمود (برای مثال برنامه را مجددا راه‌اندازی کرده و یا تنظیمات را عوض کرد تا نتایج مطلوب به دست آیند).

با این حال، اگر نرم‌افزار تحقیقاتی ما کار نکند، آنگاه باید نرم‌افزار را خودمان درست کنیم. برای این کار، باید درک درستی از نرم‌افزار داشته باشیم. مهمتر آنکه حتی اگر به نظر برسد که نرم‌افزار کار می‌کند، ممکن است عملکرد صحیح را نداشته باشد. اگر درک درستی از نرم‌افزار نداشته باشیم، هیچ‌گاه نخواهیم توانست از

---

[1] Debugger

کارکرد صحیح آن مطمئن باشیم و به همین دلیل هیچ گاه نخواهیم توانست به نتایج به دست آمده اطمینان کنیم.

در تحقیقات الگوریتم تکاملی (یا هر تحقیق مهندسی دیگری) هیچ راه میانبری وجود ندارد. بهتر است همیشه خود برنامه‌ی مورد نیازمان را بنویسیم. اگر می‌خواهیم از کد شخص دیگری استفاده نماییم، باید کد را مطالعه کرده و آن را بفهمیم.

الف.۲ الگوریتم‌های تکاملی دارای طبیعت اتفاقی هستند

الگوریتم‌های تکاملی دارای طبیعت اتفاقی هستند. این بدین معنی است که با هر بار اجرای آن‌ها ممکن است نتایج متفاوتی به دست آیند. اگر یک الگوریتم تکاملی را برای حل یک مسئله ۱۰ بار اجرا نمودیم و الگوریتم در هیچ یک از این ۱۰ بار کار نکرد، می‌توان به طرز ساده‌انگارانه‌ای نتیجه گرفت که الگوریتم تکاملی کلاً کار نمی‌کند و یا اینکه مسئله حل نشدنی است. اما اگر احتمال حل مسئله ۲۰٪ باشد، آنگاه احتمال آنکه الگوریتم تکاملی هر ۱۰ بار با شکست مواجه شود ۱۰٪ خواهد بود.

بر عکس این قضیه نیز صادق است به این معنی که اگر الگوریتم تکاملی هر ۱۰ بار در حل مسئله موفق باشد، آنگاه می‌توان به طرز ساده‌انگارانه‌ای نتیجه گرفت که الگوریتم تکاملی همواره کار می‌کند. اما اگر احتمال شکست الگوریتم تکاملی ۲۰٪ باشد، آنگاه احتمال موفق شدن آن در هر ۱۰ بار اجرا ۱۰٪ خواهد بود. نتیجه آنکه داشتن درکی مناسب از نظریه‌ی احتمال یکی از ملزومات فهم کامل طبیعت اتفاقی الگوریتم تکاملی می‌باشد.

الف.۳ تغییرات کوچک ممکن است تأثیرات بزرگی داشته باشند

عملی مانند نحوه‌ی تغییر روش مدیریت ذرات تکراری، یا تغییرات بسیار کوچک در نرخ جهش و یا تغییر روش انتخاب که بی‌ضرر به نظر می‌رسد، می‌تواند به شدت عملکرد یک الگوریتم تکاملی را تغییر دهد. هیچ وقت نباید فرض کنیم که یک تغییر کوچک بی‌اهمیت است. این بدین معنی است که هر گاه نتیجه‌ی خوبی به دست می‌آوریم، باید تنظیماتی را که آن نتیجه‌ی خوب را به دست دادند ذخیره نماییم. حتی باید حالت اولیه‌ی عدد اتفاقی را نیز ذخیره کنیم تا بتوان نتایج را بعدا تکرار نمود (ضمیمه‌ی ب.۳-۲ را ببینید). در غیر این صورت، اگر نتایج خوبی به دست آوریم و بعد تنظیمات را جهت دستیابی به عملکرد بهتر عوض کنیم، ممکن است نتایج خوبمان را از دست داده و از آنجایی که تنظیمات پارامترها را فراموش کرده‌ایم، نتوانیم نتایج خوبمان را دوباره تکرار کنیم.

الف.٤ تغییرات بزرگ ممکن است تأثیر کوچکی داشته باشند

بر خلاف نکته‌ی قبل، گاهاً الگوریتم تکاملی نسبت به تغییرات بزرگ در تنظیمات پارامتر غیرحساس است. این ویژگی هنگامی که الگوریتم تکاملی عملکرد مناسبی دارد، ویژگی خوبی است چرا که باعث می‌شود الگوریتم تکاملی نسبت به تغییرات آن پارامتر مقاوم باشد. اما اگر عملکرد الگوریتم تکاملی ضعیف باشد، این عدم حساسیت به تغییرات پارامتر ویژگی بدی خواهد بود. عملکرد ضعیف الگوریتم تکاملی ممکن است به این دلیل باشد که حل مسئله برای الگوریتم تکاملی بسیار سخت بوده و یا مسئله به‌گونه‌ای فرمول‌بندی شده است که برای الگوریتم تکاملی قابل حل نیست. هیچ تضمینی برای به دست آمدن نتایج خوب از الگوریتم تکاملی وجود ندارد. به همین دلیل نباید تصور کنیم اگر بتوانیم تنظیمات درست پارامتر را پیدا کنیم، الگوریتم تکاملی موفق عمل خواهد کرد. در اینجاست که باید میان استقامت خود و احتمال اینکه ممکن است در حال به هدر دادن زمانمان باشیم، تعادل برقرار نماییم.

الف.٥ جمعیت‌ها دارای اطلاعات بسیار زیادی هستند

اگر الگوریتم تکاملی خوب کار نمی‌کند، باید جمعیت را در نسل‌های مختلف مطالعه کنیم. ترکیب جمعیت می‌تواند اطلاعات زیادی را در اختیار ما قرار دهد. اگر ببینیم که جمعیت در حال همگرا شدن به یک راه‌حل نامزد خاص می‌باشد، آنگاه می‌دانیم که باید با دقت بیشتری با ذرات تکراری برخورد کنیم. اگر ببینیم بهبودی در ذراتی که بازترکیب می‌شوند حاصل نمی‌شود، می‌فهمیم که باید استراتژی بازترکیب خود را اصلاح نماییم. با دنبال نمودن جهش‌ها می‌توانیم بفهمیم نرخ جهش زیاد است یا کم. اگر ببینیم که جمعیت در نسل‌های ابتدایی بهبود کمی دارد و یا اصلا بهبودی ندارد، می‌توان فهمید به کاوش بیشتر و به انتفاع کمتر نیاز داریم. محدودیت تعداد جزییاتی که می‌توانیم با مطالعه‌ی جمعیت به دست آوریم تنها به میزان خلاقیت و پشت کار ما بستگی دارد.

الف.٦ تنوع را تقویت کنید

این نکته به نکته‌ی قبل مرتبط است. ما باید از میزان تنوع جمعیت آگاه باشیم. اگر جمعیت دارای میزان تنوع کافی نباشد احتمالاً عملکرد خوبی نخواهد داشت. ما باید هر کاری که می‌توانیم جهت تقویت تنوع، بی‌آنکه مانع از بهره‌وری از راه‌حل‌های نامزد خوب شویم، انجام دهیم. باید به خاطر داشت برای داشتن یک الگوریتم تکاملی موفق، داشتن تنها یک یا چند ذره‌ی خوب کافی است. هرچه تنوع بیشتر باشد، شانس موفقیت بیشتر خواهد بود.

الف.۷ از اطلاعات خاص مسئله استفاده نمایید

هر چه درک بهتری از مسئله داشته باشیم، قادر خواهیم بود راه‌حل‌های بهتری پیدا کنیم. معمولاً یک الگوریتم تکاملی به‌عنوان بهینه‌سازی بدون مدل در نظر گرفته می‌شود. این بدین معناست که نیازی به وارد نمودن اطلاعات خاص مسئله در الگوریتم تکاملی نمی‌باشد. با این حال، اگر از اطلاعات خاص مسئله در الگوریتم تکاملی استفاده نماییم، قطعا می‌توانیم راه‌حل بهتری پیدا کنیم. یک الگوریتم تکاملی خود یک بهینه‌گر جهانی خوب است، اما استفاده از یک روش جستجوی محلی مانند تپه‌نوردی یا نزول گرادیانی می‌تواند به قدری نتایج الگوریتم تکاملی را بهبود بخشد. این موضوع خود می‌تواند مرز میان موفقیت و شکست را تعیین کند.

الف.۸ نتایج خود را مکرراً ذخیره نمایید

حافظه‌های کامپیوتری قیمت چندانی ندارند. ما باید تنظیمات مسئله، نسخه‌های قدیمی نرم‌افزار، نتایج و نتایج میانی را ذخیره نماییم. بنابراین باید منظم باشیم تا هر موقع که نیاز داشتیم، با صرف کمترین زمان، در میان اطلاعات ذخیره شده‌ی خود گشته و آنچه را که می‌خواهیم پیدا کنیم.

اجراهای الگوریتم تکاملی معمولاً به لحاظ محاسباتی پرهزینه هستند. گاهاً ممکن است مجبور باشیم الگوریتم را برای روزها و یا هفته‌ها اجرا کنیم تا نتایج خوبی به دست آوریم. یک راه مؤثر و منظم برای انجام این کار آن است که الگوریتم تکاملی را برای یک روز اجرا کرده، نتایج را ذخیره کرده و سپس الگوریتم را دوباره و با یک جمعیت جدید که با استفاده از نتایج قبلی مقداردهی اولیه شده است، آغاز کنیم. این کار به الگوریتم تکاملی اجازه می‌دهد هم از راه‌حل‌های نامزد خوبی که تا کنون کشف شده‌اند بهره برده و هم برای مدت طولانی‌تر در حال اجرا باشد.

الف.۹ اهمیت آماری را دریابید

ما باید از اهمیت آماری نتایج آزمایش‌هایمان مطلع باشیم. این موضوع بدین معنی است که باید درک خوبی از آمار و قضیه‌ی no-free-lunch داشته باشیم (ضمیمه‌ی ب را ببینید). این موضوع همچنین به این معنی است که باید صحت نتایج الگوریتم تکاملی را بر روی مجموعه‌ای از داده‌های آزمایشی امتحان نماییم. می‌توان از یک الگوریتم تکاملی جهت پیدا نمودن راه‌حل‌های خوب برای یک مسئله‌ی بهینه‌سازی استفاده نمود، اما آزمایش عملکرد راه‌حل بر روی داده‌هایی که در طول دوره‌ی آموزش الگوریتم تکاملی به کار نرفته‌اند نیز از اهمیت بالایی برخوردار است. برای مثال، ممکن است از یک الگوریتم تکاملی برای پیدا نمودن پارامترهایی که یک مرتب‌ساز را بهینه می‌کنند استفاده نماییم و همچنین مجموعه‌ای از داده‌های آزمایشی در

اختیار داریم که از آن برای ارزیابی تابع برازندگی استفاده می‌نماییم. با این حال، اگر در این مرحله متوقف شویم، چیزی جز اینکه نشان دهیم که مرتب‌سازی می‌تواند داده‌های یادگیری را به خاطر بسپارد انجام نداده‌ایم. آزمایش حقیقی مرتب‌ساز زمانی صورت می‌گیرد که ببینیم عملکرد آن بر روی داده‌ای که هنوز ندیده است، چگونه است. این داده، مجموعه‌ی اعتبارسنجی نام دارد. این ایده‌ها را در این کتاب مطرح نمی‌کنیم اما اطلاع از ایده‌های اساسی اعتبارسنجی بسیار اهمیت دارد [هیستی[1] و همکاران، ۲۰۰۹].

الف.۱۰ خوب بنویسید

اگر بهترین تحقیق ممکن در زمینه‌ی الگوریتم‌های تکاملی را انجام دهیم اما آن را با سایرین در میان نگذاریم، تحقیق هیچ ارزشی نخواهد داشت. تحقیق صورت می‌گیرد تا به اشتراک گذاشته شود. دانشمند بزرگ انگلیسی مایکل فارادی می‌گوید: "کار کن، تمام کن و منتشر کن" [بوریج[2]، ۲۰۰٤،صفحه ۱۲۱]. این جمله بدین معناست که فرایند تحقیق تا زمانی که نتایج منتشر نشده‌اند، هنوز کامل تلقی نمی‌شود. نگارش فنی مهارتی است که این روزها دانشجویان، مهندسین و محققین به طرز دردناکی از آن بی‌بهره هستند. اگر بتوانیم بهتر بنویسیم، محققین حرفه‌ای بهتری خواهیم بود. یادگیری این که چگونه بهتر بنویسیم فرایند اسرارآمیزی نیست. فقط باید تمرین کنیم.

الف.۱۱ بر اصول نظری تأکید داشته باشید

تعداد بیش از حد زیادی از تحقیقات الگوریتم‌های تکاملی این روزها به بازی با پارمترها و ترکیب الگوریتم‌های تکاملی با سایر الگوریتم‌های بهینه‌سازی اختصاص یافته است. با توجه به تعداد الگوریتم‌های تکاملی در دسترس، تعداد پارامترهای میزان‌سازی هر یک، و همچنین تعداد الگوریتم‌های بهینه‌سازی غیرتکاملی در دسترس، تقریباً بی‌نهایت روش ممکن برای اصلاح، ترکیب و میزان‌سازی الگوریتم‌های تکاملی جهت دستیابی به عملکرد بهتر وجود دارد. اما این تحقیقات چندان باعث پیشرفت علم نمی‌شوند. افق دید این گونه تحقیقات بسیار کوتاه است و بصیرت چندانی، به جز نتایج لحظه‌ای، به دست نمی‌دهند. بسیاری از الگوریتم‌های تکاملی از فقدان حمایت نظری و تحلیل ریاضی رنج می‌برند. اگر در تحقیقات خود کمی بیشتر بر نظریات و ریاضیات تأکید داشته باشیم، می‌توانیم در درک خود از الگوریتم‌های تکاملی تفاوتی اساسی ایجاد نماییم. یک مقاله که به نظریات پرداخته باشد، به اندازه‌ی یک دوجین مقاله‌ای که تنها به پارامترهای میزان‌سازی پرداخته باشند، ارزش دارد.

[1] Hastie
[2] Beveridge

الف.۱۲ بر عمل تأکید کنید

امروزه بسیاری از تحقیقات الگوریتم‌های تکاملی به محک‌ها محدود شده‌اند. اما اگر بخواهیم که تحقیقمان در دنیا تفاوتی ایجاد کند، باید همکاری نزدیکی با صنعت داشته باشیم و مسائلی را که از دید مهندسین مهم هستند حل نماییم. الگوریتم‌های تکاملی ما ممکن است تماماً به بهینه نمودن محک‌ها مشغول باشند ولی هیچگاه رنگ دنیای واقعی را نبینند. اما این دلیل اصلی وجود و اختراع الگوریتم‌های تکاملی نیست [فوگل و همکاران، ۱۹۹٦]، [فوگل،۱۹۹۹]. این الگوریتم‌ها به وجود آمدند تا مسائل دنیای واقعی را حل کنند و به جامعه خدمت کنند. محک‌ها مهم هستند اما فقط از این لحاظ که راه را برای طراحی و ایجاد الگوریتم‌های تکاملی که در دنیای واقعی کاربرد دارند، هموار سازند. هدف اصلی در تحقیقات الگوریتم تکاملی را فراموش نکنید.

ضمیمه ب: قضیه No Free Lunch و آزمایش عملکرد

انتظار آن است که جفت الگوریتم‌های A و B وجود دارند به‌طوری که A به‌صورت میانگین بهتر از B عمل می‌نماید ... یکی از نتایج اصلی این مقاله آن است که چنین انتظاراتی غلط هستند.

دیوید ولپرت[1] و ویلیام مکردی[2] [ولپرت و مکردی، ۱۹۹۷، صفحه ٦٧]

سه نوع دروغ وجود دارد: دروغ، دروغ‌های لعنتی و آمار

مارک تواین [تواین، ۲۰۱۰، صفحه ۲۲۸]

این ضمیمه به بحث در مورد دو موضوع مجزا اما مرتبط می‌پردازد. این ضمیمه برای دانشجویان و محققین الگوریتم تکاملی بسیار حائز اهمیت است. با این حال این قسمت تحت عنوان ضمیمه آورده شده است چرا که مستقیماً با الگوریتم‌های تکاملی مرتبط نیست، اما ضمیمه بودن آن از اهمیت آن نمی‌کاهد.

بخش ب.۱ قضیه‌ی No Free Lunch (NFL) را از منظری حسی و غیرریاضی مورد بررسی قرار می‌دهد. قضیه‌ی NFL بیان می‌دارد که تحت شرایطی خاص همه‌ی الگوریتم‌ها عملکرد یکسانی دارند. بخش ب.۲ چگونگی ارائه‌ی نتایج شبیه‌سازی الگوریتم‌های تکاملی به نحوی گمراه‌کننده را شرح داده و نحوه‌ی ارائه‌ی درست را نیز معرفی می‌کند. بخش ب.۳ شامل نکاتی در مورد رابطه‌ی میان روش‌های به کار رفته در این کتاب و همچنین راهنمای تحقیقاتی که در این ضمیمه مورد بحث واقع می‌شوند، می‌باشد.

ب.۱ قضیه‌ی No Free Lunch

قضیه‌ی No Free Lunch، که ابتدا توسط دیوید ولپرت و ویلیام مکردی به‌صورت رسمی معرفی شد [ولپرت و مکردی، ۱۹۹۷]، بسیار جالب است:

عملکرد تمام الگوریتم‌های تکاملی هنگامی که بر روی تمام مسائل ممکن میانگین‌گیری می‌شوند، یکسان می‌باشد.

این بدین معناست که در کل، هیچ الگوریتم تکاملی بهتر از دیگری نیست و هیچ یک نیز بدتر از دیگری نیست. توجه داشته باشید که NFL چیزی بیش از یک بیان کلی و یا یک قاعده‌ی سرانگشتی است؛ این یک قضیه‌ی ریاضی است. برای انواع خاصی از مسائل، برخی الگوریتم‌ها بهتر از سایر الگوریتم‌ها عمل می‌کنند. اما قضیه‌ی NFL باید از مطرح نمودن ادعاهای بی اساس در مورد الگوریتم تکاملی مورد علاقه‌ی کاربر جلوگیری نماید و کمک کند کاربر ادعاهای فروتنانه‌تری مطرح کند.

---

[1] David Wolpert
[2] William Macready

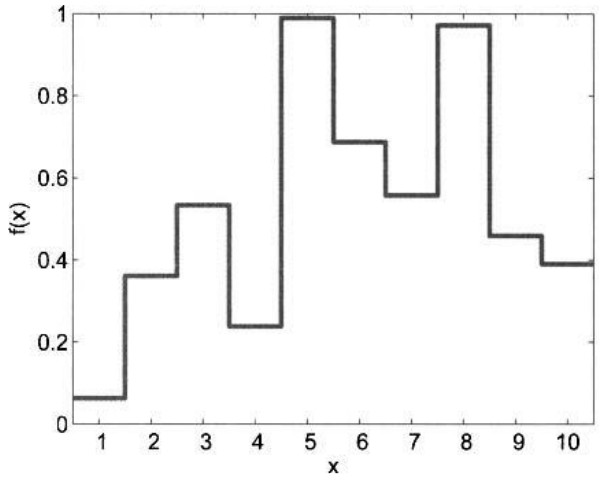
به لحاظ فنی، قضیه‌ی NFL تنها در مورد مسائل بهینه‌سازی گسسته مصداق دارد. با این حال، تمام توابع برازندگی دنیای واقعی در نهایت گسسته می‌شوند. به هر حال، ما راه‌حل‌های نامزد را معرفی کرده و برازندگی را با کامپیوترهای دیجیتالی اندازه می‌گیریم. بنابراین، می‌توان برای تمامی اهداف عملی، واژه‌ی "گسسته" را از NFL حذف کرد.

به‌صورت حسی انتظار می‌رود عملکرد میانگین برخی الگوریتم‌های بهینه‌سازی از سایر الگوریتم‌ها بهتر باشد. برای مثال، دو الگوریتم A و B را در نظر بگیرید. الگوریتم A یک الگورتم فرود از تپه مانند الگوریتم‌های معرفی شده در بخش ۲-٦ بوده و الگوریتم B نیز یک تولیدکننده اعداد اتفاقی می‌باشد. فرض کنید هر دو الگوریتم در تلاش برای یافتن مینیمم تابعی مانند $f(x)$ هستند. اگر $f(x) = x^2$ باشد، الگوریتم A بسیار بهتر از جستجوی اتفاقی عمل خواهد کرد. با این حال، جستجوی اتفاقی گاهاً ممکن است عددی را تولید کند که بسیار به مینیمم نزدیک می‌باشد. در چنین حالت‌هایی الگوریتم B بهتر از A عمل خواهد کرد.

می‌توان بحث نمود که تعداد بی‌نهایت تابع مشتق‌پذیر مشابه $f(x) = x^2$ وجود دارد و الگوریتم A عملکرد بسیار خوبی بر روی آن‌ها دارد در حالی که جستجوی اتفاقی معمولاً عملکرد چندان خوبی در مورد این توابع ندارد. با این حال، تعداد بی‌نهایت تابع غیرعادی نیز وجود دارد که جستجوی اتفاقی عملکرد بهتری بر روی آن‌ها خواهد داشت. این موضوع در مثال زیر نشان داده شده است.

مثال ۲-۱

تابع گسسته‌ی شکل ب.۱ را در نظر بگیرید. این تابع دارای فضای جستجویی با اندازه‌ی ۱۰ بوده و به‌صورت اتفاقی تولید شده است. می‌توان از یک الگوریتم فرود از تپه برای پیدا کردن مینیمم این تابع استفاده نمود. جستجو را از نقطه‌ای اتفاقی در فضای جستجو آغاز می‌کنیم. الگوریتم برای تعیین جهت حرکت شروع به آزمایش مقادیر همسایه می‌کند. هنگامی که الگوریتم در یکی از مقادیر مینیمم محلی $x \in \{4.7.10\}$ به دام می‌افتد، با یک مقدار اتفاقی مجددا شروع می‌نماید. الگوریتم نزول از تپه به‌طور میانگین به ۱۹ ارزیابی تابع برای پیدا کردن مینیمم نیاز دارد. این در حالی است که تعداد متوسط ارزیابی‌های تابع مورد نیاز برای جستجوی اتفاقی دقیقاً ۱۰ می‌باشد. دلیل طولانی شدن پیدا کردن مینیمم توسط الگوریتم فرود از تپه آن است که این الگوریتم برای این که بتواند مینیمم کلی را پیدا کند، باید از نقطه‌ی $x \in [1.3]$ شروع نماید، در غیر این صورت الگوریتم به یک مینیمم محلی همگرا می‌شود و باید با یک مقدار اتفاقی جدید دوباره از اول شروع کند. هر نقطه‌ی شروع دیگری غیر از $x \in [1.3]$ باعث می‌شود الگوریتم فرود از تپه زمان خود را در همسایگی محلی تلف کرده و به مینیمم جهانی نرسد.



شکل ب.۱ مثال ۲-۱: در این مسئله‌ی مینیمم‌سازی، تعداد ارزیابی‌های تابع برای پیدا کردن مینیمم جهانی توسط الگوریتم فرود از تپه برابر ۱۹ بوده، در حالی که این عدد برای جستجوی اتفاقی برابر ۱۰ می‌باشد.

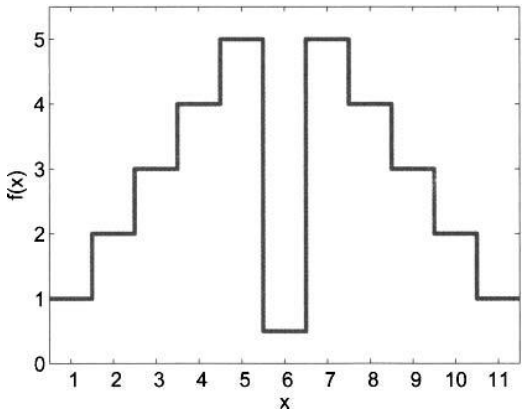
بی‌نهایت تابع غیرعادی مانند شکل ب.۱ وجود دارد. جستجوی اتفاقی عملکرد بهتری بر روی این توابع نسبت به الگوریتم فرود از تپه دارد. به ازای هر تابع عادی، که عملکرد الگوریتم فرود از تپه بر روی آن بهتر از جستجوی اتفاقی می‌باشد، یک تابع غیرعادی وجود دارد که عملکرد جستجوی اتفاقی در مورد آن بهتر از الگوریتم فرود از تپه است. می‌توان دید که اگر بر روی تمام توابع ممکن میانگین بگیریم، عملکرد هر دو الگوریتم به یک میزان خوب خواهد بود.

شاید این موضوع که جستجوی اتفاقی در مینیمم کردن تابعی غیرعادی مانند تابع شکل ب.۱ عملکردی مؤثرتر از الگوریتم فرود از تپه دارد، چندان موجب شگفتی نباشد. چیزی که بیشتر باعث شگفتی است آن است که قضیه‌ی NFL بیان می‌دارد، به‌طور میانگین، عملکرد الگوریتم تپه‌نوردی و الگوریتم فرود از تپه در مینیمم‌سازی تابع کاملاً مشابه هم است. بنابراین، اگر می‌خواهیم تابعی را مینیمم کنیم، باید از کدام یک از این دو الگوریتم باید استفاده نماییم؟ به‌صورت حسی می‌توان گفت که برای این کار باید از الگوریتم فرود از تپه استفاده کنیم. با این حال، قضیه‌ی NFL به ما می‌گوید که اگر بر روی تمام توابع ممکن میانگین بگیریم، الگوریتم تپه‌نوردی به همان خوبی الگوریتم فرود از تپه عمل خواهد کرد. این موضوع در مثال زیر نشان داده شده است.

مثال ۲-۲

تابع گسسته‌ی نشان داده شده در شکل ب.۲ را در نظر بگیرید. این تابع می‌تواند گمراه‌کننده باشد چرا که تغییرات محلی در تابع هزینه باعث می‌شود انتظار داشته باشیم مینیمم در یکی از مقادیر اکسترمم x واقع

شده باشد. الگوریتم فرود از تپه از اطلاعات محلی در جستجوی خود استفاده می‌کند و به همین دلیل معمولاً در $x = 1$ یا $x = 11$ به دام می‌افتد و باید دوباره از نو شروع کند. الگوریتم فرود از تپه تنها زمانی موفق خواهد بود که از نقطه‌ی $x \in [3.5]$ شروع نماید. از سوی دیگر، اگر $x < 5$ باشد الگوریتم تپه‌نوردی همواره به $x = 5$ و اگر $x > 7$ باشد الگوریتم تپه‌نوردی همواره به $x = 7$ خواهد رسید. پس از آن که الگوریتم به $x = 5$ یا $x = 7$ رسید، در حرکت بعدی به مینیمم جهانی خواهد رسید چرا که به دنبال جهت صعودی خواهد گشت. الگوریتم تپه‌نوردی هیچ گاه به بیش از ٦ ارزیابی تابع برای رسیدن به مینیمم جهانی نخواهد داشت. به‌صورت میانگین الگوریتم فرود از تپه به ١٦، جستجوی اتفاقی به ١١ و الگوریتم تپه‌نوردی به ٥ ارزیابی تابع برای رسیدن به مینیمم جهانی نیاز خواهد داشت. می‌توان بی‌نهایت تابع با توپولوژی مشابه تابع شکل ب.٢ ایجاد نمود. اگر چه که به‌صورت حسی به نظر می‌رسد باید از الگوریتم فرود از تپه جهت مینیمم نمودن توابع استفاده نمود، اما می‌توان دید که با میانگین‌گیری بر روی تمام توابع ممکن، الگوریتم تپه‌نوردی دارای عملکردی به خوبی الگوریتم فرود از تپه می‌باشد.



شکل ب.٢ مثال ٢-٢: برای این مسئله‌ی مینیمم‌سازی و به‌طور میانگین، الگوریتم فرود از تپه به ١٦، جستجوی اتفاقی به ١١ و الگوریتم تپه‌نوردی به ٥ ارزیابی تابع جهت رسیدن به مینیمم جهانی در $x = 6$ نیاز دارد. بنابراین، در این مسئله الگوریتم تپه‌نوردی عملکرد بهتری نسبت به الگوریتم فرود از تپه دارد.

مثال ٢-٣

یک GA، که برای حل یک مسئله‌ی دشوار در دنیای واقعی بسیار ظریف و به خوبی میزان‌سازی شده است، را در نظر بگیرید. انتظار می‌رود الگوریتم GA عملکرد بهتری نسبت به یک شخص که به‌صورت اتفاقی راه‌حل‌ها را حدس می‌زند، داشته باشد. اما فرض کنید به خیابان می‌روید و از اولین نفری که می‌بینید می‌خواهید یک عدد انتخاب کند. عددی که آن شخص انتخاب می‌کند راه‌حل یک مسئله‌ی بهینه‌سازی خواهد

بود. در حقیقت، عددی که آن شخص انتخاب می‌کند، راه‌حل تعداد بی‌نهایت مسئله‌ی بهینه‌سازی خواهد بود. ممکن است مسئله‌ی خاصی که ما در تلاش برای حل آن هستیم با GA بهتر حل شود تا یک تولیدکننده‌ی اعداد اتفاقی، اما اگر بر روی تمام مسائل ممکن میانگین بگیریم، حدس زدن اتفاقی عملکردی به خوبی GA خواهد داشت.

می‌توان با پرسیدن سؤالاتی ساده در مورد مجموعه‌ی تمام توابع ممکن، مجموعه‌ی تمام الگوریتم‌های بهینه‌سازی ممکن و مجموعه‌ی تمام رفتارهای ممکن از الگوریتم‌های بهینه‌سازی، قضیه‌ی NFL را کمی شفاف نمود [وایتلی و واتسون، ۲۰۰۵]. این رویکرد برای درک قضیه‌ی NFL در مثال زیر نشان داده شده است.

مثال ۲-٤

یک مسئله‌ی مینیمم‌سازی ساده با دامنه‌ی $x \in [1.3]$ را در نظر بگیرید. فرض کنید تمام الگوریتم‌های جستجوی ممکن را با $x = 1$ آغاز کنیم. تعداد توابعی که در آن‌ها $f(1) > f(2) > f(3)$ باشد بی‌نهایت است. به همین ترتیب، تعداد توابعی که در آن‌ها $f(1) > f(3) > f(2)$ باشد نیز بی‌نهایت است. بنابراین، تناظری یک به یک میان تعداد توابعی که در $x = 3$ مینیمم شده با تعداد توابعی که در $x = 2$ مینیمم می‌شوند، وجود دارد.

حال مجموعه‌ای از تمام الگوریتم‌های بهینه‌سازی مانند A را در نظر بگیرید که در قدم اولشان در فرایند جستجو، $x = 2$ را امتحان می‌نمایند. به‌طور مشابه مجموعه‌ای از تمام الگوریتم‌های بهینه‌سازی مانند $\bar{A}$ را در نظر بگیرید که در قدم اول $x = 3$ را امتحان می‌نمایند. از آنجایی که تناظری یک به یک میان تعداد توابعی که در $x = 3$ مینیمم شده و تعداد توابعی که در $x = 2$ مینیمم می‌شوند وجود دارد، می‌توان دید که $\bar{A}$ در نیمی از توابع ممکن زودتر از A مینیمم را پیدا کرده و در نیمی دیگر دیرتر از A مینیمم را خواهد یافت.

بحث‌هایی که تا به اینجا در مورد قضیه‌ی NFL ارائه شد، اثباتی برای این قضیه ارائه نمی‌دهند. این مباحث بیشتر توضیحاتی در مورد این قضیه بوده و به‌صورت حسی دلیل درستی این قضیه را نشان می‌دهند. اثبات ریاضی این قضیه در [رادکلیف[1] و ساری، ۱۹۹۵] و [ولپرت، مکردی، ۱۹۹۷] ارائه شده است. قضیه‌ی NFL را می‌توان به طرق مختلف بیان نمود. در اینجا چند بیان مختلف این قضیه آورده شده است [شوماخر[2] و همکاران، ۲۰۰۱].

[1] Radcliffe
[2] Schumacher

- در صورت میانگین‌گیری بر روی تمام مسائل ممکن، تمام الگوریتم‌های بهینه‌سازی دارای عملکرد یکسانی هستند (همان‌طور که در ابتدای ضمیمه نیز بیان شد).
- برای هر دو الگوریتم بهینه‌سازی مانند A و B، اگر عملکرد A بر روی یک مسئله مانند f را با $V(A.f)$ نمایش دهیم، آنگاه مسئله‌ای مانند g وجود خواهد داشت که $V(A.f) = V(B.g)$. این بدین معناست که صرف نظر از اینکه عملکرد A بر روی یک مسئله چگونه است، الگوریتم دیگری مانند B وجود دارد که دارای عملکرد مشابه A بر روی یک مسئله‌ی دیگر می‌باشد. عکس این موضوع نیز صادق است.
- الگوریتمی که عملکرد بهتری از جستجوی اتفاقی دارد مانند یک ماشین حرکت دائمی می‌باشد [شافر، ۱۹۹٤]. این موضوع قاعده‌ی پایداری عملکرد کلی[1] نام دارد.
- تلاش برای طراحی الگوریتمی که بهتر از جستجوی اتفاقی باشد بیهوده است، مگر آنکه بتوان از اطلاعات خاص مسئله در الگوریتم استفاده نمود [انگلیش[2]، ۱۹۹۹]. این موضوع قانون پایداری اطلاعات[3] نام دارد.
- در غیاب اطلاعات خاص مسئله باید فرض کنیم تمام راه‌حل‌های ممکن به یک اندازه محتمل هستند [دمبسکی و مارکس، ۲۰۰۹b]. این موضوع قاعده‌ی دلیل ناکافی برنولی[4] نام دارد.

قضیه‌ی NFL باید در ادعاهای ما کمی تعادل به وجود بیاورد. برای مثال، فرض کنید یک الگوریتم جدید و یا نسخه‌ای بهبودیافته از یک الگوریتم را به وجود آورده‌ایم و می‌خواهیم نشان دهیم این الگوریتم از سایر الگوریتم‌ها بهتر بوده تا بتوانیم نتایج خود را منتشر کنیم. فرض کنید مجموعه‌ای از توابع محک را، که با F نمایش می‌دهیم، در اختیار داریم. از $\bar{F}$ برای نشان دادن مجموعه‌ی متمم F استفاده می‌نماییم. بنابراین، $\bar{F}$ مجموعه‌ی تمام توابعی است که در F نیستند. اگر الگوریتم مورد علاقه‌ی ما، که آن را با A نمایش می‌دهیم، بر روی F دارای عملکرد بهتری نسبت به الگوریتم B باشد، آنگاه قضیه‌ی NFL حکم می‌کند که الگوریتم B دارای عملکرد بهتری بر روی مجموعه‌ی $\bar{F}$ نسبت به الگوریتم A خواهد بود.

تا به اینجا در مورد نحوه‌ی اندازه‌گیری عملکرد صحبت نکرده‌ایم. راه‌های زیادی برای اندازه‌گیری عملکرد یک الگوریتم وجود دارد. برای مثال:

[1] The Law of Conservation of General Performance
[2] English
[3] The Law of Conservation of Information
[4] Bernoulli's Principal of Insufficient Reason

- می‌توان عملکرد را بر اساس بهترین ذره‌ی پیدا شده پس از تعداد مشخصی نسل و تعداد مشخصی شبیه‌سازی اندازه‌گیری نمود. می‌توان این راه را *بهترینِ بهترین* نام‌گذاری نمود.
- می‌توان عملکرد را بر اساس میانگین بهترین راه‌حل‌های به دست آمده پس از تعداد مشخصی شبیه‌سازی، که هر کدام دارای تعداد مشخصی نسل می‌باشند، اندازه‌گیری نمود. می‌توان این روش را *میانگینِ بهترین* نام‌گذاری نمود.
- می‌توان عملکرد را با محاسبه‌ی میانگین برازندگی تمام ذرات موجود در جمعیت پس از گذشت تعداد مشخصی نسل، و سپس پیدا کردن بهترین این میانگین‌ها پس از تعداد مشخصی شبیه‌سازی، محاسبه نمود. این روش را می‌توان *بهترینِ میانگین* نامید.
- می‌توان عملکرد را با محاسبه‌ی میانگین برازندگی تمام ذرات موجود در جمعیت پس از گذشت تعداد مشخصی نسل و سپس محاسبه‌ی میانگین این میانگین‌ها پس از تعداد مشخصی شبیه‌سازی، محاسبه نمود. این روش را می‌توان *میانگینِ میانگین* نامید.
- می‌توان عملکرد را بر اساس انحراف استاندارد بهترین راه‌حل‌های پیدا شده پس از تعداد مشخصی نسل و تعداد مشخصی شبیه‌سازی، اندازه‌گیری نمود.
- هر چند روش از روش‌های اندازه‌گیری بالا را می‌توان ترکیب نمود تا اندازه‌گیری‌های عملکرد ترکیبی حاصل شود.
- هر یک از روش‌های اندازه‌گیری بالا را می‌توان بر روی چند مسئله میانگین‌گیری نمود.

به نظر می‌رسد به تعداد مسائل بهینه‌سازی راه برای اندازه‌گیری عملکرد وجود داشته باشد. در حقیقت، تعداد هر دوی این‌ها بی‌نهایت است.

این موضوع یکی دیگر از ویژگی‌های مهم قضیه‌ی NFL را آشکار می‌سازد: قضیه‌ی NFL را می‌توان مستقل از نحوه‌ی اندازه‌گیری عملکرد، اعمال نمود. گاهی گفته می‌شود یک الگوریتم از الگوریتم دیگری مقاوم‌تر و پایدارتر است. واژه‌ی *مقاومت و پایداری* ممکن است به این معنا باشد که الگوریتم دارای عملکرد خوبی بر روی گستره‌ی وسیعی از مسائل بوده، و یا اینکه الگوریتم به‌صورت نسبی نسبت به نویز ارزیابی تابع برازندگی، تغییرات پارامترهای تابع برازندگی و تغییرات پارامترهای الگوریتم، غیرحساس است. قضیه‌ی NFL این اطمینان را به ما می‌دهد که تمامی الگوریتم‌ها به یک اندازه مقاوم و پایدار هستند. این قضیه همچنین به ما می‌گوید که تمام الگوریتم‌ها به یک اندازه تخصصی هستند [شوماخر و همکاران، ۲۰۰۱]. اگر الگوریتم A بر روی مجموعه‌ی تابع F مقاوم‌تر از الگوریتم B باشد، آنگاه الگوریتم B بر روی مجموعه‌ی $\bar{F}$ از الگوریتم A مقاوم‌تر است.

قضیه‌ی NFL از برخی زوایا بسیار غیر بصری و در تناقض با درک حسی جلوه می‌کند، اما اگر زاویه‌ی دید خود را عوض کنیم، می‌توان دید که این قضیه کاملاً با درک حسی در یک راستا قرار دارد. با در نظر گرفتن این دید حسی از قضیه‌ی NFL، ظاهر شدن آن در مقالات و متون، پیش از اثبات رسمی آن در اواسط دهه‌ی ۹۰، چندان جای شگفتی ندارد. برای مثال، گرگوری رالینز[1] نوشته است [رالینز، ۱۹۹۱، صفحه ۷]:

... گاهاً گفته می‌شود که GAها را می‌توان برای بهینه‌سازی هر تابعی به کار برد. چنین گفته‌هایی به‌صورت خیلی محدودی صادق هستند. انتظار می‌رود هر الگوریتمی که یکی از این ادعاها را برآورده نماید، عملکردی بهتر از جستجوی اتفاقی بر روی مجموعه‌ی تمام توابع نداشته باشد.

عکس‌العملی که محققین بهینه‌سازی نسبت به قضیه‌ی NFL نشان داده‌اند، متفاوت بوده است. برخی می‌گویند قضیه‌ی NFL به دلیل شرط "میانگین گرفته شده بر روی مجموعه تمام مسائل ممکن"، فاقد جذابیت عملی می‌باشد. در دنیای واقعی (بر خلاف دنیای ریاضی)، ما به تمام مسائل ممکن علاقه‌ای نداریم، بلکه بیشتر به مسائلی علاقه داریم که از کاربردهای عملی نشئت می‌گیرند. مسائل دنیای واقعی معمولاً اتفاقی و گمراه‌کننده نیستند و دارای ساختار می‌باشند. این بدین معناست که در مسائل مینیمم‌سازی دنیای واقعی، الگوریتم فرود از تپه بهتر از جستجوی اتفاقی یا الگوریتم تپه‌نوردی عمل می‌نماید. این واقعاً همان چیزی است که بیشتر اوقات در دنیای واقعی شاهد آن هستیم:

با میانگین‌گیری بر روی مسائل مورد علاقه می‌توان دید که همه‌ی الگوریتم‌های بهینه‌سازی دارای عملکرد یکسانی نیستند.

اما این موضوع دال بر این نیست که قضیه‌ی NFL با مهندسی عملی ارتباطی ندارد. قضیه‌ی NFL زیرساخت محکمی برای آنچه که فکر می‌کنیم به‌صورت حسی می‌دانیم، فراهم می‌آورد. این زیرساخت آن است که جهت دستیابی به یک راه‌حل خوب برای یک مسئله‌ی بهینه‌سازی باید اطلاعات خاص مسئله را در الگوریتم جستجوی خود به کار ببریم و بدین ترتیب به عملکردی بهتر از جستجوی اتفاقی دست یابیم. اگر بدانیم که راه‌حل‌های خوب یک مسئله تمایل به خوشه شدن دارند (یعنی فضای جستجو دارای نوعی نظم است)، آنگاه خواهیم دانست که بازترکیب عملکرد خوبی از خود نشان خواهد داد. اگر بدانیم که نظم فضای جستجو کمتر است، آنگاه می‌توان دریافت که جهش دارای اهمیت بیشتری است.

برای مثال، فرض کنید می‌خواهیم پارامترهای یک کنترل‌کننده‌ی PID را جهت بهینه نمودن عملکرد یک سیستم کنترلی بهینه نماییم. برای این کار می‌توانیم جستجویی را طراحی کنیم که اطلاعات گرادیانی (یا به عبارتی میزان حساسیت عملکرد به پارامترهای PID) را محاسبه کرده و سپس از این اطلاعات در فرایند

[1] Gregory Rawlins

جستجوی خود استفاده می‌نماید. این به معنی استفاده از اطلاعات خاص مسئله در فرایند جستجو می‌باشد. این الگوریتم میزان‌سازی PID از قضیه‌ی NFL پرهیز می‌نماید چرا که اطلاعات گرادیانی تقریباً برای هیچ تابعی در دسترس نیست. می‌توان گفت با فراهم آوردن اطلاعات خاص مسئله از قلمروی قضیه‌ی NFL خارج شده و به همین علت احتمالاً الگوریتم تکاملی ما بهتر از جستجوی اتفاقی عمل خواهد نمود. یک راه دیگر برای بیان این موضوع وجود دارد و آن این است که قضیه‌ی NFL فرایند بهینه‌سازی را مانند یک جستجوی کورکورانه و بدون حضور هیچ گونه اطلاعات خاص مسئله در فرایند جستجو در نظر می‌گیرد. اما الگوریتم‌های جستجوی مؤثر در دنیای واقعی کورکورانه نیستند، بدین معنی که این الگوریتم‌ها *شامل* اطلاعات خاص مسئله می‌باشند [کالبرسون[1]، ۱۹۹۸].

یک مثال دیگر، واژگونی در مسئله‌ی فروشنده‌ی دوره‌گرد (بخش ۱۸-٤-۱) می‌باشد. کاربردهای چندگانه‌ی واژگونی متضمن مسیری بدون لبه‌های متقاطع است. به همین دلیل، واژگونی بیشتر وقت خود را صرف مسیرهایی می‌کند که از میانگین بهتر هستند.

قضیه‌ی NFL همچنین به‌صورت غیرمستقیم به ما می‌گوید که چرا نحوه‌ی ارائه‌ی مسئله حائز اهمیت است (بخش ۸-۳ را ببینید). اگر فضای جستجوی یک مسئله دارای ساختاری منظم باشد، آنگاه باید مسئله را به‌گونه‌ای ارائه نمود که نظم فضای جستجو حفظ شود. پس می‌توان با یک جستجوی ساخت‌یافته به نتایج خوبی دست یافت. اگر مسئله را به‌گونه‌ای بدون ساختار ارائه نماییم، می‌توان از همان جستجوی اتفاقی استفاده نمود.

وایتلی و واتسون نتایج قضیه‌ی NFL را به‌صورت زیر بیان نموده‌اند [وایتلی و واتسون، ۲۰۰۵]:

- طراحی یک الگوریتم بهینه‌سازی شامل تعادلی میان کلیت و تأثیر الگوریتم بر مسئله‌ای خاص می‌شود. یک الگوریتم که بر روی گستره‌ی وسیعی از توابع محک عملکرد خوبی دارد، ممکن است برای یک مسئله‌ی خاص از دنیای واقعی چندان مؤثر نباشد. عکس این قضیه نیز صادق است، بدین معنی که الگوریتم‌هایی که دارای عملکرد ضعیفی بر روی محک‌های استاندارد می‌باشند ممکن است نتایج خوبی برای مسائل دنیای واقعی در پی داشته باشند. الگوریتم‌ها ساده اغلب نتایج خوبی به دست می‌دهند. الگوریتم‌های پیچیده برای دستیابی به نتایج بهتر در یک مسئله خاص طراحی می‌شوند. یکی از تصمیماتی که باید گرفت آن است که چه قدر زمان و تلاش برای میزان‌سازی الگوریتم جهت یک مسئله‌ی خاص صرف کنیم. به عبارت دیگر، دستیابی به یک نتیجه‌ی بهینه‌سازی بهتر چه قدر اهمیت دارد.

[1] Culberson

- در صورت استفاده از اطلاعات خاص مسئله در الگوریتم بهینه‌سازی، باید نتایج بهتری را نسبت به یک الگوریتم کلی‌تر به دست آوریم (البته به شرط آن که از اطلاعات خاص مسئله به درستی استفاده شود). این مثالی از یک قاعده‌ی کلی است که مهندسین و دانشمندان برای مدتی طولانی مشاهده نموده‌اند. همیشه تعادلی میان کلیت و عملکرد الگوریتم وجود دارد. ابزار و الگوریتم‌هایی که جهت گستره‌ی وسیعی از مسائل طراحی شده‌اند، در نهایت عملکرد خوبی بر روی هر مسئله نخواهند داشت.
- نحوه‌ی ارائه‌ی مسئله‌ی بهینه‌سازی می‌تواند تأثیر قابل توجهی بر روی الگوریتم بهینه‌سازی داشته باشد. هر مسئله‌ی بهینه‌سازی را می‌توان به بی‌نهایت روش ممکن ارائه نمود. انتخاب نحوه‌ی ارائه می‌تواند کار زیادی را حتی پیش از اینکه الگوریتم پیاده‌سازی شود، بطلبد. اما این کار در طولانی مدت بسیار مفید خواهد بود (بخش ۸–۳ را ببینید).
- اگر یک الگوریتم بهینه‌سازی دارای عملکرد خوبی بر روی توابع محک بود، تصور نکنید که بر روی مسائل دنیای واقعی نیز عملکرد خوبی دارد. همچنین، اگر یک الگوریتم بهینه‌سازی دارای عملکرد ضعیفی بر روی توابع محک بود، تصور نکنید که بر روی مسائل دنیای واقعی نیز عملکرد ضعیفی خواهد داشت. این نتیجه‌ی قضیه‌ی NFL باعث ایجاد تردید در مورد اهمیت بیشتر مقالات مرتبط با الگوریتم‌های تکاملی می‌شود چرا که بیشتر این مقالات تنها بر روی توابع محک تمرکز کرده و فاقد تأکید کافی بر روی مسائل دنیای واقعی می‌باشند.

تحقیقات فعلی در مورد قضیه‌ی NFL شامل تلاش برای یافتن مجموعه‌ای از توابعی می‌باشد که قضیه‌ی NFL در مورد آن‌ها صادق نیست. به یاد آورید که قضیه‌ی NFL در مورد تمام مسائل بهینه‌سازی ممکن صادق است. این قضیه در مورد تمام مجموعه‌های مسائل بهینه‌سازی صادق نیست. برای مثال، واضح است که یک الگوریتم فرود از تپه در مورد مسائلی که دارای یک مینیمم می‌باشند بهتر از جستجوی اتفاقی عمل می‌کند. یک نتیجه‌ی دیگر آن است که قضیه‌ی NFL برای توابعی که می‌توان آن‌ها را با چندجمله‌ای‌های تک متغیره و با پیچیدگی محدود بیان نمود، صادق نیست [کریستنسن[1] و اوپاکر، ۲۰۰۱]. همچنین، قضیه‌ی NFL در مورد نوعی خاص از مسائل هم–تکاملی نیز صادق نیست [ولپرت و مکردی، ۲۰۰۵].

یافتن مجموعه‌ای دیگر از توابع که قضیه‌ی NFL در موردآن‌ها صادق نیست می‌تواند نتایج مهمی را برای طراحی الگوریتم‌های بهینه‌سازی عملی به دنبال داشته باشد. این سؤال‌ها به مواردی همچون قابلیت

[1] Christensen

فشرده‌سازی توابع، طول توصیف مسائل و همچنین تفاوت میان مجموعه‌های نامتناهی و متناهی توابع، بستگی دارد [شوماخر و همکاران، ۲۰۰۱]، [لتیمور[1] و هاتر[2]، ۲۰۱۱].

ب.۲ آزمایش عملکرد

این بخش به بحث در مورد برخی موارد مرتبط با آمار و ارائه‌ی نتایج شبیه‌سازی‌های الگورتم تکاملی در رساله‌ها، مقالات، پایان‌نامه‌ها و متون فنی می‌پردازد. هر کسی که به مطالعه و یا انجام تحقیقات الگوریتم‌های تکاملی می‌پردازد باید درک خوبی از موضوعات این بخش داشته باشد. این بخش به ما نشان می‌دهد چگونه از تمایلات و گرایشات در ارائه‌ی نتایجمان بکاهیم. این بخش همچنین به ما نشان می‌دهد چگونه چنین تمایلات و گرایشاتی را در مقالات و نوشته‌های دیگران تشخیص دهیم. شاید بهتر باشد بگوییم این بخش بر این موضوع تأکید دارد که چه در هنگام خواندن نوشته‌ها و نتایج دیگران و چه در هنگام خواندن نتایج و نوشته‌های خودمان، باید نوعی شک سالم به خرج دهیم.

بخش ب.۱-۲ مروری از برخی مسائلی که به‌طور معمول در مقالات تحقیقاتی الگوریتم‌های تکاملی مشاهده می‌شوند را ارائه می‌دهد. بخش ب.۲-۲ نشان می‌دهد چگونه افراد از نحوه‌ی ارائه‌ی نتایج شبیه‌سازی برای هر گونه نتیجه‌گیری که می‌خواهند استفاده می‌نمایند. به عبارت دیگر این بخش نشان می‌دهد چگونه میتوان نتایج را به‌گونه‌ای گمراهکننده ارائه داد (البته امیدواریم این اتفاق بدون غرض و سهوا باشد). این بخش همچنین نشان می‌دهد چگونه می‌توان نتایج شبیه‌سازی را به‌گونه‌ای واضح و صادقانه ارائه داد. بخش ب.۳-۲ به برخی نکات مهم در مورد تولیدکننده‌های اعداد اتفاقی که در شبیه‌سازی‌ها از آن‌ها استفاده می‌نماییم، اشاره خواهد نمود. بخش ب.٤-۲ به مرور آزمون T[3]، که میزان قابل توجه بودن تفاوت میان دو مجموعه نتایج شبیه‌سازی را مشخص می‌کند، می‌پردازد. بخش ب.۵-۲ نیز به مرور آزمون F[4]، که میزان قابل توجه بودن تفاوت میان بیش از دو مجموعه نتایج شبیه‌سازی را مشخص می‌کند، خواهد پرداخت.

ب.۱-۲ اغراق بر اساس نتایج شبیه‌سازی

جمله‌ی مارک تواین که در اول این فصل آورده شده است، کتاب *چگونه با آمار زندگی کنیم* [هاف[5] و گیس[6]، ۱۹۹۳] و بخش حاضر، همگی سخن مشابهی در مورد تحقیقات الگوریتم تکاملی می‌گویند. هنگامی

[1] Lattimore
[2] Hutter
[3] T-test
[4] F-test
[5] Huff
[6] Geis

که نتایج الگوریتم تکاملی را در یک مقاله و یا یک کتاب مشاهده می‌کنید مطمئن باشید که نوعی گرایش و تمایل در آن نتایج وجود دارد. این گرایش ممکن است عمدی و یا سهوی باشد، ممکن است مستقیم و یا غیرمستقیم باشد، ممکن است پررنگ و یا کمرنگ باشد، ممکن است ناچیز و یا قابل توجه باشد، اما همیشه نوعی گرایش و تمایل وجود دارد. هنگامی که یک مقاله را می‌خوانیم و بر اساس خروجی آن نتیجه‌گیری می‌نماییم، باید به خاطر داشته باشیم که اگر نویسنده مجموعه‌ی دیگری از نتایج را ارائه می‌داد و یا حتی اگر همان نتایج را به نحو دیگری ارائه می‌داد، نتیجه‌گیری‌های ما نیز کاملاً متفاوت از آب در می‌آمد.

با توجه به قضیه‌ی NFL، که بیان می‌دارد در صورت میانگین‌گیری بر روی تمامی مسائل بهینه‌سازی ممکن عملکرد تمامی الگوریتم‌های بهینه‌سازی یکسان می‌باشد، آزمایش نمودن الگوریتم‌ها بر روی توابع محک بیهوده به نظر می‌رسد. بنا بر گفته‌ی وایتلی "از دید فنی، ارزیابی مقایسه‌ای الگوریتم‌های جستجو امری خطیر است" [واتلی و واتسون، ۲۰۰۵، صفحه۳۳۳]. اگر ما یک مقاله در مورد الگوریتم تکاملی A بنویسیم و در آن نشان دهیم که این الگوریتم بر روی مجموعه‌ای از توابع محک مانند F دارای عملکرد بهتری نسبت به الگوریتم B می‌باشد، آنگاه قضیه‌ی NFL بیان می‌دارد که الگوریتم B بر روی مجموعه توابع $\bar{F}$ دارای عملکرد بهتری نسبت به A می‌باشد.

با این حال، این موضوع نباید بیش از اندازه ما را دلسرد کند. اگر نکته‌ی اصلی مقاله ما در نشان دادن بهتر بودن A نسبت به B بر روی مجموعه‌ی F باشد، آنگاه مقاله‌ی ما موفقیت‌آمیز خواهد بود. به یاد داشته باشید که در قضیه‌ی NFL گفته نشده تمامی الگوریتم‌ها بر روی تمام مجموعه‌های ممکن F دارای عملکرد یکسانی می‌باشند، بلکه در این قضیه بیان شده است که با میانگین گرفتن بر روی تمامی مسائل ممکن، تمامی الگوریتم‌ها دارای عملکرد یکسان خواهند بود. این در واقع بدین معنی است که واقعاً ممکن است الگوریتم A برای مسائلی خاص و یا نوع خاصی از مسائل عملکرد بهتری داشته باشد.

این موضوع این نکته اشاره دارد که اگرچه محک زدن الگوریتم‌های تکاملی کاری ارزنده است، اما باید قضیه‌ی NFL را در ذهن داشته باشیم تا بتوانیم نتیجه‌گیری خود را تعدیل نماییم. یک مقاله معمول در زمینه‌ی الگوریتم‌های تکاملی نشان می‌دهد که الگوریتم A بر روی مجموعه‌ای از توابع محک مانند F دارای عملکرد بهتری نسبت به الگوریتم B است و سپس نتیجه‌ای مانند نتیجه‌ی زیر را ارائه می‌دهد:

بنابراین می‌توان دید که A برای بهینه‌سازی تابع از B بهتر است.

ادعاهایی مانند این از دیدگاه قضیه‌ی NFL اساساً نادرست هستند. اما مقالات معمولی که در زمینه‌ی الگوریتم‌های تکاملی وجود دارند شامل عبارت‌هایی این چنینی در چکیده، مقدمه و نتیجه‌گیری خود می‌باشند. یک ادعای خاضعانه‌تر می‌تواند این چنین باشد:

بنابراین می‌توان دید که A برای بهینه‌سازی توابعی که دارای خصیصه‌های بحث شده در این مقاله می‌باشند، بهتر از B عمل میکند.

با این حال، مقالاتی که شامل آزمایش عملکرد بر روی توابع محک می‌شوند، هیچ تلاشی در جهت بررسی خواص و ویژگی‌های توابع محک ارائه نمی‌دهند. این محک‌ها چه ویژگی دارند که باعث می‌شود الگوریتم A عملکرد بهتری از B داشته باشد؟ آیا علت این موضوع این است که توابع موجود در F مشتق‌پذیرند یا چند پیمانه‌ای هستند یا دارای مشتق دوم پیوسته‌اند و یا اینکه دارای محدودیت‌اند؟ پاسخ به چنین سؤالاتی معمولاً دشوار است و به همین علت نادیده گرفته می‌شوند. به همین دلیل، یک ادعای بهتر (خاضعانه‌تر) که می‌توان در یک مقاله‌ی مرتبط با الگوریتم تکاملی مطرح نمود به‌صورت زیر است:

بنابراین می‌توان دید که A برای بهینه‌سازی توابعی که در این مقاله مورد بحث قرار گرفته‌اند عملکرد بهتری نسبت به B دارد.

این ادعای بهتری است چرا که در آن سعی بر تعمیم نتایج به فرای نتایج تجربی نشده است. به عبارت دیگر، ادعای اضافی صورت نگرفته است. با این حال، حتی این ادعا هم کمی زیادی به نظر می‌رسد چرا که می‌توان الگوریتم‌ها را به طرق بسیاری میزان و پیاده‌سازی نمود.

یکی دیگر از مشکلات مرتبط با ارائه‌ی نتایج محک آن است که ممکن است توابع محک در کلاس مسائل جالب دنیای واقعی نباشد. فرض بر آن است که بیشتر محققین الگوریتم‌های تکاملی به کاربرد نهایی تحقیقشان در مسائل دنیای واقعی علاقه‌مند هستند. اگر یک مقاله نشان دهد که الگوریتم A دارای عملکرد خوبی بر روی مجموعه‌ای از محک‌ها مانند F می‌باشد، این موضوع چه ارتباطی با دنیای واقعی خواهد داشت؟ در حقیقت، با توجه به قضیه‌ی NFL، ممکن است بتوان نتیجه گرفت که الگوریتم A دارای عملکرد ضعیفی بر روی مسائل دنیای واقعی می‌باشد. بالاخره، اگر الگوریتم A دارای عملکردی بهتری بر روی مجموعه‌ی F، که تنها از توابع محک تشکیل شده است، باشد، آنگاه الگوریتم B دارای عملکرد بهتری بر روی مجموعه‌ی $\bar{F}$، که شامل مسائل دنیای واقعی می‌باشد، خواهد بود. این مسابقه‌ی همیشگی برای دستیابی عملکرد بهتر بر روی مسائل محک باعث شده است به مجموعه‌ای از الگوریتم‌ها دست یابیم که عملکردشان بر روی توابع محک بسیار خوب است، اما به لحاظ مهندسی بلااستفاده هستند.

از لحاظ مباحثی که در این بخش ارائه شد، مقالات الگوریتم‌های تکاملی باید بیشتر بر کاربردها تأکید داشته باشند تا توابع و مسائل محک. گاهاً ممکن است در صورت عدم درج نتایج از توابع محک متداول، قبول شدن مقاله با مشکل مواجه شود، اما این اتکای بیش از حد به توابع محک باعث ایجاد اعتماد به نفس کاذب می‌شود. امروزه به سختی می‌توان کاربردی حقیقی از الگوریتم‌های تکاملی را در ژورنال‌ها مشاهده

نمود. اما با دیدی که از قضیه‌ی NFL به دست آوردیم، میتوانیم به جرئت بگوییم موضوع اصلی مقالات باید کاربردهای الگوریتم تکاملی در مسائل دنیای واقعی باشد. تنها با آزمایش عملکرد الگوریتم تکاملی بر روی مسائل دنیای واقعی می‌توان با اعتماد به نفس گفت که الگوریتم‌های تکاملی مفید هستند.

ب.۲-۲ چگونه نتایج شبیه‌سازی را گزارش کنیم و چگونه گزارش نکنیم

این بخش نشان می‌دهد که معمولاً چگونه نویسندگان نتایج شبیه‌سازی را به‌گونه‌ای گمراه‌کننده ارائه داده تا پیام مورد نظر وی را بازتاب دهند. این بخش همچنین نشان می‌دهد که چگونه می‌توان با استفاده از آمار نتایج را به‌صورتی شفاف و صادقانه ارائه نمود. بحث بیشتر و پرمایه‌تر در مورد آمار را به بخش ب.٤-۲ موکول می‌نماییم.

فرض کنید می‌خواهیم عملکرد دو الگوریتم تکاملی A و B را بر روی یک محک مقایسه نماییم. فرض کنید که این دو الگوریتم را بر روی یک محک اجرا می‌نماییم. هر الگوریتم برای T نسل اجرا می‌شوند. در هر نسل هزینه‌ی بهترین ذره‌ی جمعیت را اندازه گرفته و آن را با f_{min} نشان می‌دهیم. با این کار مجموعه‌ای از داده‌ها را خواهیم داشت که به شکل زیر خواهد بود:

$$\begin{aligned} &\text{الگوریتم } A: f_{A,min} = \{f_{A0}, f_{A1}, f_{A2}, \dots, f_{AT}\} \\ &\text{الگوریتم } B: f_{B,min} = \{f_{B0}, f_{B1}, f_{B2}, \dots, f_{BT}\} \end{aligned} \quad (\text{ب.۱})$$

در معادله‌ی بالا f_{A0} و f_{B0} به ترتیب هزینه‌ی بهترین ذره پس از مقداردهی اولیه برای الگوریتم‌های A و B بوده و f_{Ai} و f_{Bi} نیز هزینه‌ی بهترین ذره پس از نسل iاُم می‌باشند. عملکرد الگوریتم A را می‌توان با هزینه‌ی بهترین ذره‌ی پیدا شده توسط این الگوریتم در طول T نسل اندازه گرفت:

$$\text{متریک الگوریتم } A: \min_{i\in[0,T]} f_{Ai} \quad (\text{ب.۲})$$

با فرض این که از نخبه‌گرایی استفاده می‌نماییم، f_{Ai} به‌صورت یکنواختی غیر افزایشی خواهد بود، بنابراین

$$\text{متریک الگوریتم } A: \min_{i\in[0,T]} f_{Ai} = f_{AT} \quad (\text{ب.۳})$$

اگر الگوریتم‌های A و B را بر روی یک محک اجرا کرده و بخواهیم ببینیم کدام عملکرد بهتری دارد، می‌توان به سادگی f_{AT} را با f_{BT} مقایسه نمود. می‌توانیم نتیجه بگیریم که

$$\text{منطق غلط} \begin{cases} f_{AT} < f_{BT} \text{ اگر } A \text{ از } B \text{ بهتر است} \\ f_{BT} < f_{AT} \text{ اگر } B \text{ از } A \text{ بهتر است} \end{cases} \quad (ب.٤)$$

واضح است که چنین استدلالی معتبر نیست. ما نباید هیچ گاه از چنین منطق معیوبی استفاده نماییم. یکی از نکاتی که اهمیت بسیار دارد آن است که الگوریتم‌های تکاملی اتفاقی هستند، بدین معنی که به یک تولیدکننده‌ی اعداد اتفاقی بستگی دارند. بنابراین و به‌صورت کلی، الگوریتم تکاملی در هر بار اجرا نتایج متفاوتی تولید خواهد نمود. اگر آزمایش بالا را تنها یکبار انجام دهیم ممکن است نتیجه بگیریم که الگوریتم A بهتر از الگوریتم B است. اما اگر آزمایش را دوباره تکرار نماییم، ممکن است نتیجه بگیریم الگوریتم B بهتر از الگوریتم A است. به همین دلیل است که اجرای چندباره‌ی شبیه‌سازی‌ها و استفاده از نتایج تمام شبیه‌سازی‌ها در هنگام مقایسه‌ی الگوریتم‌های تکاملی مهم است. استفاده از چندین شبیه‌سازی، به‌طوری که هر یک به‌صورت اتفاقی آغاز شود، شبیه‌سازی مونت کارلو، آزمایش شبیه‌سازی و یا روش مونت کارلو نام دارد [رابرت و کاسلا[1]، ۲۰۱۰].

حال فرض کنید مفهوم شبیه‌سازی مونت کارلو را درک کرده‌ایم. سپس، الگوریتم‌های A و B را بر روی یک مسئله و به تعداد M بار اجرا می‌نماییم. M تعداد شبیه‌سازی‌های مونت کارلو است. هر بار که الگوریتم‌ها اجرا می‌شوند نتایج متفاوتی برای f_{AT} و f_{BT} به دست می‌آید. از f_{ATk} و f_{BTk} برای نشان دادن هزینه‌ی الگوریتم‌های A و B در انتهای نسل Tاُم از شبیه‌سازی مونت کارلوی kاُم استفاده می‌نماییم. می‌توان اطلاعات شبیه‌سازی را به‌صورت زیر نوشت:

$$\begin{aligned} &\text{نتایج الگوریتم } A: \{f_{AT1}, f_{AT2}, \dots, f_{ATM}\} \\ &\text{نتایج الگوریتم } B: \{f_{BT1}, f_{BT2}, \dots, f_{BTM}\} \end{aligned} \quad (ب.٥)$$

می‌توان متوسط عملکرد دو الگوریتم را به‌صورت زیر مقایسه نمود:

$$\begin{aligned} \bar{f}_A &= \frac{1}{M}\sum_{k=1}^{M} f_{ATK} \\ \bar{f}_B &= \frac{1}{M}\sum_{k=1}^{M} f_{BTK} \end{aligned} \quad (ب.٦)$$

[1] Casella

همچنین، می‌توان واریانس عملکرد دو الگوریتم را به‌صورت زیر مقایسه نمود[1]:

$$\sigma_A^2 = \frac{1}{M-1}\sum_{k=1}^{M}\left(f_{ATK} - \bar{f}_A\right)^2$$
$$\sigma_B^2 = \frac{1}{M-1}\sum_{k=1}^{M}\left(f_{BTK} - \bar{f}_B\right)^2 \qquad (ب.۷)$$

عملکرد بهترین حالت دو الگوریتم نیز به‌صورت زیر مقایسه می‌شود:

$$f_{A,best} = \min_{k\in[1,M]} f_{ATK}$$
$$f_{B,best} = \min_{k\in[1,M]} f_{BTK} \qquad (ب.۸)$$

در نهایت، عملکرد بدترین حالت دو الگوریتم را نیز می‌توان به‌صورت زیر مقایسه نمود:

$$\bar{f}_{A,worst} = \max_{k\in[1,M]} f_{ATK}$$
$$\bar{f}_{B,worst} = \max_{k\in[1,M]} f_{BTK} \qquad (ب.۹)$$

هر یک از این متریک‌ها نوع مختلفی از عملکرد را اندازه‌گیری می‌نمایند. مقادیر هزینه‌ی متوسطی که از معادله‌ی (ب.٦) به دست می‌آیند، مطلوبیت متوسط عملکرد الگوریتم‌ها را نشان می‌دهند. واریانس‌های به دست آمده از معادله (ب.۷)، میزان تداوم عملکرد الگوریتم‌ها را تعیین می‌نمایند. مقدار هزینه بهترین حالت که از معادله‌ی (ب.۸) به دست می‌آید حاکی از آن است که در صورت اجرای چندباره‌ی الگوریتم‌ها، انتظار ارائه بهترین عملکرد از کدام الگوریتم منطقی‌تر خواهد بود.

هر یک از این متریک‌ها نوع متفاوتی از عملکرد را اندازه‌گیری می‌نمایند. مقادیر هزینه‌ی متوسط از معادله‌ی (ب.٦) به ما می‌گویند که الگوریتم‌ها به‌طور متوسط چه قدر خوب عمل می‌کنند. واریانس‌هایی که از معادله‌ی (ب.۷) به دست می‌آیند میزان تداوم عملکرد الگوریتم‌ها را اندازه می‌گیرند. مقادیر هزینه‌ی بهترین حالت از معادله‌ی (ب.۸) به ما می‌گویند در صورت اجرای چند باره‌ی الگوریتم‌ها از کدام یک انتظار عملکرد بهتری داشته باشیم. هزینه‌ی بدترین حالت که از معادله‌ی (ب.۸) به دست می‌آید نیز به ما می‌گوید در صورتی که الگوریتم‌ها را تنها یکبار اجرا نماییم و همچنین در صورتی که الگوریتم‌ها به‌گونه‌ای مقداردهی اولیه شوند که عملکردهای بد از الگوریتم‌ها حاصل شود، از کدام الگوریتم انتظار عملکرد بهتری داشته باشیم. در صورت

[1] به‌طور معمول انتظار می‌رود مخرج کسر معادله‌ی (ب.۷) به جای $(M-1)$، برابر M باشد. اما استفاده از $(M-1)$ در مخرج تخمین بهتری از واریانس به دست می‌دهد [سایمون، ۲۰۰٦، مسئله‌ی۳,٦].

تمایل به آگاهی از مباحث بیشتر در مورد متریک‌های عملکردی الگوریتم‌های تکاملی می‌توانید به [ایبن و اسمیت، ۲۰۱۱۱] مراجعه نمایید.

فرض کنید دو الگوریتم A و B را بر روی یک مسئله‌ی محک اجرا نماییم. همچنین فرض کنید که الگوریتم‌ها را M بار اجرا کرده و نتایج زیر را به دست آورده‌ایم:

$$\begin{matrix} \bar{f}_A = 14, & \sigma_A^2 = 4, & \bar{f}_{A,best} = 7, & \bar{f}_{A,worst} = 32 \\ \bar{f}_B = 16, & \sigma_B^2 = 3, & \bar{f}_{B,best} = 6, & \bar{f}_{B,worst} = 25 \end{matrix} \qquad (ب.۱۰)$$

اگر مقاله‌ای در مورد این نتایج بنویسیم، هر یک از عبارت‌های زیر ممکن است در این مقاله ظاهر شود.

۱. هزینه‌ی میانگین به دست آمده توسط الگوریتم A برابر ١٤ بوده در حالی که این هزینه برای الگوریتم B برابر ١٦ می‌باشد. بنابراین می‌توان گفت که الگوریتم A بهتر عمل می‌کند.

۲. الگوریتم A دارای واریانس ٤ بوده در حالی که الگوریتم B دارای واریانس ۳ می‌باشد. این نشان می‌دهد که الگوریتم B دارای تداوم و پایداری بیشتری می‌باشد.

۳. بهترین هزینه‌ی به دست آمده توسط الگوریتم A برابر ۷ بوده در حالی که این مقدار برای الگوریتم B برابر ٦ می‌باشد. بنابراین الگوریتم B عملکرد بهتری دارد.

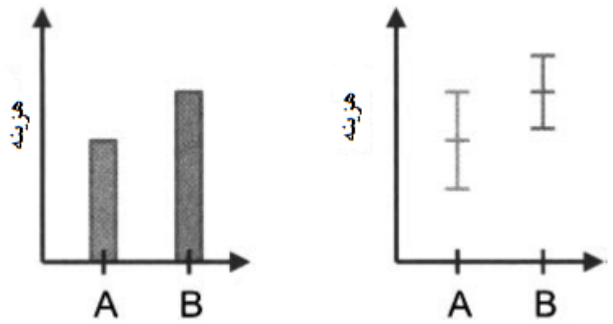
٤. هزینه‌های به دست آمده توسط الگوریتم A هیچگاه بدتر از ۲۳ نبوده است در حالی که هزینه‌های به دست آمده توسط الگوریتم B هیچگاه از ۲۵ بدتر نبوده است. این موضوع نشان می‌دهد که الگوریتم A دارای عملکرد بهتری است.

هیچ یک از عبارات بالا کاملاً غلط نیستند، ولی نشان می‌دهد که چگونه می‌توان آمار را بر حسب ایده‌های از پیش تعیین شده، تعبیر نمود. این بدین معنی است که ما هدفمند نیستیم و بیشتر تمایل به ارائه‌ی نتایجی داریم که با گرایش شخصی ما همخوانی بیشتری دارد.

اولین عبارت از چهار عبارت بالا در نظر بگیرید. این که الگوریتم A به‌طور متوسط بهتر از الگوریتم B عمل می‌کند کاملاً درست است، اما چرا باید به جای عبارت دوم یا سوم، از این عبارت در مقاله‌ی خود استفاده نماییم. به علاوه، این بهتر بودن که در عبارت ۱ از آن صحبت شده است چه قدر قابل ملاحظه است؟ الگوریتم A تنها به اندازه‌ی دو واحد از الگوریتم B بهتر است و این مقدار از انحراف استاندارد هر دو الگوریتم کمتر است. شکل ب.۳ نموداری از میانگین و انحراف استاندارد هر دو الگوریتم نشان می‌دهد. نمودار سمت چپ از این شکل مبین آن است که الگوریتم A به طرز قابل توجهی بهتر از الگوریتم B بوده در حالی که نمودار سمت راست از این شکل نشان می‌دهد که این بهتر بودن، آن اندازه که فکر می‌کنیم قابل

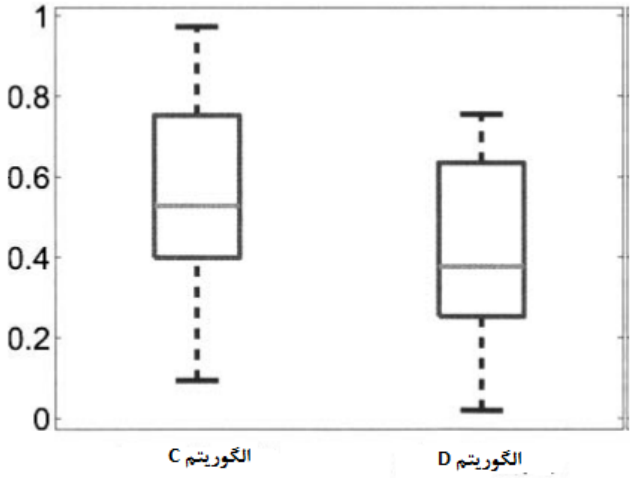
ملاحظه نیست. در هر شبیه‌سازی که دو الگوریتم تکاملی تنها به اندازه‌ی یک انحراف استاندارد متفاوت از میانگینشان عمل کنند، الگوریتم B بهتر از الگوریتم A عمل خواهد نمود.

یک راه مناسب برای نشان دادن نتایج عملکرد الگوریتم‌های تکاملی استفاده از نمودارهای جعبه‌ای است. نمودارهای جعبه‌ای شامل ۵ تکه داده برای هر الگوریتم تکاملی می‌باشند: کمترین نتیجه، چارک پایین، میانه، چارک بالا و بیشترین نتیجه از M شبیه‌سازی مونت کارلو [مک‌گیل[1] و همکاران، ۱۹۷۸]. چارک پایین مقداری است که از ۲۵٪ تمام مقادیر بزرگتر بوده، میانه مقداری است که از ۵۰٪ تمام مقادیر بزرگتر بوده، و چارک بالا مقداری است که از ۷۵٪ تمام مقادیر بزرگتر است. MATLAB Statistics Toolbox دارای تابعی با نام boxplot است که می‌توان با استفاده از آن به رسم نمودارهای جعبه‌ای پرداخت. شکل ب.٤ مثالی از یک نمودار جعبه‌ای را نشان می‌دهد. نمودارهای جعبه‌ای مقدار زیادی از اطلاعات مرتبط را در یک نمودار به‌صورت یکجا نشان می‌دهند. این موضوع نمودارهای جعبه‌ای را به روشی مختصر و مفید برای ارائه نتایج و مقایسه‌ی الگوریتم‌ها بدل ساخته است.



شکل ب.۳ مقادیر متوسط (نمودار سمت چپ)، و مقادیر متوسط به همراه انحراف‌های استاندارد (نمودار سمت راست) از عملکرد دو الگوریتم فرضی. نمودار سمت چپ الگوریتم A را بسیار بهتر از نمودار سمت راست نشان می‌دهد.

[1] McGill



شکل ب.٤ یک نمودار جعبه‌ای نوعی که عملکرد دو الگوریتم فرضی را نشان می‌دهد. هر جعبه مجموعه‌ی نتایج ۵۰٪ میانی از الگوریتم متناظر خود را نشان داده و خط وسط هر جعبه نیز میانه را نشان می‌دهد. خطوط بالا و پایین هر نمودار (که با خط‌چین به جعبه‌ها وصل شده‌اند) مقادیر ماکزیمم و مینیمم هستند.

با این حال، راه‌های بدون اشکال برای پرهیز از گرایش در ارائه‌ی نتایج شبیه‌سازی بسیار کم هستند. برای مثال، فرض کنید تعدادی شبیه‌سازی از الگوریتم‌های A و B را اجرا کرده و پنج چارک را به ازای هر یک از شبیه‌سازی‌ها به دست می‌آوریم. فرض کنید این کار را برای شش تابع محک انجام می‌دهیم. نتایجی که به دست می‌آیند احتمالاً به شکل نتایج ارائه شده در جدول ب.۱ خواهد بود. می‌بینیم که الگوریتم A (که بهترین الگوریتم شاخته شده در نظر گرفته می‌شود) بر روی محک‌های ۱، ۲ و ۳ عملکرد بهتری داشته و الگوریتم B (که به تازگی معرفی شده است) نیز بر روی محک‌های ٤، ۵ و ٦ عملکرد بهتری از خود ارائه می‌دهد. در چنین شرایطی چه باید کرد؟ ماه‌های زیادی را خرج ایجاد، تصحیح، اشکال‌زدایی و آزمودن الگوریتم جدید خود کرده‌ایم. به علاوه، برای دستیابی به مدرک خود نیاز به انتشار نتایج خود داریم. در چنین شرایطی بسیاری از محققین وسوسه می‌شوند مقاله‌ای بنویسند که به محک‌های ٤، ۵ و ٦ پرداخته و محک‌های ۱، ۲ و ۳ را نادیده می‌گیرد.

جدول ب.۱ نتایج شبیه‌سازی نمونه بر روی شش تابع محک برای دو الگوریتم. اعداد فرضی ارائه شده در این جدول نشانگر مقادیر متوسط برای مجموعه‌ای از شبیه‌سازی‌های مونت کارلو می‌باشند. الگوریتم *A* بهترین الگوریتم شناخته شده بوده و الگوریتم *B*، الگوریتم جدید ارائه شده توسط محقق می‌باشد. با توجه به این نتایج، محقق تصمیم به انتشار نتایج مربوط به توابع محک ٤، ٥ و ٦ گرفته و نتایج مربوط به محک‌های ۱، ۲ و ۳ را حذف خواهد کرد.

	F_1	F_2	F_3	F_4	F_5	F_6
الگوریتم A	۱۳	۲۱	۱٦	٤٥	۲٤	۳۳
الگوریتم B	۱٦	۳۰	۲۲	۳٦	۱۷	۲۸

این کار غیراخلاقی است. از میان تمام تحقیقات و انتشاراتی که این روزها صورت می‌گیرد، چنین کاری بدترین کار ممکن نیست، اما می‌توانیم به اتفاق نظر بگوییم که این کار متقلبانه است. با این حال، بحث اخلاق همواره سیاه یا سفید نیست. چنین شرایطی بیشتر به شکل‌هایی تلفیق شده روی می‌دهد تا به فرم ساده‌ی جدول ب.۱. برای نمونه، اگر نتایج اولیه‌ای به دست آوریم که به خوبی گویای تحقیق ما نیست، می‌توان به سادگی دریافت که چنین نتایجی به دلیلی مشکوک هستند[1] و بنابراین باید با استفاده از محک‌هایی که به خوبی گویای تحقیق ما هستند استفاده کرده و نتایج کامل‌تر و گسترده‌تری را به دست آورد.

استانداردهای داوری همتا[2] این نوع نادرستی نامحسوس را تقویت می‌کند. اگر یک الگوریتم که به تازگی ارائه شده است و یا نوع جدید از یک الگوریتم حاضر، نتایج بهتری از بهترین الگوریم موجود ارائه ندهد، مقاله منتشر نشده و رد خواهد شد. این موضوع، با اینکه بهترین الگوریتم موجود طی قرن‌ها تکامل پیدا کرده و بالغ شده و الگوریتمی که تازه ارائه می‌شود هنوز دوران طفولیت خود را شروع نکرده و زمانی برای بالغ شدن نداشته است، همچنان صادق است. پروپوزال‌ها و مقالات مرتبط با GAها که در دهه‌ی ۱۹٦۰ نوشته می‌شدند، اکثرا مورد قبول واقع نمی‌شدند چرا که GAها در آن زمان هنوز به خوبی رویکردهای بهینه‌سازی آن زمان نبودند. یک نقد معمول در آن زمان این جمله بود که: "هر آنچه که در مورد جستجوی الگوریتم ژنتیک ارائه نموده‌اید را می‌توان به نحوی واضح‌تر و با استفاده از جستجوی درختی بیان نمود" [فوگل، ۱۹۹۹، صفحه xi]. به نظر بنای تحقیق بر روی ایده‌های نو باز است، اما مانند هر بنای دیگری، تمایل بر تأکید بر موضوعات از پیش داشته و دلسرد نمودن کسانی است که خارج از جعبه و به نوعی نامتعارف فکر می‌کنند.

[1] شاید از فرمول‌بندی ریاضی محک مطمئن نبوده، شاید محک زمان زیادی از CPU را اشغال کرده و به محک‌های سریعتری نیاز داریم. شاید تابع محک به اندازه‌ی دیگر توابع محک مشهور نیست. شاید محک دارای ویژگی‌های است که به عقیده‌ی ما آن را غیرمعمول می‌سازند. اگر به خوبی جستجو کنیم، می‌توانیم دلایل زیادی را برای مستثنی کردن هر نوع محکی پیدا کنیم.

[2] Peer Review

همتاداوران به جای تأکید بر روی نتایج تدریجی و پیوسته بر روی محک‌ها باید بر خلاقیت و نوآوری تمرکز کنند چرا که نمی‌توان پیش‌بینی کرد کدامیک از الگوریتم‌ها یا فناوری‌های جدید بر آینده تأثیر خواهند داشت. همچنین، از دید قضیه‌ی NFL، همتاداوران باید به نویسندگان اصرار کنند محدودیت‌ها و اشکالات الگوریتم‌های خود را به بحث بگذارند. به بیان دیگر، نباید به نویسندگان فشار آورد تا نتایج منفی را مخفی کنند بلکه باید آن‌ها را تشویق نمود تا نتایج منفی را به اطلاع عموم رسانده تا شفافیت حاصل شود.

نرخ موفقیت

یکی از متریک‌های عملکردی معمول در مورد الگوریتم‌های تکاملی، نرخ موفقیت است [سوگانتان و همکاران، ۲۰۰۵]، [لیانگ و همکاران، ۲۰۰۶]، [مالیپدی و سوگانتان، ۲۰۱۰]. فرض کنید تابع هزینه‌ی محکی مانند $f(x)$ در اختیار داریم که مینیمم آن در f^* قرار دارد. فرض کنید M شبیه‌سازی از الگوریتم تکاملی خود را برای F_{max} ارزیابی تابع برازندگی انجا می‌دهیم. در این صورت گوییم یک شبیه‌سازی موفق بوده است اگر در آن ذره‌ی مانند x پیدا شود که $f(x) < f^* + \epsilon$. در این صورت ϵ یک عدد مثبت بوده و آستانه‌ی موفقیت نامیده می‌شود. نرخ موفقیت S نیز درصد تعداد شبیه‌سازی‌های موفق می‌باشد.

این متریک چنان محبوب شده است که بسیاری از داوران و ویراستاران هیچ مقاله‌ی را بدون آن منتشر نکرده و یا به آن اجازه‌ی انتشار نمی‌دهند. با این حال چند مشکل در مورد این متریک عملکرد وجود دارد. اول آنکه انتخاب ϵ کاملاً دلبخواهی است و این موضوع می‌تواند تأثیر زیادی بر روی شایستگی ظاهری دو الگوریتم رقیب داشته باشد. اگر $\epsilon = \epsilon_1$ باشد آنگاه ممکن است الگوریتم A از الگوریتم B بهتر عمل کرده و اگر $\epsilon = \epsilon_2 \neq \epsilon_1$ ممکن است الگوریتم B عملکرد بهتری از الگوریتم A داشته باشد. دوم آنکه، تعداد ارزیابی‌ها تابع F_{max} نیز دلبخواهی است و این موضوع نیز می‌تواند بر شایستگی ظاهری دو الگوریتم تأثیر داشته باشد. . اگر $F_{max} = F_1$ باشد آنگاه ممکن است الگوریتم A از الگوریتم B بهتر عمل کرده و اگر $F_{max} = F_2 \neq F_1$ ممکن است الگوریتم B عملکرد بهتری از الگوریتم A داشته باشد. سوم آنکه، تعداد شبیه‌سازی‌ها M باید به طرز غیرواقع‌گرایانه‌ای بزرگ باشد تا بتوان تخمین مطمئنی از نرخ موفقیت S به دست آورد، اما چنین مقادیر بزرگی از M ممکن است به روزها و یا هفته‌ها جمع‌آوری داده نیاز داشته باشد [کلرک، ۲۰۱۲b]. هرگاه در مقاله‌ای نرخ موفقیت گزارش می‌شود، انحراف استاندارد آن از قلم می‌افتد و این موضوع می‌تواند مقید بودن آن به‌عنوان یک متریک را نفی کند.

اندازه‌ی جمعیت

برای یک مسئله‌ی خاص، الگوریتم تکاملی اول با اندازه‌ی جمعیت N_1 و الگوریتم تکاملی دوم با اندازه‌ی جمعیت N_2 بهترین عملکرد خود را نشان خواهند داد. بنابراین، هنگام مقایسه‌ی دو الگوریتم تکاملی، ابتدا باید هر الگوریتم را جداگانه میزان‌سازی کرده تا مقادیر مناسب اندازه‌ی جمعیت برای هر یک به دست آید. از سوی دیگر، ممکن است هدف از مقایسه‌ی دو الگوریتم پیدا کردن الگوریتم بهتر با یک اندازه‌ی جمعیت خاص باشد. در چنین حالتی باید از اندازه‌ی جمعیت یکسان برای هر دو الگوریتم استفاده نمود. برای نمونه، عملکرد الگوریتم تکاملی با اندازه‌ی جمعیت کوچک، ممکن است موضوع تحقیقی مهمی باشد. بنابراین این سؤال که هنگام مقایسه الگوریتم‌های تکاملی از چه اندازه‌ی جمعیتی باید استفاده نمود چندان روشن نیست، اما به هدف مقایسه بستگی دارد.

ب.۳-۲ اعداد اتفاقی

برای دستیابی به نتایج شبیه‌سازی معتبر، ضروری است که تولیدکننده‌ی اعداد اتفاقی را به خوبی درک کنیم. تولیدکننده‌ی اعداد اتفاقی می‌توانند تأثیر قابل توجهی بر روی عملکرد الگوریتم تکاملی داشته باشند. کیفیت یک راه‌حل الزاماً با کیفیت تولیدکننده‌ی اعداد اتفاقی همبستگی ندارد [کلرک، ۲۰۱۲b]. برای برخی مسائل و برخی الگوریتم‌های تکاملی، استفاده از تولیدکننده‌ی اعداد اتفاقی با کیفیت باعث بهبود عملکرد الگوریتم تکاملی شده، در حالی که برای سایر مسائل و همچنین دیگر الگوریتم‌های تکاملی، با استفاده از تولیدکننده‌ی اعداد اتفاقی بدتر می‌توان به عملکرد بهتر در الگوریتم تکاملی دست یافت.

بسیاری از اعداد اتفاقی واقعاً اتفاقی نیستند. در حقیقت، تعریف و وجود تصادفی بودن موضوع بحث‌های بسیاری است. تولیدکننده‌ی اعداد اتفاقی موجود در کامپیوترها اعداد کاملاً اتفاقی تولید نمی‌نمایند. این تولیدکننده‌ها، اعداد *شبه اتفاقی* تولید می‌نماید. اعداد شبه اتفاقی به نظر تصادفی می‌آیند ولی کاملاً تصادفی نیستند چرا که توسط الگوریتم‌های قاطع به وجود می‌آیند.

باقی این بخش جهت به تصویر کشیدن این موضوع بر MATLAB تمرکز می‌نماید، اما این مباحث را می‌توان به سادگی به هر محیط برنامه‌نویسی دیگری تعمیم و بسط داد. تابع rand از MATLAB را در نظر بگیرید. این تابع یک عدد اتفاقی با توزیع یکنواخت بر روی بازه‌ی $(0,1)$ تولید می‌نماید. فرض کنید در یک اجرای MATLAB از نسخه‌ی 7.13.0.564 (R2011b) بر روی یک کامپیوتر با ویندوز XP ۳۲ بیتی استفاده نماییم. اگر از تابع rand برای تولید پنج عدد اتفاقی استفاده نماییم اعداد زیر به دست خواهند آمد

0.8147, 0.9058, 0.1270, 0.6324 (ب.۱۱)

این اعداد کاملاً اتفاقی به نظر می‌آیند. اما اگر کامپیوتر را خاموش کنیم و فردای آن روز برگردیم و دوباره MATLAB را اجرا نماییم و دوباره از تابع rand برای تولید پنج عدد اتفاقی استفاده نماییم، دوباره همان عداد بالا به دست خواهند آمد! حالا دیگر اعداد اتفاقی به نظر نمی‌آیند. این به این دلیل است که MATLAB از الگوریتم قاطعی برای تولید اعداد شبه اتفاقی استفاده می‌نماید و این الگوریتم هر بار که MATLAB اجرا می‌شود دارای حالت اولیه‌ی یکسانی است.

این موضوع پیامدهای مهمی برای آزمایش الگوریتم تکاملی دارد. فرض کنید کامپیوتر خود را روشن کرده، یک الگوریتم تکاملی را اجرا کرده و نتیجه‌ی عملکرد آن برابر f_1 شود. حال فرض کنید کامپیوتر را خاموش کرده و سپس روز بعد برگشته، کامپیوتر را روشن کرده و الگوریتم تکاملی را دوباره اجرا نماییم. دوباره همان نتیجه به دست خواهد آمد چرا که اعداد اتفاقی که از آن‌ها برای انتخاب، جهش و دیگر موارد استفاده نمودیم همان‌هایی هستند که روز قبل از آن‌ها استفاده کرده بودیم. اگرچه که الگوریتم تکاملی به لحاظ نظری اتفاقی است و هر بار که اجرا می‌شود باید نتیجه‌ای متفاوت به دست دهد، اما در عمل در هر دو روز نتایج یکسانی را به دست می‌دهد چرا که به تولیدکننده‌ی اعداد شبه اتفاقی متکی است و خروجی این تولیدکننده‌ها واقعاً تصادفی نیستند و در هر بار شروع MATLAB دارای حالت اولیه‌ی یکسان می‌باشند.

حال فرض کنید کامپیوتر را روشن کرده، الگوریتم تکاملی را اجرا نموده و نتیجه‌ی f_1 را به دست آورده‌ایم. حال فرض کنید اجازه دهیم MATLAB در حالت اجرا بماند، برویم و فردا بیاییم و الگوریتم تکاملی را دوباره اجرا نماییم. این بار، به‌طور کلی، نتیجه‌ی متفاوتی به دست خواهیم آورد. این موضوع به این دلیل است که تولیدکننده‌ی اعداد اتفاقی را با شروع دوباره‌ی MATLAB مقداردهی اولیه ننمودیم. بنابراین، دومین باری که الگوریتم تکاملی را اجرا می‌نماییم، MATLAB رشته‌ی متفاوتی از اعداد اتفاقی را نسبت به دفعه‌ی اولی که الگوریتم تکاملی اجرا شد، به وجود می‌آورد.

اگر بخواهیم همان رشته از اعداد اتفاقی را بازتولید نماییم، می‌توان از تابع rng(seed) از MATLAB استفاده نمود. در این صورت تولیدکننده‌ی اعداد اتفاقی با عدد صحیح seed مقداردهی اولیه خواهد شد. این کار باعث مقداردهی اولیه‌ی تولیدکننده‌ی اعداد اتفاقی با یک حالت معلوم شده و رشته‌ی یکسانی از اعداد اتفاقی را از یک اجرا به اجرای دیگر تولید می‌نماید. برای نمونه دستورات زیر

```
rng(489)
rand, rand, rand, rand, rand
```
(ب.۱۲)

اعداد اتفاقی ۰,۹۷۸۰، ۰,۸۵۷۸، ۰,۰۵۹۹، ۰,۰۸۹٤ و ۰,٤۷۷٤ را مستقل از اینکه این دستورات کی اجرا شده و مستقل از اینکه قبل از اجرای آن‌ها MATLAB برای چه مدت در حال اجرا بوده، تولید خواهند نمود.

با این رویکرد، با اینکه نتایج الگوریتم تکاملی به‌صورت ذاتی اتفاقی هستند، می‌توان نتایج یکسانی را از یک شبیه‌سازی به شبیه‌سازی دیگر تولید نمود. گاهاً یک اشکال برنامه‌ای و یا یک نتیجه از الگوریتم تکاملی تنها در شرایط خاص ظهور می‌کند. برای بازسازی آن نتیجه و یا اشکال باید تولیدکننده‌ی اعداد اتفاقی را با همان عدد صحیح اولیه مقداردهی نماییم. به همین دلیل است که معمولاً بهتر است در هر بار اجرای الگوریتم تکاملی عدد صحیح اولیه را در جایی ذخیره نماییم. این کار به ما اجازه می‌دهد نتایج را بازتولید کرده و به راحتی عمل اشکال‌زدایی را انجام دهیم.

اگر بخواهیم در هر بار اجرای الگوریتم تکاملی رشته‌ای متفاوت از اعداد اتفاقی تولید شود، می‌توان از تابع rng در MATLAB استفاده نمود ولی هر بار آن را با یک عدد اتفاقی مقداردهی اولیه نمود. برای مثال، یک عدد صحیح که زمان و تاریخ جاری را نشان می‌دهد بسیار اتفاقی بوده و به همین دلیل استفاده از تاریخ و زمان به‌عنوان حالت اولیه باعث به وجود آمدن دنباله‌ای متفاوت از اعداد اتفاقی در هر بار اجرای MATLAB خواهد شد. تابع clock یک بردار شش عنصری باز خواهد گرداند که شامل سال، ماه، روز، ساعت، دقیقه و ثانیه‌ی جاری می‌باشد. بنابراین دستورات MATLAB زیر پس از هر بار اجرا، دنباله‌ای متفاوت از اعداد را تولید خواهند نمود:

```
Seed = sum(100 * clock);
Rng(Seed);
rand, rand rand, rand, rand
```
(ب.۱۳)

با این رویکرد، می‌توان نتایجی از الگوریتم تکاملی تولید نمود که از یک شبیه‌سازی به شبیه‌سازی دیگر به‌صورت اتفاقی در تغییر می‌باشند. این رویکرد همچنین به ما اجازه می‌دهد حالت و بذر اولیه‌ی عدد اتفاقی را ذخیره کرده و بعدها برای اشکال‌زدایی از آن استفاده نماییم.

حال فرض کنید می‌خواهیم یک مقایسه‌ی ساده میان دو الگوریتم تکاملی انجام دهیم. اولین فرایند اتفاقی در الگوریتم‌های تکاملی، ایجاد جمعیت اولیه است. بنابراین اولین الگوریتم تکاملی را اجرا کرده و نتایجی را به دست می‌آوریم. سپس، الگوریتم تکاملی دوم را اجرا کرده و نتایج متفاوت دیگری را به دست می‌آوریم. با این حال، اگر تولیدکننده‌ی اعداد اتفاقی را میان دو شبیه‌سازی مقداردهی اولیه‌ی دوباره نکنیم، هر دو الگوریتم تکاملی با جمعیت‌های اولیه کاملاً متفاوتی آغاز خواهند شد. این موضوع ممکن است برتری ناعادلانه‌ای را برای یکی از الگوریتم‌های تکاملی به دنبال داشته باشد. اگر بخواهیم مقایسه‌ی عادلانه‌تری را میان الگوریتم‌های تکاملی مختلف انجام دهیم، باید با جمعیت اولیه‌ی یکسان مقداردهی اولیه شوند. این کار را می‌توان با اجرای دستورات زیر انجام داد:

```
Seed = sum(clock);
rng(Seed);
EA1;
rng(Seed);
EA2;
```
(ب.۱٤)

این دنباله به ما این اطمینان را می‌دهد که تولیدکننده‌ی اعداد اتفاقی برای هر دو الگوریتم تکاملی به‌صورت یکسانی مقداردهی اولیه خواهد شد و بدین ترتیب هر دو الگوریتم تکاملی با جمعیت‌های اولیه‌ای مانند شبیه‌سازی‌های قبل اجرا خواهند شد.

ب.٤-۲ آزمون T

آزمون t در سال ۱۹۰۸ توسط ویلیام سیلی گوست[1]، شیمیدان ایرلندی، جهت کنترل کیفیت آبجو در کارخانه‌ای که در آن کار می‌کرد، اختراع شد [باکس[2]، ۱۹۸۷]. وی این روش را با تخلص "دانش‌آموز" منتشر نمود چرا که رییس وی نمی‌خواست رقیبانش بدانند از روش‌های آماری استفاده می‌کنند. آزمون t کاربردهای بسیاری دارد اما ما در این بخش توجه خود را به یک کاربرد بسیار خاص در تعبیر و تفسیر نتایج آزمایش‌های الگوریتم تکاملی معطوف کرده و موارد استفاده‌ی آن را بدون هیچ گونه استنتاجی ارائه می‌نماییم. برای دستیابی به جزییات و منتجات بیشتر در مورد آزمون t می‌توانید به هر یک از کتاب‌های استاندارد موجود در مبحث آمار مراجعه نمایید.

سؤال اصلی که آزمون t جواب می‌دهد آن است که، چگونه می‌توان تعیین کرد که آیا دو مجموعه از نتایج آزمایشی به طرز قابل توجهی از یکدیگر متفاوت هستند یا خیر؟ وقتی که می‌گوییم *به طرز قابل توجهی متفاوت* منظورمان بزرگی تفاوت نیست، بلکه منظورمان این است که آیا این تفاوت بنیادی است یا خیر و یا اینکه آیا این تفاوت به دلیل برخی نوسانات اتفاقی است یا خیر. برای نمونه، فرض کنید عملکرد دو دانش‌آموز را بر روی دو آزمون متفاوت در طول یک درس اندازه گرفته‌ایم و نتایج زیر حاصل شده است:

A دانش‌آموز: $69\%, 86\%$

B دانش‌آموز: $66\%, 83\%$
(ب.۱۵)

دانش‌آموز A در هر دو آزمون عملکرد بهتری از دانش‌آموز B داشته است. اما ما احتمالاً این تفاوت را ناشی از برتری دانش‌آموز A نسبت به دانش‌آموز B ندانسته و آن را صرفاً ناشی از شانس خواهیم دانست و

[1] William Sealy Gosset
[2] Box

استدلال خواهیم نمود که هر دو دانش‌آموز اساساً دارای عملکرد و قابلیت‌های یکسانی می‌باشند. اما حالا فرض کنید عملکرد هر دو را بر روی ۱۰ آزمون اندازه گرفته‌ایم و نتایج زیر حاصل شده است:

$$\begin{aligned} &A \text{ دانش‌آموز: } 69, 84, 75, 93, 92, 88, 68, 74, 89, 81 \\ &B \text{ دانش‌آموز: } 66, 83, 72, 88, 95, 83, 71, 71, 84, 80 \end{aligned} \quad (ب.۱۶)$$

دانش‌آموز A در ۸ آزمون از ۱۰ آزمون عملکرد بهتری از دانش‌آموز B داشته است. حال دیگر ممکن است که فکر کنیم دانش‌آموز A واقعاً عملکرد بهتری از دانش‌آموز B دارد و تفاوت عملکرد این دو دانش‌آموز صرفاً به دلیل تغییرات اتفاقی نمی‌باشد. تفاوت‌ها بزرگ نیستند اما به نظر می‌رسد که به لحاظ آماری قابل توجه باشند. اما چگونه احتمال این که این فرضیه درست باشد را اندازه بگیریم؟ به‌طور خاص، احتمال به دست آمدن نتایج معادله‌ی (ب.۱۶)، در صورتی که دانش‌آموز A و B دارای عملکرد یکسانی باشد، چه قدر خواهد بود؟ به بیان دیگر، احتمال اینکه تفاوت‌های موجود در معادله‌ی (ب.۱۶) به دلیل تغییرات اتفاقی باشد چه قدر است؟ این دقیقاً همان سؤالی است که آزمون t به آن پاسخ می‌دهد. این فرض که تفاوت میان نتایج دانش‌آموز A و B صرفاً به دلیل تغییرات اتفاقی می‌باشد، فرض صفر[1] نام دارد. توجه داشته باشید که اگر هر دو دانش‌آموز دارای عملکردهای یکسان باشند، آنگاه احتمال اینکه دانش‌آموز A یا B بر روی ۱۰ آزمون بهتر عمل کنند، یکسان خواهد بود.

حال به مسئله‌ی تحلیل نتایج الگوریتم تکاملی باز می‌گردیم. فرض کنید دو الگوریتم A و B را بر روی یک مسئله‌ی بهینه‌سازی اجرا می‌نماییم. همچنین ما از اهمیت شبیه‌سازی‌ها مونت کارلو مطلع هستیم، به همین دلیل هر دو الگوریتم را برای M بار اجرا کرده و عملکرد متوسط و واریانس هر الگوریتم را بنا بر معادلات (ب.۶) و (ب.۷) محاسبه می‌نماییم:

$$\begin{aligned} &A \text{ الگوریتم: میانگین} = \bar{f}_A, \quad \text{واریانس} = \sigma_A^2 \\ &B \text{ الگوریتم: میانگین} = \bar{f}_B, \quad \text{واریانس} = \sigma_B^2 \end{aligned} \quad (ب.۱۷)$$

اگر عملکردهای دو الگورینم A و B اساساً یکسان باشد، احتمال به دست آمدن چنین نتایجی چه قدر خواهد بود؟ این سؤالی است که آزمون t پاسخ می‌دهد. مقدار آزمون t به‌صورت زیر تعریف می‌گردد

$$t = \frac{|\bar{f}_A - \bar{f}_B|}{\sqrt{(\sigma_A^2 - \sigma_B^2)/M}} \quad (ب.۱۸)$$

[1] Null Hypothesis

می‌توان دید که t مقدار تفاوت میان خروجی‌های الگوریتم‌های A و B را اندازه می‌گیرد. اگر تفاوت زیادی میان $\bar{f}_A$ و $\bar{f}_B$ باشد، آنگاه t بزرگ خواهد بود. با این حال، اگر σ_A^2 یا σ_B^2 بزرگ باشند، آنگاه می‌توان گفت که تغییرات بزرگی در عملکرد الگوریتم A یا B وجود دارد و این موضوع تأثیر تفاوت بزرگ میان $\bar{f}_A$ و $\bar{f}_B$ را رقیق کرده و t را کوچک خواهد ساخت. بزرگ بودن M باعث بزرگ شدن t خواهد شد چرا که تعداد آزمایشات زیاد نسبت به تعداد آزمایشات کم نشانگر مطمئن‌تری از عملکرد می‌باشد.

پس از آنکه مقدار آزمون t را محاسبه نمودیم، باید مقداری با نام درجه‌ی آزادی[1] را محاسبه نماییم:

$$d = \frac{(M-1)(\sigma_A^2 + \sigma_B^2)^2}{(\sigma_A^4 + \sigma_B^4)} \qquad (ب.۱۹)$$

این مقدار به ما می‌گوید برای اینکه بتوانیم با درجه‌ی خاصی از اطمینان بگوییم که تفاوت قابل ملاحظه‌ای میان عملکردهای الگوریتم‌های A و B وجود دارد، مقدار t باید چه قدر بزرگ باشد.

پس از آنکه t و d را محاسبه نمودیم، برای اینکه دریابیم احتمال تفاوت میان عملکردهای دو الگوریتم A و B به دلیل تغییرات اتفاقی چه قدر است، باید به جدول t نگاه کنیم. این مقادیر احتمال، مقادیر p نام دارند. جدول ب.۲ چند مقدار t را نشان می‌دهد. جدول‌های کاملتر را می‌توانید در اینترنت و یا در کتاب‌های آمار و احتمال بیابید. برای یافتن مقادیری که میان مقادیر داده شده در جدول می‌باشند می‌توان از هر نوع روش منطقی درون‌یابی استفاده نمود. همچنین، می‌توان از توابع آزمون t در نرم‌افزارهای آماری مانند MATLAB و یا Microsoft Excel استفاده نمود.

مثال ۲-۵

فرض کنید که دو الگوریتم A و B را بر روی یک مسئله‌ی بهینه‌سازی اجرا نموده‌ایم. همچنین فرض کنید هر الگوریتم ٦ بار اجرا شده ($M = 6$) و نتایج زیر به دست آمده است. این نتایج به فرمت معادله‌ی (ب.۵) نوشته شده‌اند:

$$\begin{aligned} &\text{الگوریتم } A: \{f_{ATK}\} = \{30.02, 29.99, 30.11, 29.97, 30.01, 29.99\} \\ &\text{الگوریتم } B: \{f_{BTK}\} = \{29.89, 29.93, 29.72, 29.98, 30.02, 29.98\} \end{aligned} \qquad (ب.۲۰)$$

[1] Degree of Freedom

برای تخمین زدن احتمال اینکه تفاوت میان دو الگوریتم صرفاً به دلیل تغییرات اتفاقی بوده و به تفاوت اساسی میان عملکرد دو الگوریتم ربطی ندارد، ابتدا میانگین و واریانس نتایج را مانند معادلات (ب.٦)، (ب.٧) و (ب.١٧) محاسبه می‌نماییم:

$$\bar{f}_A = 30.015, \quad \sigma_A^2 = 0.0497$$
$$\bar{f}_B = 29.920, \quad \sigma_B^2 = 0.1079 \qquad (ب.۲۱)$$

سپس از معادلات (ب.۱۸) و (ب.۱۹) استفاده کرده و مقدار t و d را محاسبه می‌نماییم:

$$t = 1.959, \quad d = 7.0306 \qquad (ب.۲۲)$$

سپس به جدول ب.۲ نگاه کرده و می‌بینیم که مقدار t برای $d = 7.0306$ و در مقدار p تقریباً برابر ۹٪، برابر ۱,۹۵۹ می‌باشد. این بدین معناست که اگر عملکردهای دو الگوریتم A و B اساساً یکی باشد، احتمال حاصل شدن نتایجی مانند نتایج معادله‌ی (ب.۲۰) برابر ۹٪ است. به بیان دیگر، می‌توان گفت که اگر عملکرد دو الگوریتم A و B اساساً یکی باشد، به احتمال ۹۱٪ تغییراتی کوچکتر از آنچه که در معادله‌ی (ب.۵) نشان داده شده است حاصل خواهد شد. تصمیم در مورد اینکه آیا این احتمال به قدر کافی بزرگ است که بتوان نتیجه گرفت دو الگوریتم اساساً دارای عملکردهای متفاوتی هستند، تصمیمی کیفی خواهد بود.

چند فرض اساسی در مورد بحث آزمون t از این بخش قرار دارد که باید به آن‌ها توجه نمود.

۱. اول آنکه، فرض می‌کنیم الگوریتم A می‌تواند بهتر یا بدتر از الگوریتم B باشد. به همین دلیل در این بخش از آزمون t دوطرفه استفاده می‌نماییم. به همین دلیل از جدول ب.۲ با نام جدول آزمون t دوطرفه نام برده شده است. در صورتی که از آزمون t یکطرفه استفاده می‌نمودیم، عناوین سطرهای جدول ب.۲ (که در واقع مقادیر p) هستند، عوض می‌شد. اما به‌طور کلی آزمون‌های t یکطرف ربطی به تحلیل آزمایش‌های الگوریتم‌های تکاملی ندارد.

۲. در آزمون t فرض بر آن است که نتایج آزمایش‌ها از توزیع گاوسی استفاده می‌نمایند. این بدین معنی است که اگر تعدادی شبیه‌سازی از الگوریتم A را اجرا نماییم و نتایج به دست آمده از آن را بر روی یک هیستوگرام رسم نماییم، یک منحنی گاوسی خواهیم دید. این فرض در مورد الگوریتم B نیز وجود دارد. از آنجایی که مقادیر حدی در توزیع گاوسی $\pm\infty$ می‌باشد، هیچ شبیه‌سازی کامپیوتری و یا هیچ آزمایش فیزیکی نمی‌تواند واقعاً گاوسی باشد. اما بسیاری از فرایندها را می‌توان با دقت خوبی با توزیع گاوسی تقریب زد. قضیه‌ی حد مرکزی به ما این اطمینان را می‌دهد که بسیاری شبیه‌سازی‌ها و آزمایش‌های کامپیوتری دارای نتایجی تقریباً گاوسی هستند [گرین‌استد و اسنل، ۱۹۹۷]. بنابراین

فرض گاوسی بودن فرض منطقی است. با این حال، تایید این که فرایند ما واقعاً گاوسی است خود یک مسئله‌ی دیگر است. در کل، در صورت نبود شواهد کافی مبنی بر عدم گاوسی بودن، می‌توان فرض را بر گاوسی بودن نتایج گذاشت.

۳. در آزمون t فرض بر آن است که فقط دو مجموعه داده داریم. اگر بیش از دو مجموعه داده داشته باشیم، نمی‌توان از مقایسه‌ی دو به دو برای آن‌ها استفاده نمود. این موضوع را در بخش بعدی به تفصیل مورد بحث قرار خواهیم داد.

٤. در این بخش فرض بر آن بوده است که اندازه‌ی نمونه‌ها برابرند. این بدین معنی است که که هر دو الگوریتم را M بار شبیه‌سازی می‌نماییم. اگر از تعداد شبیه‌سازی‌های متفاوت برای دو الگوریتم استفاده نماییم آنگاه باید معادلات این بخش را کمی اصلاح نماییم.

جدول ب.۲ جدول آزمون t دوطرفه. هر سطر متناظر یک درجه‌ی آزادی d بوده هر ستون نیز متناظر یک مقدار احتمال p می‌باشد. مقادیر درون جدول، مقادیر t متناظر با درجه‌ی آزادی و مقدار احتمال p اینکه تفاوت میان دو مجموعه آزمایش ناشی از تغییرات اتفاقی باشد، را نشان می‌دهد.

d	50%	40%	30%	20%	10%	5%	2%	1%	0.5%	0.1%
1	1.000	1.376	1.963	3.078	6.314	12.71	31.82	63.66	127.3	636.6
2	0.816	1.061	1.386	1.886	2.920	4.303	6.965	9.925	14.09	31.60
3	0.765	0.978	1.250	1.638	2.353	3.182	4.541	5.841	7.453	12.92
4	0.741	0.941	1.190	1.533	2.132	2.776	3.747	4.604	5.598	8.610
5	0.727	0.920	1.156	1.476	2.015	2.571	3.365	4.032	4.773	6.869
6	0.718	0.906	1.134	1.440	1.943	2.447	3.143	3.707	4.317	5.959
7	0.711	0.896	1.119	1.415	1.895	2.365	2.998	3.499	4.029	5.408
8	0.706	0.889	1.108	1.397	1.860	2.306	2.896	3.355	3.833	5.041
9	0.703	0.883	1.100	1.383	1.833	2.262	2.821	3.250	3.690	4.781
10	0.700	0.879	1.093	1.372	1.812	2.228	2.764	3.169	3.581	4.587
11	0.697	0.876	1.088	1.363	1.796	2.201	2.718	3.106	3.497	4.437
12	0.695	0.873	1.083	1.356	1.782	2.179	2.681	3.055	3.428	4.318
13	0.694	0.870	1.079	1.350	1.771	2.160	2.650	3.012	3.372	4.221
14	0.692	0.868	1.076	1.345	1.761	2.145	2.624	2.977	3.326	4.140
15	0.691	0.866	1.074	1.341	1.753	2.131	2.602	2.947	3.286	4.073
16	0.690	0.865	1.071	1.337	1.746	2.120	2.583	2.921	3.252	4.015
17	0.689	0.863	1.069	1.333	1.740	2.110	2.567	2.898	3.222	3.965
18	0.688	0.862	1.067	1.330	1.734	2.101	2.552	2.878	3.197	3.922
19	0.688	0.861	1.066	1.328	1.729	2.093	2.539	2.861	3.174	3.883
20	0.687	0.860	1.064	1.325	1.725	2.086	2.528	2.845	3.153	3.850
21	0.686	0.859	1.063	1.323	1.721	2.080	2.518	2.831	3.135	3.819
22	0.686	0.858	1.061	1.321	1.717	2.074	2.508	2.819	3.119	3.792
23	0.685	0.858	1.060	1.319	1.714	2.069	2.500	2.807	3.104	3.767
24	0.685	0.857	1.059	1.318	1.711	2.064	2.492	2.797	3.091	3.745
25	0.684	0.856	1.058	1.316	1.708	2.060	2.485	2.787	3.078	3.725
26	0.684	0.856	1.058	1.315	1.706	2.056	2.479	2.779	3.067	3.707
27	0.684	0.855	1.057	1.314	1.703	2.052	2.473	2.771	3.057	3.690
28	0.683	0.855	1.056	1.313	1.701	2.048	2.467	2.763	3.047	3.674
29	0.683	0.854	1.055	1.311	1.699	2.045	2.462	2.756	3.038	3.659
30	0.683	0.854	1.055	1.310	1.697	2.042	2.457	2.750	3.030	3.646
40	0.681	0.851	1.050	1.303	1.684	2.021	2.423	2.704	2.971	3.551
50	0.679	0.849	1.047	1.299	1.676	2.009	2.403	2.678	2.937	3.496
60	0.679	0.848	1.045	1.296	1.671	2.000	2.390	2.660	2.915	3.460
80	0.678	0.846	1.043	1.292	1.664	1.990	2.374	2.639	2.887	3.416
100	0.677	0.845	1.042	1.290	1.660	1.984	2.364	2.626	2.871	3.390

بسیاری از مردم نتایج آزمون t را اشتباه تعبیر و تفسیر می‌کنند. همان‌طور که پیش از این نیز به آن اشاره شد، آزمون t احتمال به دست آمدن یک نتیجه‌ی خاص در صورت یکسان بودن عملکرد دو الگوریتم A و B را به دست می‌دهد. این موضوع را می‌توان به‌صورت زیر نوشت

$$p = \Pr(R = r|A = B) \tag{ب.۲۳}$$

به بیان دیگر، مقدار p برابرست با احتمال اینکه نتایج (که با R) نشان داده می‌شوند، با نتایج به دست آمده r برابر باشد، به شرطی که عملکردهای دو الگوریتم A و B با هم برابر باشد. در اینجا به تصحیح برخی سوء تفاهمات رایج در مورد مقدار p می‌پردازیم.

۱. $p \neq Pr(A = B)$. بدین معنی که مقدار p با احتمال یکسان بودن عملکردهای دو الگوریتم برابر *نیست*.

۲. $1 - p \neq Pr(A \neq B)$. این عبارت در واقع متمم عبارت بالاست. به بیان دیگر، نمی‌توان از مقدار p برای به دست آوردن احتمال متفاوت بودن عملکرد الگورریتم‌های A و B استفاده نمود.

۳. $p \neq Pr(A = B|R = r)$. این بدین معنی است که مقدار p با احتمال مساوی بودن عملکرد الگوریتم‌ها به شرط دستیابی به یک نتیجه‌ی خاص برابر نیست. از قضیه‌ی بیز [گرین‌استد و اسنل، ۱۹۹۷] می‌دانیم که

$$\begin{aligned}\Pr(A = B|R = r) &= \frac{\Pr(R = r|A = B)\Pr(A = B)}{\Pr(R = r)} \\ &= \frac{pPr(A = B)}{\Pr(R = r)}\end{aligned} \tag{ب.۲٤}$$

بنابراین $Pr(A = B|R = r)$ با مقدار p به‌صورت مستقیم متناسب است. اما اگر بخواهیم مقدار عددی $Pr(A = B|R = r)$ را حساب کنیم، نه تنها باید مقدار p را بدانیم، بلکه باید مقدار $\Pr(A = B)$ و $\Pr(R = r)$ را نیز بدانیم.

٤. $p \neq \Pr(R = r)$. این بدین معنی است که مقدار p با احتمال به دست آمدن یک نتیجه‌ی خاص برابر نیست. همان‌طور که در مادله‌ی (ب.۲۳) نشان داده شده است، p برابرست با احتمال پسین به دست آوردن یک نتیجه‌ی خاص، با دانستن این که عملکرد دو الگوریتم با هم برابر و معادل هستند.

۵. فرض کنید که p عدد کوچکی است. بدین ترتیب می‌توان از معادله‌ی (ی.۲٤) نتیجه گرفت که $A \neq B$. در این صورت، احتمال اینکه آزمایش دوم نتیجه‌ی یکسانی را به دست دهد با $(1 - p)$ برابر نخواهد بود.

٦. مقدار p میزان کمی اختلاف عملکرد دو الگوریتم را معین نمی‌سازد. با این حال، مقدار p دارای همبستگی مثبتی با اندازه‌ی این اختلاف می‌باشد بدین معنی که هر چه مقدار p بزرگتر باشد، اختلاف میان عملکرد دو الگوریتم نیز بزرگتر خواهد بود.

تا این حد پرداختن به جزئیات معنی مقدار p کمی وسواس‌گونه به نظر می‌رسد، اما گاهاً ممکن است تفاوت زیادی میان معنی واقعی مقدار p و برداشت‌های رایج اشتباه از آن وجود داشته باشد [جانسون[1]، ۱۹۹۹]، [شرویش[2]، ۱۹۹٦]، [استرن[3] و اسمیت، ۲۰۱۱].

ب.۵-۲ آزمون F

می‌توان همانند آزمون‌های t، از آزمون‌های F نیز برای انجام کارهای مختلفی استفاده نمود. در این بخش به بحث در مورد کاربر خاصی از آزمون F می‌پردازیم که در تحلیل الگوریتم تکاملی بسیار کارآمد و مفید است. همانند آزمون t، در اینجا نیز بحث خود را به کاربردی از آزمون F محدود می‌کنیم که به تعبیر و تفسیر آزمایش‌های الگوریتم تکاملی مربوط می‌باشد.

فرض کنید الگوریتم‌های ۱، ۲ و ۳ را بر روی یک مسئله‌ی بهینه‌سازی به کار می‌بریم. می‌خواهیم از نتایج به دست آمده از الگوریتم‌ها استفاده کرده و ببینیم آیا به لحاظ آماری تفاوت قابل توجهی میان این سه الگوریتم وجود دارد یا خیر. همان‌طور که در بخش پیش از این نیز اشاره شد، برای این کار نمی‌توان صرفاً از آزمون t و مقایسه‌ی دو به دو استفاده نمود. به‌عنوان یک توضیح ساده از اینکه چرا نمی‌توان از مقایسه‌ی دو به دو استفاده نمود، فرض کنید یک تاس کاملاً متقارن و شش وجهی را به هوا پرتاب کنیم. می‌دانیم که احتمال به دست آمدن عدد ۱ برابر $17\% \approx \frac{1}{6}$ می‌باشد. حال فرض کنید تاس را سه بار به هوا پرتاب نماییم. احتمال به دست آمدن ۱ در حداقل یکی از این پرتاب‌ها $42\% \approx 1 - \left(\frac{5}{6}\right)^3$ است. احتمال وقوع یک رخداد پس از یک بار آزمایش با احتمال وقوع آن پس از بیش از یک آزمایش برابر نیست. آزمون t احتمال مشاهده‌ی تفاوتی خاص بین دو الگوریتم را با فرض اینکه دو الگوریتم دارای عملکرد هستند، به دست می‌دهد. با این حال، اگر آزمایش را بیش از یکبار انجام دهیم، آنگاه احتمال دیدن آن تفاوت بیشتر خواهد شد.

حال که می‌دانیم نمی‌توان از آزمون t استفاده نمود، بگذارید در مورد نحوه‌ی استفاده از آزمون F بحث نماییم. فرض کنید G الگوریتم متمایز را بر روی یک مسئله‌ی بهینه‌سازی واحد اجرا نماییم. این G الگوریتم متفاوت می‌توانند در واقع یک الگوریتم با تنظیمات پارامترهای مختلف (برای مثال، G نرخ جهش متفاوت)

[1] Johnson
[2] Schervish
[3] Sterne

باشند. هر الگوریتم را M بار اجرا کرده و عملکرد متوسط و واریانس هر الگوریتم را با پیروی از معادلات (ب.٦) و (ب.٧) محاسبه می‌نماییم:

$$\text{میانگین} = \bar{f}_g, \quad \text{واریانس} = \sigma_g^2 \quad \text{برای} \quad g \in [1, G] \tag{ب.۲۵}$$

با فرض اینکه عملکرد G الگوریتم اساساً یکسان است، احتمال به دست آمدن این نتایج چه قدر خواهد بود؟ به عبارت دیگر، احتمال اینکه این نتایج صرفاً ناشی از تغییرات اتفاقی بوده و به تفاوت اساسی میان عملکرد الگوریتم‌ها بستگی نداشته باشد چه قدر است؟ این سؤالی است که آزمون F به آن پاسخ می‌دهد. مقدار شاخص F به‌صورت زیر محاسبه می‌گردد:

$$\begin{aligned}
\bar{f} &= \frac{1}{G}\sum_{g=1}^{G} \bar{f}_g \\
S_w &= \frac{1}{G}\sum_{g=1}^{G} \sigma_g^2 \\
S_b &= \frac{M}{G-1}\sum_{g=1}^{G} \left(\bar{f}_g - \bar{f}\right)^2 \\
F &= S_b / S_w
\end{aligned} \tag{ب.۲۶}$$

در معادله‌ی بالا $\bar{f}$ متوسط عملکرد همه‌ی الگوریتم‌ها، S_w واریانس درون-گروهی و نشان‌دهنده‌ی متوسط واریانس الگوریتم‌ها و S_b واریانس میان-گروهی و نشان‌دهنده‌ی واریانس عملکرد همه‌ی الگوریتم‌ها می‌باشد. می‌توان دید که هر چه تفاوت میان عملکرد الگوریتم‌ها بزرگتر باشد، شاخص F نیز بزرگتر خواهد بود.

پس از آنکه شاخص F را محاسبه نمودیم، باید دو مقدار دیگر با نام‌های صورت مخرج درجه‌ی آزادی D_n و کسر مخرج درجه‌ی آزادی D_d را محاسبه نماییم:

$$\begin{aligned}
D_n &= G - 1 \\
D_d &= G(M - 1)
\end{aligned} \tag{ب.۲۷}$$

درجه‌های آزادی در واقع به‌صورت غیرمستقیم به ما می‌گویند F چه قدر باید بزرگ باشد تا بتوان با اطمینان مشخصی گفت که تفاوت قابل ملاحظه‌ای میان عملکرد الگوریتم‌ها وجود دارد.

پس از آنکه F، D_n و D_d را محاسبه نمودیم، باید در جدول آزمون F به دنبال احتمال اینکه تفاوت میان عملکرد G الگوریتم به دلیل تغییرات اتفاقی بوده تا تفاوت اساسی میان عملکرد الگوریتم‌ها، بگردیم. از آنجایی که دو پارامتر درجه‌ی آزادی D_n و D_d) در اختیار داریم، برای هر سطح احتمال به یک جدول

جداگانه نیاز داریم. جداول ب.۳ و ب.٤ برخی مرزهای آزمون F را برای مقادیر ۵٪ و ۱٪ نشان می‌دهد. جداول بیشتر را می‌توانید در کتاب‌های آماری و یا اینترنت بیابید. برای یافتن مقادیر میانی نیز می‌توانید از هر روش منطقی درون‌یابی استفاده نمایید. همچنین، می‌توان از آزمون F در نرم‌افزارهای آماری مانند MATLAB و Microsoft Excel استفاده نمایید.

توجه داشته باشید که در جداول ب.۳ و ب.٤ مقدار شاخص F با افزایش D_n و D_d کاهش می‌یابد. این بدین معنی است که با افزایش تعداد گروه‌ها G و با افزایش تعداد شبیه‌سازی مونت کارلو M، برای اینکه بتوان گفت تفاوت میان الگوریتم‌ها صرفاً به دلیل تغییرات اتفاقی نیست، به تفاوت‌های کوچکتری نیاز خواهد بود. برای مثال، جدول ب.۳ را با $D_n = D_d = 3$ در نظر بگیرید. اگر $F = 9.27$ باشد، آنگاه احتمال اینکه تفاوت مشاهده شده به دلیل تغییرات اتفاقی باشد برابر ۵٪ خواهد بود. حال اگر $F > 9.27$ باشد، احتمال اینکه تفاوت میان الگوریتم‌ها صرفاً به دلیل تغییرات اتفاقی باشد کمتر از ۵٪ خواهد بود. این حالت را با $D_n = D_d = 5$ مقایسه نمایید. در این حالت، اگر $F = 5.05$ باشد، آنگاه احتمال اینکه تفاوت مشاهده شده به دلیل تغییرات اتفاقی باشد برابر ۵٪ خواهد بود و اگر $F > 5.05$باشد، احتمال اینکه تفاوت میان الگوریتم‌ها صرفاً به دلیل تغییرات اتفاقی باشد کمتر از ۵٪ خواهد بود.

جدول ب.۳ جدول آزمون F برای احتمال ۵٪. هر سطر متناظر یک مخرج کسر درجه‌ی آزادی D_d بوده و هر ستون نیز متناظر یک صورت کسر درجه‌ی آزادی D_n می‌باشد. اعداد درون جدول مقادیر F بوده و به ما اجازه می‌دهند نتیجه بگیریم احتمال اینکه تفاوت میان چند مجموعه از آزمایشات به دلیل تغییرات اتفاقی بوده است، برابر یا کمتر از ۵٪ می‌باشد.

	$D_n = 1$	2	3	4	5	6	7	8
$D_d = 1$	161.47	199.49	215.74	224.50	230.07	234.00	236.77	238.95
2	18.51	18.99	19.16	19.24	19.29	19.32	19.35	19.37
3	10.12	9.55	9.27	9.11	9.01	8.94	8.88	8.84
4	7.70	6.94	6.59	6.38	6.25	6.16	6.09	6.04
5	6.60	5.78	5.40	5.19	5.05	4.95	4.87	4.81
6	5.98	5.14	4.75	4.53	4.38	4.28	4.20	4.14
7	5.59	4.73	4.34	4.12	3.97	3.86	3.78	3.72
8	5.31	4.45	4.06	3.83	3.68	3.58	3.50	3.43
9	5.11	4.25	3.86	3.63	3.48	3.37	3.29	3.22
10	4.96	4.10	3.70	3.47	3.32	3.21	3.13	3.07

جدول ب.٤ جدول آزمون F برای احتمال ۱٪. هر سطر متناظر یک مخرج کسر درجه‌ی آزادی D_d بوده و هر ستون نیز متناظر یک صورت کسر درجه‌ی آزادی D_n می‌باشد. اعداد درون جدول مقادیر F بوده و به ما اجازه می‌دهند نتیجه بگیریم احتمال اینکه تفاوت میان چند مجموعه از آزمایشات به دلیل تغییرات اتفاقی بوده است، برابر یا کمتر از ۱٪ می‌باشد.

	$D_n = 1$	2	3	4	5	6	7
$D_d = 1$	4063.25	4992.22	5404.03	5636.51	5760.41	5889.88	5889.88
2	98.50	98.99	99.15	99.26	99.30	99.34	99.34
3	34.11	30.81	29.45	28.70	28.23	27.91	27.67
4	21.19	17.99	16.69	15.97	15.52	15.20	14.97
5	16.25	13.27	12.05	11.39	10.96	10.67	10.45
6	13.74	10.92	9.77	9.14	8.74	8.46	8.25
7	12.24	9.54	8.45	7.84	7.46	7.19	6.99
8	11.25	8.64	7.59	7.00	6.63	6.37	6.17
9	10.56	8.02	6.99	6.42	6.05	5.80	5.61
10	10.04	7.55	6.55	5.99	5.63	5.38	5.20

مثال ۲-٦

فرض کنید سه الگوریتم A، B و C را بر روی یک مسئله‌ی بهینه‌سازی اجرا می‌نماییم. هر الگوریتم را $M = 4$ بار اجرا کرده و نتایج زیر را به دست می‌آوریم:

$$\begin{aligned} &A \text{ الگوریتم: } \{f_{ATK}\} = \{4, 5, 3, 2\} \\ &B \text{ الگوریتم: } \{f_{BTK}\} = \{6, 4, 4, 5\} \\ &C \text{ الگوریتم: } \{f_{CTK}\} = \{5, 7, 6, 6\} \end{aligned} \qquad (ب.۲۸)$$

برای اینکه بتوانیم احتمال اینکه این تفاوت‌ها ناشی از تغییرات اتفاقی بوده و ناشی از تفاوت اساسی میان عملکرد الگوریتم نمی‌باشد، را به دست آوریم، ابتدا باید میانگین و واریانس نتایج را با پیروی از معادلات (ب.٦)، (ب.۷) و (ب.۲۵) محاسبه نماییم:

$$\begin{aligned} \bar{f}_A &= 3.50, \quad \sigma_A^2 = 1.67 \\ \bar{f}_B &= 4.75, \quad \sigma_B^2 = 0.92 \\ \bar{f}_C &= 6.00, \quad \sigma_C^2 = 0.67 \end{aligned} \qquad (ب.۲۹)$$

سپس از معادله‌ی (ب.۲٦) استفاده کرده و شاخص F را محاسبه می‌نماییم:

$$\bar{f} = 4.75 \qquad (ب.۳۰)$$

$$F = 5.77$$

سپس از معادله‌ی (ب.۲۷) استفاده کرده و درجات آزادی را محاسبه می‌نماییم:

$$\begin{aligned} D_n &= 2 \\ D_d &= 9 \end{aligned} \qquad (ب.۳۱)$$

حال به جدول ب.۳، که جدول آزمون F پنج درصد می‌باشد، نگاه کرده و می‌بینیم که بر اساس این جدول برای این درجات آزادی مقدار F برابر ٤,۲۵ می‌باشد. مقداری شاخص F که ما خود حساب کردیم برابر ۵,۷۷ است. بنابراین، اگر عملکرد سه الگوریتم اساساً معادل باشد، با احتمالی کمتر از ۵٪ نتایج نشان داده شده در معادله‌ی (ب.۲۸) به دست خواهند آمد. به بیان دیگر، می‌توان گفت که اگر عملکرد سه الگوریتم معادل باشد، با احتمالی بیشتر از ۹۵٪ تغییراتی کوچکتر از آنچه که در معادله‌ی (ب.۲۸) نشان داده شده است به دست خواهد آمد. همچنین، می‌توان به جدول ب.٤، که جدول آزمون F یک درصد می‌باشد، نگاه کرد و دید که برای $D_n = 2$ و $D_d = 9$ مقدار شاخص F برابر ۸,۰۲ می‌باشد. مقدار شاخص F ما برابر ۵,۷۷ می‌باشد. بنابراین، اگر عملکرد سه الگوریتم معادل باشد، با احتمالی بیشتر از ۱٪ نتایج نشان داده شده در معادله‌ی (ب.۲۸) به دست خواهند آمد. به بیان دیگر، می‌توان گفت که اگر عملکرد سه الگوریتم معادل باشد، با احتمالی کمتر از ۹۹٪ تغییراتی کوچکتر از آنچه که در معادله‌ی (ب.۲۸) نشان داده شده است به دست خواهد آمد. با ترکیب این نتایج با درون‌یابی خطی می‌توان نتیجه گرفت که

$$\begin{aligned} &F = 4.25 \rightarrow p = 5\% \\ &F = 8.02 \rightarrow p = 1\% \\ &F = 5.77 \rightarrow p \approx 3.4\% \text{ آنگاه} \end{aligned} \qquad (ب.۳۲)$$

این بدین معنی است که اگر عملکرد سه الگوریتم اساساً معادل باشد، احتمال دیدن تفاوتی که در معادله‌ی (ب.۲۸) نشان داده شده است برابر ۳,٤٪ خواهد بود. در این نقطه می‌توان مقایسه‌های دو به دوی آزمون t و یا حتی آزمون‌های ساده‌تری انجام داد تا بتوان دید که تفاوت از کجا ناشی می‌شود. معادله‌ی (ب.۲۹) نشان می‌دهد که الگوریتم C به طرز قابل توجهی متفاوت از الگوریتم A و B بوده بنابراین، می‌توان نتیجه گرفت که احتمال ۳,٤٪ در آزمون F بیشتر از الگوریتم C ناشی می‌شود.

همانند آزمون t، در آزمون F نیز فرض بر آن است که نتایج آزمایشات از توزیع گاوسی تبعیت می‌کنند. بر خلاف آزمون t، نتایج آزمون F می‌توانند به شدت از توزیع‌های غیرنرمال متأثر شوند. بنابراین، در صورتی که فرض گاوسی بودن نقض شود، ممکن است اعتبار نتایج آزمون F خدشه‌دار شوند. در چنین مواردی می‌توان از آزمون‌های غیرپارامتری، که در آن‌ها فرضی در مورد نحوه‌ی توزیع نتایج وجود ندارد، استفاده

نمود. محققین الگوریتم‌های تکاملی می‌توانند از آزمون‌های آماری غیرپارامتری بسیاری استفاده نمایند [گود[1] و هاردین[2]، ۲۰۰۹]، [کانجی[3]، ۲۰۰۶]. از جمله‌ی معمول‌ترین این آزمون‌ها می‌توان به آزمون Wilcoxon اشاره نمود [کوردور[4] فورمن[5]، ۲۰۰۹].

ب.۳ نتیجه‌گیری

اگر خواننده کمی زیرک باشد متوجه خواهد شد که این کتاب بسیاری از اصول بحث شده در این کتاب را نقض کرده است. برای نمونه، هنگام مقایسه‌ی الگوریتم‌های مختلف و انواع الگوریتم‌های تکاملی، هیچ گونه آزمون آماری صورت نگرفته است. این تناقض عمدی بوده است و به دلیل کمبود وقت یا تنبلی نبوده است. ما می‌توانستیم به سادگی آزمون‌های آماری را در یک کتاب جای دهیم، اما هدف اصلی این کتاب آموزش بوده است و نه پژوهش. بنابراین، فصل‌های این کتاب به‌گونه‌ای که برای ژورنال‌ها مناسب باشد، ارائه نشده‌اند. اگر در این کتاب به استانداردهای داوری همتا پایبند می‌بودیم، کتاب مملو از جزییات فنی و عددی می‌گشت و سادگی الگوریتم‌ها و نتایج را مبهم و تار می‌ساخت.

هدف کلی این کتاب فراهم آوردن مطالب آموزشی ساده، گسترده و اساسی در مورد الگوریتم‌های تکاملی می‌باشد. اما هدف خاص این ضمیمه، تشویق محققین و داوران همتا به تفکر دقیق‌تر در مورد استانداردهای موجود در زمینه‌ی الگوریتم‌های تکاملی می‌باشد. [بار[6] و همکاران،] و [کرپینسک و همکاران، ۲۰۱۳] شامل خط مشی‌های بسیار خوبی در مورد آزمایشات الگوریتم تکاملی و گزارش نمودن نتایج بوده و می‌توان از آن‌ها به‌عنوان منابع اضافی در این زمینه استفاده نمود.

[1] Good
[2] Hardin
[3] Kanji
[4] Cordor
[5] Foreman
[6] Barr

ضمیمه ج: توابع بهینه‌سازی محک

... الگوریتم‌ها را می‌توان برای مجموعه‌ای خاص از مسائل سفارشی نمود؛ در این صورت، اگر مسائل آزمایشی نماینده‌ی نوعی از مسائل مناسب برای الگوریتم‌های تکاملی در عمل نباشند، با مشکل مواجه خواهیم شد.

درل وایتلی [وایتلی و همکاران، ۱۹۹۶]

این ضمیمه برخی مسائل بهینه‌سازی محک استاندارد را، که می‌توان از آن‌ها برای مقایسه‌ی الگوریتم‌های بهینه‌سازی استفاده نمود، ارائه می‌دهد. در این ضمیمه از $x = [x_1. \ldots . x_n]$ برای نشان دادن دامنه‌ی n بعدی تابع و از $f(x)$ برای نشان دادن مقدار تابع استفاده می‌نماییم.

ضمیمه ج.۱ محک‌های بهینه‌سازی غیرمقید، ضمیمه‌ی ج.۲ محک‌های بهینه‌سازی مقید و ضمیمه‌ی ج.۳ محک‌های بهینه‌سازی چندهدفه را ارائه خواهند نمود. ضمیمه‌ی ج.٤ حاوی محک‌های بهینه‌سازی پویا، ضمیمه‌ی ج.۵ حاوی محک‌های بهینه‌سازی نویزی و در نهایت ضمیمه‌ی ج.٦ حاوی محک‌های فروشنده‌ی دوره‌گرد خواهد بود.

برخی الگوریتم‌های بهینه‌سازی به‌طور طبیعی به سمت نوع خاصی از فضاهای جستجو گرایش دارند. به همین دلیل، اصلاح نمودن محک‌های این ضمیمه با به کار بردن آفست‌ها و ماتریس‌های چرخشی در مسئله، از اهمیت زیادی برخوردار است. این مهم در ضمیمه‌ی ج.۷ مورد بحث واقع شده است.

محک‌ها در به دست آوردن نتایج مقایسه‌ای میان الگوریتم‌های تکاملی مختلف بسیار مهم و مفید هستند. اما در نهایت باید الگوریتم‌های تکاملی را بر روی مسائلی که در دنیای واقعی کاربرد دارند نیز آزمایش نمود.

ج.۱ محک‌های غیرمقید

مسئله، مینیمم نمودن $f(x)$ بر روی تمامی xها می‌باشد. از x^* برای نشان دادن مقدار بهینه‌ساز x و از $f(x^*)$ برای نشان دادن مقدار مینیمم $f(x)$ استفاده می‌نماییم:

$$x^* = argmin\, f(x) \qquad (ج.۱)$$

بسیاری از محک‌هایی که در این بخش معرفی می‌نماییم از [باک، ۱۹۹٦]، [کای و وانگ، ۲۰۰٦] و [یائو و همکاران، ۱۹۹۹] گرفته شده‌اند. اطلاعات دقیق در مورد محک‌های غیرمقید و ماتریس‌های تکامل برای مسابقات الگوریتم تکاملی در کنگره‌ی محاسبات تکاملی IEEE 2005 را می‌توان در [سوگانتان و همکاران، ۲۰۰۵] یافت. همچنین، [علی و همکاران، ۲۰۰۵] نیز شامل بسیاری محک‌های غیرمقید می‌شود. [فلوداس و

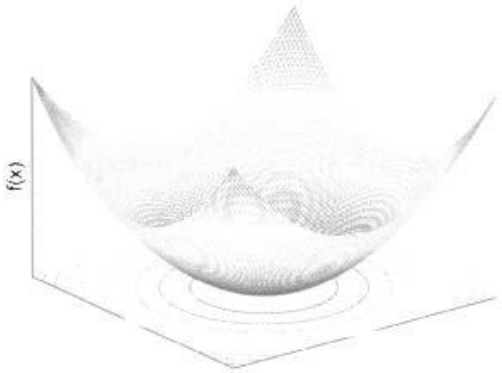
همکاران، ۲۰۱۰] یک کتاب کامل، مختص تعاریف محک‌های بهینه‌سازی غیرمقید می‌باشد. محک‌هایی که در این بخش معرفی شده‌اند به توابعی که بتوان آن‌ها را بر روی هر تعداد بعد n تعریف نمود، محدود شده است. تعداد بسیاری محک‌های دیگر، از جمله محک‌هایی که دارای تعداد مشخصی بعد هستند، ارائه شده‌اند. اما ما بر این باور بودیم که آزمودن الگوریتم‌های بهینه‌سازی بر روی محک‌هایی که دارای بعد متغیر هستند جالب‌تر است چرا که در این صورت می‌توان عملکرد را بر اساس تعداد بعدها مورد کاوش قرار داد. همچنین، دامنه‌هایی که در زیربخش‌های آتی مشخص نموده‌ایم، دامنه‌های معمول هستند، اما محققین از دامنه‌های دیگری نیز استفاده نموده‌اند.

ج.۱-۱ تابع کروی

تابع کروی به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-5.12. +5.12]$ تعریف می‌گردد:

$$f(x) = \sum_{i=1}^{n} x_i^2 \qquad (ج.۲)$$
$$x^* = 0$$
$$f(x^*) = 0$$

معادله در رساله‌ی دکترای کن دِجونگ تابع ۱ بوده [دِجونگ، ۱۹۷۵] و در [اشوفل، ۱۹۹۵] مسئله‌ی ۱-۱ و ۲-۱۷ می‌باشد. شکل ج.۱ نمودار $f(x)$ را در دو بعد نشان می‌دهد. این یک مسئله‌ی بهینه‌سازی بسیار ساده بوده و تقریباً هر الگوریتم منطقی خواهد توانست مینیمم آن را بیابد. با این حال، این تابع آزمون اولیه‌ی خوبی برای آزمودن الگوریتم‌های بهینه‌سازی می‌باشد. این تابع همچنین محک خوبی را برای مقایسه‌ی میان الگوریتم‌ها فراهم می‌آورد چرا که بسیاری از مسائل بهینه‌سازی حول مقدار مینیمم‌شان تقریباً از درجه‌ی دوم می‌باشند.



شکل ج.۱ تابع دو بعدی *کروی*.

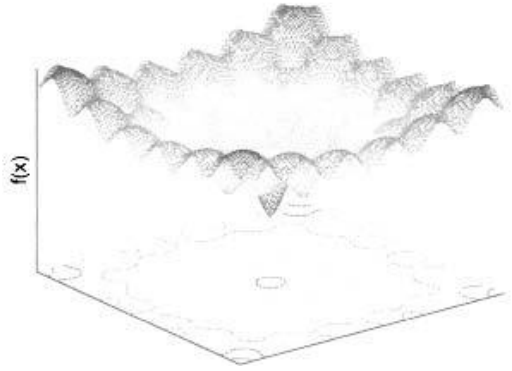
ج.۲-۱ تابع اَکلی

تابع *اَکلی* به‌صورت زیر نوشته شده و بر روی دامنه‌ی $x_i \in [-30. +30]$ تعریف می‌گردد:

$$f(x) = 20 + e - 20\exp\left(-0.2\sum_{i=1}^{n}\frac{x_i^2}{n}\right) - \exp(\sum_{i=1}^{n}(\cos 2\pi x_i)/n) \qquad (ج.۳)$$

$$x^* = 0$$
$$f(x^*) = 0$$

این محک در [اکلی، ۱۹۸۷b] ارائه شده است. شکل ج.۲ نموداری از $f(x)$ در دو بعد را نشان می‌دهد. تعداد زیاد مینیمم‌های محلی این تابع را به یک چالش برای الگوریتم‌های بهینه‌سازی بدل می‌سازد.



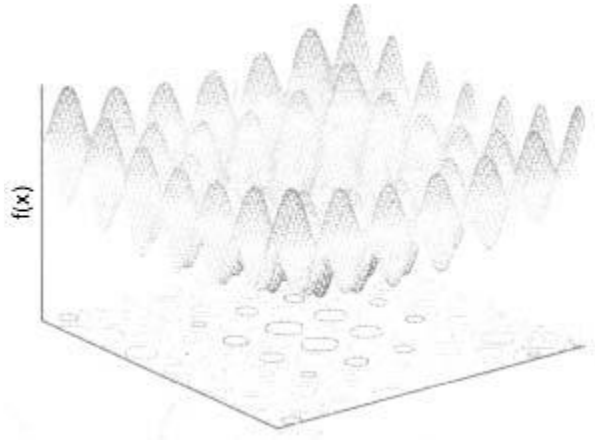
شکل ج.۲ تابع دو بعدی *اَکلی*.

ج.۳-۱ تابع آزمون اَکلی

تابع آزمون *اَکلی* به‌صورت زیر نوشته شده و بر روی دامنه‌ی $x_i \in [-30. +30]$ تعریف می‌گردد:

$$f(x) = \sum_{i=1}^{n-1} 3(\cos(2x_i) + \sin(2x_{i+1})) + \exp(-0.2)\sqrt{x_i^2 + x_{i+1}^2} \qquad (ج.٤)$$

توجه داشته باشید که x^* و $f(x^*)$ برای ان مسئله شناخته شده نیستند. این محک با قله‌ها و دره‌های بسیارش شبیه تابع *اَکلی* می‌باشد. این محک در شکل ج.۳ نشان داده شده است.



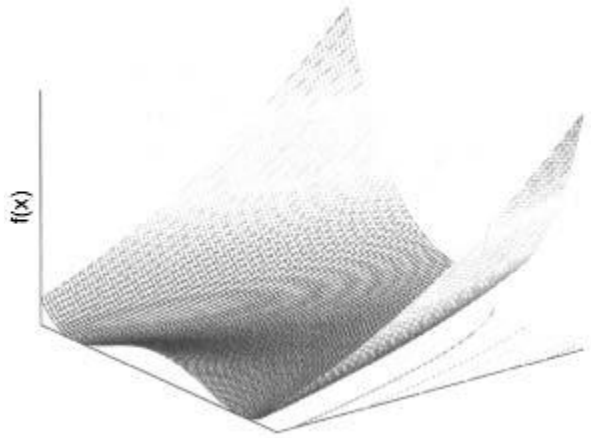
شکل ج.۳ تابع آزمون *آکلی* دو بعدی.

ج.٤-١ تابع روزنبروک

تابع *روزنبروک* به‌صورت زیر نوشته شده و بر روی دامنه‌ی $x_i \in [-2.048. +2.048]$ تعریف می‌گردد:

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \qquad (ج.۵)$$
$$x^* = [1, \dots, 1]$$
$$f(x^*) = 0$$

این محک در [روزنبروک، ۱۹٦۰] ارائه شده و در رساله‌ی دکتری دِجونگ تابع ۲ می‌باشد. در [اشوفل، ۱۹۹۵] نیز این تابع در مسائل ۲-٤، ۲-۲٤ و ۲-۲۵ ظاهر شده است. شکل ج.٤ شکل $f(x)$ در دو بعد را نشان می‌دهد. این شکل دارای دره‌ی باریک و موزی شکلی است که آن را به یک چالش برای الگوریتم‌های بهینه‌سازی بدل می‌سازد.



شکل ج.٤ تابع روزنبروک دو بعدی.

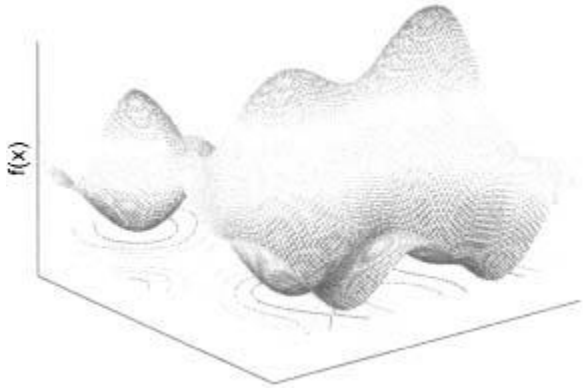
ج.۵-۱ تابع فلچر

تابع فلچر، که با نام فلچر-پاول نیز شناخته می‌شود، به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-\pi. +\pi]$ تعریف می‌گردد:

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{n} (A_i - B_i)^2 \\
A_i &= \sum_{i=1}^{n} (a_{ij} \sin \alpha_j + b_{ij} \cos \alpha_j) \\
B_i &= \sum_{i=1}^{n} (a_{ij} \sin x_j + b_{ij} \cos x_j) \\
\alpha_i &\in [-\pi, \pi], \qquad i \in \{1, \dots, n\} \\
a_{ij}, b_{ij} &\in [-100, 100], \qquad i, j \in \{1, \dots, n\} \\
x^* &= \alpha \\
f(x^*) &= 0
\end{aligned}
\qquad (ج.٦)
$$

این محک در [فلتچر[1] و پوول، ۱۹٦۳] ارائه شده و در [اشوفل، ۱۹۹۵]، مسئله‌ی ۲-۱۳ می‌باشد. شکل ج.۵ نمودار $f(x)$ را در دو بعد و برای مقادیر خاص a_{ij}، b_{ij} و α_i نشان می‌دهد. این تابع از آن جهت جالب است که با هر مقدار a_{ij}، b_{ij} و α_i تغییر می‌کند. مقدار این پارامترها معمولاً با استفاده از یک تولیدکننده‌ی اعداد اتفاقی تعیین می‌شود.

[1] Fletcher



شکل ج.۵ تابع فلچر دو بعدی.

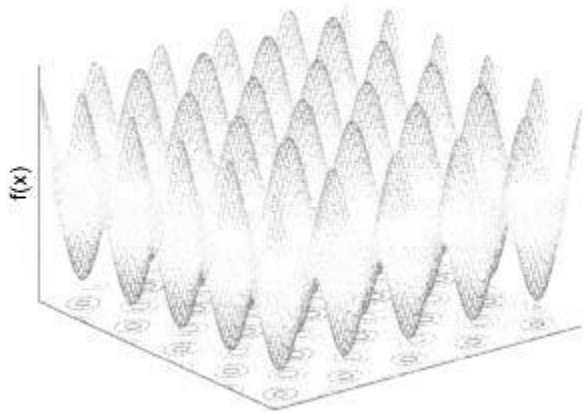
ج.٦-١ تابع Griewank

تابع Griewank به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-600. +600]$ تعریف می‌گردد:

$$f(x) = 1 + \sum_{i=1}^{n} x_i^2/4000 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \qquad (ج.۷)$$

$$x^* = 0$$
$$f(x^*) = 0$$

این محک در [باک و همکاران، ۱۹۹۷a، بخش ب۲-۷] مورد بحث واقع شده است. شکل ج.٦ نمودار دو بعدی $f(x)$ را نشان می‌دهد. این تابع دارای بهینه‌های محلی بسیاری بوده و جمله‌ی ضربی موجود در معادله‌ی $f(x)$ باعث به وجود آمدن مقدار زیادی وابستگی متقابل میان عناصر x می‌شود.



شکل ج.٦ تابع گرینوانک دو بعدی.

ج.۷-۱ تابع مجازات شماره ۱

تابع مجازات شماره ۱ به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-50.+50]$ تعریف می‌گردد:

$$
\begin{aligned}
f(x) = \frac{\pi}{n}\Bigg\{ & 10\sin^2(\pi x_1) \\
& + \sum_{i=1}^{n-1}(x_i-1)^2[1+10sin^2(\pi x_{i+1})] + (x_n-1)^2\Bigg\} \\
& + \sum_{i=1}^{n} u_i \qquad (ج.۸)
\end{aligned}
$$

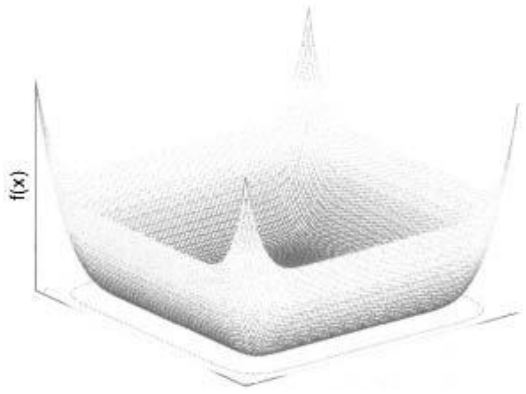
$$
u_i = \begin{cases} k(x_i-a)^m & x_i > a \\ 0 & -a \le x_i \le a \\ k(-x_i-a)^m & x_i < -a \end{cases}
$$

$$
y_i = 1 + \frac{x_i+1}{4}
$$

$$
x^* = [1,\dots,1]
$$

$$
f(x^*) = 0
$$

این محک در [یائو و همکاران، ۱۹۹۹] داده شده است. مقادیر k، m و a داده نشده است اما معمولاً از $k=100$، $m=4$ و $a=10$ استفاده می‌نماییم. شکل ج.۷ نموداری دو بعدی از $f(x)$ را نشان می‌دهد. این تابع دارای تنها یک مینیمم می‌باشد، اما همان‌طور که از شکل نیز پیداست، پیدا نمودن این مینیمم با دقت بالا بسیار دشوار است.



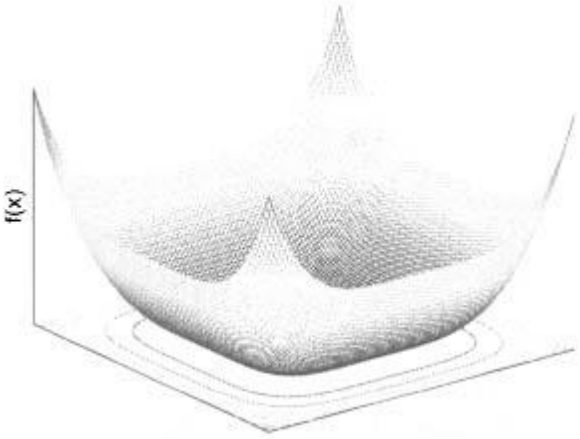
شکل ج.۷ تابع مجازات شماره ۱ دوبعدی.

ج.۸-۱ تابع مجازات شماره ۲

تابع مجازات شماره ۱ به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-50, +50]$ تعریف می‌گردد:

$$f(x) = \sum_{i=1}^{n} u_i + 0.1\{sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + sin^2(3\pi x_{i+1})] + (x_n - 1)^2[1 + sin^2(2\pi x_n)]\} \quad (ج.۹)$$
$$x^* = [1, \dots, 1]$$
$$f(x^*) = 0$$

در معادله‌ی بالا u_i را می‌توان از معادله‌ی (ج.۸) به دست آورد. این محک در [یائو و همکاران، ۱۹۹۹] آورده شده است. همانند تابع مجازات شماره ۱، در اینجا نیز از مقادیر k، m و a داده نشده است اما معمولاً از $k = 100$، $m = 4$ و $a = 5$ استفاده می‌نماییم. شکل ج.۷ نموداری دو بعدی از $f(x)$ را نشان می‌دهد. این تابع دارای تنها یک مینیمم می‌باشد، اما همان‌طور که از شکل نیز پیداست، پیدا نمودن این مینیمم با دقت بالا بسیار دشوار است.



شکل ج.۸ تابع مجازات شماره ۲ دو بعدی.

ج.۹-۱ تابع درجه چهارم

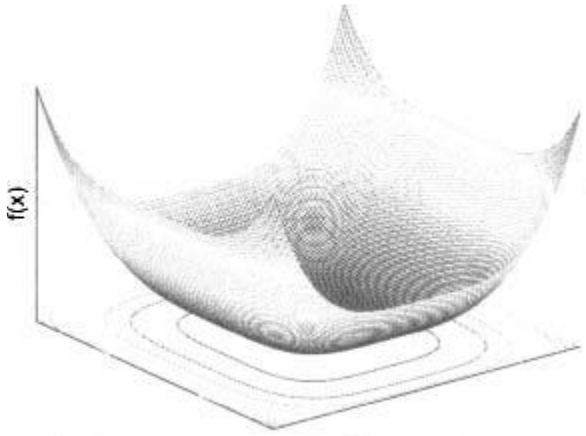
تابع درجه چهارم به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-1.28. +1.28]$ تعریف می‌گردد:

$$f(x) = \sum_{i=1}^{n} i x_i^4 \quad (ج.۱۰)$$
$$x^* = 0$$
$$f(x^*) = 0$$

معمولاً $f(x)$ با نویز همراه است اما این موضوع تأثیری بر مینیمم این تابع نخواهد داشت. این محک در رساله‌ی دکتری دِجونگ تابع ٤ بوده [دِجونگ، ۱۹۷۵] و در [یائو و همکاران، ۱۹۹۹] نیز حضور دارد. اگر در تابع درجه چهارم x_i به توان دو برسد، آنگاه با نام تابع فرا-بیضوی و یا تابع کروی وزنی شناخته خواهد شد [راس و هنسن، ۲۰۰۸]. با این حال، گاهاً تابع فرا-بیضوی به‌صورت زیر نوشته می‌شود:

$$f(x) = \sum_{i=1}^{n} 2^i x_i^2 \qquad (ج.۱۱)$$

شکل ج.۹ نموداری دو بعدی از $f(x)$ را نشان می‌دهد. مانند توابع مجازات، تابع درجه چهارم نیز دارای تنها یک مینیمم است. با این حال این مینیمم نیز به‌گونه‌ای است که تعیین آن با دقت بالا چالش‌برانگیز است.



شکل ج.۹ تابع درجه چهارم در دو بعد.

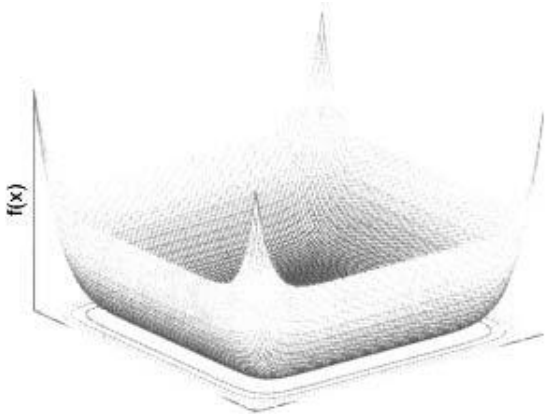
ج.۱۰-۱ تابع درجه دهم

تابع درجه دهم به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-5.12. +5.12]$ تعریف می‌گردد:

$$f(x) = \sum_{i=1}^{n} x_i^{10} \qquad (ج.۱۲)$$
$$x^* = 0$$
$$f(x^*) = 0$$

این محک در [اشوفل، ۱۹۹۵] تحت عنوان مسئله‌ی ۲-۲۳ ارائه شده و در [یائو و همکاران، ۱۹۹۹] نیز وجود دارد. شکل ج.۱۰ نموداری دو بعدی از $f(x)$ را نشان می‌دهد. همانند تابع درجه چهارم و توابع

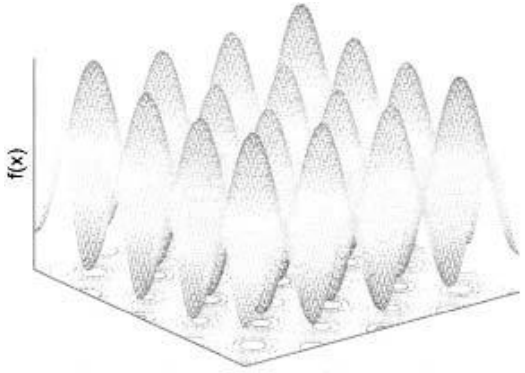
مجازات، این تابع نیز دارای تنها یک مینیمم بوده و این مینیمم به‌گونه‌ای است که تعیین آن با دقت بالا بسیار چالش‌برانگیز است.



شکل ج.۱۰ تابع درجه دهم در دو بعد.

ج.۱۱-۱ تابع رستریجین

تابع رستریجین به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-5.12. +5.12]$ تعریف می‌گردد. این محک در [راستریجین[1]، ۱۹۷٤] ارائه شده و در [یائو و همکاران، ۱۹۹۹] نیز وجود دارد. شکل ج.۱۱ نموداری از $f(x)$ را در دو بعد نشان می‌دهد. تابع رستریجین بسیار مشابه تابع Grienwank است. در تابع رستریجین، تعداد مینیمم‌های محلی به‌صورت نمایی با n افزایش می‌یابد [بیر و اشوفل، ۲۰۰۲].



شکل ج.۱۱ تابع رستریجین در دو بعد.

[1] Rastrigin

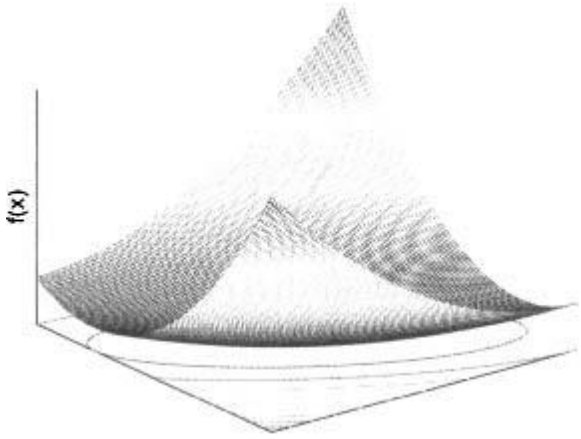
$$f(x) = 10n + \sum_{i=1}^{n} x_i^2 - 10\cos(2\pi x_i) \qquad (ج.۱۳)$$
$$x^* = 0$$
$$f(x^*) = 0$$

ج.۱۲-۱ تابع جمع دوبل اشوفل

تابع جمع دوبل اشوفل، که با نام تابع خط الرأس اشوفل نیز شناخته می‌شود [پرایس و همکاران، ۲۰۰۵]، به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-65.536. +65.536]$ تعریف می‌گردد:

$$f(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2 \qquad (ج.۱٤)$$
$$x^* = 0$$
$$f(x^*) = 0$$

این محک همچنین با تابع فرا-بیضوی چرخشی نیز شناخته می‌شود [راس و هنسن، ۲۰۰۸]. این محک در [اشوفل، ۱۹۹۵] تحت عنوان مسائل ۱-۲ و ۲-۹ ارائه شده است و در [یائو و همکاران، ۱۹۹۹] نیز وجود دارد. این تابع از درجه‌ی دوم بوده و عدد وضعیت آن با n^2 متناسب است. شکل ج.۱۲ نموداری از $f(x)$ را در دو بعد نشان می‌دهد.

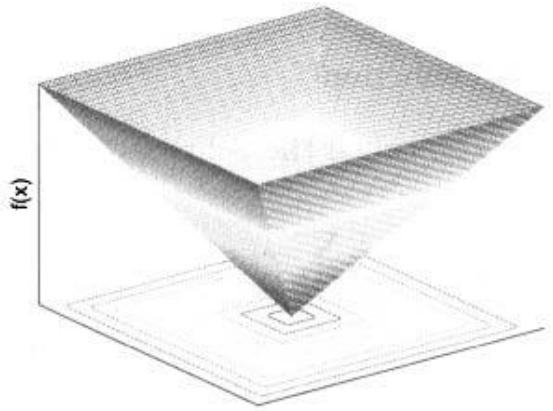


شکل ج.۱۲ تابع جمع دوبل اشوفل در دو بعد.

ج.۱۳-۱ تابع ماکزیمم اشوفل

تابع ماکزیمم اشوفل، که با نام تابع اشوفل2.21 نیز شناخته می‌شود، به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-100. +100]$ تعریف می‌گردد. این محک در [اشوفل، ۱۹۹۵] ارائه شده و در [یائو و همکاران، ۱۹۹۹] نیز وجود دارد. این تابع مشتق‌ناپذیر است. شکل ج.۱۳ نموداری دو بعدی از $f(x)$ را نشان می‌دهد.

$$\begin{aligned} f(x) &= \max_i(|x_i| : i \in \{1, \dots, n\}) \\ x^* &= 0 \\ f(x^*) &= 0 \end{aligned} \qquad (ج.۱۵)$$



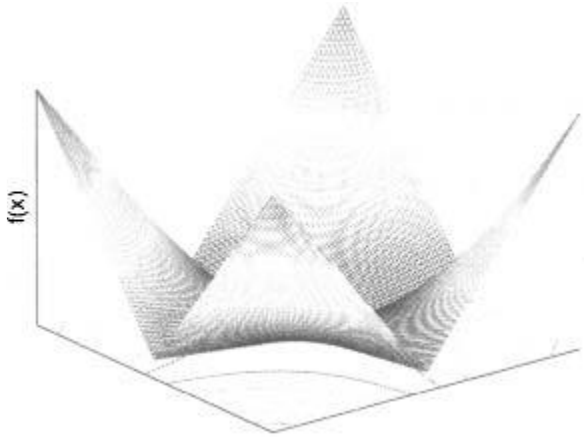
شکل ج.۱۳ تابع ماکزیمم اشوفل در دو بعد.

ج.۱۴-۱ تابع قدر مطلق اشوفل

تابع قدر مطلق اشوفل، که با نام تابع اشوفل 2.22 نیز شناخته می‌شود، به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-10. +10]$ تعریف می‌گردد:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i| \\ x^* &= 0 \\ f(x^*) &= 0 \end{aligned} \qquad (ج.۱۶)$$

این محک در [اشوفل، ۱۹۹۵] در مسئله‌ی ۲-۲۲ ارائه شده است و در [یائو و همکاران، ۱۹۹۹] نیز وجود دارد. این تابع مشتق‌ناپذیر است. شکل ج.۱۴ نموداری دو بعدی از این تابع را نشان می‌دهد.



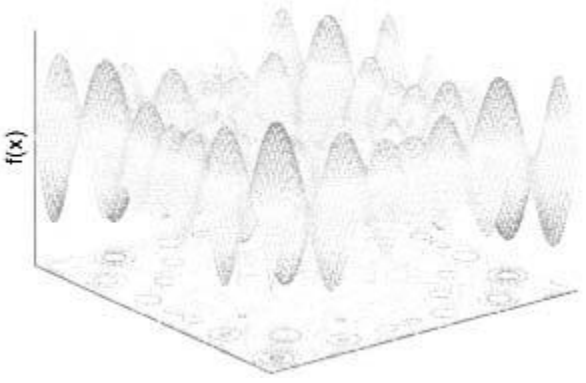
شکل ج.۱٤ تابع قدر مطلق اشوفل در دو بعد.

ج.۱۵-۱ تابع سینوسی اشوفل

تابع سینوسی اشوفل، که با نام تابع اشوفل 2.26 نیز شناخته می‌شود، به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-500. +500]$ تعریف می‌گردد:

$$f(x) = -\sum_{i=1}^{n} x_i \sin\sqrt{|x_i|} \qquad (ج.۱۷)$$
$$x^* = [420.9687, \dots, 420.9867]$$
$$f(x^*) = -12965.5$$

این محک در [اشوفل، ۱۹۹۵] در مسائل ۲-۳ و ۲-۲٦ ارائه شده است و در [یائو و همکاران، ۱۹۹۹] نیز وجود دارد. این تابع دارای مینیمم‌های محلی بسیاری است. شکل ج.۱۵ نموداری دو بعدی از این تابع را نشان می‌دهد.



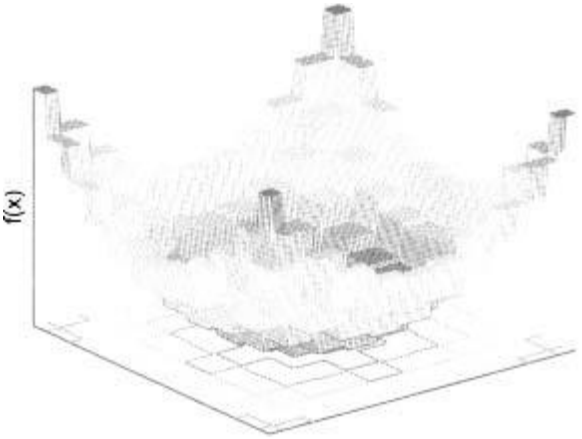
شکل ج.۱۵ تابع سینوسی اشوفل در دو بعد.

ج.۱۶-۱ تابع پله

تابع پله به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-100. +100]$ تعریف می‌گردد:

$$f(x) = \sum_{i=1}^{n} \left(floor(x_i + 0.5)\right)^2 \qquad \text{(ج.۱۸)}$$
$$x^* = 0$$
$$f(x^*) = 0$$

در معادله‌ی بالا تابع floor عدد را به سمت پایین رند می‌کند. این محک در رساله‌ی دکتری دِجونگ تابع ۳ بوده [دِجونگ، ۱۹۷۵] و در [یائو و همکاران، ۱۹۹۹] نیز وجود دارد. این تابع مشتق‌پذیر نبود و دارای سطوح مسطح بسیاری است. شکل ج.۱۶ نمودار دو بعدی $f(x)$ را نشان می‌دهد.



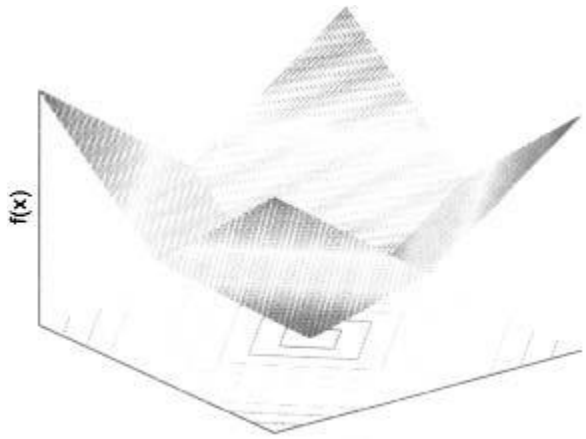
شکل ج.۱۶ تابع پله در دو بعد.

ج.۱۷-۱ تابع قدر مطلق

تابع قدر مطلق به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-10. +10]$ تعریف می‌گردد:

$$f(x) = \sum_{i=1}^{n} |x_i| \qquad \text{(ج.۱۹)}$$
$$x^* = 0$$
$$f(x^*) = 0$$

این محک در [اشوفل، ۱۹۹۵] در مسئله‌ی ۲-۲۰ ارائه شده است. این تابع مشتق‌پذیر نیست. شکل ج.۱۷ نموداری دو بعدی از $f(x)$ را نشان می‌دهد.



شکل ج.۱۷ تابع قدر مطلق در دو بعد.

ج.۱۸-۱ تابع لانه روباه شِکِل[1]

تابع لانه روباه شِکِل به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-65.536. +65.536]$ تعریف می‌گردد:

$$f(x) = \left[\frac{1}{500} + \sum_{j=1}^{2} 5 \frac{1}{j + \sum_{i=1}^{n} (x_i - a_{ij})^6}\right]^{-1} \qquad (ج.۲۰)$$

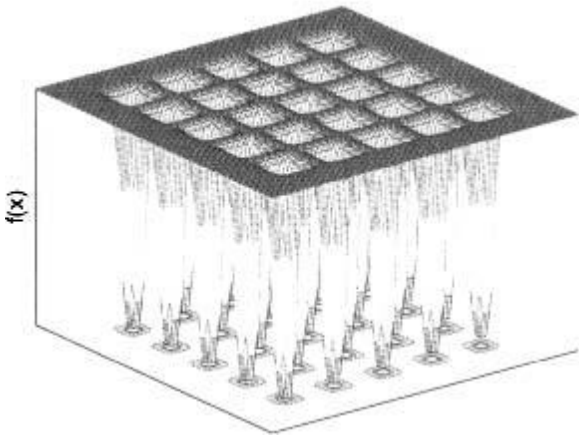
$$x^* = [-32, \dots, -32]$$
$$f(x^*) \approx 1$$

در معادله‌ی بالا a_{ij} عنصر سطر iاُم و ستون jاُم از ماتریس a می‌باشد. در حالتی که تعداد ابعاد برابر $n = 2$ باشد، ماتریس a را می‌توان به‌صورت زیر به دست آورد:

$$a = \begin{bmatrix} b_0 & \dots & b_0 \\ b_1 & \dots & b_5 \end{bmatrix} \qquad (ج.۲۱)$$
$$b_0 = [-32 \quad -16 \quad 0 \quad 16 \quad 32]$$
$$b_i = (16(i-1) - 32)[1 \quad 1 \quad 1 \quad 1 \quad 1]$$

این محک در رساله‌ی دکتری دِجونگ تابع ۵ بوده [دِجونگ، ۱۹۷۵] و در [یائو و همکاران، ۱۹۹۹] نیز وجود دارد. این تابع دارای چندین مینیمم محلی می‌باشد. مقدار همه‌ی این مینیمم‌ها با هم برابر نیست. همان‌طور که در شکل ج.۱۸ نیز واضح است، این تابع دارای شیب تندی به سمت مینیمم‌هایش است.

[1] Shekel



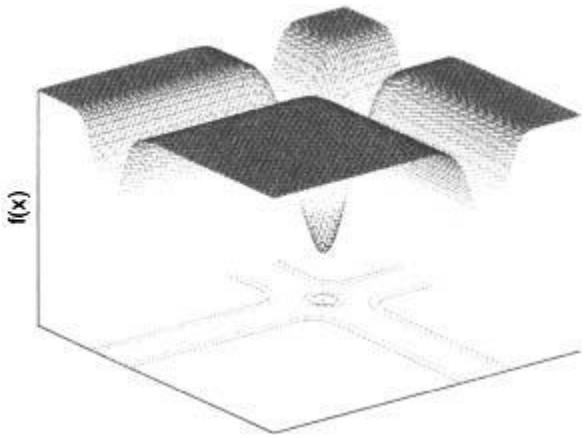
شکل ج.۱۸ تابع لانه روباه شِکِل در دو بعد.

ج.۱۹-۱ تابع مایکلویکز

تابع مایکلویکز به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [0.\pi]$ تعریف می‌گردد:

$$f(x) = -\sum_{i=1}^{n} \sin x_i sin^{\,2m}\left(\frac{ix_i^2}{\pi}\right) \qquad (ج.۲۲)$$

در معادله‌ی بالا m پارامتری است که میزان دشواری جستجو را کنترل می‌نماید. توجه داشته باشید که x^* و $f(x^*)$ برای این مسئله معلوم نیستند. این محک در [مایکلویکز، ۱۹۹۶] آورده شده است. این تابع دارای دره‌هایی باریک و عمیق با افت ناگهانی به سمت مینیمم می‌باشد. نمودار این تابع در شکل ج.۱۹ نشان داده شده است.



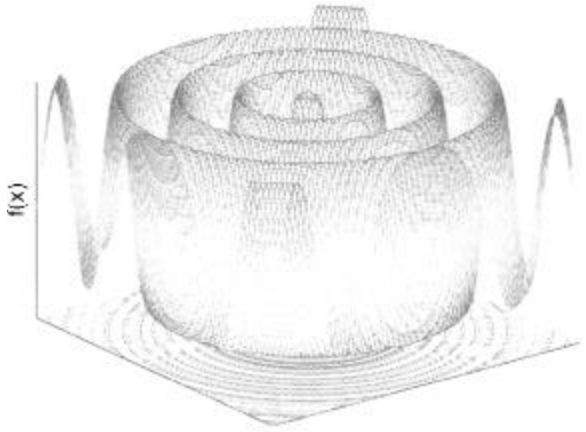
شکل ج.۱۹ تابع مایکلویکز در دو بعد با $m = 10$.

ج.۲۰-۱ تابع پوش سینوسی

تابع پوش سینوسی به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-100. +100]$ تعریف می‌گردد:

$$f(x) = -\sum_{i=1}^{n-1} \frac{sin^2\sqrt{x_i + x_{i+1} - 0.5}}{(0.001(x_i^2 + x_{i+1}^2) + 1)^2} \qquad (ج.۲۳)$$

توجه داشته باشید که x^* و $f(x^*)$ برای این مسئله معلوم نیستند. این محک که در [چنگ و همکاران، ۲۰۰۸] با نام تابع Schaeffer ارائه شده است، دارای دره‌ها و مینیمم‌های محلی بسیاری است. نمودار این تابع در شکل ج.۲۰ نشان داده شده است.



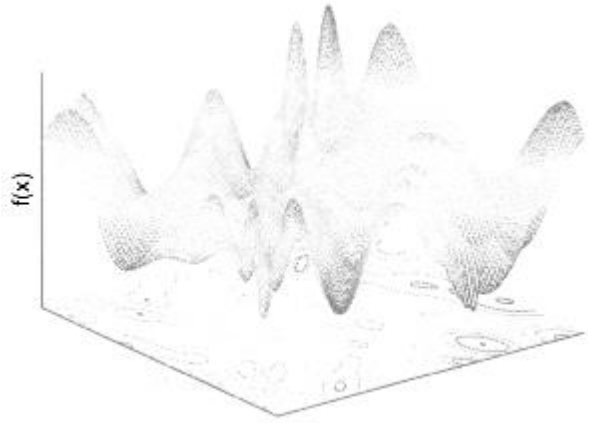
شکل ج.۲۰ تابع پوش سینوسی در دو بعد.

ج.۲۱-۱ تابع سینی تخم مرغ

تابع سینی تخم مرغ به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-512. +512]$ تعریف می‌گردد:

$$f(x) = -\sum_{i=1}^{n-1} (x_{i+1} + 47)\sin\sqrt{\left|x_{i+1} + \frac{x_i}{2} + 47\right|} + x_i \sin\sqrt{|x_i - x_{i+1} - 47|} \qquad (ج.۲٤)$$

توجه داشته باشید که x^* و $f(x^*)$ برای این مسئله معلوم نیستند. این محک در [وو و چو، ۲۰۰۷] آورده شده است. نمودار دو بعدی این تابع در شکل ج.۲۱ نشان داده شده است.



شکل ج.۲۱ تابع سینی تخم مرغ در دو بعد.

ج.۲۲-۱ تابع ویراسترس[1]

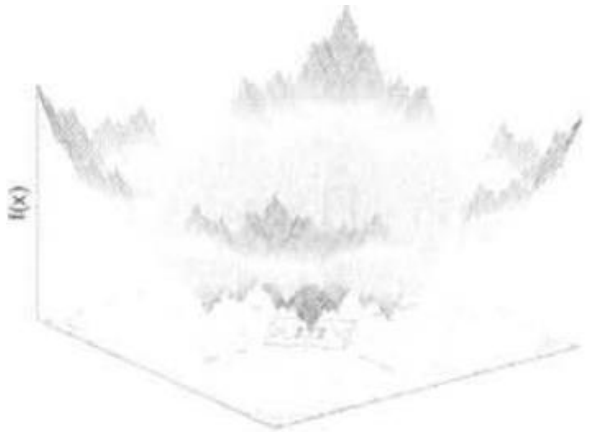
تابع ویراسترس به‌صورت زیر بوده و بر روی دامنه‌ی $x_i \in [-5.+5]$ تعریف می‌گردد:

$$f(x) = \sum_{i=1}^{n} \left\{ \sum_{k=0}^{k_{max}} \left[a^k \cos\left(2\pi b^k (x_i + 0.5)\right) \right] \right\} \qquad (ج.۲۵)$$

$$x^* = 0$$
$$f(x^*) = 0$$

در معادله‌ی بالا $a = 0.5$، $b = 3$ و $k_{max} = 20$ می‌باشد. این محک در [لیانگ و همکاران، ۲۰۰۵] آورده شده است. این تابع دارای خاصیت جالبی است و آن این است که با میل کردن n به سمت بی‌نهایت، تابع در همه‌جا پیوسته بوده ولی در هیچ نقطه‌ای مشتق‌پذیر نخواهد بود. همچنین این تابع در سراسر دامنه‌اش غیریکنوا است. نموداری دو بعدی از این تابع در شکل ج.۲۲ نشان داده شده است.

[1] Weierstrass



شکل ج.۲۲ تابع دوبعدی Weiestrass.

ج.۲ محک‌های مقید

در بهینه‌سازی مقید هدف مینیمم نمودن تابع $f(x)$ بر روی تمام $x \in \mathcal{F} \in \mathcal{R}^n$ می‌باشد. در این صورت $\mathcal{F}$ مجموعه‌ی امکان‌پذیر بوده و n نیز ابعاد مسئله است. از x^* برای نشان دادن مقدار بهینه‌ساز x و از $f(x^*)$ برای نشان دادن مقدار بهینه‌ی مقید $f(x)$ استفاده می‌نماییم:

$$x^* = arg\min_x f(x)$$

به‌طوری که: (ج.۲۶)

$$g_i(x) \leq 0 \text{ برای } i \in [1,m] \text{ و } h_j(x) = 0 \text{ برای } j \in [1,p]$$

این مسئله شامل $(m+p)$ قید می‌باشد. m تعداد قیدهای نامساوی بوده و p نیز تعداد قیدهای مساوی است. بسیاری از مسائلی که از این دست هستند دارای فرم‌هایی پیچیده و طولانی از $f(x)$، $g_i(x)$ و $h_i(x)$ بوده و به همین دلیل تنها برای نوشتن آن‌ها به فضای زیادی نیاز است. بنابراین، در این بخش تنها محک‌های مقید ساده را مورد بحث قرار داد و برای اطلاع از محک‌های پیچیده‌تر مراجعی را معرفی می‌نماییم.

توابع محک مقید را می‌توان در [آرائوجو[1] و همکاران، ۲۰۰۹]، [کوئلو کوئلو، ۲۰۰۰a]، [کوئلو کوئلو، ۲۰۰۲]، [دب، ۲۰۰۰]، [مزورا–مونتس و کوئلو کوئلو، ۲۰۰۵] و [رونارسون و یائو، ۲۰۰۰] یافت. اطلاعات دقیق و کامل در مورد محک‌های مقید و متریک‌های ارزیابی برای مسابقات الگوریتم‌های تکاملی در کنگره‌ی محاسبات تکاملی IEEE 2006 و IEEE 2010 را می‌توان در [لیانگ و همکاران، ۲۰۰۶] و [مالیپدی و

[1] Araujo

سوگانتان، ۲۰۱۰] پیدا نمود. توجه داشته باشید که [فلوداس و پاردالوس، ۱۹۹۰] یک کتاب کامل، مختص تعریف محک‌های بهینه‌سازی مقید می‌باشد. برای اطلاع از محک‌های چندهدفه‌ی مقید نیز می‌توانید به [دب و همکاران، ۲۰۰۱] مراجعه نمایید.

محک‌هایی مقیدی که در این بخش ذکر شده‌اند تماماً از [مالیپدی و سوگانتان، ۲۰۱۰] گرفته شده و همگی در مسابقه‌ی الگوریتم‌های تکاملی مقید که در سال ۲۰۱۰ و در کنگره‌ی محاسبات تکاملی برگزار شد، مورد استفاده واقع شده‌اند. در زیربخش‌های آتی از نماد o_i برای نشان دادن آفستی اتفاقی و از M برای اشاره به یک ماتریس چرخشی اتفاقی استفاده نموده‌ایم (بخش ج.۷ را ببینید).

ج.۱-۲ تابع C01

تابع C01 به‌صورت زیر است:

$$\begin{aligned} f(x) &= -\left|\frac{\sum_{i=1}^{n} cos^4 z_i - 2\prod_{i=1}^{n} cos^2 z_i}{\sum_{i=1}^{n} i z_i^2}\right| \\ g_1(x) &= 0.75 - \prod_{i=1}^{n} z_i \le 0 \\ g_2(x) &= \sum_{i=1}^{n} z_i - 7.5n \le 0 \\ x_i &\in [0,10] \end{aligned} \qquad (ج.۲۷)$$

در معادله‌ی بالا، برای $i \in [1.n]$، $z_i = x_i - o_i$ می‌باشد.

ج.۲-۲ تابع C02

تابع C02 به‌صورت زیر است:

$$\begin{aligned} f(x) &= \max_i z_i \\ g_1(x) &= 10 - \frac{1}{n}\sum_{i=1}^{n}(z_i^2 - 10\cos(2\pi z_i) + 10) \le 0 \\ g_2(x) &= \frac{1}{n}\sum_{i=1}^{n}(z_i^2 - 10\cos(2\pi z_i) + 10) - 15 \le 0 \\ h(x) &= \frac{1}{n}\sum_{i=1}^{n}(y_i^2 - 10\cos(2\pi y_i) + 10) - 20 \le 0 \\ x_i &\in [-5.12, +5.12] \end{aligned} \qquad (ج.۲۸)$$

در معادله‌ی بالا، برای $i \in [1.n]$، $z_i = x_i - o_i$ و $y_i = z_i - 0.5$ می‌باشد.

ج.۳-۲ تابع C03

تابع C03 به‌صورت زیر است:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n-1} [100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2] \\ h(x) &= \sum_{i=1}^{n-1} (z_{\square} - z_{i+1})^2 = 0 \\ x_i &\in [-1000,1000] \end{aligned} \quad (ج.۲۹)$$

در معادله‌ی بالا، برای $i \in [1.n]$، $z_i = x_i - o_i$ می‌باشد.

ج.٤-۲ تابع C04

تابع C04 به‌صورت زیر است:

$$\begin{aligned} f(x) &= \max_i z_i \\ h_1(x) &= \frac{1}{n}\sum_{i=1}^{n} z_i \cos\sqrt{|z_i|} = 0 \\ h_2(x) &= \sum_{i=1}^{\frac{n}{2}-1} (z_i - z_{i+1})^2 = 0 \\ h_3(x) &= \sum_{i=\frac{n}{2}+1}^{n} (z_i^2 - z_{i+1}) = 0 \\ h_4(x) &= \sum_{i=1}^{n} z_i = 0 \\ x_i &\in [-50,50] \end{aligned} \quad (ج.۳۰)$$

در معادله‌ی بالا، برای $i \in [1.n]$، $z_i = x_i - o_i$ می‌باشد.

ج.۵-۲ تابع C05

تابع C05 به‌صورت زیر است:

$$
\begin{aligned}
&f(x) = \max_i z_i \\
&h_1(x) = \frac{1}{n}\sum_{i=1}^{n}[-z_i \sin\left(\sqrt{|z_i|}\right)] = 0 \\
&h_2(x) = \frac{1}{n}\sum_{i=1}^{n}[-z_i \sin\left(0.5\sqrt{|z_i|}\right)] = 0 \\
&x_i \in [-600,600]
\end{aligned}
\qquad (ج.۳۱)
$$

در معادله‌ی بالا، برای $i \in [1.n]$، $z_i = x_i - o_i$ می‌باشد.

ج.۶-۲ تابع C06

تابع C06 به‌صورت زیر است:

$$
\begin{aligned}
&f(x) = \max_i z_i \\
&y_i = (z_i + 483.6106156535)\mathrm{M} - 483.6106156535 \\
&h_1(x) = \frac{1}{n}\sum_{i=1}^{n}[-y_i \sin\left(\sqrt{|z_i|}\right)] = 0 \\
&h_2(x) = \frac{1}{n}\sum_{i=1}^{n}[-y_i \sin\left(0.5\sqrt{|z_i|}\right)] = 0 \\
&x_i \in [-600,600]
\end{aligned}
\qquad (ج.۳۲)
$$

در معادله‌ی بالا، برای $i \in [1.n]$، $z_i = x_i - o_i$ می‌باشد.

ج.۷-۲ تابع C07

تابع C07 به‌صورت زیر است:

$$
\begin{aligned}
&f(x) = \sum_{i=1}^{n-1}[100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2] \\
&g(x) = 0.5 - \exp\left[\frac{0.1}{n}\sum_{i=1}^{n} y_i^2\right] - 3\exp\left[\frac{1}{n}\sum_{i=1}^{n}\cos(0.1y_i)\right] + \exp(1) \le 0 \\
&x_i \in [-140,140]
\end{aligned}
\qquad (ج.۳۳)
$$

در معادله‌ی بالا برای $i \in [1.n]$، $y_i = x_i - o_i$ و $z_i = x_i - o_i + 1$ می‌باشد.

ج.۸-۲ تابع C08

تابع C08 به‌صورت زیر است:

$$f(x) = \sum_{i=1}^{n-1}[100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2]$$
$$g(x) = 0.5 - \exp\left[\frac{0.1}{n}\sum_{i=1}^{n} y_i^2\right] - 3\exp\left[\frac{1}{n}\sum_{i=1}^{n}\cos(0.1y_i)\right] + \exp(1) \le 0 \quad (ج.۳۴)$$
$$x_i \in [-140,140]$$

در معادله‌ی بالا برای $i \in [1.n]$، $y_i = (x_i - o_i)M$ و $z_i = x_i - o_i + 1$ می‌باشد.

ج.۹-۲ تابع C09

تابع C09 به‌صورت زیر است:

$$f(x) = \sum_{i=1}^{n-1}[100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2]$$
$$g(x) = \sum_{i=1}^{n} y\ \sin\sqrt{|y_i|} = 0 \quad (ج.۳۵)$$
$$x_i \in [-500,500]$$

در معادله‌ی بالا برای $i \in [1.n]$، $y_i = x_i - o_i$ و $z_i = x_i - o_i + 1$ می‌باشد.

ج.۱۰-۲ تابع C10

تابع C10 به‌صورت زیر است:

$$f(x) = \sum_{i=1}^{n-1}[100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2]$$
$$g(x) = \sum_{i=1}^{n} y_i\ \sin\sqrt{|y_i|} = 0 \quad (ج.۳۶)$$
$$x_i \in [-500,500]$$

در معادله‌ی بالا برای $i \in [1.n]$، $y_i = (x_i - o_i)M$ و $z_i = x_i - o_i + 1$ می‌باشد.

ج.۱۱-۲ تابع C11

تابع C11 به‌صورت زیر است:

$$f(x) = \frac{1}{n}\sum_{i=1}^{n}\left[-z_i \cos\left(2\sqrt{|z_i|}\right)\right]$$
$$h(x) = \sum_{i=1}^{n-1}[100(y_i^2 - y_{i+1})^2 + (y_i - 1)^2] = 0 \quad (ج.۳۷)$$
$$x_i \in [-100,100]$$

در معادله‌ی بالا برای $i \in [1.n]$، $z_i = (x_i - o_i)M$ و $y_i = x_i - o_i + 1$ می‌باشد.

ج.۱۲-۲ تابع C12

تابع C12 به‌صورت زیر است:

$$f(x) = \sum_{i=1}^{n} z_i sin\sqrt{|z_i|}$$
$$h(x) = \sum_{i=1}^{n}(z_i^2 - z_{i+1})^2 = 0 \quad (ج.۳۸)$$
$$g(x) = \sum_{i=1}^{n-1}[z_i - 100\cos(0.1z_i) + 10] \le 0$$
$$x_i \in [-1000,1000]$$

در معادله‌ی بالا، برای $i \in [1.n]$، $z_i = x_i - o_i$ می‌باشد.

ج.۱۳-۲ تابع C13

تابع C13 به‌صورت زیر است:

$$f(x) = \frac{1}{n}\sum_{i=1}^{n}\left[-z_i sin\sqrt{|z_i|}\right]$$
$$g_1(x) = -50 + \frac{1}{100n}\sum_{i=1}^{n} z_i^2 \le 0$$
$$g_2(x) = \frac{50}{n}\sum_{i=1}^{n-1}\sin\left(\frac{\pi z_i}{50}\right) \le 0 \quad (ج.۳۹)$$
$$g_3(x) = 75 - 50[\sum_{i=1}^{n}\frac{z_i^2}{4000} - \prod_{i=1}^{n}\cos\left(\frac{z_i}{\sqrt{i}}\right) + 1] \le 0$$
$$x_i \in [-1000,1000]$$

در معادله‌ی بالا، برای $i \in [1.n]$، $z_i = x_i - o_i$ می‌باشد.

ج.۱۴-۲ تابع C14

تابع C14 به‌صورت زیر است:

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{n-1} [100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2] \\
g_1(x) &= \sum_{i=1}^{n} \left[-y_i \cos\sqrt{|y_i|}\right] - n \le 0 \\
g_2(x) &= \sum_{i=1}^{n} \left[y_i \cos\sqrt{|y_i|}\right] - n \le 0 \\
g_3(x) &= \sum_{i=1}^{n} \left[y_i \sin\sqrt{|y_i|}\right] - 10n \le 0 \\
x_i &\in [-1000,1000]
\end{aligned}
\qquad (ج.۴۰)
$$

در معادله‌ی بالا برای $i \in [1.n]$، $y_i = x_i - o_i$ و $z_i = x_i - o_i + 1$ می‌باشد.

ج.۱۵-۲ تابع C15

تابع C15 به‌صورت زیر است:

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{n-1} [100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2] \\
g_1(x) &= \sum_{i=1}^{n} \left[-y_i \cos\sqrt{|y_i|}\right] - n \le 0 \\
g_2(x) &= \sum_{i=1}^{n} \left[y_i \cos\sqrt{|y_i|}\right] - n \le 0 \\
g_3(x) &= \sum_{i=1}^{n} \left[y_i \sin\sqrt{|y_i|}\right] - 10n \le 0 \\
x_i &\in [-1000,1000]
\end{aligned}
\qquad (ج.۴۱)
$$

در معادله‌ی بالا برای $i \in [1.n]$، $y_i = (x_i - o_i)M$ و $z_i = x_i - o_i + 1$ می‌باشد.

ج.۱۶-۲ تابع C16

تابع C16 به‌صورت زیر است:

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{n} \frac{z_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{z_i}{\sqrt{i}}\right) + 1 \\
g_1(x) &= \sum_{i=1}^{n} [z_i^2 - 100\cos(\pi z_i) + 10] \le 0 \\
g_2(x) &= \prod_{i=1}^{n} z_i \le 0 \\
h_1(x) &= \sum_{i=1}^{n} \left[z_i \sin\sqrt{|z_i|}\right] = 0 \\
h_2(x) &= \sum_{i=1}^{n} \left[z \sin\sqrt{|z_i|}\right] = 0 \\
x_i &\in [-10,10]
\end{aligned}
\qquad (ج.۴۲)
$$

در معادله‌ی بالا، برای $i \in [1.n]$، $z_i = x_i - o_i$ می‌باشد.

ج.۱۷-۲ تابع C17

تابع C17 به‌صورت زیر است:

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{n-1} (z_i - z_{i+1})^2 \\
g_1(x) &= \prod_{i=1}^{n} z_i \le 0 \\
g_2(x) &= \sum_{i=1}^{n} z_i \le 0 \\
h(x) &= \sum_{i=1}^{n} z_i \sin\left(4\sqrt{|z_i|}\right) = 0 \\
x_i &\in [-10,10]
\end{aligned}
\qquad (ج.۴۳)
$$

در معادله‌ی بالا، برای $i \in [1.n]$، $z_i = x_i - o_i$ می‌باشد.

ج.۱۸-۲ تابع C18

تابع C18 به‌صورت زیر است:

$$f(x) = \sum_{i=1}^{n-1}(z_i - z_{i+1})^2$$
$$g(x) = \frac{1}{n}\sum_{i=1}^{n}[-z_i \sin\left(\sqrt{|z_i|}\right)] = 0 \qquad (ج.٤٤)$$
$$h(x) = \frac{1}{n}\sum_{i=1}^{n}[z_i \sin\left(\sqrt{|z_i|}\right)] = 0$$
$$x_i \in [-50,50]$$

در معادله‌ی بالا، برای $i \in [1.n]$ ،$z_i = x_i - o_i$ می‌باشد.

ج.۱۹-۲ خلاصه‌ی محک‌های مقید

در اینجا، خلاصه‌ای از محک‌های معرفی شده در این بخش را ارائه می‌دهیم. نسبت تخمینی میان اندازه‌ی مجموعه‌ی امکان‌پذیر و اندازه‌ی فضای جستجو از میزان سختی برآورده کردن قیود حکایت دارد (معادله‌ی (۱۹-۵۳) را ببینید). جدول ج.۱ خلاصه‌ای از ۱۸ محک مقید را نشان می‌دهد.

جدول ج.۱ خلاصه‌ای از ۱۸ محک بهینه‌سازی مقید. N_e تعداد قیود مساوی را نشان داده و N_i تعداد قیود نامساوی را نشان می‌دهد. ρ نیز نسبت میان اندازه‌ی مجموعه‌ی امکان‌پذیر و اندازه‌ی فضای جستجو در نسخه‌های ۱۰ و ۳۰ بعدی از هر مسئله می‌باشد.

$\rho(n=30)$	$\rho(n=10)$	N_i	N_e	تابع
1.000000	0.997689	2	0	C01
0.000000	0.000000	2	1	C02
0.000000	0.000000	0	1	C03
0.000000	0.000000	0	4	C04
0.000000	0.000000	0	2	C05
0.000000	0.000000	0	2	C06
0.503725	0.505123	1	0	C07
0.375278	0.379512	1	0	C08
0.000000	0.000000	0	1	C09
0.000000	0.000000	0	1	C10
0.000000	0.000000	0	1	C11
0.000000	0.000000	1	1	C12
0.000000	0.000000	3	0	C13

0.006123	0.000000	3	0	C14
0.006023	0.003210	3	0	C15
0.000000	0.000000	2	2	C16
0.000000	0.000000	2	1	C17
0.000000	0.000000	1	1	C18

ج.۳ محک‌های چندهدفه

در یک مسئله‌ی بهینه‌سازی چندهدفه (MOP)، هدف مینیمم نمودن تابع $f(x)$ بر روی تمام xهاست، به‌گونه‌ای که $f(x)$ یک بردار و x نیز یک بردار تصمیم n بعدی است. مینیمم‌سازی بردار موضوعی تعریف نشده است و به همین دلیل در بخش ۲۰-۱ مجموعه‌ی پرتو P_s و مرز پرتو P_f را تعریف نمودیم. در این صورت می‌توان MOP را به‌صورت پیدا نمودن بهترین P_s و P_f ممکن تعریف نمود. همان‌طور که در بخش ۲۰-۲ نیز بحث نمودیم، "بهترین" را می‌توان به چند روش مختلف تعریف کرد.

اطلاعات دقیق و کامل در مورد محک‌های چندهدفه و متریک‌های ارزیابی برای مسابقات الگوریتم‌های تکاملی در کنگره‌ی محاسبات تکاملی IEEE 2007 و IEEE 2009 را می‌توان در [هوآنگ و همکاران، ۲۰۰۷] و [ژانگ و همکاران، ۲۰۰۹] یافت. همچنین، برای دستیابی به مسائل محک چندهدفه‌ی بیشتر می‌توانید به [زیتزلر و همکاران، ۲۰۰۰] مراجعه نمایید. محک‌های چندهدفه‌ی مقید را در [دب و همکاران، ۲۰۰۱] خواهید یافت. همچنین، [دب و همکاران، ۲۰۰۲b] و [ژانگ و همکاران، ۲۰۰۹] رویکردهای طراحی مسائل آزمون چندهدفه را ارائه می‌دهند. محک‌های چندهدفه‌ی بسیاری در ادبیات مربوطه وجود دارد و همه روزه نیز محک‌های جدید بسیاری ارائه می‌شوند. در این بخش تنها MOPهای غیرمقید از کنگره‌ی محاسبات تکاملی سال ۲۰۰۹ را ارائه می‌دهیم [ژانگ و همکاران، ۲۰۰۹]. در صورت تمایل، خواننده می‌تواند محک‌های چندهدفه‌ی بیشتری را از مراجع بالا مطالعه نماید. ابعاد متغیر مستقل در محک‌هایی که در زیربخش‌های آتی معرفی می‌نماییم متغیر است. اما در کنگره‌ی محاسبات تکاملی در سال ۲۰۰۹ از $n = 30$ استفاده شده بود.

ج.۱-۳ مسئله‌ی اول بهینه‌سازی چندهدفه‌ی نامقید

این مسئله‌ی دو هدفه به‌صورت زیر تعریف می‌گردد:

$$f_1(x) = x_1 + \frac{2}{|J_1|}\sum_{j\in J_1}\left[x_j - \sin\left(6\pi x_1 + \frac{j\pi}{n}\right)\right]^2$$
$$f_2(x) = 1 - \sqrt{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2}\left[x_j - \sin\left(6\pi x_1 + \frac{j\pi}{n}\right)\right]^2 \qquad (ج.۴۵)$$

در معادله‌ی بالا J_1 و J_2 به‌صورت زیر تعریف می‌گردند:

$$J_1 = \{j \in [2,n] : j \text{ فرد}\}$$
$$J_1 = \{j \in [2,n] : j \text{ زوج}\} \quad (ج.٤٦)$$

در این مسئله فضا جستجو برابرست با:

$$x_1 = [0,1]$$
$$x_j = [-1,1] \text{ برای } j \in [2,n] \quad (ج.٤٧)$$

مرز پرتو برابرست با:

$$f_1^* \in [0,1]$$
$$f_2^* = 1 - \sqrt{f_1^*} \quad (ج.٤٨)$$

و در نهایت، مجموعه‌ی پرتو برابرست با:

$$x_1^* \in [0,1]$$
$$x_j^* = \sin\left(6\pi x_1 + \frac{j\pi}{n}\right) \text{ برای } j \in [2,n] \quad (ج.٤٩)$$

ج.۲-۳ مسئله‌ی دوم بهینه‌سازی چندهدفه‌ی نامقید

این مسئله‌ی دو هدفه به‌صورت زیر تعریف می‌گردد:

$$f_1 = x_1 + \frac{2}{|J_1|}\sum_{j\in J_1} y_j^2$$
$$f_2 = 1 - \sqrt{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2} y_j^2 \quad (ج.٥٠)$$

در معادله‌ی بالا J_1 و J_2 به همان صورت مسئله‌ی اول تعریف شده و y_j نیز به‌صورت زیر تعریف می‌گردد:

$$y_j = \begin{cases} x_j - \left[0.3x_1^2\cos\left(24\pi x_1 + \frac{4j\pi}{n}\right) + 0.6x_1\right]\cos\left(6\pi x_1 + \frac{j\pi}{n}\right) & \text{اگر } j \in J_1 \\ x_j - \left[0.3x_1^2\cos\left(24\pi x_1 + \frac{4j\pi}{n}\right) + 0.6x_1\right]\sin\left(6\pi x_1 + \frac{j\pi}{n}\right) & \text{اگر } j \in J_2 \end{cases} \quad (ج.٥١)$$

در این مسئله فضای جستجو برابرست با:

$$x_1 = [0,1]$$
$$x_j = [-1,1] \text{ برای } j \in [2,n] \qquad (ج.۵۲)$$

مرز پرتو برابرست با:

$$f_1^* \in [0,1]$$
$$f_2^* = 1 - \sqrt{f_1^*} \qquad (ج.۵۳)$$

و در نهایت، مجموعه‌ی پرتو برابرست با:

$$x_1^* \in [0,1]$$
$$x_j^* = \begin{cases} \left[0.3(x_1^*)^2 \cos\left(24\pi x_1^* + \frac{4j\pi}{n}\right) + 0.6x_1^*\right] \cos\left(6\pi x_1^* + \frac{j\pi}{n}\right) & \text{اگر } j \in J_1 \\ \left[0.3(x_1^*)^2 \cos\left(24\pi x_1^* + \frac{4j\pi}{n}\right) + 0.6x_1^*\right] \csin\left(6\pi x_1^* + \frac{j\pi}{n}\right) & \text{اگر } j \in J_2 \end{cases} \qquad (ج.۵۴)$$

ج.۳-۳ مسئله‌ی سوم بهینه‌سازی چندهدفه‌ی نامقید

این مسئله‌ی دو هدفه به‌صورت زیر تعریف می‌گردد:

$$f_1 = x_1 + \frac{2}{|J_1|}\left[4\sum_{j\in J_1} y_j^2 - 2\prod_{j\in J_1}\cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right]$$
$$f_2(x) = 1 - \sqrt{x_1} + \frac{2}{|J_2|}\left[4\sum_{j\in J_2} y_j^2 - 2\prod_{j\in J_2}\cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right] \qquad (ج.۵۵)$$

در معادله‌ی بالا J_1 و J_2 به همان صورت مسئله‌ی اول تعریف شده و y_j نیز به‌صورت زیر تعریف می‌گردد:

$$y_j = x_j - x_1^{0.5\left[1+\frac{3(j-2)}{n-2}\right]} \text{ برای } j \in [2,n] \qquad (ج.۵۶)$$

در این مسئله فضای جستجو برابرست با:

$$x_j = [-1,1] \text{ برای } j \in [1,n] \qquad (ج.۵۷)$$

مرز پرتو برابرست با:

$$f_1^* \in [0,1]$$
$$f_2^* = 1 - \sqrt{f_1^*} \qquad (ج.۵۸)$$

و در نهایت، مجموعه‌ی پرتو برابرست با:

$$x_1^* \in [0,1]$$
$$x_j^* = (x_1^*)^{0.5\left[1+\frac{3(j-2)}{n-2}\right]} \text{ برای } j \in [2,n] \quad (ج.۵۹)$$

ج.٤-۳ مسئله‌ی چهارم بهینه‌سازی چندهدفه‌ی نامقید

این مسئله‌ی دو هدفه به‌صورت زیر تعریف می‌گردد:

$$f_1 = x_1 + \frac{2}{|J_1|}\sum_{j\in J_1} h(y_j) \quad (ج.۶۰)$$

$$f_2 = 1 - \sqrt{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2} h(y_j) \quad (ج.۶۱)$$

در معادله‌ی بالا J_1 و J_2 به همان صورت مسئله‌ی اول تعریف شده و y_j نیز به‌صورت زیر تعریف می‌گردد:

$$y_j = x_j - sin\left(6\pi x_1 + \frac{j\pi}{n}\right) \text{ برای } j \in [2,n] \quad (ج.۶۲)$$

همچنین، $h(.)$ به‌صورت زیر تعریف می‌گردد:

$$h(t) = \frac{|t|}{1+e^{2|t|}} \quad (ج.۶۳)$$

در این مسئله فضای جستجو برابرست با:

$$x_1 \in [0,1]$$
$$x_j \in [-2,2] \text{ برای } j \in [2,n] \quad (ج.۶۴)$$

مرز پرتو برابرست با:

$$f_1^* \in [0,1]$$
$$f_2^* = 1 - (f_1^*)^2 \quad (ج.۶۵)$$

و در نهایت، مجموعه‌ی پرتو برابرست با:

$$x_1^* \in [0,1]$$
$$x_j^* = \sin\left(6\pi x_1^* + \frac{j\pi}{n}\right) \text{ برای } j \in [2,n] \quad (ج.۶۶)$$

ج.۵-۳ مسئله‌ی پنجم بهینه‌سازی چندهدفه‌ی نامقید

این مسئله‌ی دو هدفه به‌صورت زیر تعریف می‌گردد:

$$f_1 = x_1 + \left(\frac{1}{2N} + \epsilon\right)|\sin(2N\pi x_1)| + \frac{2}{|J_1|}\sum_{j\in J_1} h(y_j) \quad (ج.۶۷)$$

$$f_2 = 1 - x_1 + \left(\frac{1}{2N} + \epsilon\right)|\sin(2N\pi x_1)| + \frac{2}{|J_2|}\sum_{j\in J_2} h(y_j) \quad (ج.۶۸)$$

در معادله‌ی بالا J_1 و J_2 به همان صورت مسئله‌ی اول تعریف شده، N یک عدد صحیح بوده (در کنگره‌ی محاسبات تکاملی $N = 10$ بوده است) و ϵ یک عدد حقیقی مثبت می‌باشد (در کنگره‌ی محاسبات تکاملی $\epsilon = 0.5$ بوده است). همچنین y_j به‌صورت زیر تعریف می‌گردد:

$$y_j = x_j - sin\left(6\pi x_1 + \frac{j\pi}{n}\right) \text{ برای } j \in [2, n] \quad (ج.۶۹)$$

$h(.)$ نیز به‌صورت زیر تعریف می‌گردد:

$$h(t) = 2t^2 - \cos(4\pi t) + 1 \quad (ج.۷۰)$$

در این مسئله فضای جستجو برابرست با:

$$x_1 \in [0,1]$$
$$x_j \in [-1,1] \text{ برای } j \in [2, n] \quad (ج.۷۱)$$

مرز پرتو دارای $(2N + 1)$ نقطه‌ی گسسته می‌باشد:

$$(f_{1i}^*, f_{2i}^*) = \left(\frac{i}{2N}, 1 - \frac{i}{2N}\right) \text{ برای } i \in [1, 2N + 1] \quad (ج.۷۲)$$

مجموعه‌ی پرتو نیز دارای $(2N + 1)$ نقطه‌ی گسسته است، اما به دلیل اینکه نمی‌توان این نقاط را به‌صورت تحلیلی بیان نمود، آن‌ها را در اینجا نشان نداده‌ایم.

ج.۶-۳ مسئله‌ی ششم بهینه‌سازی چندهدفه‌ی نامقید

این مسئله‌ی دو هدفه به‌صورت زیر تعریف می‌گردد:

$$f_1 = x_1 + \max\left\{0, 2\left(\frac{1}{2N} + \epsilon\right) sin(2N\pi x_1)\right\} + z_1$$
$$f_2 = 1 - x_1 + \max\left\{0, 2\left(\frac{1}{2N} + \epsilon\right) sin(2N\pi x_1)\right\} + z_2 \quad (ج.۷۳)$$

N یک عدد صحیح بوده (در کنگره‌ی محاسبات تکاملی $N = 2$ بوده است) و ϵ یک عدد حقیقی مثبت می‌باشد (در کنگره‌ی محاسبات تکاملی $\epsilon = 0.1$ بوده است). همچنین z_i به‌صورت زیر تعریف می‌گردد:

$$z_i = \frac{2}{|J_i|}\left(4\sum_{j\in J_i} y_j^2 - 2\prod_{j\in J_i}\cos\left(\frac{20y_j\pi}{\sqrt{j}}\right)\right) \quad \text{برای } i \in [1,2] \tag{ج.۷۴}$$

در معادله‌ی بالا J_1 و J_2 به همان صورت مسئله‌ی اول تعریف شده و y_j به‌صورت زیر تعریف می‌گردد:

$$y_j = x_j - sin\left(6\pi x_1 + \frac{j\pi}{n}\right) \quad \text{برای } j \in [2,n] \tag{ج.۷۵}$$

در این مسئله فضای جستجو برابرست با:

$$\begin{aligned} &x_1 \in [0,1] \\ &x_j \in [-1,1] \quad \text{برای } j \in [2,n] \end{aligned} \tag{ج.۷۶}$$

مرز پرتو دارای یک نقطه‌ی گسسته (0.1) بوده و شامل N تکه‌ی جدا از هم می‌شود:

$$\begin{aligned} f_1^* &= \bigcup_{i=1}^{N}\left[\frac{2i-1}{2N}, \frac{2i}{2N}\right] \\ f_2^* &= 1 - f_1^* \end{aligned} \tag{ج.۷۷}$$

مجموعه‌ی پرتو نیز دارای یک نقطه‌ی گسسته می‌باشد، اما به دلیل اینکه نمی‌توان این نقاط را به‌صورت تحلیلی بیان نمود، آن‌ها را در اینجا نشان نداده‌ایم.

ج.۷-۳ مسئله‌ی هفتم بهینه‌سازی چندهدفه‌ی نامقید

این مسئله‌ی دو هدفه به‌صورت زیر تعریف می‌گردد:

$$\begin{aligned} f_1 &= x_1^{1/5} + \frac{2}{|J_1|}\sum_{j\in J_1} y_j^2 \\ f_2 &= 1 - x_1^{1/5} + \frac{2}{|J_1|}\sum_{j\in J_1} y_j^2 \end{aligned} \tag{ج.۷۸}$$

در معادله‌ی بالا J_1 و J_2 به همان صورت مسئله‌ی اول تعریف شده و y_j نیز به‌صورت زیر تعریف می‌گردد:

$$y_j = x_j - \sin\left(6\pi + \frac{j\pi}{n}\right) \quad \text{برای } j \in [2.n] \tag{ج.۷۹}$$

در این مسئله فضای جستجو برابرست با:

$$\begin{aligned} &x_1 \in [0,1] \\ &x_j \in [-1,1] \text{ برای } j \in [2,n] \end{aligned} \quad (ج.۸۰)$$

مرز پرتو برابرست با

$$\begin{aligned} &f_1^* \in [0,1] \\ &f_2^* = 1 - f_1^* \end{aligned} \quad (ج.۸۱)$$

مجموعه‌ی پرتو نیز به‌صورت زیر است:

$$\begin{aligned} &x_1^* \in [0,1] \\ &x_j^* = \sin\left(6\pi x_1 + \frac{j\pi}{n}\right) \text{ برای } j \in [2,n] \end{aligned} \quad (ج.۸۲)$$

ج.۸-۳ مسئله‌ی هشتم بهینه‌سازی چندهدفه‌ی نامقید

این مسئله‌ی سه هدفه به‌صورت زیر تعریف می‌گردد:

$$\begin{aligned} f_1 &= \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|}\sum_{j\in J_1}\left[x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right]^2 \\ f_2 &= \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|}\sum_{j\in J_2}\left[x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right]^2 \\ f_3 &= \sin(0.5x_1\pi) + \frac{2}{|J_3|}\sum_{j\in J_3}\left[x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right]^2 \end{aligned} \quad (ج.۸۳)$$

در معادله بالا J_1، J_2 و J_3 به‌صورت زیر تعریف می‌شوند:

$$\begin{aligned} J_1 &= \left\{j \in [3,n] : j-1 \text{ مضربی از } 3 \text{ است}\right\} \\ J_2 &= \left\{j \in [3,n] : j-2 \text{ مضربی از } 3 \text{ است}\right\} \\ J_3 &= \left\{j \in [3,n] : j \text{ مضربی از } 3 \text{ است}\right\} \end{aligned} \quad (ج.۸۴)$$

در این مسئله فضای جستجو برابرست با:

$$\begin{aligned} &x_1 \in [0,1] \\ &x_2 \in [0,1] \\ &x_j = [-2,2] \text{ برای } j \in [3,n] \end{aligned} \quad (ج.۸۵)$$

مرز پرتو برابرست با:

$$(f_1^*, f_2^*, f_3^*) \text{ به‌طوری که } f_1^* \in [0,1], f_2^* \in [0,1], f_3^* \in [0,1] \text{ و } (f_1^*)^2 + (f_2^*)^2 + (f_3^*)^2 = 1 \quad (ج.۸۶)$$

و در نهایت مجموعه‌ی پرتو برابرست با

$$\begin{aligned} & x_1^* \in [0,1] \\ & x_2^* \in [0,1] \\ & x_j^* = 2x_2^* \sin\left(2\pi x_1^* + \frac{j\pi}{n}\right) \text{ برای } j \in [3,n] \end{aligned} \quad (ج.۸۷)$$

ج.۹-۳ مسئله‌ی نهم بهینه‌سازی چندهدفه‌ی نامقید

این مسئله‌ی سه هدفه به‌صورت زیر تعریف می‌گردد:

$$\begin{aligned} f_1 &= 0.5[\max\{0,(1+\epsilon)(1-4(2x_1-1)^2)\} + 2x_1]x_2 + z_1 \\ f_2 &= 0.5[\max\{0,(1+\epsilon)(1-4(2x_1-1)^2)\} - 2x_1 + 2]x_2 + z_2 \\ f_3 &= 1 - x_2 + \frac{2}{|J_3|}\sum_{j\in J_3}\left[x_j - 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right]^2 \end{aligned} \quad (ج.۸۸)$$

ϵ یک عدد حقیقی مثبت بوده (در کنگره‌ی محاسبات تکاملی سال ۲۰۰۹، $\epsilon = 0.1$ بوده است) و z_i به‌صورت زیر تعریف می‌گردد:

$$z_i = \frac{2}{|J_1|}\sum_{j\in J_1}\left[x_j - 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right]^2 \quad \text{برای } i \in [1,2] \quad (ج.۸۹)$$

مجموعه‌های J_1، J_2 و J_3 همانند MOP نامقید هشتم تعریف می‌شوند. در این مسئله فضای جستجو برابرست با:

$$\begin{aligned} & x_1 \in [0,1] \\ & x_2 \in [0,1] \\ & x_j = [-2,2] \text{ برای } j \in [3,n] \end{aligned} \quad (ج.۹۰)$$

مرز پرتو دارای دو بخش است. اولین بخش به‌صورت زیر خواهد بود:

$$\begin{aligned} & f_3^* \in [0,1] \\ & f_1^* \in [0,(1-f_3)/4] \\ & f_2^* = 1 - f_3^* - f_1^* \end{aligned} \quad (ج.۹۱)$$

بخش دوم نیز به‌صورت زیر خواهد بود:

$$\begin{aligned} &f_3^* \in [0,1] \\ &f_1^* \in [3(1-f_3)/4,1] \\ &f_2^* = 1 - f_3^* - f_1^* \end{aligned} \qquad (ج.۹۲)$$

در نهایت مجموعه پرتو به‌صورت زیر می‌باشد:

$$\begin{aligned} &x_1^* \in [0,0.25] \cup [0.75,1] \\ &x_2^* \in [0,1] \\ &x_j^* = 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{n}\right) \text{ برای } j \in [3,n] \end{aligned} \qquad (ج.۹۳)$$

ج.۱۰-۳ مسئله‌ی دهم بهینه‌سازی چندهدفه‌ی نامقید

این مسئله‌ی سه هدفه به‌صورت زیر تعریف می‌گردد:

$$\begin{aligned} f_1 &= \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|}\sum_{j\in J_1}\left[4y_j^2 - \cos(8\pi y_j) + 1\right] \\ f_2 &= \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|}\sum_{j\in J_2}\left[4y_j^2 - \cos(8\pi y_j) + 1\right] \\ f_3 &= \sin(0.5x_1\pi) + \frac{2}{|J_3|}\sum_{j\in J_3}\left[4y_j^2 - \cos(8\pi y_j) + 1\right] \end{aligned} \qquad (ج.۹٤)$$

مجموعه‌های J_1، J_2 و J_3 همانند MOP نامقید هشتم تعریف شده و y_j به‌صورت زیر تعریف می‌گردد:

$$y_j = x_j - 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{n}\right) \text{ برای } j \in [3,n] \qquad (ج.۹۵)$$

فضای جستجو، مرز پرتو و مجموعه‌ی پرتوی این مسئله مشابه MOP نامقید هشتم می‌باشد.

ج.٤ محک‌های پویا

طی سال‌ها مسائل محک پویای بسیاری توسط محققین ارائه شده است [برانک، ۱۹۹۹]. برخی مسائل پویای مقید در [نگویان[1] و یائو، ۲۰۰۹] و برخی مسائل پویای چندهدفه در [ری و همکاران، ۲۰۰۹a] آورده شده است.در [یانگ، ۲۰۰۸a] نیز تعدادی از محک‌های پویای ترکیبی ذکر شده‌اند. این مسائل شامل مسئله‌ی

[1] Nguyan

کوله‌پشتی پویا و مسئله‌ی فروشنده‌ی دوره‌گرد پویا می‌شوند [برانک و همکاران، ۲۰۰۶]، [ماوروونیوتیس[1] و همکاران، ۲۰۱۱]. اما ما در اینجا بحث خود را به محک‌های پویای پیوسته محدود می‌سازیم.

این بخش خلاصه‌ای از مسائل بهینه‌سازی [لی و همکاران، ۲۰۰۸] به دست می‌دهد. این مرجع شامل محک‌های پیوسته‌ای می‌باشد که در مسابقه‌ی بهینه‌سازی پویا که در سال ۲۰۰۹ و در کنگره‌ی محاسبات تکاملی برگزار شد، مورد استفاده واقع شده‌اند. محک‌های پویا بر پایه‌ی برخی مسائل نامقید از ضمیمه‌ی ج.۱ می‌باشند. محک‌های پویا شامل آفست‌ها و ماتریس‌های چرخشی (ضمیمه‌ی ج.۷ را ببینید)، توابع متغیر با زمان و جمع برخی از این توابع، می‌باشند. در بخش ج.۱-٤ توصیف کاملی از محک‌های پویای کنگره‌ی محاسبات تکاملی سال ۲۰۰۹ را در اختیار خواننده قرار داده و سپس در بخش ج.۲-٤ نسخه‌ی بسیاری ساده‌ای از این نوع محک‌ها را ارائه خواهیم نمود.

ج.۱-٤ توصیف کامل محک پویا

یکی از توابع n بعدی بخش ج.۱ را در نظر بگیرید. ابتدا اندازه‌ی (دامنه‌ی) تابع را نرمالیزه می‌نماییم. این کار را از این جهت انجام می‌دهیم که مطمئن باشیم تابع متغیر با زمانی که بعداً اضافه خواهد شد، تأثیر نسبی مطلوب را داشته باشد. اندازه‌ی محک را به‌صورت زیر نرمالیزه می‌نماییم:

$$f'(x) = \frac{Cf(x)}{fmax} \quad \text{که در آن } C = 2000 \qquad (ج.۹٦)$$

ثابت C در معادله‌ی بالا به‌گونه‌ای انتخاب می‌شود که تمام محک‌های مقیاس شده دارای اندازه‌ی یکسانی بوده تا بدین ترتیب تأثیر عنصر متغیر با زمان، که بعدا اضافه خواهد شد، بر روی تمام محک‌های مقیاس شده یکسان باشد.

حال به بحث در مورد نحوه‌ی تعیین f_{max} در معادله‌ی (ج.۹٦) می‌پردازیم. برای محک‌های پویا معمولاً از تابع پایه‌ای استفاده می‌شود که با افزایش x، افزایش می‌یابد. با اینکه بسیاری از توابع ضمیمه‌ی ج.۱ تعداد زیادی قله و دره‌ی محلی دارند، اما بسیاری از آن‌ها هنگامی به ماکزیمم خود نزدیک هستند که تمام عناصر x در مقدار بیشینه‌ی خود هستند. بنابراین، مقدار f_{max} در معادله‌ی (ج.۹٦) به‌صورت زیر تخمین زده می‌شود:

$$f_{max} \approx f(x_{max}Q) \qquad (ج.۹۷)$$

Q ماتریس چرخش بوده و کمی جلوتر در مورد آن سخن خواهیم راند. x_{max} نیز به‌صورت عنصر-به-عنصر تعریف می‌گردد:

[1] Mavrovouniotis

$$x_i \in [x_{i,min}, x_{i,max}] \text{ که در آن } x = [x_1 \dots x_n]$$
$$\Longrightarrow x_{max} = [x_{1,max} \dots x_{n,max}] \quad (ج.۹۸)$$

سپس $f'(x)$ را شیفت داده و $f'(x-\theta)$ را به دست می‌آوریم. θ یک بردار گرایش n عنصری اتفاقی است. هر عنصر بردار گرایش دارای توزیع یکنواختی است که باعث می‌شود بهینه‌ی $f'(x-\theta)$ دارای توزیع یکنواخت بر روی دامنه‌ی x باشد. برای نمونه، فرض کنید که از تابع آکلی به‌عنوان تابع پایه استفاده نماییم. دامنه‌ی تابع آکلی برای $i \in [1.n]$ برابر $x_i \in [-30.30]$ می‌باشد. مقدار بهینه‌ی تابع آکلی بدون گرایش برای $i \in [1.n]$، $x_i^* = 0$ می‌باشد. بنابراین، هر عنصر θ باید برای همه‌ی iها دارای توزیع یکنواخت بر روی $[-30.30]$ باشد. همانظور که در بخش ج.۱-۷ نیز بحث شده است، این کار باعث می‌شود مقایسه‌ی عادلانه‌ای میان الگوریتم‌های تکاملی صورت بپذیرد.

سپس، محک شیفت یافته و مقیاس شده را می‌چرخانیم تا $f'((x-\theta)Q)$ به دست آید. Q یک ماتریس چرخشی عمودی اتفاقی است. همان‌طور که در بخش ج.۲-۷ نیز بحث شده است، این یکی دیگر از مواردی است که باعث می‌شود مقایسه‌ی عادلانه‌ای میان الگوریتم‌های تکاملی صورت بپذیرد. توجه داشته باشید که از Q در معادله‌ی (ج.۹۷) برای تخمین f_{ma} نیز استفاده شده است.

سپس، تابع متغیر با زمان $\phi(t)$ را اضافه می‌نماییم تا $f'((x-\theta)Q) + \phi(t)$ به دست آید. می‌توان تابع $\phi(t)$ را از یک نسل به نسل دیگر به‌صورت زیر اصلاح نمود:

$$\begin{aligned} &\phi(t) \leftarrow \phi(t-1) + \Delta\phi \\ &\phi(t) \leftarrow \min(\phi(t) + \phi_{max}) \\ &\phi(t) \leftarrow \max(\phi(t) + \phi_{min}) \end{aligned} \quad (ج.۹۹)$$

در معادله‌ی بالا t شماره‌ی دفعه‌ی به‌روزرسانی تابع بوده (این مقدار الزاما شماره نسل الگوریتم تکاملی نمی‌باشد) و ϕ_{min} و ϕ_{max} نیز کمترین و بیشترین مقدار مجاز $\phi(t)$ می‌باشند. مقدار تغییرات $\Delta\phi$ می‌تواند اشکال مختلفی داشته باشد. ابتدا پویایی‌هایی را که در [لی و همکاران، ۲۰۰۸] و [لی و یانگ، ۲۰۰۸] به آن‌ها به‌عنوان پویایی‌های کوچک-گام اشاره شده است، مورد بحث قرار می‌دهیم:

$$\text{گام کوچک: } \Delta\phi = \alpha\phi_{range} r(t-1)\phi_s \quad (ج.۱۰۰)$$

در معادله‌ی بالا α یک ثابت، ϕ_{range} برد مجاز $\phi(t)$، ϕ_s ثابتی برای تعریف شدت تغییر $\phi(t)$ و $r(t-1)$ عددی اتفاقی با توزیع یکنواخت بر روی $[-1.1]$ می‌باشد. [لی و همکاران، ۲۰۰۸] از مقادیر زیر برای این پارامترهای استفاده می‌نماید:

$$\begin{aligned}&\alpha = 0.04\\&\phi_s = 5\\&\phi_{min} = 10\\&\phi_{max} = 100\\&\phi_{range} = \phi_{max} - \phi_{min}\end{aligned} \qquad (ج.۱۰۱)$$

مقدار اولیه‌ی $\phi(t)$ در $t = 0$ به‌صورت اتفاقی و از یک توزیع یکنواخت میان ϕ_{min} و ϕ_{max} به دست می‌آید. می‌توان از معادلات (ج.۹۹)-(ج.۱۰۰) دید که پویایی‌های کوچک-گام نمی‌توانند تغییری بیش از ۱۸ واحد در هر نسل را برای $\phi(t)$ ایجاد کنند. همچنین، از معادله‌ی (ج.۹۶) می‌بینیم که تقریباً $f'(x) \in [-2000.2000]$ می‌باشد. بنابراین، بیشترین تغییر $\phi(t)$ نسبت به بیشترین مقدار $f'(x)$ برای تغییر کوچک-گام برابر $\frac{18}{2000} = 0.9\%$ می‌باشد.

توجه داشته باشید که معادله‌ی (ج.۹۹) در هر نسل اعمال نمی‌شود و تنها در حلقه‌ی while اجرا می‌گردد. در [لی و همکاران، ۲۰۰۸] پیشنهاد شده است معادله‌ی (ج.۹۹) به ازای هر ۱۰۰۰۰ بار ارزیابی تابع، یکبار اجرا شود. در این مرجع همچنین پیشنهاد شده است کل الگوریتم تکاملی برای ۶۰۰۰۰۰ ارزیابی تابع اجرا شود. به علاوه، هر ۱۰۰۰۰ به ازای هر ۱۰۰۰۰ بار ارزیابی تابع برازندگی، یکبار از ماتریس Q برای چرخاندن بردار θ استفاده می‌شود:

$$\theta(t) \leftarrow \theta(t-1)Q \qquad (ج.۱۰۲)$$

در آخر، m عدد از این توابع مقیاس شده، شیفت داده شده، چرخیده و متغیر با زمان تولید کرده، آن‌ها را به هم اضافه نموده تا یک تابع ترکیبی پویا به دست آید:

$$F(x,t) = \sum_{i=1}^{m} w_i \left[f'\left(\left(x - \theta_i(t) \right) Q_i \right) + \phi_i(t) \right] \qquad (ج.۱۰۳)$$

در معادله‌ی بالا هر w_i یک مقدار وزن‌دهی است و به‌صورت زیر به دست می‌آید:

$$w_i \leftarrow \exp\left[-\left(\frac{\sum_{k=1}^{n}\left(x_k-\theta_{ik}(t)\right)^2}{2n}\right)^{\frac{1}{2}}\right]$$
$$w_{max} \leftarrow \max\{w_i\}$$
$$w_i \leftarrow \begin{cases} w_i & \text{اگر } w_i = w_{max} \\ w_i(1-w_{max}^{10}) & \text{اگر } w_i \neq w_{max} \end{cases} \qquad (ج.۱۰٤)$$
$$w_i \leftarrow \frac{w_i}{\sum_{j=1}^{m} w_j}$$

معادله‌ی بالا برای $i \in [1,m]$ صادق است. توجه داشته باشید که $w_i \in [0,1]$ بوده و هر چه x از θ_i دورتر شود، کوچکتر خواهد شد. در [لی و همکاران، ۲۰۰۸] از $m = 10$ استفاده شده است. هر بردار $\theta_i(t)$ در معادله‌ی (ج.۱۰۳) یک بردار اتفاقی n عنصری بوده که به ازای هر ۱۰۰۰۰ بار ارزیابی تابع، یکبار دچار چرخش می‌گردد. همچنین، $\theta_{ik}(t)$ در معادله‌ی (ج.۱۰٤) معرف kاُمین عنصر $\theta_i(t)$ می‌باشد. هر Q_i یک ماتریس چرخش اتفاقی $n \times n$ نامتغیر با زمان بوده و هر تابع $\phi_i(t)$ نیز تابعی اتفاقی و اسکالر بوده و مقدار آن از معادله‌ی (ج.۱۰۰) به دست می‌آید. این تابع به ازای هر ۱۰۰۰۰ بار ارزیابی تابع، یکبار به‌روزرسانی می‌شود. هر یک از m تابعی که در معادله‌ی (ج.۱۰۳) جمع شده است دارای عنصر متغیر با زمان متفاوتی می‌باشد. بنابراین، هنگامی که این m تابع را با یکدیگر جمع می‌کنیم تابع مرکبی به دست می‌آید که مقدار مینیمم آن از یک نسل به نسل دیگر متفاوت خواهد بود.

با جمع‌بندی نتایجی که در پاراگراف‌های بالا ارائه شد، الگوریتم شکل ج.۲۳ برای ایجاد تابع محک پویا به دست خواهد آمد.

شکل ج.۲۳ تعریف توابع محک پویا برای پویایی گام-کوچک می‌باشد. در [لی و همکاران، ۲۰۰۸] و [لی و یانگ، ۲۰۰۸] شش نوع مختلف پویایی معرفی شده است.

۱. پویایی گام-کوچک که در معادلات (جو۹۹)-(ج.۱۰۱) خلاصه شده است.
۲. پویایی گام-بزرگ که به‌صورت زیر تعریف می‌گردد:

$$\Delta\phi = \phi_{range}\left[\alpha\, sign\left(r(t-1)\right) + (\alpha_{max}-\alpha)r(t-1)\right]\phi_s \qquad \text{گام بزرگ:} \quad (ج.۱۰۵)$$

در معادله‌ی بالا $r(t-1)$ یک عدد اتفاقی با توزیع یکنواخت بر روی $[-1,1]$ می‌باشد. تنها ثابت جدید در معادله‌ی بالا α_{max} می‌باشد. این مقدار در [لی و همکاران، ۲۰۰۸] برابرست با:

$$\alpha_{max} = 0.1 \qquad (ج.۱۰۶)$$

می‌توان از معادلات (ج.۱۰۱)، (ج.۱۰۵) و (ج.۱۰۶) دید که در صورتی که پویایی گام-بزرگ باشد، مقدار $\phi(t)$ بیشتر از ٤٥ واحد در یک نسل تغییر نمی‌کند. همچنین از معادله‌ی (ج.۹٦) واضح است که $f'(x) \in [-2000,2000]$. بنابراین، بیشترین تغییر $\phi(t)$ در یک نسل نسبت به بیشترین مقدار $f'(x)$ برای تغییر گام-بزرگ برابر $\frac{45}{2000} = 2.25\%$ می‌باشد.

مقداردهی اولیه را آغاز کن

تابع پایه از قسمت ۱.ج $= f(.)$

فضای جستجوی n بعدی $= [x_{min}, x_{max}]$

مقدار بهینه‌ساز n بعدی $= x^* = f(x)$

تعداد ارزیابی‌های تابع بین به‌روزرسانی‌های پویا $= E_{update}$

تعداد توابعی که در محک ترکیب می‌شوند $= m$

برای $i = 1$ تا m

یک ماتریس چرخشی اتفاقی مانند Q_i تولید کن (بخش ج.۷-۲ را ببینید)

یک بردار اتفاقی بایاس مانند θ_i تولید کن به‌طوری که $x^* + \theta_i \in [x_{min}, x_{max}]$

i بعدی

$f_{max} = f(x_{max} Q)$

$C = 2000$

تعریف تابع: $f'(x) = Cf(x)/f_{max}$

$\phi(0) \leftarrow U[\phi_{min}, \phi_{max}]$

تعداد ارزیابی‌های تابع $= E \leftarrow 0$

پایان مقداردهی اولیه

زمانی که آماده‌ی ارزیابی تابع محک برای راه‌حل نامزد x می‌شویم

از معادله‌ی (ج.۱۰٤) برای محاسبه‌ی w_i برای $i \in [1,m]$ استفاده کن

$E \leftarrow E + 1$

اگر $(E \bmod E_{update}) = 0$ آنگاه

از معادلات (ج.۹۹) – (ج.۱۰۱) برای به‌روزرسانی $\phi_i(t)$ برای $i \in [1,m]$ استفاده کن

از معادله‌ی (ج.۱۰۲) برای به‌روزرسانی $\theta_i(t)$ برای $i \in [1, m]$ استفاده کن

پایان اگر

از معادله‌ی (ج.۱۰۳) برای ارزیابی راه‌حل نامزد x استفاده کن

ارزیابی محک بعدی

شکل ج.۲۳ تعریف یک تابع پویای n بعدی بر اساس تابع محک استاندارد $f(.)$ با پویایی گام-کوچک. $(E \bmod E_{update})$ باقیمانده‌ی تقسیم دو عدد صحیح E و E_{update} می‌باشد.

۳. پویایی‌های اتفاقی به‌صورت زیر تعریف می‌گردد:

$$\Delta\phi = \phi_s \rho(t-1) \quad \text{اتفاقی:} \qquad (ج.۱۰۷)$$

در معادله‌ی بالا $\rho(t-1)$ عددی اتفاقی است که از یک توزیع گاوسی با واریانس واحد و میانگین صفر گرفته شده است. از آنجایی که یک عدد گاوسی بدون کران است، $\phi(t)$ می‌تواند در یک نسل از مقدار مینیمم به مقدار ماکزیمم تغییر کند. با این حال، در ۹۹٫۷٪ از مواقع تغییرات $\phi(t)$ در حدود 3σ واقع می‌شود. در صورت پویا بودن تغییرات، تغییرات 3σ در $\phi(t)$ در یک نسل نسبت به بیشترین مقدار $f'(x)$ برابرست با $\frac{15}{2000} = 0.75\%$.

٤. پویایی‌های آشوبی به‌صورت زیر تعریف می‌گردند:

$$\phi(t) = A[\phi(t-1) - \phi_{min}][1-l] \quad \text{آشوبی:} \qquad (ج.۱۰۸)$$

در معادله‌ی بالا $r(t-1)$ عددی اتفاقی با توزیع یکنواخت بر روی $[-1.1]$ می‌باشد. تنها ثابت جدید در معادله‌ی بالا A بوده که این ثابت در [لی و همکاران، ۲۰۰۸] به‌صورت زیر تعریف شده است

$$A = 3.67 \qquad (ج.۱۰۹)$$

۵. پویایی‌های بازگشتی به‌صورت زیر تعریف می‌شوند:

$$\phi(t) = \phi_{min} + \frac{\phi_{range}\left[\sin\left(\frac{2\pi(t-1)}{P} + \zeta\right) + 1\right]}{2} \quad \text{بازگشتی:} \qquad (ج.۱۱۰)$$

این‌ها تنها پویایی‌های قطعی هستند که در [لی و همکاران، ۲۰۰۸] تعریف شده‌اند. تنها ثابت‌های جدید در معادله‌ی بالا P (دوره) و ζ (فاز اولیه) می‌باشند. این دو ثابت در [لی و همکاران، ۲۰۰۸] به‌صورت زیر تعریف شده‌اند:

$$P = 12E_{update}$$
$$\zeta = U[0,2\pi] \qquad (ج.۱۱۱)$$

در معادله‌ی بالا E_{update} تعداد ارزیابی‌ها میان به‌روزرسانی پویایی‌ها را نشان داده و $U[0.2\pi]$ نیز عددی اتفاقی با توزیع یکنواخت میان ۰ و 2π می‌باشد.

٦. پویایی‌های بازگشتی نویزی به‌صورت زیر تعریف می‌شوند:

$$\phi(t) = \phi_{min} + \frac{\phi_{range}\left[\sin\left(\frac{2\pi(t-1)}{P}+\zeta\right)+1\right]}{2} \quad \text{بازگشتی نویزی:} \qquad (ج.۱۱۲)$$

در معادله‌ی بالا $\rho(t-1)$ عددی اتفاقی است که از یک توزیع گاوسی با واریانس واحد و میانگین صفر گرفته شده است. تنها ثابت جدید در معادله‌ی بالا ρ_s بوده که شدت پویایی نویزی را تعیین می‌نماید. این ثابت در [لی و همکاران، ۲۰۰۸] به‌صورت زیر تعریف شده است:

$$\rho_s = 0.8 \qquad (ج.۱۱۳)$$

می‌توان شکل ج.۲۳ را جهت پیاده‌سازی هر یک از پویایی‌های بالا اصلاح نمود. برای این کار تنها باید خط "use equation (c.99)-(c.101) to update $\phi_i(t)$ for $i \in [1.m]$" از شکل ج.۲۳ را عوض نماییم.

۱. اگر پویایی گام-کوچک مد نظرمان باشد باید شکل ج.۲۳ را به همان شکلی که هست پیاده‌سازی نماییم.
۲. اگر پویایی گام-بزرگ مد نظرمان باشد باید از معادله‌ی (ج.۱۰۵) برای به‌روزرسانی $\phi_i(t)$ استفاده نماییم.
۳. اگر پویایی اتفاقی مد نظرمان باشد باید از معادله‌ی (ج.۱۰۷) برای به‌روزرسانی $\phi_i(t)$ استفاده نماییم.
٤. اگر پویایی آشوبی مد نظرمان باشد باید از معادله‌ی (ج.۱۰۸) برای به‌روزرسانی $\phi_i(t)$ استفاده نماییم.
۵. اگر پویایی بازگشتی مد نظرمان باشد باید از معادله‌ی (ج.۱۱۰) برای به‌روزرسانی $\phi_i(t)$ استفاده نماییم.
٦. اگر پویایی بازگشتی نویزی مد نظرمان باشد باید از معادله‌ی (ج.۱۱۲) برای به‌روزرسانی $\phi_i(t)$ استفاده نماییم.

در [لی و همکاران، ۲۰۰۸] پنج تابع مختلف برای استفاده به‌عنوان تابع پایه در شکل ج.۲۳ پیشنهاد شده است: تابع کروی (بخش ج.۱-۱)، تابع رستریجین (بخش ج.۱۱-۱)، تابع ویراسترس (بخش ج.۲۲-۱)، تابع

گرینوانک (بخش ج.٦-١) و تابع اَکلی (بخش ج.٢-١). توجه داشته باشید که نسخه‌های اصلی و شیفت نیافته از این توابع دارای راه‌حل بهینه در x^* می‌باشند.

ج.٤-٢ توصیف ساده شده محک پویا

شکل ج.٢٣ نشان می‌دهد چند پویایی متعامل در توابع محک وجود دارد. این پویایی‌ها شامل وزن‌های $\{w_i\}$، که خود تابعی از راه‌حل نامزد هستند، متغیرهای پویای $\phi_i(t)$ و متغیرهای گرایش پویای $\theta_i(t)$، می‌شوند. با این حال، به نظر می‌رسد بتوان ماهیت پویایی‌ها را از متغیرهای گرایش به دست آورد. سایر متغیرها تنها تأثیرات جانبی دارند. به علاوه، برای به دست آوردن تابعی با پویایی بسیار نیازی به اضافه نمودن چندین تابع به یکدیگر نیست. به بیان دیگر، می‌توان در شکل ج.٢٣ از $m = 1$ استفاده نمود و همچنان محک پویای خوبی به دست آورد. این موضوع منجر به شکل ج.٢٤، که یک تولیدکننده‌ی تابع محک پویای مؤثر است، می‌شود.

در آخر، متذکر می‌شویم که می‌توان از روش‌های دیگری غیر از معادله‌ی (ج.١٠٢) برای به‌روزرسانی $\theta(t)$ استفاده نمود. معادله‌ی (ج.١٠٢) شامل چرخش $\theta(t)$ به دور مبدا فضای جستجو می‌باشد. $\theta(t)$ را می‌توان به روش قابل پیش‌بینی دیگری (برای مثال خطی و یا متناوب) تغییر داد. همچنین می‌توان یک $\theta(t)$ اتفاقی در هر بار تغییر پویا ایجاد نمود. در مسائل دنیای واقعی می‌توان از روش‌های مختلفی برای تغییر دادن $\theta(t)$ جهت ارائه نمودن پویایی استفاده نمود.

مقداردهی اولیه را آغاز کن

تابع پایه از بخش ۱.ج $= f(.)$

فضای جستجوی n بعدی $= [x_{min}, x_{max}]$

مقدار بهینه‌ساز n بعدی $x^* = f(x)$

تعداد ارزیابی‌های تابع بین به‌روزرسانی‌های پویا $= E_{update}$

یک ماتریس چرخشی اتفاقی مانند Q تولید کن (بخش ج.۷-۲ را ببینید)

یک بردار اتفاقی بایاس مانند θ تولید کن به‌طوری که $x^* + \theta \in [x_{min}, x_{max}]$

پایان مفداردهی اولیه

زمانی که آماده‌ی ارزیابی تابع محک برای راه‌حل نامزد x می‌شویم

$E \leftarrow E + 1$

اگر $(E \ mod \ E_{update}) = 0$ آنگاه

از معادله‌ی (ج.۱۰۲) برای به‌روزرسانی $\theta(t)$ استفاده کن

پایان اگر

از $F(x,t) = f\left(\left(x - \theta(t)\right)Q\right)$ جهت ارزیابی x استفاده کن

ارزیابی محک بعدی

ج.۲٤ تعریف ساده‌ی تابع برای یک تابع پویای n بعدی بر اساس محک استاندارد $f(.)$.

ج.۵ محک‌های نویزی

مسائل محک نویزی را می‌توان به سادگی تولید نمود. برای این کار کافی است به توابع استاندارد محک نویز اضافه نماییم. می‌توان از نویزهای مختلفی برای این کار استفاده نمود: برخی نویزها دارای شاخصه‌های آماری مسقل از راه‌حل نامزد x می‌باشند (معادله‌ی (۲۱-۳۹) را ببینید)، برخی نویزها به‌گونه‌ای با x تغییر می‌کنند (معادله‌ی (۲۱-٤۲) را ببینید)، نویز گاوسی و یا هر نویز دیگری که مطلوبمان باشد.

ج.٦ مسائل فروشنده دوره‌گرد

وبسایت TSPLIB دارای مجموعه‌ای از بیش از ۱۰۰ محک TSP می‌باشد [رینلت[1]، ۲۰۰۸]. ساده‌ترین نوع TSP در این مجموعه محک Ulysses16 است که بر پایه‌ی ١٦ شهری است که پادشاه افسانه‌ای یونان در طول سفر خود به مدیترانه از آن‌ها گذر کرده بود [گروتشل[2] و پدبرگ[3]، ۲۰۰۱]. بزرگترین TSP این وبسایت یک مسئله‌ی ارائه منطقی قابل برنامه‌نویسی با بیش از ۸۵۹۰۰ گره می‌باشد. مرکز ریاضیات گسسته و علوم کامپیوتر نظری دارای وبسایتی در مورد محک‌های TSP مقیاس بزرگ بوده که بزرگترین آن‌ها دارای بیش از ۲۰ میلیون گره می‌باشد [دمترسکیو[4]، ۲۰۱۲].

هر TSP در یک فایل با پسوند TSP تعریف شده است – برای مثال، ULYSSES.TSP. هر فایل TSP شامل مختصات هر شهر به فرم DD.MM می‌باشد. به این ترتیب DD زاویه و MM نیز دقیقه‌ی زاویه‌ای را معین می‌نماید. فایل TSP همچنین "نوع وزن لبه" مسئله را نیز تعیین می‌کند. این مقدار یا EUC_2D و یا GEO بوده و نحوه‌ی محاسبه فاصله میان شهرها را تعیین می‌نماید.

برای مسائل EUC_2D باید از فاصله‌ی اقلیدسی میان شهرها، که به‌صورت زیر محاسبه می‌شود، استفاده نمود:

$$\begin{aligned} \Delta B &= B_i - B_k \\ \Delta L &= L_i - L_k \\ D(i,k) &= round\sqrt{\Delta B^2 + \Delta L^2} \end{aligned} \qquad (ج.١١٤)$$

در معادله‌ی بالا B_i و L_i به ترتیب عرض و طول جغرافیایی شهر iاُم بوده و تابع round نیز عدد ورودی خود را به نزدیکترین عدد صحیح رند می‌نماید. رندسازی الزامی نیست اما به‌صورت سنتی برای مسائل TSPLIB انجام می‌شود. به همین دلیل بهتر است برای انجام مقایسه‌ی عادلانه میان الگوریتم‌های TSP و نتایجی که پیش از این منتشر شده‌اند، از رندسازی استفاده نماییم.

برای مسائل GEO، باید فاصله‌ی جغرافیایی میان شهرهای i و k را محاسبه نماییم. با فرض این که زمین یک کره‌ی کامل است این فاصله به‌صورت زیر محاسبه خواهد شد:

$$q_1 = \cos(L_i - L_k) \qquad (ج.١١٥)$$

[1] Reinlet
[2] Grotschel
[3] Padberg
[4] Demetrescu

$$D(i,k) = \left\lfloor R \arccos\left\{\frac{[(1+q_1)q_2 - (1-q_1)q_3]}{2}\right\} + 1\right\rfloor$$

در معادله‌ی بالا $R = 6378.388$ شعاع کره‌ی زمین بوده . تابع $\lfloor . \rfloor$ نیز عدد صحیحی را که کوچکتر از آرگومان است، به دست می‌دهد. این تابع و همچنین به علاوه‌ی ۱ که در انتهای محاسبه‌ی $D(i.k)$ وجود دارد الزامی نیست، اما در محک‌های TSPLIB و در محاسبه‌ی فاصله‌ی جغرافیایی استاندارد لحاظ می‌شوند.

استنتاج فاصله‌ی جغرافیایی

معادله‌ی (ج.۱۱۵) را می‌توان با تبدیل عرض و طول جغرافیایی به مختصات کارتزین، استنتاج نمود. بدین ترتیب خواهیم داشت:

$$\begin{aligned} x_i &= \mathrm{Rcos}(B_i)\cos(L_i) \\ y_i &= \mathrm{Rcos}(B_i)\sin(L_i) \\ z_i &= \mathrm{Rsin}(B_i) \end{aligned} \qquad (ج.۱۱۶)$$

به همین ترتیب مختصات مشابهی برای x_k، y_k و z_k، که مختصات kاُمین شهر می‌باشند، به دست خواهد آمد. به خاطر آورید که می‌توان ضرب داخلی دو بردار A و B را به‌صورت زیر تعریف نمود

$$A.B = |A|.|B|cos\theta \qquad (ج.۱۱۷)$$

در معادله‌ی بالا θ زاویه‌ی میان دو بردار است. بنابراین، می‌توان ضرب داخلی میان بردارهای دو شهر i و k را به‌صورت زیر نوشت

$$\begin{bmatrix} \mathrm{Rcos}(B_i)\cos(L_i) \\ \mathrm{Rcos}(B_i)\sin(L_i) \\ \mathrm{Rsin}(B_i) \end{bmatrix} \begin{bmatrix} \mathrm{Rcos}(B_k)\cos(L_k) \\ \mathrm{Rcos}(B_k)\sin(L_k) \\ \mathrm{Rsin}(B_k) \end{bmatrix} = R^2 cos\theta \qquad (ج.۱۱۸)$$

در معادله‌ی بالا θ زاویه‌ی میان دو شهر i و k می‌باشد. با ضرب دو طرف معادله‌ی بالا در R^2 و بسط دادن آن خواهیم داشت

$$cosB_i cosL_i cosB_k cosl_k + cosB_i sinL_i cosB_k sinL_k + sinB_i sinB_k = cos\theta \qquad (ج.۱۱۹)$$

این معادله را می‌توان به‌صورت زیر ساده‌سازی نمود

$$cosB_i cosB_k (cosL_i cosL_k + sinL_i sinL_k) + sinB_i sinB_k = cos\theta \qquad (ج.۱۲۰)$$

می‌توان از تبدیلات مثلثاتی استفاده نمود و معادله‌ی بالا را به‌صورت زیر نوشت

$$\frac{1}{2}[\cos(B_i + B_k) + \cos(B_i - B_k)]\cos(L_i - L_k) + \frac{1}{2}[\cos(B_i - B_k) - \cos(B_i + B_k)] = cos\theta \quad (ج.۱۲۱)$$

با حل معادله‌ی بالا برای θ خواهیم داشت

$$\theta = \arccos\{\frac{1}{2}[q_1(q_2 + q_3) + q_2 - q_3]\} \quad (ج.۱۲۱)$$

دو نقطه بر روی یک کره با شعاع R که با زاویه‌ی θ از یکدیگر جدا شده‌اند، دارای فاصله‌ای برابر $R\theta$ بر روی سطح کره بوده و معادله‌ی (ج.۱۱۵) را به دست می‌دهد.

سایر متریک‌های فاصله

در [رینلت، ۲۰۰۸] متریک‌های فاصله‌ی دیگری نیز وجود دارد. این متریک‌ها شامل فاصله‌ی اقلیدسی سه بعدی، فاصله‌ی منهتن، که در آن فرض بر آن است که مسیرها بر روی یک شبکه‌ی متعامد قرار دارند، فاصله‌ی ماکزیمم، که فاصله را در طول مختصاتی که بیشترین فاصله را شامل می‌شوند اندازه می‌گیرد، فاصله‌ی شبه‌اقلیدسی، که مشابه معادله‌ی (ج.۱۱٤) می‌باشد، و نوع خاصی از تابع فاصله که به بلورشناسی اشعه ایکس مربوط می‌باشد، می‌شوند.

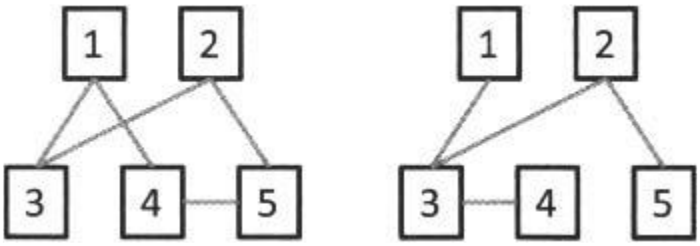
مسائل ترکیبی دیگر

هم TSPهای متقارن و هم نامتقارن را می‌توان در [رینلت، ۲۰۰۸] یافت. وبسایت همچنین شامل انواع مرتبط مسائل می‌شود. از این جمله می‌توان به موارد زیر اشاره نمود.

۱. مسئله‌ی مرتب‌سازی ترتیبی نوعی TSP نامتقارن است که شامل محدودیت تقدم می‌باشد [دوریگو و استاتزل، ۲۰۰٤]. این بدین معنی است که با فرض در اختیار داشتن n شهر، باید کوتاه‌ترین مسیر را به‌گونه‌ای یافت که برای $m \in [1.m]$، شهر i_m قبل از شهر k_m بازدید شود. M تعداد قیدها می‌باشد.

۲. مسئله‌ی مسیریابی وسیله نقلیه شامل $(n-1)$ گره و یک انبار می‌باشد [توث و ویگو، ۲۰۰۲]. در این مسئله باید از کامیون‌ها برای رساندن محموله‌ها از انبار به گره‌ها استفاده نمود. هر گره دارای میزان تقاضای خاصی برای محموله‌ها بوده و کامیون‌ها دارای ظرفیت یکسان می‌باشند. هر مسیر از انبار آغاز شده، از برخی از گره‌ها عبور کرده و در نهایت به انبار باز می‌گردد. تابع هزینه در این مسئله را می‌توان مجموع مسافت کل طی شده و یا مدت زمان کل مورد نیاز برای تحویل محموله‌ها تعریف نمود.

۳. مسئله مسیر همیلتونی شامل پیدا کردن مسیری است که از هر گره تنها یک بار عبور نماید [بالاکریشنان[1]، ۱۹۹۷]. مسئله‌ی چرخه‌ی همیلتونی نیز تنها دارای یک شرط اضافی است و آن این است که مسیر باید به نقطه‌ی آغازین بازگردد. شکل ج.۲۵ نمونه‌ای از یک مسئله‌ی مسیر همیلتونی را نشان می‌دهد. راه‌حل گراف سمت چپ $4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$ می‌باشد. با این حال، گراف سمت چپ مسیر همیلتونی ندارد. در گراف سمت راست می‌توان مسیری را یافت که از همه‌ی گره‌ها یک بار عبور نماید، اما هر مسیری که در این گراف در نظر بگیرید، از برخی گره‌ها بیش از یک بار عبور خواهد نمود (برای مثال، $4 \rightarrow 3 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 5$).



شکل ج.۲۵ دو گراف متصل. گراف سمت چپ دارای مسیر همیلتونی است در حالی که گراف سمت راست مسیر همیلتونی ندارد.

ج.۷ آنباياس نمودن فضای جستجو

در این بخش به بحث مسائل بهینه‌سازی پیوسته باز می‌گردیم. برخی الگوریتم‌های تکاملی به‌صورت طبیعی بر روی برخی توابع محک عملکرد خوبی دارند. علت این موضوع صرفاً آن است که تغییرات آن محک‌ها در راستای تغییرات الگوریتم تکاملی قرار می‌گیرند. این موضوع مبین خوبی برای عملکرد الگوریتم‌های تکاملی نیست و تنها بدین معنی است که خواص بسیاری از محک‌های مصنوعی نماینده‌ی مسائل دنیای واقعی نیستند. این بخش به بحث در مورد استفاده از آفست‌ها و ماتریس‌های چرخش در محک‌های بهینه‌سازی جهت چالشی‌تر و واقع‌گرایانه‌تر نمودن آن‌ها می‌پردازد.

ج.۱-۷ آفست‌ها

برخی الگوریتم‌های تکاملی به‌صورت طبیعی به نوع خاصی از نواحی جستجو گرایش دارند. برای مثال، در بخش ۱۶-۲ مشاهده نمودیم که برخی انواع خاص یادگیری مقابله-محور (OBL) تمایل دارند راه‌حل‌های

[1] Balakrishnan

نامزد را به سمت مرکز فضای جستجو سوق دهند. بنابراین، OBL به‌صورت طبیعی بر روی مسائلی که راه‌حلشان در نزدیکی مرکز فضای جستجو واقع شده است، عملکرد مناسبی دارد. با این حال، این عملکرد خوب OBL گمراه کننده است چرا که این موضوع یکی از تأثیرات جانبی مصنوعی این حقیقت است که راه‌حل بسیاری از توابع محک در نزدیکی مرکز فضای جستجو واقع شده است. یک مثال دیگر، تکاملی دیفرانسیلی (DE) است. این الگوریتم راه‌حل‌های نامزد را بر اساس بردارهای تفاوت اصلاح می‌نماید. بنابراین DE بر روی مسائل مقیدی که نواحی امکان‌پذیرشان به‌صورت موازی با هم قرار دارند، عملکرد خوبی دارد. در اینجا نیز عملکرد خوب DE گمراه کننده است چرا که این عملکرد خوب در واقع اثر جانبی مصنوعی این حقیقت است که بسیاری از توابع محک دارای نواحی امکان‌پذیر موازی می‌باشند. راه‌حل بسیاری از محک‌های ارائه شده در این بخش درست در مرکز فضای جستجو واقع شده است. بنابراین، استفاده از چنین محک‌های برای ارزیابی عملکرد الگوریتم تکاملی عادلانه نیست.

گاهاً، و نه همیشه، می‌توان تشخیص داد که یک الگوریتم دارای گرایش (بایاس) است و هنگامی که راه‌حل در مرکز جستجو واقع شده باشد، الگوریتم آن را راحت‌تر پیدا می‌کند [کلرک، ۲۰۱۲b]. برای مثال، می‌توان الگوریتم را بر روی یک مسئله‌ی دو بعدی با فضای جستجوی $x_1 \in [-1,+1]$ و $x_2 \in [-1,+1]$ و تابع هزینه $f(x) = 1$ برای تمامی xها، اجرا نماییم. حال فرض کنید پس از گذشت تعداد زیادی نسل، نمودار جمعیت را بر روی دامنه‌ی جستجوی دو بعدی رسم نماییم. اگر الگوریتم دارای گرایش نباشد، باید توزیعی یکنواخت به دست آید. با این حال، برای بسیاری از الگوریتم‌ها توزیع در اطراف نقطه‌ی $x = (0,0)$ دارای تمرکز بیشتری است. در چنین حالتی می‌توان نتیجه گرفت که الگوریتم دارای گرایش است. تفاوت تمرکز برای چند الگوریتم مختلف ممکن است قابل تفکیک نباشد، بنابراین تنها راه مطمئن برای ارزیابی الگوریتم‌های تکاملی آن است که هیچگاه از توابع هزینه‌ای که راه‌حلشان در مرکز دامنه‌ی جستجو واقع شده است، استفاده ننماییم.

می‌توان با اضافه نمودن آفست به متغیرهای مستقل مسائل محک، بایاس و گرایش آن‌ها را از بین برد و به اصطلاح آن‌ها را آنبایاس نمود [لیانگ و همکاران، ۲۰۰۵]، [سوگانتان و همکاران، ۲۰۰۵]. تابع کروی از معادله‌ی (ج.۲) را در نظر بگیرید

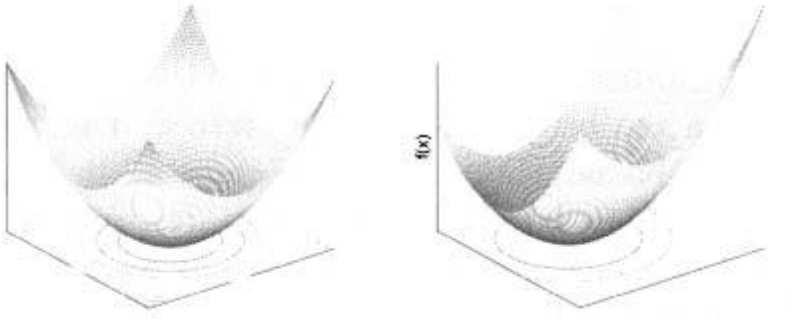
$$f(x) = \sum_{i=1}^{n} x_i^2 \qquad (ج.۱۲۳)$$

بهینه‌ی این تابع در $x^* = 0$ واقع شده است. اگر فضای جستجو برای تمامی iها در $x_i \in [-C, C]$ باشد، آنگاه بهینه در مرکز فضای جستجو واقع خواهد شد. بنابراین معادله‌ی (ج.۱۲۳) را به‌صورت زیر اصلاح می‌نماییم:

$$f(x) = \sum_{i=1}^{n}(x_i - o_i)^2 \quad \text{که در آن } o_i \sim U[-C, C] \text{ برای } i \in [1, n] \qquad (ج.۱۲٤)$$

همچنین می‌توان از توزیع دیگری به غیر از توزیع یکنواخت برای تولید o_i استفاده نمود. اما معمولاً o_i را به $[-C, C]$ محدود می‌نماییم تا مطمئن باشیم بهینه‌ی جهانی معادله‌ی (ج.۱۲٤) در فضای جستجو واقع می‌شود. شکل تابع کروی شیفت یافته‌ی معادله‌ی (ج.۱۲٤) همانند شکل تابع کروی اصلی معادله‌ی (ج.۱۲۳) می‌باشد اما راه‌حل آن در محلی اتفاقی در فضای جستجو واقع شده است. بدین ترتیب اطمینان خواهیم داشت که هیچ الگوریتم تکاملی دارای برتری ناعادلانه در ارزیابی محک نخواهد بود. شکل ج.۲٦ نسخه‌ی شیفت یافته‌ی تابع کروی را نشان می‌دهد.

هنگام ارزیابی الگوریتم‌های تکاملی بر روی تابع کروی شیفت یافته باید چند شبیه‌سازی مونت کارلو، هر کدام با یک شیفت متفاوت، اجرا نماییم. این رویکرد، که در شکل ج.۲۷ نشان داده شده است، به ما این اجازه را می‌دهد که بهترین الگوریتم تکاملی برای توابع شبه-کروی را، با پرهیز از گرایشی که ناشی از محل بهینه است، تعیین نماییم. پس از تکمیل حلقه‌ی شکل ج.۲۷، M نتیجه برای الگوریتم تکاملی اول، M نتیجه برای الگوریتم تکاملی دوم و ... به دست خواهیم آورد. الگوریتم‌های تکاملی را می‌توان با میانگین‌گیری از مجموعه‌ی M نتیجه‌ی حاصل شده برای هر الگوریتم، یا با استفاده از بهترین نتیجه، یا با استفاده از بدترین نتیجه، و یا با استفاده از هر اندازه‌گیری دیگری، مقایسه نمود. انتخاب کمیت مورد مقایسه به میزان اهمیت آن کمیت بستگی دارد (ضمیمه‌ی ب.۲ را ببینید).



شکل ج.۲٦ شکل سمت چپ تابع کروی دو بعدی اصلی بدون شیفت و شکل سمت راست تابع کروی شیفت یافته است.

تعداد ارزیابی‌های الگوریتم‌های تکاملی $= P$

تعداد شبیه‌سازی‌های مونت کارلو $= M$

برای M تا $j = 1$

o_i اتفاقی را برای $i \in [1,n]$ تولید کن

برای P تا $p = 1$

pاُمین عملکرد الگوریتم تکاملی بر روی $f(x - o)$ را محاسبه کن

الگوریتم تکاملی بعد

شبیه‌سازی مونت کارلوی بعد

شکل ج.۲۷ شبیه‌سازی مونت کارلو برای ارزیابی عملکرد الگوریتم تکاملی بر روی توابع n بعدی شیفت یافته. گرایش ناشی از محل قرارگیری بهینه حذف گردیده است.

ج.۲-۷ ماتریس‌های چرخش

فرایند جستجوی برخی الگوریتم‌های تکاملی به سمت برخی متغیرهای مستقل گرایش دارد. برای مثال، یک استراتژی جهش و یا یک استراتژی تپه‌نوردی که در هر زمان تنها یک متغیر مستقل را تغییر می‌دهد، در هر دوره در راستای تنها یک بعد به جستجو می‌پردازد. این نوع الگوریتم‌های بهینه‌سازی بر روی مسائلی که گرادیانشان به موازات متغیرهای مستقل قرار دارند، عملکرد خوبی دارند. با این حال، این عملکرد خوب می‌تواند گمراه‌کننده باشد چرا که ناشی از از این حقیقت است که بسیاری از محک‌ها دارای گرادیان موازی با بردارهای یکه‌ی فضای جستجو می‌باشند. بسیاری از محک‌های این ضمیمه دارای چنین گرادیان‌هایی می‌باشند. استفاده از چنین محک‌هایی برای ارزیابی عملکرد الگوریتم تکاملی عادلانه نیست.

بنابراین، باید محک‌ها را با استفاده از ماتریس‌های چرخش در مسئله اصلاح نمود [سالومون، ۱۹۹۶]، [سوگانتان و همکاران، ۲۰۰۵]. تابع ماکزیمم اشوفل از معادله‌ی (ج.۲۵) را در نظر بگیرید:

$$f(x) = \max_i(|x_i| : i \in \{1, \dots, n\}) \qquad (ج.۱۲۵)$$

در این مسئله می‌خواهیم $f(x)$ را مینیمم نماییم. یک فرایند جستجوی ساده که هر بار یک عنصر از x کم می‌کند بر روی این مسئله عملکرد بسیار خوبی خواهد داشت. چنین فرایند جستجوی ساده‌ای حتی می‌تواند بر روی مسائل دنیای واقعی نیز عملکرد خوبی داشته باشد. اما مسائل بسیاری در دنیای واقعی وجود

دارد که به فرایندهای جستجوی پیچیده‌تری نیاز دارند. بنابراین، معادله‌ی (ج.۱۲۵) را به‌صورت زیر اصلاح می‌نماییم:

$$f(x) = \max_i(|y_i| : i \in \{1, \dots, n\}) \quad \text{که در آن} \quad y = xQ \qquad (ج.۱۲۶)$$

بردارهای n عنصری x و y بردارهای سطری بوده و Q نیز یک ماتریس چرخش $n \times n$ می‌باشد. یک ماتریس چرخش ماتریسی است که وقتی در یک بردار ضرب می‌شود آن بردار را در دامنه‌ی n بعدی‌اش می‌چرخاند [گولان، ۲۰۰۷]. یک ماتریس چرخش معادل یک ماتریس متعامد است و یک ماتریس متعامد ماتریسی است که ترنسپوز آن با دترمینان آن برابر است و دترمینان آن نیز برابر ۱ است:

$$|Q| = 1 \text{ و } Q^{-1} = Q^T \qquad (ج.۱۲۷)$$

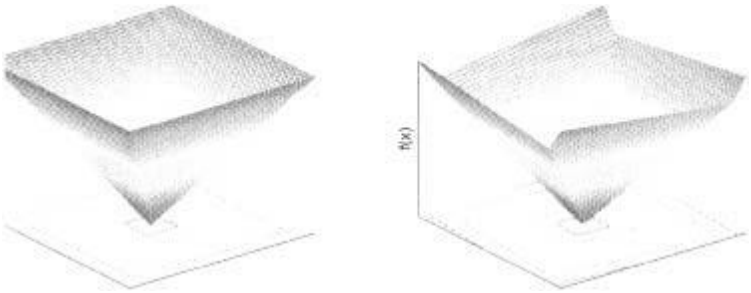
یک ماتریس چرخش اتفاقی را می‌توان با تجزیه QR تولید نمود [گولان، ۲۰۰۷]. تجزیه‌ی QR شامل پیدا نمودن یک ماتریس متعامد Q و یک ماتریس بالامثلثی R است به‌صورتی که برای یک ماتریس مشخص مانند D، $QR = D$ شود. هر ماتریس حقیقی D دارای تجزیه‌ی QR می‌باشد. اگر یک ماتریس D با ابعاد $n \times n$ و با درایه‌های اتفاقی تولید نماییم و سپس تجزیه‌ی QR آن را محاسبه نماییم، نگاه ماتریس Q یک ماتریس چرخش اتفاقی خواهد بود. بنابراین، در محیط MATLAB می‌توان به‌صورت زیر یک ماتریس چرخش اتفاقی Q، با ابعاد $n \times n$، تولید نمود:

```
D = rand(n);
[Q, R] = QR(D);
```

در بالا $rand(n)$ تابعی است در MATLAB که یک ماتریس $n \times n$ تولید می‌نماید، به‌صورتی که تمامی درایه‌ها از یک توزیع گاوسی با میانگین صفر و واریانس یک گرفته می‌شوند. همچنین تابع QR در بالا تابع تجزیه‌ی QR در MATLAB می‌باشد. دیگر کتابخانه‌های جبر خطی و نرم‌افزارها دارای توابع مشابه می‌باشند. شکل تابع ماکزیمم اشوفل شیفت یافته از معادله‌ی (ج.۱۲۶) همانند شکل تابع ماکزیمم اشوفل اصلی است تنها با این تفاوت که نسبت به مبدأ فضای جستجو چرخیده است. بنابراین، گرادیان تابع هدف موازی ابعاد متغیر مستقل نخواهد بود. این موضوع این اطمینان را به ما خواهد داد که هیچ الگوریتم تکاملی خاصی دارای برتری ناعادلانه در ارزیابی محک نخواهد بود. شکل ج.۲۸ تابع ماکزیمم اشوفل را که به اندازه‌ی چند درجه در جهت خلاف عقربه‌های ساعت چرخیده است، نشان می‌دهد.

هنگام مقایسه‌ی الگوریتم‌های تکاملی بر روی تابع ماکزیمم اشوفل باید از چند شبیه‌سازی مونت کارلو استفاده نماییم در هر شبیه‌سازی نیز ار ماتریس‌های چرخش متفاوت استفاده نماییم. این رویکرد مشابه رویکرد

نشان داده شده در شکل ج.۲۷ بوده در شکل ج.۲۹ نشان داده شده است. این رویکرد به ما اجازه می‌دهد با پرهیز از گرایش ناشی از طبیعت موازی گرادیان تابع اصلی، بهترین الگوریتم تکاملی را تعیین نماییم. پس از کامل شدن حلقه‌ی شکل ج.۲۹، M نتیجه برای الگوریتم تکاملی اول، M نتیجه برای الگوریتم تکاملی دوم و ... به دست خواهیم آورد. الگوریتم‌های تکاملی را می‌توان با میانگین‌گیری از مجموعه‌ی M نتیجه‌ی حاصل شده برای هر الگوریتم، یا با استفاده از بهترین نتیجه، یا با استفاده از بدترین نتیجه، و یا با استفاده از هر اندازه‌گیری دیگری، مقایسه نمود. انتخاب کمیت مورد مقایسه به میزان اهمیت آن کمیت بستگی دارد (ضمیمه‌ی ب.۲ را ببینید). می‌توان منطق‌های شکل ج.۲۷ و ج.۲۹ را ترکیب کرد و از آن برای مقایسه‌ی الگوریتم‌های تکاملی بر روی توابع $f((x-o)Q)$، که هم شیفت داده شده و هم چرخیده‌اند، استفاده نمود.



شکل ج.۲۸ شکل سمت چپ تابع ماکزیمم اشوفل دو بعدی و شکل سمت راست همان تابع را با چرخش چند درجه‌ای در جهت خلاف عقربه‌های ساعت را نشان می‌دهد.

تعداد ارزیابی‌های الگوریتم‌های تکاملی $= P$

تعداد شبیه‌سازی‌های مونت کارلو $= M$

برای M تا $j = 1$

یک ماتریس چرخشی اتفاقی مانند Q تولید کن

برای P تا $p = 1$

pاُمین عملکرد الگوریتم تکاملی بر روی $f(xQ)$ را محاسبه کن

الگوریتم تکاملی بعد

شبیه‌سازی مونت کارلوی بعد

شکل ج.۲۹ شبیه‌سازی مونت کارلو برای ارزیابی عملکرد الگوریتم تکاملی بر روی مسائل چرخیده. بایاس ناشی از موازی بودن گرادیان با سیستم مختصات حذف شده است.

واژه نامه انگلیسی به فارسی

به‌طور مشابه لغات پانویسی شده باید در اینجا به ترتیب حروف الفبای انگلیسی و از چپ به راست آورده شود.

Acceptance probability constant	ثابت احتمال پذیرش
Acceptance test	آزمایش پذیرش
Ackley	اکلی
Active Learning	یادگیری فعال
Adaptive Culture Model	مدل فرهنگی انطباقی
Adaptive Penalty Methods	روش‌های مجازات انطباقی
Adaptive Segregational Constraint Handling Evolutionary Algorithm	الگوریتم تکاملی اداره‌ی تبعیضی و انطباقی قید
Additive Penalty Method	روش مجازات افزایشی
Adjacency Representation	نمایش مجاورت
All Topology	توپولوژیِ همه
Alternating Edges Crossover	برش شاخه‌های متناوب
Ant Colony System	سیستم کلونی مورچگان
Approximated non-deterministic Tree Search	درخت جستجوی غیرقاطع تخمینی
Argentine Ant	مورچه‌های آرژانتین
Arithmetic crossover	برش حسابی
Artificial Fish Swarm Algorithm	الگوریتم تجمع مصنوعی ماهی‌ها
Artificial Immune Systems	سیستم دفاعی مصنوعی
Bacterial Chemotaxis Model	مدل کموتاکسی باکتری
Bacterial Foraging Optimization Algorithm	الگوریتم بهینه‌سازی کاوش باکتریایی
Barrier Methods	روش‌های مانع

Bat-inspired Algorithm	الگوریتم الهام گرفته شده از خفاش
Bayesian Optimization Algorithm	الگوریتم بهینه‌سازی بیزی
Behavioral Memory	حافظه‌ی رفتاری
Belief Space	فضای عقیده
Bernoulli’s Principal of Insufficient Reason	قاعده‌ی دلیل ناکافی برنولی
Big Bang Big Crunch Algorithm	الگوریتم بحران بزرگ مهبانگ
Bin Packing Problem	مسئله‌ی بسته‌بندی صندوق
Bivariate Marginal Distribution Algorithm	الگوریتم‌های تخمین حاشیه‌ای دومتغیره
Blended crossover	برش مخلوط شده
Boltzmann’s constant	ثابت بولتزمن
Building Block	بلوک سازه
Capacitated Vehicle Routing Problem	مسئله‌ی مسیریابی وسیله نقلیه تقویت‌شده
Central Force Optimization	بهینه‌سازی نیروی مرکزی
Charged System Search	جستجوی سیستم باردارشده
Chemical Reaction Optimization	بهینه‌سازی واکنش شیمیایی
Cluster Topology	توپولوژیِ خوشه‌ای
Coevolutionary Penalty	مجازات هم‌تکاملی
Combining Optimizers with Mutual Information Trees	ترکیب بهینه‌سازها با استفاده از درخت اطلاعات متقابل
Compact Genetic Algorithm	الگوریتم ژنتیک فشرده
Contour Matching	تطبیق شکل
Corridor Problem	مسئله‌ی دهلیز
Covariance Matrix Adaptation	انطباق ماتریس کوواریانس

Cross Validation	تأیید متقابل
Cuckoo Search	جستجوی فاخته
Cycle crossover	برش چرخشی
Death Penalty Approaches	رویکردهای مجازات مرگ
Decoder Approach	رویکرد کدبردار
Design and Analysis of Computer Experiments	طراحی و تحلیل آزمایش‌های کامپیوتری
Discrete Sexual Crossover	برش گسسته‌ی جنسی
Displacement	جابه‌جایی
Diversity Evolutionary Multi-objective Optimizer	بهینه‌سازی تکاملی چندهدفه‌ی تنوع
Divide-and-Conquer Method	روش تقسیم-و-غلبه
Dominant Crossover	برش غالب
Dynamic Approximate Fitness Hybrid EA	الگوریتم تکاملی ترکیبی مبتنی بر تخمین پویای برازندگی
Equilibrium-energy configurations	پیکربندی‌های انرژی- متعادل
Estimation of Bayesian networks algorithm	الگوریتم تخمین شبکه‌ی بیزی
Estimation of Distribution Algorithm	الگوریتم تخمین توزیع
Evolutionary Algorithms	الگوریتم‌های تکاملی
Evolutionary Program	برنامه‌ی تکاملی
Evolutionary Programming	برنامه‌نویسی تکاملی
Evolutionary Strategiy	استراتژی تکامل
Expected Value	امید ریاضی
Exponential Dynamic Penalties	مجازات‌های پویای نمایی
Extended Compact Genetic Algorithm	الگوریتم ژنتیک فشرده‌ی تعمیم‌یافته

Exterior Approaches	روش‌های خارجی
Factorized Distribution Algorithm	الگوریتم توزیع فاکتور گرفته شده
Feasible Set	مجموعه‌ی امکان‌پذیر
Feral children	کودکان یاغی
Finite State Machines	ماشین‌های حالتِ متناهی
Fitness Inheritance	وراثت برازندگی
Fitness Landscape	دورنمای برازندگی
Flat crossover	برش مسطح
Fully Informed Particle Swarm	تجمع ذرات کاملاً آگاه
Fuzzy Logic	منطق فازی
Fuzzy Logic Systems	سیستم‌های با منطق فازی
Fuzzy recombination	بازترکیب فازی
Fuzzy Systems	سیستم‌های فازی
Gaussian Adaptation	انطباق گاوسی
Gaussian Mutation centered at the middle of search domain	جهش گاوسی متمرکز در مرکز محدوده‌ی جستجو
Gene pool recombination	بازترکیب استخر ژن
Genetic programming	برنامه‌نویسی ژنتیک
Genocop	ژنوکوپ
Genocop Algorithm	الگوریتم ژنوکوپ
Genotype	ژنوتیپ
Global Crossover	برش سراسری
Global uniform crossover	برش یکنواخت جهانی

Glow worm Swarm Optimization	بهینه‌سازی تجمع کرم شب‌تاب
Goal Programming	برنامه‌نویسی هدف
Grammatical Evolution	تکامل گرامری
Graph Coloring Problem	مسئله‌ی رنگ‌آمیزی گراف
Gravitational Search Algorithm	الگوریتم جستجوی گرانشی
Great Deluge Algorithm and record-to-record Travel	الگوریتم طوفان بزرگ و سفر رکورد-به-رکورد
Greedy	حریص
Habitat Suitability Index	شاخص تناسب زیستگاهی
Hajela-Lin Genetic Algprithm	الگوریتم ژنتیک حاجلا-لین
Hamiltonian Path Problem	مسئله‌ی مسیر همیلتون
Hard Computing	محاسبات سخت
Harmony Seacrh	جستجوی هارمونی
Heuristic Crossover	برش ابتکاری
Hill Climbing	تپه‌نوردی
Hill Descending	دره‌نوردی
Hybrid Methods	روش‌های ترکیبی
Hypermutation	فراجهش
Hypervolume	فراتوده
Immigrant-Based Eas	الگوریتم‌های تکاملی مهاجرت-محور
Imperialist Competitive Algorithm	الگوریتم رقابتی امپریالیست
Inertia	اینرسی
Infeasibility Driven Evolutionary Algorithm	الگوریتم تکاملی منتج از امکان‌ناپذیری
Infeasible Set	مجموعه‌ی امکان ناپذیر

Informed Operator	اپراتور آگاه
Insertion	الحاق
Integrated Radiation Optimization	بهینه‌سازی تشعشع یکپارچه
Intelligent Water Drops	قطرات آب هوشمند
Interior Point Approaches	روش‌های نقطه‌ی داخلی
Intermediate Global Crossover	برش سراسری میانی
Intermediate Sexual Crossover	برش جنسی میانی
Intersection Crossover	برش اشتراکی
Invasive Weed Optimization	بهینه‌سازی علف مهاجم
Inversion	واژگونی
Job Shop Scheduling	مسئله‌ی زمانبندی کار کارگاهی
Knapsack Problem	مسئله‌ی کوله‌پشتی
Krill Herd	گله‌ی کریل
Latin Hypercube Sampling	نمونه‌گیری فرامکعب لاتین
Lexicographic Ordering	رویکردهای مرتب‌سازی وابسته به واژه‌نگاری
Linear crossover	برش خطی
Markov Network Factorized Distribution Algorithm	الگوریتم توزیع فاکتوریزه شبکه‌ی مارکوف
Matrix Representation	نمایش ماتریسی
Maximum Likelihood Estimate	برآورد درست‌نمایی بیشینه
Maximum Likliehood	درست‌نمایی بیشینه
Membrane Computing	محاسبات غشایی
Memetic Algorithms	الگوریتم‌های ممتیک
Memory-Based Eas	الگوریتم‌های تکاملی حافظه–محور

Metaheuristic	فرااکتشافی
Minimum Spanning Tree Problem	مسئله‌ی درخت پوشای مینیمم
Multimembered Evolutionary Strategy	استراتژی تکامل چند عضوه
Multimodal	چندپیمانه‌ای
Multi-Objective Biogeography-Based Optimization	الگوریتم بهینه‌سازی چندهدفه‌ی زیست‌جغرافی-محور
Multi-Objective Evolutionary Algorithms	الگوریتم‌های تکاملی چندهدفه
Multi-Objective Genetic Algorithm	الگوریتم ژنتیک چندهدفه
Multi-Objective Optimization	بهینه‌سازی چندهدفه
Multi-Objective Optimization Problem	مسائل بهینه‌سازی چندهدفه
Multi-parent crossover	برش چند- والده
Multi-point crossover	برش چند نقطه‌ای
Mutual Information maximization for Input Clustering	بیشینه‌سازی اطلاعات متقابل برای خوشه‌سازی ورودی
Neural Network	شبکه‌های عصبی
Niched Pareto BBO	الگوریتم بهینه‌سازی زیست‌جغرافی-محور نیچه
Niched Pareto Genetic Algorithm	الگوریتم ژنتیک پرتوی نیچه
Niched-Penalty Approach	رویکرد جریمه‌ی نیچه
Nondominated Sorting Biogeography-Based Optimization	الگوریتم بهینه‌سازی زیست‌جغرافی-محور مرتب‌کننده‌ی نامغلوب
Non-dominated Sorting Genetic Algorithm	الگوریتم ژنتیک مرتب‌سازی بر حسب ذرات غیرمغلوب
None-Death-Penalty Approaches	رویکردهای مجازات غیر-مرگ

Online Surrogate Updating	به روز رسانی جانشین
Opposition Based Learning	یادگیری مقابله محور
Order crossover	برش ترتیبی
Order-Based crossover	برش ترتیب–محور
Ordinal Representation	نمایش ترتیبی
Other Dynamic Penalty Approaches	رویکردهای مجازات پویای دیگر
Overfitting	بیش برازش
Over-Selecting	انتخاب بیش از حد
Panmictic Crossover	برش پانمیکتیک
Pareto	پَرِتو
Pareto Archived Evolutionary Strategy	استراتژی تکاملی پرتوی آرشیو شده
Pareto Front	*مرز پرتو*
Pareto Optimality	بهینگی پرتو
Partially Matched crossover	برش تطبیق‌یافته جزیی
Particle Swarm Optimization	بهینه‌سازی تجمع ذرات
Path Representation	نمایش مسیر
Peer Review	داوری همتا
Peer-to-Peer File Sharing	اشتراک گذاری فایل به‌صورت نظیر به نظیر
Penalized Cost Function	تابع هزینه‌ی مجازات شده
Penalty Function Approaches	روش‌های تابع مجازات
Phenotype	فنوتیپ
Population Based Incremental Learning	یادگیری افزایشی جمعیت–محور
Pre-Allocating Arrays	ارائه‌های پیش تخصیص

Predictive EA	الگوریتم تکاملی پیشگو
Quasi Opposites	مخالفان کوشی
Rank-Based Selection	انتخاب رتبه- محور
Reciprocal Exchange	تبادل هم‌پاسخ
Recursive Least Squares Algorithm	الگوریتم بازگشتی حداقل مربعات
Relative Coverage	پوشش نسبی
Repair Algorithms	الگوریتم‌های بازسازی
Ring Topoogy	توپولوژی حلقه
River Foundation Dynamic	دینامیک تشکیل رودخانه
Segmented crossover	برش قطعه قطعه شده
Segregated Genetic Algorithm	الگوریتم ژنتیک تبعیضی
Self-Adaptive Fitness Formulation	فرمول‌بندی برازندگی خود انطباق
Self-Adaptive Penalty Function	تابع مجازات خود انطباق
Sequential Ordering Problem	مسئله‌ی مرتب‌سازی متوالی
Shuffle crossover	برش شافل
Sigma Scaling	مقیاس‌گذاری سیگما
Simple Evolutionary Multi-objective Optimizer	بهینه‌ساز تکاملی چندهدفه ساده
Simple Genetic Algorithm	الگوریتم ژنتیک ساده
Simulated Binary crossover	برش شبیه‌سازی شده‌ی دودویی
Single-point crossover	برش تک نقطه‌ای
Society and Civilization Algorithm	الگوریتم جامعه و تمدن
Soft Computing	محاسبات نرم
Space Gravitational Optimization	بهینه‌سازی گرانش فضایی

Special Operators	عملگرهای خاص
Special Representations	نمایش‌های خاص
Square Topology	توپولوژی مربعی
Squeaky Wheel Optimization	بهینه‌سازی چرخ جیرجیری
Steepest Ascent Hill Climbing	تپه‌نوردی با شدیدترین شیب
Stochastic Diffusion Search	جستجوی انتشار اتفاقی
Stochastic Hill Climbing with Learning by Vectors of Normal Distribution	تپه‌نوردی اتفاقی یا یادگیری با استفاده از بردارهای توزیع نرمال
Stochastic Ranking	رتبه‌بندی اتفاقی
Stochastic Sampling	نمونه‌برداری اتفاقی
Strength Pareto BBO	الگوریتم بهینه‌سازی زیست‌جغرافی–محور استحکام پرتو
Strength Pareto Evolutionary Algorithm	الگوریتم تکاملی استحکام پرتو
Suitability Index Variables	متغیرهای شاخص سازگاری
Super Opposites	فرامخالفان
Superiority of Feasible Points	برتری نقاط امکان‌پذیر
Supported Vector Machines	ماشین‌های بردار پشتیبان
Survival of The Fittest	بقای برازنده‌ترین
Survival of the Mediocre	بقای متوسط
Tabu Search	جستجوی ممنوعه
Teaching-learning Based Optimization	بهینه‌سازی تعلیم و تعلم محور
The Eclectic Evolutionary Algorithms	الگوریتم تکاملی گزینشی
The Law of Conservation of General Performance	قاعده‌ی پایداری عملکرد کلی

The Law of Conservation of Information	قانون پایداری اطلاعات
The Messenger Problem	مسئله‌ی پیام‌رسان
Threshold Accepting	آستانه پذیرش
Traveling Salesman Problem	مسئله‌ی فروشنده‌ی دوره گرد
Trust Regions	نواحی اعتماد
Uniform crossover	برش یکنواخت
Uniform Mutation centered	جهش یکنواخت متمرکز
Uniform Mutation centered at the middle of search domain	جهش یکنواخت متمرکز در مرکز محدوده‌ی جستجو
Union Crossover	برش اجتماع
Univariate Marginal Distribution Algorithm	الگوریتم تخمین حاشیه‌ای تک‌متعیره
Vector Evaluated Biogeography-Based Optimization	الگوریتم بهینه‌سازی زیست‌جغرافی–محور برداری سنجیده
Vector Evaluated Genetic Algorithm	الگوریتم ژنتیک برداری سنجیده شده
Wheel Topology	توپولوژی چرخ

کتابنامه

Aarts, E. and Korst, J. (1989). *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing.* John Wiley &; Sons.

Aarts, E., Lenstra, J., and van Laarhoven, P. (2003). Simulated annealing. In Aarts, E. and Lenstra, J., editors, *Local Search in Combinatorial Optimization,* pages 91-120. Princeton University Press.

Ackley, D. (1987a). *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers.

Ackley, D. (1987b). An empirical study of bit vector function optimization. In Davis, L., editor, *Genetic Algorithms and Simulated Annealing,* pages 170-215. Pitman Publishing. Adami, C. (1997). *Introduction to Artificial Life.* Springer. Adler, F. and Nuernberger, B. (1994). Persistence in patchy irregular landscapes. *Theoretical Population Biology,* 45(l):41-75.

Aguirre, A., Rionda, S., Coello Coello, C , Lizârraga, G., and Mezura-Montes, E. (2004). Handling constraints using multiobjective optimization concepts. *International Journal for Numerical Methods in Engineering,* 59(15): 1989-2017.

Ahn, C. and Ramakrishna, R. (2007). Multiobjective real-coded Bayesian optimization algorithm revisited: Diversity preservation. *Genetic and Evolutionary Computation Conference,* London, England, pages 593-600.

Akat, S. and Gazi, V. (2008). Particle swarm optimization with dynamic neighborhood topology: Three neighborhood strategies and preliminary results. *IEEE Swarm Intelligence Symposium,* St. Louis, Missouri, pages 1-8.

Alami, J. and El Imrani, A. (2008). Using cultural algorithm for the fixed-spectrum frequency assignment problem. *Journal of Mobile Communication,* 2(1): 1-9.

Alami, J., El Imrani, A., and Bouroumi, A. (2007). A multipopulation cultural algorithm using fuzzy clustering. *Applied Soft Computing,* 7(2):506-519.

Alexander, R. (1996). *Optima for Animals.* Princeton University Press. Ali, M., Khompatraporn, C , and Zabinsky, Z. (2005). A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization,* 31(4):635-672.

Allenson, R. (1992). Genetic algorithms with gender for multi-function optimisation. Technical report, Edinburgh Parallel Computing Centre. EPCC-SS92-01.

Altenberg, L. (1994). Emergent phenomena in genetic programming. *Conference on Evolutionary Programming,* San Diego, California, pages 233-241.

Anderson, M. and Oates, T. (2007). A review of recent research in metareasoning and metalearning. *AI Magazine,* 28(1):7-16.

Andre, D., Bennett, F., and Koza, J. (1996). Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. *Genetic Programming Conference,* Palo Alto, California, pages 28-31.

Angeline, P. (1996a). An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover. *Genetic Programming Conference,* Palo Alto, California, pages 21-29.

Angeline, P. (1996b). Two self-adaptive crossover operators for genetic programming. In Angeline, P. and Kinnear, K., editors, *Advances in Genetic Programming: Volume 2,* pages 89-110. The MIT Press.

Angeline, P. (1997). Subtree crossover: Building block engine or macromutation? *Genetic Programming Conference,* Palo Alto, California, pages 9-17.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (2007). *The Traveling Salesman Problem.* Princeton University Press.

Araujo, M., Wanner, E., Guimaräes, F., and Takahashi, R. (2009). Constrained optimization based on quadratic approximations in genetic algorithms. In Mezura-Montes, E., editor, *Constraint-Handling in Evolutionary Optimization,* pages 193-217. Springer.

Arnold, D. (2002). *Noisy Optimization with Evolution Strategies.* Kluwer Academic Publishers.

Ashlock, D. (2009). *Evolutionary Computation for Modeling and Optimization.* Springer. Aström, K. and Wittenmark, B. (2008). *Adaptive Control.* Dover Publications. Atashpaz-Gargari, E. and Lucas, C. (2007). Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. *IEEE Congress on Evolutionary Computation,* Singapore, pages 4661-4667.

Auger, A., Bader, J., Brockhoff, D., and Zitzler, E. (2012). Hypervolume-based multiobjective optimization: Theoretical foundations and practical implications. *Theoretical Computer Science,* 425:75-103.

Axelrod, R. (1997). The dissemination of culture: A model with local convergence and global polarization. *Journal of Conflict Resolution,* 41(2):203-226.

Axelrod, R. (2006). *The Evolution of Cooperation: Revised Edition.* Basic Books. First published in 1984. Back, T. (1996). *Evolutionary Algorithms in Theory and Practice.* Oxford University Press.

Back, T., Fogel, D., and Michalewicz, Z. (1997a). *Handbook of Evolutionary Computation.* Taylor and Francis.

Back, T., Hammel, U., and Schwefel, H. (1997b). Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation,* 1(1):3-17.

Back, T. and Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation,* 1(1): 1-23.

Baker, J. (1987). Reducing bias and inefficiency in the selection algorithm. *International Conference on Genetic Algorithms and Their Application,* Cambridge, Massachusetts, pages 14-21.

Balakrishnan, V. (1997). *Schaum's Outline of Graph Theory.* McGraw-Hill, 13th edition. Balasubramaniam, P. and Kumar, A. (2009). Solution of matrix Riccati differential

equation for nonlinear singular system using genetic programming. *Genetic Programming and Evolvable Machines*, 10(l):71-89.

Ball, W. and Coxeter, H. (2010). *Mathematical Recreations and Essays.* Dover, 13th edition.

Baluja, S. (1994). Population-based incremental learning. Technical report, Carnegie Mellon University. CMU-CS-94-163.

Baluja, S. and Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. *12th International Conference on Machine Learning,* Tahoe City, California, pages 38-46.

Baluja, S. and Davies, S. (1998). Fast probabilistic modeling for combinatorial optimization. *Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence,* pages 469-476.

Bandyopadhyay, S., Saha, S., Maulik, U., and Deb, K. (2008). A simulated annealingbased multiobjective optimization algorithm: AMOSA. *IEEE Transactions on Evolutionary Computation,* 12(3):269-283.

Banks, A., Vincent, J., and Anyakoha, C. (2007). A review of particle swarm optimization. Part I: Background and development. *Natural Computing,* 6(4):467-484.

Banks, A., Vincent, J., and Anyakoha, C. (2008). A review of particle swarm optimization. Part II: Hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. *Natural Computing,* 7(1): 109-124.

Bankston, J. (2005). *Gregor Mendel and the Discovery of the Gene.* Mitchell Lane

Publishers. Banzhaf, W. (1990). The "molecular" traveling salesman. *Biological Cybernetics,* 64(1):7- 14.

Banzhaf, W., Nordin, P., Keller, R., and Francone, F. (1998). *Genetic Programming.* Morgan Kauffman Publishers.

Barr, R., Golden, B., Kelly, J., Resende, M., and Stewart, W. Designing and reporting on computational experiments with heuristic methods. *Journal of Metaheuristics,* 1(1).

Barricelli, N. (1954). Esempi numerici di processi di evoluzione. *Methodos,* 6:45-68.

The English translation of the title is *Numerical models of evolutionary processes.* Bastürk, B. and Karaboga, D. (2006). An artificial bee colony (ABC) algorithm for numeric function optimization. *IEEE Swarm Intelligence Symposium,* Indianapolis, Indiana.

Becerra, R. and Coello Coello, C. (2004). A cultural algorithm with differential evolution to solve constrained optimization problems. In Lemaitre, C , Reyes, C , and Gonzalez, J., editors, *Advances in Artificial Intelligence - IBERAMIA 2004_' ^th Ibero-American Conference on AI, Puebla, Mexico, November 22-26, 2004,* pages 881-890. Springer.

Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour.* Princeton University Press.

Benatchba, K., Admane, L., and Koudil, M. (2005). Using bees to solve a data-mining problem expressed as a max-sat one. In Mira, J. and Alvarez, J., editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach,* pages 212-220. Springer.

Bernstein, D. (2006). Optimization r us. *IEEE Control Systems Magazine,* 26(5):6-7.

Betts, J. (2009). *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming.* Society for Industrial *h* Applied Mathematics, 2nd edition.

Beveridge, W. (2004). *The Art of Scientific Investigation.* Blackburn Press.

Beyer, H.-G. (1998). On the dynamics of EAs without selection. *Foundations of Genetic Algorithms,* Amsterdam, The Netherlands, pages 5-26.

Beyer, H.-G. (2010). *The Theory of Evolution Strategies.* Springer.

Beyer, H.-G. and Deb, K. (2001). On self-adaptive features in real-parameter evolutionary algorithms. *IEEE Transactions on Evolutionary Computation,* 5(3):250-269.

Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strategies: A comprehensive introduction. *Natural Computing,* l(l):3-52.

Beyer, H.-G. and Sendhoff, B. (2008). Covariance matrix adaptation revisited: The CMSA evolution strategy. In Rudolph, G., Jansen, T., Lucas, S., Poloni, C., and Beume, N., editors, *Parallel Problem Solving from Nature - PPSN X,* pages 123-132. Springer.

Bhattacharya, M. (2008). Reduced computation for evolutionary optimization in noisy environment. *Genetic and Evolutionary Computation Conference,* Atlanta, Georgia, pages 2117-2122.

Bishop, C. (2006). *Pattern Recognition and Machine Learning.* Springer.

Bishop, J. (1989). Stochastic searching networks. *First IEE Conference on Artificial Neural Networks,* London, England, pages 329-331.

Biswas, A., Dasgupta, S., Das, S., and Abraham, A. (2007a). A synergy of differential evolution and bacterial foraging optimization for global optimization. *Neural Network World,* 17(6):607-626.

Biswas, A., Dasgupta, S., Das, S., and Abraham, A. (2007b). Synergy of PSO and bacterial foraging optimization - A comparative study on numerical benchmarks. In Corchado, E., Corchado, J., and Abraham, A., editors, *Innovations in Hybrid Intelligent Systems,* pages 255-263. Springer.

Blum, C. (2005a). Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews,* 2(4):353-373.

Blum, C. (2005b). Beam-ACO - Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research,* 32(6): 1565-1591.

Blum, C. (2007). Ant colony optimization: Introduction and hybridizations. *Seventh International Conference on Hybrid Intelligent Systems,* Kaiserlautern, Germany, pages 24-29.

Blum, C. and Dorigo, M. (2004). The hypercube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics,* 34(2) 1161—1172.

Bonabeau, E., Theraulaz, G., and Dorigo, M. (1999). *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press.

Bonacich, P., Shure, G., Kahan, J., and Meeker, R. (1976). Cooperation and group size

In = the n-person prisoners' dilemma. *The Journal of Conflict Resolution,* 20(4):687-706.

Boslaugh, S. and Watters, P. (2008). *Statistics in a Nutshell.* O'Reilly Media.

Bosman, P. and Thierens, D. (2003). The balance between proximity and diversity in multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation,* 7(2):174-188.

Box, G. (1957). Evolutionary operation: A method for increasing industrial productivity. *Journal of the Royal Statistical Society, Series C (Applied Statistics),* 6(2):81-101.

Box, J. (1987). Guinness, Gösset, Fisher, and small samples. *Statistical Science,* 2(1):45 52.

Branke, J. (1998). Creating robust solutions by means of evolutionary algorithms. In Eiben, A., Back, T., Schoenauer, M., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN V,* pages 119-128. Springer.

Branke, J. (1999). Efficient fitness estimation in noisy environments. *Memory enhanced evolutionary algorithms for changing optimization problems,* Washington, District of Columbia, pages 1875-1882.

Branke, J. (2002). *Evolutionary Optimization in Dynamic Environments.* Kluwer Academic Publishers.

Branke, J. (2012). Evolutionary Algorithms for Dynamic Optimization Problems (EvoDOP). h t t p : / / p e o p l e . a i f b . k i t . edu/jbr/EvoDOP.

Branke, J., Orbayi, M., and Uyar, S. (2006). The role of representations in dynamic knapsack problems. In Rothlauf, F., editor, *Applications of Evolutionary Computing,* pages 764-775. Springer.

Branke, J., Schmidt, C , and Schmec, H. (2001). Efficient fitness estimation in noisy environments. *Genetic and Evolutionary Computation Conference,* San Francisco, California, pages 243-250.

Bratton, D. and Kennedy, J. (2007). Defining a standard for particle swarm optimization. *IEEE Swarm Intelligence Symposium,* Honolulu, Hawaii, pages 120-127.

Bremermann, H., Rogson, M., and Salaff, S. (1966). Global properties of evolution processes. In Pattee, H., Edlsack, E., Fein, L., and Callahan, A., editors, *Natural Automata and Useful Simulations,* pages 3-41. Spartan Books.

Brest, J. (2009). Constrained real-parameter optimization with e-self-adaptive differential evolution. In Mezura-Montes, E., editor, *Constraint-Handling in Evolutionary Optimization,* pages 73-93. Springer.

Brest, J., Zamuda, A., Boskovic, B., Maucec, M., and Zumer, V. (2009). Dynamic optimization using self-adaptive differential evolution. *IEEE Congress on Evolutionary Computation,* Trondheim, Norway, pages 415-422.

Bringmann, K. and Friedrich, T. (2010). An efficient algorithm for computing hypervolume contributions. *Evolutionary Computation*, 18(3):383-402.

Bui, L., Abbass, H., and Essam, D. (2005). Fitness inheritance for noisy evolutionary multi-objective optmization. *Genetic and Evolutionary Computation Conference*, Washington, District of Columbia, pages 779-785.

Bureerat, S. and Sriworamas, K. (2007). Population-based incremental learning for multiobjective optimisation. In Saad, A., Dahal, K., Sarfraz, M., and Roy, R., editors, *Soft Computing in Industrial Applications*, pages 223-232. Springer.

Burke, E. (2003). *High-Tech Cycling.* Human Kinetics, 2nd edition.

Cai, C. and Wang, Y. (2006). A multiobjective optimization-based evolutionary algorithm for constrained optimization. *IEEE Transactions on Evolutionary Computation*, 10(6):658-675.

Cakir, B., Altiparmak, F., and Dengiz, B. (2011). Multi-objective optimization of a stochastic assembly line balancing: A hybrid simulated annealing algorithm. *Computers & Industrial Engineering*, 60(3):376-384.

Carlisle, A. and Dozier, G. (2001). An off-the-shelf PSO. *Particle Swarm Optimization Workshop*, Indianapolis, Indiana, pages 1-6.

Carlson, S. and Shonkwiler, R. (1998). Annealing a genetic algorith over constraints. *IEEE International Conference on Systems, Man, and Cybernetics*, San Diego, California, pages 3931-3936.

Cerny, V. (1985). Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41- 51.

Chafekar, D., Shi, L., Rasheed, K., and Xuan, J. (2005). Multiobjective GA optimization using reduced models. *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, 35(2):261-265.

Chen, S. and Montgomery, J. (2011). Selection strategies for initial positions and initial velocities in multi-optima particle swarms. *Genetic and Evolutionary Computation Conference*, Dublin, Ireland, pages 53-60.

Chen, Y.-L. and Liu, C.-C. (1994). Multiobjective VAr planning using the goalattainment method. *IEE Proceedings on Generation, Transmission and Distribution*, 141(3):227-232.

Cheng, C , Wang, W., Xu, D., and Chau, K. (2008). Optimizing hydropower reservoir operation using hybrid genetic algorithm and chaos. *Water Resources Management*, 22(7):895-909.

Choi, S. and Moon, B. (2003). Normalization in genetic algorithms. *Genetic and Evolutionary Computation Conference*, Chicago, Illinois, pages 862-873.

Chow, C. and Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462-467.

Christensen, S. and Oppacher, F. (2001). What can we learn from no free lunch? A first attempt to characterize the concept of a searchable function. *Genetic and Evolutionary Computation Conference*, San Francisco, California, pages 1219-1226.

Chuan-Chong, C. and Khee-Meng, K. (1992). *Principles and Techniques in Combinatorics.* World Scientific.

Chuang, C.-L. and Jiang, J.-A. (2007). Integrated radiation optimization: Inspired by the gravitational radiation in the curvature of space-time. *IEEE Congress on Evolutionary Computation*, Singapore, pages 3157-3164.

Chung, H.-S. and Alonso, J. (2004). Multiobjective optimization using approximation model-based genetic algorithms. *10th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Albany, New York.

Chung, H.-S., Choi, S., and Alonso, J. (2003). Supersonic business jet design using a knowledge-based genetic algorithm with an adaptive, unstructured grid methodology. *21st AIAA Applied Aerodynamics Conference*, Orlando, Florida.

Clement, P. (1959). A class of triple-diagonal matrices for test purposes. *SIAM Review*, l(l):50-52.

Clerc, M. (1999). The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In *IEEE Congress on Evolutionary Computing*, pages 1951- 1957.

Clerc, M. (2004). Discrete particle swarm optimization, illustrated by the traveling salesman problem. In Onwubolu, G. and Babu, R., editors, *New Optimization Techniques in Engineering*, pages 219-239. Springer.

Clerc, M. (2006). *Particle Swarm Optimization.* John Wiley & Sons.

Clerc, M. (2012a). Particle Swarm Optimization, http://clerc.maurice.free.fr/pso.

Clerc, M. (2012b). Randomness matters. Technical report, http://clerc.maurice.free.fr/pso.

Clerc, M. and Kennedy, J. (2002). The particle swarm - Explosion, stability, and convergencein a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*,6(l):58-73.

Clerc, M. and Poli, R. (2006). Stagnation analysis in particle swarm optimisation or what happens when nothing happens. Technical report, University of Essex, http://clerc.maurice.free.fr/pso.

Cobb, H. and Grefenstette, J. (1993). Genetic algorithms for tracking changing environments. *International Conference on Genetic Algorithms*, Urbana-Champaign, Illinois, pages 523-530.

Coello Coello, C. (1999). A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, l(3):269-308.

Coello Coello, C. (2000a). Constraint-handling using an evolutionary multiobjective optimization technique. *Civil Engineering and Environmental Systems*, 17(4):319-346.

Coello Coello, C. (2000b). Use of a self-adaptive penalty approach for engineering optimization problems. *Computers in Industry*, 41 (2): 113-127.

Coello Coello, C. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191 (11-12): 1245-1287.

Coello Coello, C. (2006). Evolutionary multi-objective optimization: A historical view of the field. *IEEE Computational Intelligence Magazine*, l(l):28-36.

Coello Coello, C. (2009). Evolutionary multi-objective optimization: Some current research trends and topics that remain to be explored. *Frontiers of Computer Science in China*, 3(l):18-30.

Coello Coello, C. (2012a). List of references on constraint-handling techniques used with evolutionary algorithms, www.cs.cinvestav.mx/~constraint. Coello Coello, C. (2012b). List of references on evolutionary multiobjective optimization, www.lania.mx/~ccoello/EM00/EM00bib.html.

Coello Coello, C. and Becerra, R. (2002). Constrained optimization using an evolutionary programming-based cultural algorithm. In Parmee, I., editor, *Adaptive Computing in Design and Manufacture V*, pages 317-328. Springer.

Coello Coello, C. and Becerra, R. (2003). Evolutionary multiobjective optimization using a cultural algorithm. *Swarm Intelligence Symposium*, Indianapolis, Indiana, pages 6-13.

Coello Coello, C , Lamont, G., and Van Veldhuizen, D. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems.* Springer.

Coello Coello, C. and Mezura-Montes, E. (2011). Constraint-handling in nature-inspired numerical optimization: Past, present and future. *Swarm and Evolutionary Computation*, 1(4):173-194.

Coit, D. and Smith, A. (1996). Penalty guided genetic search for reliability design optimization. *Computers and Industrial Engineering*, 30(4):895-904.

Coit, D., Smith, A., and Täte, D. (1996). Adaptive penalty methods for genetic optimization of constrained combinatorial problems. *INFORMS Journal on Computing*, 8(2):173-182.

Collard, P. and Aurand, J. (1994). DGA: An efficient genetic algorithm. *11th European Conference on Artificial Intelligence*, Amsterdam, The Netherlands, pages 487-492.

Collard, P. and Gaspar, A. (1996). "Royal-road" landscapes for a dual genetic algorithm. *12th European Conference on Artificial Intelligence*, Budapest, Hungary, pages 213-217.

Collette, Y. and Siarry, P. (2004). *Multiobjective Optimization: Principles and Case Studies.* Springer.

Colorni, A., Dorigo, M., and Maniezzo, V. (1991). Distributed optimization by ant colonies. *European Conference on Artificial Life*, Paris, Prance, pages 134-142.

Corder, G. and Foreman, D. (2009). *Nonparametric Statistics for Non-Statisticians.* John Wiley & Sons.

Cordon, O., Herrera, F., de Viana, F., and Moreno, L. (2000). A new ACO model integrating evolutionary computation concepts: The best-worst ant system. *From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms,* Brussels, Belgium, pages 22-29.

Corfman, K. and Lehmann, D. (1994). The prisoner's dilemma and the role of information in setting advertising budgets. *Journal of Advertising,* 23(2):35-48.

Courant, R. (1943). Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American Mathematical Society,* 49(l):l-23.

Cover, T. and Thomas, J. (1991). *Elements of Information Theory.* Wiley-Interscience

Cramer, N. (1985). A representation for the adaptive generation of simple sequential programs. *International Conference on Genetic Algorithms and Their Application,* Pittsburgh, Pennsylvania, pages 183-187.

Crepinsek, M., Liu, S.-H., and Mernik, L. (2012). A note on teaching-learning-based optimization algorithm. *Information Sciences,* 212:79-93.

Crepinsek, M., Liu, S.-H., and Mernik, M. (2013). Replication and comparison of computational experiments in applied evolutionary computing: Common pitfalls and guidelines to avoid them. *Information Sciences,* submitted for publication.

Culberson, J. (1998). On the futility of blind search. *Evolutionary Computation,* 6(2):109-127.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection, or The Preservation of Favoured Races in the Struggle for Life.* John Murray, Albemarle Street.

Darwin, C , Neve, M., and Messenger, S. (2002). *Autobiographies.* Penguin Classics.

Das, S., Biswas, A., Dasgupta, S., and Abraham, A. (2009). Bacterial foraging optimization algorithm: Theoretical foundations, analysis, and applications. In Abraham, A., Hassanien, A.-E., Siarry, P., and Engelbrecht, A., editors, *Foundations of Computational Intelligence - Volume 3: Global Optimization,* pages 23-56. Springer.

Das, S. and Suganthan, P. (2011). Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation,* 15(1):4-31.

Das, S., Suganthan, P., and Coello Coello, C. (2011). Guest editorial: Special issue on differential evolution. *IEEE Transactions on Evolutionary Computation,* 15(l):l-3.

Dasgupta, S., Das, S., Abraham, A., and Biswas, A. (2009). Adaptive computational chemotaxis in bacterial foraging optimization: An analysis. *IEEE Transactions on Evolutionary Computation,* 13(4) :919-941.

Davis, L. (1985). Job shop scheduling with genetic algorithms. *International Conference on Genetic Algorithms and Their Application,* Pittsburgh, Pennsylvania, pages 136-140.

Davis, L. and Steenstrup, M. (1987). Genetic algorithms and simulated annealing: An overview. In Davis, L., editor, *Genetic Algorithms and Simulated Annealing,* pages 1-11. Pitman Publishing.

Davis, T. and Principe, J. (1991). A simulated annealing like convergence theory for the simple genetic algorithm. *International Conference on Genetic Algorithms*, San Diego, California, pages 174-181.

Davis, T. and Principe, J. (1993). A Markov chain framework for the simple genetic algorithm. *Evolutionary Computation*, l(3):269-288.

De Bonet, J., Isbell, C , and Viola, P. (1997). MMIC: Finding optima by estimating probability densities. In Mozer, M., Jordan, M., and Petsche, T., editors, *Advances in Neural Information Processing Systems 9*, pages 424-430. MIT Press.

de Franca, F., Coelho, G., Von Zuben, F., and Attux, R. (2008). Multivariate ant colony optimization in continuous search spaces. In *Genetic and Evolutionary Computation Conference*, pages 9-16.

de Garis, H. (1990). Genetic programming: Building artificial nervous systems with genetically programmed neural network modules. *Seventh International Conference on Machine Learning*, Austin, Texas, pages 132-139.

De Jong, K. (1975). *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan.

De Jong, K. (1992). Genetic algorithms are NOT function optimizers. *Second Workshop on Foundations of Genetic Algorithms*, Vail, Colorado, pages 5-17.

De Jong, K. (2002). *Evolutionary Computation.* The MIT Press.

De Jong, K., Fogel, D., and Schwefel, H.-P. (1997). A history of evolutionary computation. In Back, T., Fogel, D., and Michalewicz, Z., editors, *Handbook of Evolutionary Computation*, pages A2.3:l-12. Oxford University Press.

de Oca, M. and Stützle, T. (2008). Convergence behavior of the fully informed particle swarm optimization algorihtm. *Genetic and Evolutionary Computation Conference*, Atlanta, Georgia, pages 71-78.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4) :311-338.

Deb, K. (2009). *Multi-Objective Optimization using Evolutionary Algorithms.* John Wiley & Sons.

Deb, K. and Agrawal, R. (1995). Simulated binary crossover for continuous search space. *Complex Systems*, 9(2): 115-148.

Deb, K. and Agrawal, S. (1999). A niched-penalty approach for constraint handling in genetic algorithms. *International Conference on Artificial Neural Nets and Genetic Algorithms*, Portoroz, Slovenia, pages 235-242.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN VI*, pages 849-858. Springer.

Deb, K. and Goldberg, D. (1989). An investigation of niche and species formation in genetic function optimization. *International Conference on Genetic Algorithms,* Fairfax, Virginia, pages 42-50.

Deb, K., Mohan, M., and Mishra, S. (2005). Evaluating the e-domination based multiobjective evolutionary algorithm for a quick computation of Pareto-optimal solutions. *Evolutionary Computation,* 13(4): 501-525.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002a). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation,* 6(2):182-197.

Deb, K., Pratap, A., and Meyarivan, T. (2001). Constrained test problems for multiobjective evolutionary optimization. In Zitzler, E., Deb, K., Thiele, L., Coello Coello, C , and Corne, D., editors, *Evolutionary Multi-Criterion Optimization: First International Conference, EMO 2001,* pages 284-298. Springer.

Deb, K., Thiele, L., Laumanns, M., and Zitzler, E. (2002b). Scalable multi-objective optimization test problems. *World Congress on Computational Intelligence,* Honolulu, Hawaii, pages 825-830.

Dechter, R. (2003). *Constraint Processing.* Morgan Kaufmann.

Deep, K. and Thakur, M. (2007). A new crossover operator for real coded genetic algorithms. *Applied Mathematics and Computation,* 188(1):895-911.

del Valle, Y., Venayagamoorthy, G., Mohagheghi, S., Hernandez, J.-C, and Harley, R. (2008). Particle swarm optimization: Basic concepts, variants and applications in power systems. *IEEE Transactions on Evolutionary Computation,* 12(2): 171-195.

Delahaye, J.-P. and Mathieu, P. (1995). Complex strategies in the iterated prisoner's dilemma. In Albert, A., editor, *Chaos and Society,* pages 283-292. IOS Press.

Delsuc, F. (2003). Army ants trapped by their evolutionary history. *Public Library of Science Biology,* l(2):e37.

Dembski, W. and Marks, R. (2009a). Bernoulli's principle of insufficient reason and conservation of information in computer search. *IEEE Conference on Systems, Man and Cybernetics,* San Antonio, Texas, pages 2647-2652.

Dembski, W. and Marks, R. (2009b). Conservation of information in search: Measuring the cost of success. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans,* 39(5): 1051-1060.

Dembski, W. and Marks, R. (2010). The search for a search: Measuring the information cost of higher level search. *Journal of Advanced Computational Intelligence and Intelligent Informatics,* 14(5):475-486.

Demetrescu, C. (2012). 9th DIMACS Implementation Challenge - Shortest Paths, www. d i s . u n i r o m a l . i t / c h a l l e n g e 9 .

Deneubourg, J.-L., Aron, S., Goss, S., and Pasteeis, J. (1990). The self-organizing exploratory pattern of the Argentine ant. *Journal of Insect Behavior,* 3(2): 159-168.

DePaulo, B., Kashy, D., Kirkendol, S., Wyer, M., and Epstein, J. (1996). Lying in everyday life. *Journal of Personality and Social Psychology*, 70(5):979-995.

Devroye, L. (1978). Progressive global random search of continuous functions. *Mathematical Programming*, 15(l):330-342.

Di Pietro, A., While, L., and Barone, L. (2004). Applying evolutionary algorithms to problems with noisy, time-consuming fitness functions. *IEEE Congress on Evolutionary Computation*, Portland, Oregon, pages 1254-1261.

Dominguez, J. and Pulido, G. (2011). A comparison on the search of particle swarm optimization and differential evolution on multi-objective optimization. *IEEE Congress on Evolutionary Computation*, New Orleans, Louisiana, pages 1978-1985.

Doran, R. (2007). The gray code. *Journal of Universal Computer Science*, 13(11):1573- 1597.

Dorigo, M., Birattari, M., and Stützle, T. (2006). Ant colony optimization: Artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine*, l(4):28-39.

Dorigo, M. and Gambardella, L. (1997a). Ant colonies for the traveling salesman problem. *BioSystems*, 43(2):73-81.

Dorigo, M. and Gambardella, L. (1997b). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, l(l):53-66.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 26(1):29-41.

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization.* The MIT Press.

Dorigo, M. and Stützle, T. (2010). Ant colony optimization: Overview and recent advances. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 227-263. Springer.

Droste, S., Jansen, T., and Wegener, I. (2002). On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1-2):51-81.

Du, D., Simon, D., and Ergezer, M. (2009). Biogeography-based optimization combined with evolutionary strategy and immigration refusal. *IEEE Conference on Systems, Man, and Cybernetics*, San Antonio, Texas, pages 1023-1028.

Duan, Q., Gupta, V., and Sorooshian, S. (1993). Shuffled complex evolution approach for effective and efficient global minimization. *Journal of Optimization Theory and Applications*, 76(3):501-521.

Duan, Q., Sorooshian, S., and Gupta, V. (1992). Effective and efficient global optimization for conceptual rainfall-runoff models. *Water Resources Research*, 28(4): 1015-1031.

Ducheyne, E., De Baets, B., and De Wulf, R. (2003). Is fitness inheritance useful for realworld applications? *Second International Conference on Evolutionary Multi-Criterion Optimization*, Faro, Portugal, pages 31-42.

Dueck, G. (1993). New optimisation heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104(86):86-92.

Dueck, G. and Scheuer, T. (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90(1):161-175.

Dunham, B., Pridshal, D., Pridshal, R., and North, J. (1963). Design by natural selection. *Synthese*, 15(2):254-259.

Durham, W. (1992). *Coevolution: Genes, Culture, and Human Diversity.* Stanford University Press.

Dyson, G. (1998). *Darwin Among the Machines.* Basic Books. Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. *International Symposium on Micro Machine and Human Science*, Nagoya, Japan, pages 39-43.

Eberhart, R. and Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. *IEEE Congress on Evolutionary Computation*, San Diego, California, pages 84-88.

Eberhart, R. and Shi, Y. (2001). Particle swarm optimization: Developments, applications and resources. *IEEE Congress on Evolutionary Computation*, Seoul, Korea, pages 81-86.

Edgeworth, F. (1881). *Mathematical Physics.* Kegan Paul.

Ehrgott, M. (2005). *Multicriteria Optimization.* Springer.

Ehrnborg, C. and Rosén, T. (2009). The psychology behind doping in sport. *Growth Hormone & IGF Research*, 19(4):285-287.

Eiben, A. (2000). Multiparent recombination. In Back, T., Fogel, D., and Michalewicz, Z., editors, *Evolutionary Computation 1: Basic Algorithms and Operators*, pages 289- 307. Institute of Physics Publishing.

Eiben, A. (2001). Evolutionary algorithms and cnstraint satisfaction: Definitions, survey, methodology, and research directions. In Kallel, L., Naudts, B., and Rogers, A., editors, *Theoretical Aspects of Evolutionary Computating*, pages 13-30. Springer.

Eiben, A. (2003). Multiparent recombination in evolutionary computing. In Ghosh, A. and Tsutsui, S., editors, *Advances in Evolutionary Computing*, pages 175-192. Springer-Verlag.

Eiben, A. and Back, T. (1998). Empirical investigation of multiparent recombination operators in evolution strategies. *Evolutionary Computation*, 5(3):347-365.

Eiben, A. and Schippers, C. (1996). Multi-parent's niche: n-ary crossovers on nklandscapes. In Ebeling, W., Rechenberg, I., Schwefel, H.-P., and Voigt, H.-M., editors, *Parallel Problem Solving from Nature - PPSN IV*, pages 319-328. Springer.

Eiben, A. and Smit, S. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1): 19-31.

Eiben, A. and Smith, J. (2010). *Introduction to Evolutionary Computing.* Springer.

Elbeltagi, E., Hegazy, T., and Grierson, D. (2005). Comparison among five evolutionarybased optimization algorithms. *Advanced Engineering Informatics,* 19(l):43 53.

Ellis, T. and Yao, X. (2007). Evolving cooperation in the non-iterated prisoner's dilemma: A social network inspired approach. *IEEE Congress on Evolutionary Computation,* Singapore, pages 736-743.

Elton, C. (1958). *Ecology of Invasions by Animals and Plants.* Chapman & Hall. Emre, E. and Knowles, G. (1987). A Newton-like approximation algorithm for the steadystate solution of the Riccati equation for time-varying systems. *Optimal Control Applications and Methods,* 8(2): 191-197.

Engelbrecht, A. (2003). *Computational Intelligence.* John Wiley *L·* Sons.

English, T. (1999). Some information theoretic results on evolutionary optimization. *IEEE Congress on Evolutionary Computation,* Washington, District of Columbia, pages 788-795.

Ergezer, M. (2011). Oppositional biogeography-based optimization. Technical report, Cleveland State University. Doctoral dissertation proposal, unpublished.

Ergezer, M. and Simon, D. (2011). Oppositional biogeography-based optimization for combinatorial problems. *IEEE Congress on Evolutionary Computation,* New Orleans, Louisiana, pages 1496-1503.

Ergezer, M., Simon, D., and Du, D. (2009). Oppositional biogeography-based optimization. *IEEE Conference on Systems, Man, and Cybernetics,* San Antonio, Texas, pages 1035-1040.

Erol, O. and Eksin, I. (2006). New optimization method: Big bang-big crunch. *Advances in Engineering Software,* 37(2): 106-111.

Eshelman, L., Caruana, R., and Schaffer, J. (1989). Biases in the crossover landscape. *International Conference on Genetic Algorithms,* Fairfax, Virginia, pages 10-19.

Eshelman, L. and Schaffer, J. (1993). Real-coded genetic algorithms and interval schemata. In Whitley, D., editor, *Foundations of Genetic Algorithms 2,* pages 187-202.

Morgan Kaufmann. Eskandari, H. and Geiger, C. (2008). A fast Pareto genetic algorithm approach for solving expensive multiobjective optimization problems. *Journal of Heuristics,* 14(3):203-241.

Eusuff, M. and Lansey, K. (2003). Optimization of water distribution network design using the shuffled frog leaping algorithm (SFLA). *Journal of Water Resources Planning and Management,* 129(3):210-225.

Eusuff, M., Lansey, K., and Pasha, F. (2006). Shuffled frog-leaping algorithm: A memetic meta-heuristic for discrete optimization. *Engineering Optimization,* 38(2): 129-154.

Evans, M., Hastings, N., and Peacock, B. (2000). *Statistical Distributions.* Wiley-Interscience.

Farmani, R. and Wright, J. (2003). Self-adaptive fitness formulation for constrained optimization. *IEEE Transactions on Evolutionary Computation,* 7(5):445-455.

Fausett, L. (1994). *Fundamentals of Neural Networks.* Prentice Hall.

Fealy, M. (2006). *The Great Pawn Hunter Chess Tutorial.* AuthorHouse.

Feoktistov, V. (2006). *Differential Evolution: In Search of Solutions.* Springer.

Fernandes, M., Martins, T., and Rocha, A. (2009). Fish swarm intelligent algorithm for bound constrained global optimization. *International Conference on Computational and Mathematical Methods in Science and Engineering,* Gijon, Spain.

Fish, F. (1995). Kinematics of ducklings swimming in formation: Consequences of position. *Journal of Experimental Zoology,* 273(1):1-11.

Fleming, P., Purshouse, R., and Lygoe, R. (2005). Many-objective optimization: An engineering design perspective. In Coello Coello, C., Hernandez Aguirre, A., and Zitzler, E., editors, *Evolutionary Multi-Criterion Optimization,* pages 14-32. Springer.

Fletcher, R. and Powell, M. (1963). A rapidly convergent descent method for minimization. *The Computer Journal,* 6(2): 163-168.

Floudas, C. and Pardalos, P. (1990). *A Collection of Test Problems for Constrained Global Optimization Algorithms.* Springer.

Floudas, C., Pardalos, P., Adjiman, C., Esposito, W., Gümüs, Z., Harding, S., Klepeis, J., Meyer, C., and Schweiger, C. (2010). *Handbook of Test Problems in Local and Global Optimization.* Springer.

Fogel, D. (1988). An evolutionary aproach to the traveling salesman problem. *Biological Cybernetics,* 60(2): 139-144.

Fogel, D. (1990). A parallel pocessing approach to a multiple traveling salesman problem using evolutionary programming. *Fourth Annual Parallel Processing Symposium,* Fullerton, California, pages 318-326.

Fogel, D., editor (1998). *Evolutionary Computation: The Fossil Record.* Wiley-IEEE Press.

Fogel, D. (2000). What is evolutionary computation? *IEEE Spectrum,* 37(2):26-32.

Fogel, D. (2006). George Friedman - Evolving circuits for robots. *IEEE Computational Intelligence Magazine,* l(4):52-54.

Fogel, D. and Anderson, R. (2000). Revisiting Bremermann's genetic algorithm: I. Simultaneous mutation of all parameters. *IEEE Congress on Evolutionary Computation,* San Diego, California, pages 1204-1209.

Fogel, L. (1999). *Intelligence through Simulated Evolution: Forty Years of Evolutionary Programming.* John Wiley *L·* Sons.

Fogel, L., Owens, A., and Walsh, M. (1966). *Artificial Intelligence through Simulated Evolution.* John Wiley & Sons.

Fonseca, C. and Fleming, P. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. *International Conference on Genetic Algorithms,* Urbana-Champaign, Illinois, pages 416-423.

Fonseca, C. and Fleming, P. (1995). An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1): 1-16.

Formato, R. (2007). Central force optimization: A new metaheuristic with applications in applied electromagnetics. *Progress in Electromagnetics Research*, 77:425-491.

Formato, R. (2008). Central force optimization: A new nature inspired computational framework for multidimensional search and optimization. In Krasnogor, N., Nicosia, G., Pavone, M., and Pelta, D., editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, pages 221-238. Springer.

Forsyth, R. (1981). BEAGLE - A Darwinian approach to pattern recognition. *Kybernetes*, 10(3): 159-166.

Fourman, M. (1985). Compaction of symbolic layout using genetic algorithms. *International Conference on Genetic Algorithms*, Pittsburgh, Pennsylvania, pages 141-153.

Fox, B. and McMahon, M. (1991). Genetic operators for sequencing problems. In Rawlins, G., editor, *Foundations of Genetic Algorithms*, pages 284-300. Morgan Kaufmann Publishers.

Francçis, 0 . (1998). An evolutionary strategy for global minimization and its Markov chain analysis. *IEEE Transactions on Evolutionary Computation*, 2(3):77-90.

Fraser, A. (1957). Simulation of genetic systems by automatic digital computers: I. Introduction. *Australian Journal of Biological Sciences*, 10(3):484-491.

Friedberg, R. (1958). A learning machine: Part I. *IBM Journal of Research and Development*, 2(1):2-13.

Friedberg, R., Dunham, B., and North, J. (1958). A learning machine: Part II. *IBM Journal of Research and Development*, 3(3):282-287.

Friedman, G. (1998). Selective feedback computers for engineering synthesis and nervous system analogy. In Fogel, D., editor, *Evolutionary Computation: The Fossil Record*, pages 30-84. Wiley-IEEE Press.

Furuta, H., Maeda, K., and Watanabe, E. (1995). Application of genetic algorithm to aesthetic eesign of bridge structures. *Computer-Aided Civil and Infrastructure Engineering*, 10(6):415-421.

Galinier, P., Hamiez, J.-P., Hao, J.-K., and Porumbel, D. (2013). Recent advances in graph vertex coloring. In Zelinka, L, Snâsel, V., and Abraham, A., editors, *Handbook of Optimization*, **ebooks**. **com.**

Gallagher, M., Wood, I., Keith, J., and Sofronov, G. (2007). Bayesian inference in estimation of distribution algorithms. *IEEE Congress on Evolutionary Computation*, Singapore, pages 127-133.

Gambardella, L. and Dorigo, M. (1995). Ant-Q: A reinforcement learning approach to the traveling salesman problem. *Twelfth International Conference on Machine Learning*, Tahoe City, California, pages 252-260.

Gandomi, A. and Alavi, A. (2012). Krill herd: A new bio-inspired optimization algorithm. *Communications in Nonlinear Science and Numerical Simulation*, 17(12) :4831-4845.

Gathercole, C. and Ross, P. (1994). Dynamic training subset selection for supervised learning in genetic programming. In Davidor, Y., Schwefel, H.-P., and Männer, R., editors, *Parallel Problem Solving from Nature - PPSN HI,* pages 312-321. Springer.

Gathercole, C. and Ross, P. (1997). Small populations over many generations can beat large populations over few generations in genetic programming. *Second Annual Conference on Genetic Programming,* Palo Alto, California, pages 111-118.

Geem, Z., editor (2010a). *Harmony Search Algorithms for Structural Design Optimization.* Springer.

Geem, Z., editor (2010b). *Music-Inspired Harmony Search Algorithm.* Springer.

Geem, Z. (2010c). *Recent Advances in Harmony Search Algorithm.* Springer.

Geem, Z., Kim, J.-H., and Loganathan, G. (2001). A new heuristic optimization algorithm: Harmony search. *Simulation,* 76(2):60-68.

Geisser, S. (1993). *Predictive Inference.* Chapman *h* Hall.

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 6(6):721-741.

Gendreau, M. (2003). An introduction to tabu search. In Glover, F. and Kochenberger, G., editors, *Handbook of Metaheuristics,* pages 37-54. Springer.

Gendreau, M. and Potvin, J.-Y. (2010). Tabu search. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics,* pages 41-59. Springer.

Giraldeau, L.-A. and Caraco, T. (2000). *Social Foraging Theory.* Princeton University Press.

Glover, F. and Laguna, M. (1998). *Tabu Search.* Springer.

Glover, F. and McMillan, C. (1986). The general employee scheduling problem: An integration of MS and AI. *Computers and Operations Research,* 13(5):563-573.

Goh, C. and Tan, K. (2007). An investigation on noisy environments in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation,* 11(3):354- 381.

Golan, J. (2007). *The Linear Algebra a Beginning Graduate Student Ought to Know.* Springer.

Goldberg, D. (1989a). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley.

Goldberg, D. (1989b). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems,* 3(5):493-530.

Goldberg, D. (1991). Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems,* 5(2): 139-167.

Goldberg, D. and Lingle, R. (1985). Alleles, loci, and the traveling salesman problem. *International Conference on Genetic Algorithms and Their Application,* Pittsburgh, Pennsylvania, pages 154-159.

Gomez, J., Barrera, J., Rojas, J., Macias-Samano, J., Liedo, J., Cruz-Lopez, L., and Badii, M. (2005). Volatile compounds released by disturbed females of Cephalonomia stephanoderis (Hymenoptera: Bethylidae): A parasitoid of the coffee berry borer Hypothenemus hampei (Coleoptera: Scolytidae). *Florida Entomologist*, 88(2): 180-187.

Gonzalez, C , Lozano, J., and Larranaga, P. (2000). Analyzing the PBIL algorithm by means of discrete dynamical systems. *Complex Systems*, 12(4):465-479.

Gonzalez, C , Lozano, J., and Larranaga, P. (2001). The convergence behavior of the PBIL algorithm: A preliminary approach. In Kurkova, V., Steele, N., Neruda, R., and Kârny, M., editors, *Artificial Neural Nets and Genetic Algorithms*, pages 228-231. Springer-Ver lag.

Gonzalez, C., Lozano, J., and Larranaga, P. (2002). Mathematical modeling of discrete estimation of distribution algorithms. In Larranaga, P. and Lozano, J., editors, *Estimation of Distribution Algorithms*, pages 147-163. Kluwer Academic Publishers.

Good, P. and Hardin, J. (2009). *Common Errors in Statistics.* John Wiley & Sons, 3rd edition.

Goss, S., Aron, S., Deneubourg, J., and Pasteeis, J. (1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76(12):579-581.

Gotelli, N. (2008). *A Primer of Ecology.* Sinauer Associates.

Gray, R. (2011). *Entropy and Information Theory.* Springer.

Greene, M. and Gordon, D. (2007). Structural complexity of chemical recognition cues affects the perception of group membership in the ants Linephithema humile and Aphaenogaster cockerelli. *Journal of Experimental Biology*, 210(5):897-905.

Grefenstette, J., Gopal, R., Rosmaita, B., and Van Gucht, D. (1985). Genetic algorithms for the TSP. *International Conference on Genetic Algorithms and Their Application*, Cambridge, Massachusetts, pages 160-165.

Gregory, R. and Karney, D. (1969). *A Collection of Matrices for Testing Computational Algorithms.* John Wiley & Sons.

Grieco, J. (1988). Realist theory and the problem of international cooperation: Analysis with an amended prisoner's dilemma model. *The Journal of Politics*, 50(3):600-624.

Grinstead, C. and Snell, J. (1997). *Introduction to Probability.* American Mathematical Society.

Grötschel, M. and Padberg, M. (2001). The optimized odyssey. *AIROnews*, 6(2): 1-7.

Guntsch, M. and Middendorf, M. (2002). Applying population based ACO to dynamic optimization problems. *Third International Workshop on Ant Algorithms*, Brussels, Belgium, pages 111-122.

Gustafson, S. and Burke, E. (2006). Speciating island model: An alternative parallel evolutionary algorithm. *Parallel and Distributed Computing*, 66(8): 1025-1036.

Gutierrez, A., Lanza, M., Barriuso, I., Valle, L., Domingo, M., Perez, J., and Basterrechea, J. (2002). Comparison of different PSO initialization techniques for high dimensional search

space problems: A test with FSS and antenna arrays. *5th European Conference on Antennas and Propagation*, Rome, Italy, pages 965-969.

Gutin, G. and Punnen, A., editors (2007). *The Traveling Salesman Problem and Its Variations.* Springer.

Gutjahr, W. (2000). A graph-based ant system and its convergence. *Future Generation Computer Systems*, 16(9):873-888.

Gutjahr, W. (2008). First steps to the runtime complexity analysis of ant colony optimization. *Computers & Operations Research*, 35(9) :2711-2727.

Hadj-Alouane, A. and Bean, J. (1993). A genetic algorithm for the multiple choice integer program. Technical report, Department of Industrial &; Operations Engineering, University of Michigan. http://ioe.engin.umich.edu/techrprt/pdf/TR92-50.pdf.

Hadj-Alouane, A. and Bean, J. (1997). A genetic algorithm for the multiple choice integer program. *Operations Research*, 45(1):92-101.

Hajela, P. and Lin, C.-Y. (1997). Genetic search strategies in multicriterion optimal design. *Structural and Multidisciplinary Optimization*, 4(2):99-107.

Hamida, S. and Schoenauer, M. (2000). An adaptive algorithm for constrained optimization problems. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN VI*, pages 529-538. Springer.

Hamida, S. and Schoenauer, M. (2002). ASCHEA: New results using adaptive segregational constraint handling. *IEEE Congress on Evolutionary Computation*, Honolulu, Hawaii, pages 884-889.

Hamilton, W. (1971). Geometry for the selfish herd. *Journal of Theoretical Biology*, 31(2):295-311.

Hansen, N. (2010). The CMA evolution strategy: A comparing review. In Lozano, J., Larranga, P., Inza, I., and Bengoetxea, E., editors, *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms*, pages 75-102. Springer.

Hansen, N., Müller, S., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1): 1-18.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2): 159-195.

Hanski, I. (1999). Habitat connectivity, habitat continuity, and metapopulations in dynamic landscapes. *Oikos*, 87(2):209-219.

Hanski, I. and Gilpin, M. (1997). *Metapopulation Biology.* Academic Press.

Hao, J.-K. and Middendorf, M., editors (2012). *Evolutionary Computation in Combinatorial Optimization.* Springer.

Harding, S. (2006). *Animate Earth.* Chelsea Green Publishing Company.

Harik, G. (1995). Finding multimodal solutions using restricted tournament selection. *International Conference on Genetic Algorithms*, Pittsburgh, Pennsylvania, pages 24-31.

Harik, G. (1999). Linkage learning via probabilistic modeling in the ECGA. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois. IlliGAL Report No. 99010.

Harik, G., Lobo, F., and Goldberg, D. (1999). The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):287-297.

Harik, G., Lobo, F., and Sastry, K. (2010). Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ecga). In Pelikan, M., Sastry, K., and Cantu- Paz, E., editors, *Scalable Optimization via Probabilistic Modeling*, pages 39-62. Springer.

Harrald, P. and Fogel, D. (1996). Evolving continuous behaviors in the iterated prisoner's dilemma. *Biosystems*, 37(1-2): 135-145.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning.* Springer, 2nd edition.

Hastings, A. and Higgins, K. (1994). Persistence of transients in spatially structured models. *Science*, 263(5150):1133-1136.

Hastings, W. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97-109.

Hatzakis, I. and Wallace, D. (2006). Dynamic multi-objective optimization with evolutionary algorithms: A forward-looking approach. *Genetic and Evolutionary Computation Conference*, Seattle, Washington, pages 1201-1208.

Haupt, R. and Haupt, S. (2004). *Practical Genetic Algorithms.* John Wiley &; Sons, 2nd edition.

Hauptman, A., Elyasaf, A., Sipper, M., and Karmon, A. (2009). GP-Rush: Using genetic programming to evolve solvers for the rush hour puzzle. *Genetic and Evolutionary Computation Conference*, Montreal, Canada, pages 955-962.

Hauptman, A. and Sipper, M. (2007). Evolution of an efficient search algorithm for the mate-in-n problem in chess. *European Conference on Genetic Programming*, Valencia, Spain, pages 78-89.

He, S., Wu, Q., and Saunders, J. (2009). Group search optimizer: An optimization algorithm inspired by animal searching behavior. *IEEE Transactions on Evolutionary Computation*, 13(5):973-990.

Heinrich, B. (2002). *Why We Run.* Harper Perennial.

Helwig, S. and Wanka, R. (2008). Theoretical analysis of initial particle swarm behavior. In Rudolph, G., Jansen, T., Lucas, S., Poloni, C., and Beume, N., editors, *Parallel Problem Solving from Nature - PPSN X*, pages 889-898. Springer.

Henderson, D., Jacobson, S., and Johnson, A. (2003). The theory and practice of simulated annealing. In Glover, F. and Kochenberger, G., editors, *Handbook of Metaheuristics*, pages 287-320. Springer.

Herrera, F., Lozano, M., and Verdegay, J. (1998). Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4) :265-319.

Hofmeyr, S. and Forrest, S. (2000). Architecture for an artificial immune system. *Evolutionary Computation*, 8(4):443-473.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* The University of Michigan Press.

Hölldobler, B. and Wilson, E. (1990). *The Ants.* The Belknap Press of Harvard University Press.

Hölldobler, B. and Wilson, E. (1994). *Journey to the Ants.* The Belknap Press of Harvard University Press.

Hölldobler, B. and Wilson, E. (2008). *The Superorganism: The Beauty, Elegance, and Strangeness of Insect Societies.* W. W. Norton & Company.

Homaifar, A., Qi, C., and Lai, S. (1994). Constrained optimization via genetic algorithms. *Simulation*, 62(4):242-253.

Hooker, J. (1995). Testing heuristics: We have it all wrong. *Journal of Heuristics*, l(l):33-42.

Horn, J., Nafpliotis, N., and Goldberg, D. (1994). A niched Pareto genetic algorithm for multiobjective optimization. *IEEE Conference on Evolutionary Computation*, Orlando, Florida, pages 82-87.

Home, E. and Jaeger, R. (1988). Territorial pheromones of female red-backed salamanders. *Ethology*, 78(2): 143-152.

Horoba, C. and Neumann, F. (2010). Approximating Pareto-optimal sets using diversity strategies in evolutionary multi-objective optimization. In Coello Coello, C , Dhaenens, C, and Jourdan, L., editors, *Advances in Multi-Objective Nature Inspired Computing*, pages 23-44. Springer.

Houck, C , Joines, J., and Kay, M. (1995). A genetic algorithm for function optimization: A Matlab implementation. Technical report, North Carolina State University.

Hsiao, Y.-T., Chuang, C.-L., Jiang, J.-A., and Chien, C.-C. (2005). A novel optimization algorithm: Space gravitational optimization. *IEEE International Conference on Systems, Man and Cybernetics*, Waikoloa, Hawaii, pages 2323-2328.

Hu, T., Harding, S., and Banzhaf, W. (2010). Variable population size and evolution acceleration: A case study with a parallel evolutionary algorithm. *Genetic Programming and Evolvable Machines*, ll(2):205-225.

Huang, H. and Wang, F. (2002). Fuzzy decision-making design of chemical plant using mixed-integer hybrid differential evolution. *Computers and Chemical Engineering*, 26(12):1649-1660.

Huang, V., Qin, A., Deb, K., Zitzler, E., Suganthan, P., Liang, J., Preuss, M., and Huband, S. (2007). Problem definitions for performance assessment on multi-objective optimization algorithms. Technical report. www.ntu.edu.sg/home/EPNSugan/index_ files/cec-benchmarking.htm.

Huff, D. and Geis, I. (1993). *How to Lie with Statistics.* W. W. Norton & Company.

Iba, H. and de Garis, H. (1996). Extending genetic programming with recombinative guidance. In Angeline, P. and Kinnear, K., editors, *Advances in Genetic Programming: Volume 2*, pages 69-88. The MIT Press.

Igelnik, B. and Simon, D. (2011). The eigenvalues of a tridiagonal matrix in biogeography. *Applied Mathematics and Computation,* 218(1): 195-201.

Ingber, L. (1996). Adaptive simulated annealing: Lessons learned. *Control and Cybernetics*, 25(l):33-54.

Ito, K., Akagi, S., and Nishikawa, M. (1983). A multiobjective optimization approach to a design problem of heat insulation for thermal distribution piping network systems. *Journal of Mechanisms, Transmissions, and Automation in Design,* 105(2):206-213.

Jaszkiewicz, A. and Zielniewicz, P. (2006). Pareto memetic algorithm with path relinking for bi-objective traveling salesperson problem. *European Journal of Operational Research,* 193(3) :885-890.

Jayalakshmi, G., Sathiamoorthy, S., and Rajaram, R. (2001). A hybrid genetic algorithm - A new approach to solve traveling salesman problem. *International Journal of Computational Engineering Science,* 2(2):339-355.

Jefferson, D., Collins, R., Cooper, C , Dyer, M., Flowers, M., Korf, R., Taylor, C , and Wang, A. (2003). Evolution as a theme in artificial life: The genesys/tracker system. In Langton, C , Taylor, C , Farmer, J., and Rasmussen, S., editors, *Artificial Life II,* pages 549-578. Westview Press.

Jensen, T. and Toft, B. (1994). *Graph Coloring Problems.* John Wiley & Sons.

Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing,* 9(1):3-12.

Jin, Y. and Branke, J. (2005). Evolutionary optmization in uncertain environments – A survey. *IEEE Transactions on Evolutionary Computation,* 9(3):303-317.

Jin, Y., Hüskin, M., and Sendhoff, B. (2003). Quality measures for approximate models in evolutionary computation. *Genetic and Evolutionary Computation Conference,* Chicago, Illinois, pages 170-173.

Jofré, P., Reisenegger, A., and Fernandez, R. (2006). Constraining a possible time variation of the gravitational constant through "gravitochemical heating" of neutron stars. *Physical Review Letters,* 97(13):131102.

Johnson, D. (1999). The insignificance of statistical significance testing. *Journal of Wildlife Management,* 63(3):763-772.

Joines, J. and Houck, C. (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's. *IEEE World Congress on Computational Intelligence,* Orlando, Florida, pages 579-584.

Jones, D., Schonlau, M., and Welch, W. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization,* 13(4):455-492.

Joslin, D. and Clements, D. (1999). Squeaky wheel optimization. *Journal of Artificial Intelligence Research*, 10:353-373.

Kanji, G. (2006). *100 Statistical Tests.* Sage Publications.

Karaboga, D. and Akay, B. (2009). A comparative study of artificial bee colonyalgorithm. *Applied Mathematics and Computation,* 214(1):108-132.

Karaboga, D. and Bastürk, B. (2007). A powerful and efficient agorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *Journal of Global Optimization,* 39(3):459-471.

Karaboga, D. and Bastürk, B. (2008). On the performance of artificial bee colony (ABC) algorithm. *Applied Soft Computing,* 8(l):687-697.

Karaboga, D., Gorkemli, B., Ozturk, C , and Karaboga, N. (2013). A comprehensive survey: Artificial bee colony (ABC) algorithm and applications. *Artificial Intelligence Review,* in print. Kaveh, A. and Talatahari, S. (2010). A novel heuristic optimization method: Charged system search. *Ada Mechanica,* 213(3-4):267-289.

Kazarlis, S. and Petridis, V. (1998). Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms. In Eiben, A., Back, T., Schoenauer, M., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN V,* pages 211-220. Springer.

Keel, L. and Bhattacharyya, S. (1997). Robust, fragile, or optimal? *IEEE Transactions on Automatic Control,* 42(8): 1098-1105.

Kemeny, J., Snell, J., and Thompson, G. (1974). *Introduction to Finite Mathematics.* Prentice-Hall.

Kennedy, J. (1998). Thinking is social: Experiments with the adaptive culture model. *Journal of Conflict Resolution,* 42(l):56-76.

Kennedy, J. and Eberhart, R. (1997). A discrete binary version of the particle swarm algorithm. *IEEE Conference on Systems, Man, and Cybernetics,* Orlando, Florida, pages 4104-4109.

Kennedy, J. and Eberhart, R., editors (2001). *Swarm Intelligence.* Morgan Kaufmann.

Kern, S., Müller, S., Hansen, N., Bûche, D., Ocenasek, J., and Koumoutsakos, P. (2004). Learning probability distributions in continuous evolutionary algorithms – A comparative review. *Natural Computing,* 3(1):77-112.

Keynes, R., editor (2001). *Charles Darwin's Beagle diary.* Cambridge University Press.

Khare, V., Yao, X., and Deb, K. (2003). Performance scaling of multi-objective evolutionary algorithms. In Fonseca, C , Fleming, P., Zitzler, E., Thiele, L., and Deb, K., editors, *Evolutionary Multi-Criterion Optimization: Second International Conference, EMO 2003,* pages 376-390. Springer.

Khatib, W. and Fleming, P. (1998). The stud G A: A mini revolution? In Eiben, A., Back, T., Schoenauer, M., and Schwefel, H.-P., editors, *Parallel Problem Solving rom Nature - PPSN V,* pages 683-691. Springer.

Kim, D. (2006). Memory analysis and significance test for agent behaviours. *Genetic and Evolutionary Computation Conference,* Seattle, Washington, pages 151-158.

Kim, H.-S. and Cho, S.-B. (2000). Application of interactive genetic algorithm to fashion design. *Engineering Applications of Artificial Intelligence,* 13(6):635-644.

Kinnear, K. (1993). Evolving a sort: Lessons in genetic programming. *International Conference on Neural Networks,* San Francisco, California, pages 881-888.

Kinnear, K. (1994). Alternatives in automatic function definition: A comparison of performance. In Kinnear, K., editor, *Advances in Genetic Programming,* pages 119 141. MIT Press.

Kirk, D., editor (2004). *Optimal Control Theory.* Dover.

Kirkpatrick, S., Gelatt, C , and Vecchi, M. (1983). Optimization by simmulated annealing. *Science,* 220(4598):671-680.

Kjellström, G. (1969). Network optimization by random variation of component values. *Ericsson Technics,* 25(3):133-151.

Kleidon, A. (2004). Amazonian biogeography as a test for Gaia. In Schneider, S., Miller, J., Crist, E., and Boston, P., editors, *Scientists Debate Gaia,* pages 291-296. MIT Press.

Knowles, J. (2005). ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation,* 10(l):50-66.

Knowles, J. and Corne, D. (2001). Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation,* 8(2): 149-172.

Knowles, J. and Nakayama, H. (2008). Meta-modeling in multiobjective optimization. In Branke, J., Deb, K., Miettinen, K., and Slowinski, R., editors, *Multiobjective Optimization,* pages 245-284. Springer.

Konak, A., Coit, D., and Smith, A. (2006). Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering and System Safety,* 91(9):992-1007.

Kondoh, M. (2006). Does foraging adaptation create the positive complexity-stability relationship in realistic food-web structure? *Journal of Theoretical Biology,* 238(3) :646-651.

Koza, J., editor (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press.

Koza, J., editor (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs.* The MIT Press.

Koza, J. (1997). Classifying protein segments as transmembrane domains using genetic programming and architecture-altering operations. In Back, T., Fogel, D., and Michalewicz, Z., editors, *Handbook of Evolutionary Computation,* pages G6.1:1-5. Oxford University Press.

Koza, J. (2010). Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines,* ll(3-4):251-284.

Koza, J., Al-Sakran, L., and Jones, L. (2008). Automated ab initio synthesis of complete designs of four patented optical lens systems by means of genetic programming. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 22(3):249-273.

Koza, J., Bennett, F., Andre, D., and Keane, M., editors (1999). *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann. Koza, J., Keane, M., Streeter, M., Mydlowec, W., Yu, J., and Lanza, G., editors (2005).

Genetic Programming IV: Routine Human-Competitive Machine Intelligence. The MIT Press.

Koziel, S. and Michalewicz, Z. (1998). A decoder-based evolutionary algorithm for constrained parameter optimization problems. In Eiben, A., Back, T., Schoenauer, M., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN V,* pages 231-240. Springer.

Koziel, S. and Michalewicz, Z. (1999). Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation,* 7(1): 19-44.

Krause, J. and Ruxton, G., editors (2002). *Living in Groups.* Oxford University Press. Krige, D. (1951). A statistical approach to some basic mine valuation problems on the Witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa,* 52(6): 119-139.

Krishnanand, K. and Ghose, D. (2009). Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. *Swarm Intelligence,* 3(2) :87-124.

Krogh, A. (2008). What are artificial neural networks? *Nature Biotechnology,* 6(2): 195- 197.

Kursawe, F. (1991). A variant of evolution strategies for vector optimization. In Schwefel, H.-P. and Männer, R., editors, *Parallel Problem Solving from Nature - PPSN I,* pages 193-197. Springer.

Kvasnicka, V., Pelikan, M., and Pospichal, J. (1996). Hill climbing with learning (an abstraction of genetic algorithm). *Neural Network World,* 6(5):773-796.

Lam, A. and Li, V. (2010). Chemical-reaction-inspired metaheuristic for optimization. *IEEE Transactions on Evolutionary Computation,* 14(3):381-399.

Lampinen, J. (2002). A constraint handling approach for the differential evolution algorithm. *IEEE Congress on Evolutionary Computation,* Honolulu, Hawaii, pages 1468- 1473.

Langdon, W. (2000). Size fair and homologous tree genetic programming crossovers. *Genetic Programming and Evolvable Machines,* l(l-2):95-119.

Langdon, W. and Poli, R., editors (2002). *Foundations of Genetic Programming.* Springer.

Larranaga, P. (2002). A review on estimation of distribution algorithms. In Larranaga, P. and Lozano, J., editors, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation,* pages 57-100. Kluwer Academic Publishers.

Larranaga, P., Etxeberria, R., Lozano, J., and Pena, J. (1999a). Optimization by learning and simulation of Bayesian and Gaussian networks. Technical report, University of the Basque Country, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41. 1895.

Larranaga, P., Etxeberria, R., Lozano, J., and Pena, J. (2000). Combinatorial optimization by learning and simulation of Bayesian networks. *Sixteenth Conference on Uncertainty in Artificial Intelligence*, Stanford, California, pages 343-352.

Larranaga, P., Karshenas, H., Bielza, C , and Santana, R. (2012). A review on probabilistic graphical models in evolutionary computation. *Journal of Heuristics*, 18(5):795-819.

Larranaga, P., Kuijpers, C , Murga, R., Inza, I., and Dizdarevic, S. (1999b). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13(2): 129-170.

Larranaga, P. and Lozano, J., editors (2002). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Kluwer Academic Publishers. Latané, B., Nowak, A., and Liu, J. (1994). Measuring emergent social phenomena: Dynamism, polarization, and clustering as order parameters of social systems. *Behavioral Science*, 39(l):l-24.

Lattimore, T. and Hutter, M. (2011). No free lunch versus Occam's razor in supervised learning. *Solomonoff 85th Memorial Conference*, Melbourne, Australia.

Laumanns, M., Thiele, L., and Zitzler, E. (2003). Running time analysis of evolutionary agorithms on vector-valued pseudo-Boolean functions. *IEEE Transactions on Evolutionary Computation*, 8(2):170-182.

Lawler, E., Lenstra, J., Rinnooy Kan, A., and Shmoys, D., editors (1985). *The Traveling Salesman Problem.* John Wiley *h* Sons.

Le Riche, R., Knopf-Lenoir, C, and Haftka, R. (1995). A segregated genetic algorithm for constrained structural optimization. *International Conference on Genetic Algorithms*, Pittsburgh, Pennsylvania, pages 558-565.

Lee, K. and Geem, Z. (2006). A new meta-heuristic algorithm for continuous engineering optimization: Harmony search theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 194(36-38):3902-3933.

Leguizamon, G. and Coello Coello, C. (2009). Boundary search for constrained numerical optimization problems. In Mezura-Montes, E., editor, *Constraint-Handling in Evolutionary Optimization*, pages 25-49. Springer.

Lehman, J. and Stanley, K. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2): 189-223.

Lenton, T. (1998). Gaia and natural selection. *Nature*, 394(6692) :439-447.

Li, C. and Yang, S. (2008). A generalized approach to construct benchmark problems for dynamic optimization. In Li, X., editor, *Simulated Evolution and Learning*, pages 391-400. Springer.

Li, C., Yang, S., Nguyen, T., Yu, E., Yao, X., Jin, Y., Beyer, H.-G., and Suganthan, P. (2008). Benchmark generator for CEC'2009 competition on dynamic optimization. Technical report, **www.ntu.**edu.**sg/home/EPNSugan/index_files/cec-benchmarking.htm.**

Li, X., Shao, Z., and Qian, J. (2003). An optimizing method based on autonomous animats: Fish-swarm algorithm. *Systems Engineering - Theory & Practice*, 22(11):32- 38.

Li, Y., Zhang, S., and Zeng, X. (2009). Research of multi-population agent genetic algorithm for feature selection. *Expert Systems with Applications,* 36(9): 11570-11581.

Liang, J., Runarsson, T., Mezura-Montes, E., Clerc, M., Suganthan, P., Coello Coello, C, and Deb, K. (2006). Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization. Technical report, **www.ntu.** edu.sg/home/EPNSugan/index_files/cec-benchmarking.htm.

Liang, J., Suganthan, P., and Deb, K. (2005). Novel composition test functions for numerical global optimization. *Swarm Intelligence Symposium,* Pasadena, California, pages 68-75.

Lim, D., Jin, Y., Ong, Y.-S., and Sendhoff, B. (2010). Generalizing surrogate-assisted evolutionary computation. *IEEE Transactions on Evolutionary Computation,* 14(3):329-355.

Lim, D., Ong, Y.-S., Jin, Y., Sendhoff, B., and Lee, B.-S. (2007). Efficient hierarchical parallel genetic algorithms using grid computing. *Future Generation Computer Systems,* 23(4):658-670.

Lima, C. and Lobo, F. (2004). Parameter-less optimization with the extended compact genetic algorithm and iterated local search. *Genetic and Evolutionary Computation Conference,* Seattle, Washington, pages 1328-1339.

Lima, S. (1995). Back to the basics of anti-predatory vigilance: The group-size effect. *Animal Behaviour,* 49(1): 11-20.

Lis, J. and Eiben, A. (1997). A multi-sexual genetic algorithm for multiobjective optimization. *IEEE International Conference on Evolutionary Computation,* Indianapolis, Indiana, pages 59-64.

Löbbing, M. and Wegener, I. (1995). The number of knight's tours equals 33,439,123,484,294 - counting with binary decision diagrams. *Electronic Journal of Combinatorics,* 3(1):5.

Lohn, J., Hornby, G., and Linden, D. (2004). An evolved antenna for deployment on NASA's space technology 5 mission. In O'Reilly, U.-M., Riolo, R., Yu, G., and Worzel, W., editors, *Genetic Programming Theory and Practice II,* pages 301-315. Kluwer Academic Publishers.

Lomolino, M. (2000a). A call for a new paradigm of island biogeography. *Global Ecology and Biogeography,* 9(1): 1-6.

Lomolino, M. (2000b). A species-based theory of insular zoogeography. *Global Ecology and Biogeography,* 9(l):39-58.

Lopez Jaimes, A., Coello Coello, C., and Urias Barrientos, J. (2009). Online objective reduction to deal with many-objective problems. *5th International Conference on Evolutionary Multi-Criterion Optimization,* Nantes, France, pages 423-437.

Lovelock, J. (1990). Hands up for the Gaia hypothesis. *Nature,* 344(6262): 100-102.

Lovelock, J., editor (1995). *Gaia.* Oxford University Press.

Lozano, J., Larranaga, P., Inza, I., and Bengoetxea, E., editors (2006). *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms.* Springer.

Lukasik, S. and Zak, S. (2009). Firefly algorithm for continuous constrained optimization tasks. *1st International Conference on Computational Collective Intelligence*, Wroclaw, Poland, pages 97-106.

Lundy, M. and Mees, A. (1986). Convergence of an annealing algorithm. *Mathematical Programming*, 34(1): 111-124.

Ma, H. (2010). An analysis of the equilibrium of migration models for biogeography-based optimization. *Information Sciences*, 180(18):3444-3464.

Ma, H., Ni, S., and Sun, M. (2009). Equilibrium species counts and migration model tradeoffs for biogeography-based optimization. *IEEE Conference on Decision and Control*, Shanghai, China, pages 3306-3310.

Ma, H. and Simon, D. (2010). Biogeography-based optimization with blended migration for constrained optimization problems. *Genetic and Evolutionary Computation Conference*, Portland, Oregon, pages 417-418.

Ma, H. and Simon, D. (2011a). Analysis of migration models of biogeography-based optimization using markov theory. *Engineering Applications of Artificial Intelligence*, 24(6):1052-1060.

Ma, H. and Simon, D. (2011b). Blended biogeography-based optimization for constrained optimization. *Engineering Applications of Artificial Intelligence*, 24(3):517-525.

Ma, H. and Simon, D. (2013). Variations of biogeography-based optimization and markov analysis. *Information Sciences*, 220:492-506.

Ma, H., Simon, D., and Fei, M. (2013). On the statistical mechanics approximation of biogeography-based optimization. *Submitted for publication.*

MacArthur, R. (1955). Fluctuations of animal populations and a measure of community stability. *Ecology*, 36(3):533-536.

MacArthur, R. and Wilson, E. (1963). An equilibrium theory of insular zoogeography. *Evolution*, 17(4):373-387.

MacArthur, R. and Wilson, E. (1967). *The Theory of Island Biogeography.* Princeton University Press.

Mahfoud, S. (1992). Crowding and preselection revisited. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois. IlliGAL Report No. 92004.

Mahfoud, S. (1995a). A comparison of parallel and sequential niching methods. *International Conference on Genetic Algorithms*, Pittsburgh, Pennsylvania, pages 136-143.

Mahfoud, S. (1995b). Niching methods for genetic algorithms. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois. IlliGAL Report No. 95001.

Mahnig, T. and Mühlenbein, H. (2000). Mathematical analysis of optimization methods using search distributions. *Genetic and Evolutionary Computation Conference*, Las Vegas, Nevada, pages 205-208.

Malisia, A. (2008). Improving the exploration ability of ant-based algorithms. In Tizhoosh, H. and Ventresca, M., editors, *Oppositional Concepts in Computational Intelligence*, pages 121-142. Springer.

Mallipeddi, R. and Suganthan, P. (2010). Problem definitions and evaluation criteria for the CEC 2010 competition on constrained real-parameter optimization. Technical report, Nanyang Technological University. www.ntu.edu.sg/home/EPNSugan/index_files/cec-benchmarking.htm.

Maniezzo, V., Gambardella, L., and de Luigi, F. (2004). Ant colony optimization. In Onwubolu, G. and Babu, R., editors, *New Optimization Techniques in Engineering*, pages 101-122. Springer.

Margulis, L. (1996). Gaia is a tough bitch. In Brockman, J., editor, *The Third Culture: Beyond the Scientific Revolution*, pages 129-151. Touchstone.

Marriott, K. and Stuckey, P. (1998). *Programming with Constraints: An Introduction*. The MIT Press.

Mavrovouniotis, M. and Yang, S. (2011). Ant colony optimization with immigrants schemes in dynamic environments. In Schaefer, R., Cotta, C., Kolodziej, J., and Rudolph, G., editors, *Parallel Problem Solving from Nature - PPSN XI*, pages 371-380. Springer. May, R. (1973). *Stability and Complexity in Model Ecosystems*. Princeton University Press.

McCann, K. (2000). The diversity-stability debate. *Nature*, 405(6783):228-233.

McConaghy, T., Palmers, P., Gielen, G., and Steyaert, M. (2008). Genetic programming with reuse of known designs for industrially scalable, novel circuit design. In Riolo, R., Soûle, T., and Worzel, B., editors, *Genetic Programming Theory and Practice V*, pages 159-184. Springer.

McGill, R., Tukey, J., and Larsen, W. (1978). Variations of box plots. *The American Statistician*, 32(1):12-16.

McNab, B. (2002). *The Physiological Ecology of Vertebrates*. Cornell University.

McTavish, T. and Restrepo, D. (2008). Evolving solutions: The genetic algorithm and evolution strategies for finding optimal parameters. In Smolinski, T., Milanova, M., and Hassanien, A., editors, *Applications of Computational Intelligence in Biology*, pages 55-78. Springer.

Mehrabian, R. and Lucas, C. (2006). A novel numerical optimization algorithm inspired from weed colonization. *Ecological Informatics*, l(4):355-366.

Melab, N., Cahon, S., and Talbi, E.-G. (2006). Grid computing for parallel bioinspired algorithms. *Journal of Parallel and Distributed Computing*, 66(8):1052-1061.

Mendes, R., Kennedy, J., and Neves, J. (2004). The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204-210.

Metropolis, N. (1987). The beginning of the Monte Carlo method. *Los Alamos Science*, 15:125-130.

Metropolis, N., Rosenblut h, A., Rosenblut h, M., Teller, A., and Teller, E. (1953). Equations of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087-1092.

Meuleau, N., Peshkin, L., Kim, K.-E., and Kaelbling, L. (1999). Learning finite-state controllers for partially observable environments. *Conference on Uncertainty in Artificial Intelligence*, Stockholm, Sweden, pages 427-436.

Meuleau02, N. and Dorigo, M. (2002). Ant colony optimization and stochastic gradient descent. *Artificial Life*, 8(2): 103-121.

Mezura-Montes, E. and Coello Coello, C. (2005). A simple multimembered evolution strategy to solve constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, 9(1): 1-17.

Mezura-Montes, E. and Coello Coello, C. (2008). Constrained optimization via multiobjective evolutionary algorithms. In Knowles, J., Corne, D., and Deb, K., editors, *Multiobjective Problem Solving from Nature*, pages 53-75. Springer.

Mezura-Montes, E. and Palomeque-Oritiz, A. (2009). Parameter control in differential evolution for constrained optimization. *IEEE Congress on Evolutionary Computation*, Trondheim, Norway, pages 1375-1382.

Mezura-Montes, E., Reyes-Sierra, M., and Coello Coello, C. (2008). Multi-objective optimization using differential evolution: A survey of the state-of-the-art. In Chakraborty, U., editor, *Advances in Differential Evolution*, pages 173-196. Springer.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer.

Michalewicz, Z. and Attia, N. (1994). Evolutionary optimization of constrained problems. *Third Annual Conference on Evolutionary Programming*, San Diego, California, pages 98-108.

Michalewicz, Z., Dasgupta, D., Riche, R. L., and Schoenauer, M. (1996). Evolutionary algorithms for constrained engineering problems. *Computers & Industrial Engineering*, 30(4):851-870.

Michalewicz, Z. and Janikow, C. (1991). Handling constraints in genetic algorithms. *International Conference on Genetic Algorithms*, Breckenridge, Colorado, pages 151—157.

Michalewicz, Z. and Nazhiyath, G. (1995). Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. *IEEE Conference on Evolutionary Computation*, Perth, Western Australia, pages 647-651.

Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1): 1-32.

Michiels, W., Aarts, E., and Korst, J. (2007). *Theoretical Aspects of Local Search.* Springer.

Milinski, H. and Heller, R. (1978). Influence of a predator on the optimal foraging behavior of sticklebacks. *Nature*, 275(5681):642-644.

Miller, J. and Smith, S. (2006). Redundancy and computational efficiency in Cartesian genetic programming. *IEEE Transactions on Evolutionary Computation,* 10(2): 167-174. 2006.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms.* The MIT Press.

Mitzenmacher, M. and Upfal, E. (2005). *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press.

Morales, A. and Quezada, C. (1998). A univeral eclectic genetic algorithm for constrained optimization. *Sixth European Congress on Intelligent Techniques and Soft Computing,* Aachen, Germany, pages 518-522.

Morrison, R. (2004). *Designing Evolutionary Algorithms for Dynamic Environments.* Springer.

Mühlenbein, H., Mahnig, T., and Ochoa, A. (1999). Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics,* 5(2):215-247.

Mühlenbein, H. and Paaß, G. (1996). Prom recombination of genes to the estimation of distributions: I. Binary parameters. In Voigt, H.-M., Ebeling, W., Rechenberg, L,mand Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN IV,* pages 178-187. Springer.

Mühlenbein, H. and Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation,* l(l):25-49.

Mühlenbein, H. and Schlierkamp-Voosen, D. (1997). The equation for response to selection and its use for prediction. *Evolutionary Computation,* 5(3):303-346.

Mühlenbein, H. and Voigt, H.-M. (1995). Gene pool recombination for the breeder genetic algorithm. *First Metaheuristics International Conference,* Breckenridge, Colorado, pages 19-25.

Müller, S., Marchetto, J., Airaghi, S., and Kournoutsakos, P. (2002). Optimization based on bacterial chemotaxis. *IEEE Transactions on Evolutionary Computation,* 6(1): 16-29.

Munroe, E. (1948). *The Geographical Distribution of Butterflies in the West Indies.* PhD thesis, Cornell University.

Nemhauser, G. and Wolsey, L. (1999). *Integer and Combinatorial Optimization.* John Wiley & Sons.

Neri, F. and Tirronen, V. (2010). Recent advances in differential evolution: A survey and experimental analysis. *Artificial Intelligence Review,* 33(1):61-106.

Neshat, M., Adeli, A., Sepidnam, G., Sargolzaei, M., and Toosi, A. (2012). A review of artificial fish swarm optimization methods and applications. *International Journal on Smart Sensing and Intelligent Systems,* 5(1): 107-148.

Neumann, F. and Witt, C. (2009). Runtime analysis of a simple ant colony optimization algorithm. *Algorithmica,* 54(2):243-255.

Newton, M. (2004). *Savage Girls and Wild Boys.* Picador.

Nguyen, T., Yang, S., and Branke, J. (2012). Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation,* 6:1-24.

Nguyen, T. and Yao, X. (2009). Benchmarking and solving dynamic constrained problems. *IEEE Congress on Evolutionary Computation,* Trondheim, Norway, pages 690-697.

Nierhaus, G. (2010). *Algorithmic Composition: Paradigms of Automated Music Generation.* Springer.

Niknam, T. and Amiri, B. (2010). An efficient hybrid approach based on PSO, ACO and k-means for cluster analysis. *Applied Soft Computing,* 10(1):183-197.

Nix, A. and Vose, M. (1992). Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence,* 5(l):79-88.

Noel, M. and Jannett, T. (2005). A new continuous optimization algorithm based on sociological models. *American Control Conference,* Portland, Oregon, pages 237-242.

Noman, N. and Iba, H. (2008). Accelerating differential evolution using an adaptive local search. *IEEE Transactions on Evolutionary Computation,* 12(1): 107-125.

Nordin, P., Francone, F., and Banzhaf, W. (1996). Explicitly defined introns and destructive crossover in genetic programming. In Angeline, P. and Kinnear, K., editors, *Advances in Genetic Programming: Volume 2,* pages 111-134. The MIT Press.

Noren, S., Biedenbach, G., Redfern, J., and Edwards, E. (2008). Hitching a ride: The formation locomotion strategy of dolphin calves. *Functional Ecology,* 22(2):278-283.

Nourani, Y. and Andresen, B. (1998). A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General,* 31(41):8373-8385.

Okubo, A. and Levin, S. (2001). *Diffusion and Ecological Problems.* Springer.

Oliver, I., Smith, D., and Holland, J. (1987). A study of permutation crossover operators on the traveling salesman problem. *International Conference on Genetic Algorithms,* Cambridge, Massachusetts, pages 224-230.

Omran, M. (2008). Using opposition-based learning with particle swarm optimization and barebones differential evolution. In Lazinica, A., editor, *Particle Swarm Optimization,* pages 373-384. InTech.

Omran, M., Engelbrecht, A., and Salman, A. (2009). Bare bones differential evolution. *European Journal of Operational Research,* 196(1): 128-139.

Omran, M. and Mahdavi, M. (2008). Global-best harmony search. *Applied Mathematics and Computation,* 198(2):643-656.

Omran, M., Simon, D., and Clerc, M. (2013). Linearized biogeography-based optimization. *Submitted for publication.*

O'Neill, M. and Ryan, C. (2003). *Grammatical Evolution.* Springer.

Ong, Y., Nair, P., Keane, A., and Wong, K. (2004). Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems. In Jin, Y., editor, *Knowledge Incorporation in Evolutionary Computation,* pages 307-332. Springer.

Ong, Y.-S., Krasnogor, N., and Ishibuchi, H. (2007). Special issue on memetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 37(1):2-5.

Onwubolu, G. and Davendra, D., editors (2009). *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization.* Springer.

Opitz, D. and Maclin, R. (1999). Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169-198.

O'Reilly, U. and Oppacher, F. (1995). The troubling aspects of a building block hypothesis for genetic programming. In Whitley, L. and Vose, M., editors, *Foundations of Genetic Algorithms, Volume 3*, pages 73-88. Morgan Kaufmann.

Orvosh, D. and Davis, L. (1993). Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints. *International Conference on Genetic Algorithms*, Urbana-Champaign, Illinois, page 650.

Otten, R. and van Ginneken, L. (1989). *The Annealing Algorithm.* Kluwer Academic Publishers.

Palmer, C. and Kershenbaum, A. (1994). Representing trees in genetic algorithms. *IEEE Conference on Evolutionary Computation*, Orlando, Florida, pages 379-384.

Pan, Q., Tasgetiren, M., and Liang, Y. (2008). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers & Industrial Engineering*, 55(4):795-816.

Paquet, U. and Engelbrecht, A. (2003). A new particle swarm optimiser for linearly constrained optimisation. *IEEE Congress on Evolutionary Computation*, Canberra, Australia,pages 227-233.

Pardalos, P. and Mavridou, T. (1998). The graph coloring problem: A bibliographic survey. In Zhu, D.-Z. and Pardalos, P., editors, *Handbook of Combinatorial Optimization*, pages 331-395. Kluwer Academic Publishers.

Paredis, J. (2000). Coevolutionary algorithms. In Back, T., Fogel, D., and Michalewicz, Z., editors, *Evolutionary Computation 2*, pages 224-238. Institute of Physics.

Pareto, V. (1896). *Cours d'Economie Politique.* Guillaumin.

Parks, G. and Miller, I. (1998). Selective breeding in a multiobjective genetic algorithm. In Eiben, A., Back, T., Schoenauer, M., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN V*, pages 250-259. Springer.

Passino, K. (2002). Biomimicry of bacterial foraging. *IEEE Control Systems Magazine*, 22(3):52-67.

Paul, T. and Iba, H. (2003). Optimization in continuous domain by real-coded estimation of distribution algorithm. In Abraham, A., Koppen, M., and Pranke, K., editors, *Design and Application of Hybrid Intelligent Systems*, pages 262-271. IOS Press.

Pena, J., Robles, V., Larranaga, P., Herves, V., Rosales, F., and Pérez, M. (2004). GA-EDA: Hybrid evolutionary algorithm using genetic and estimation of distribution algorithms. In Orchard, B., Yang, C., and Ali, M., editors, *Innovations in Applied Artificial Intelligence*, pages 361-371. Springer.

Pedersen, M. (2010). Good parameters for particle swarm optimization. Technical report, Hvass Laboratories, www. h v a s s - l a b s . **org.**

Pedersen, M. and Chipperfield, A. (2010). Simplifying particle swarm optimization. *Applied Soft Computing,* 10(2):618-628.

Pelikan, M. (2005). *Hierarchical Bayesian Optimization Algorithm.* Springer.

Pelikan, M., Goldberg, D., and Cantu-Paz, E. (1999). BOA: The Bayesian optimization algorithm. *Genetic and Evolutionary Computation Conference*, Orlando, Florida, pages 525-532.

Pelikan, M., Goldberg, D., and Lobo, F. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications,* 21(l):5-20.

Pelikan, M. and Mühlenbein, H. (1998). The bivariate marginal distribution algorithm. In Benitez, J., Cordon, O., Hoffmann, F., and Roy, R., editors, *Advances in Soft Computing: Engineering Design and Manufacturing*, pages 521-535. Springer.

Pelikan, M. and Sastry, K. (2004). Fitness inheritance in the Bayesian optimization algorithm. *Genetic and Evolutionary Computation Conference,* Seattle, Washington, pages 48-59.

Petroski, H. (1992). *To Engineer Is Human: The Role of Failure in Successful Design.* Vintage.

Pétrowski, A. (1996). A clearing procedure as a niching method for genetic algorithms. *IEEE Conference on Evolutionary Computation,* Nagoya, Japan, pages 798-803.

Pham, D., Ghanbarzadeh, A., Koç, E., Otri, S., Rahim, S., and Zaidi, M. (2006). The bees algorithm - A novel tool for complex optimisation problems. *2nd International Virtual Conference on Intelligent Production Machines and Systems,* pages 454-459.

Pincus, M. (1968a). A closed form solution of certain programming problems. *Operations Research,* 16(3):690-694.

Pincus, M. (1968b). A Monte Carlo method for the approximate solution of certain types of constrained optimization problems. *Operations Research*, 18(6): 1225-1228.

Pitcher, T. and Parrish, J. (1993). Functions of shoaling behaviour in teleosts. In Pitcher, T., editor, *Behaviour of Teleost Fishes,* pages 363-439. Chapman & Hall.

Poli, R. (2003). A simple but theoretically-motivated method to control bloat in genetic programming. *Sixth European Conference on Genetic Programming,* Essex, England, pages 211-223.

Poli, R. (2008). Dynamics and stability of the sampling distribution of particle swarm optimisers via moment analysis. *Journal of Artificial Evolution and Applications,* 2008. Article ID 761459, 10 pages, doi: 10.1155/2008/761459.

Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization: An overview. *Swarm Intelligence,* l(l):33-57.

Poli, R., Langdon, W., and McPhee, N. (2008). *A Field Guide to Genetic Programming.* Published via h t t p : / / l u l u . c om and freely available at http://www.gp-field-guide. org.uk.

Poli, R., McPhee, N., and Rowe, J. (2004). Exact schema theory and Markov chain models for genetic programming and variable-length genetic algorithms with homologous crossover. *Genetic Programming and Evolvable Machines*, 5(l):31-70.

Poli, R., Rowe, J., and McPhee, N. (2001). Markov chain models for GP and variablelength GAs with homologous crossover. *Genetic and Evolutionary Computation Conference*, San Francisco, California, pages 112-119.

Poundstone, W. (1993). *Prisoner's Dilemma.* Anchor.

Powell, D. and Skolnick, M. (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. *International Conference on Genetic Algorithms*, Urbana-Champaign, Illinois, pages 424-431.

Preble, S., Lipson, M., and Lipson, H. (2005). Two-dimensional photonic crystals designed by evolutionary algorithms. *Applied Physics Letters*, 86(6):061111.

Price, K. (1997). Differential evolution versus the functions of the 2nd ICEO. *IEEE Conference on Evolutionary Computation*, Indianapolis, Indiana, pages 153-157.

Price, K. (2013). Differential evolution. In Zelinka, I., Snâsel, V., and Abraham, A., editors, *Handbook of Optimization*, **ebooks**. **com.**

Price, K. and Storn, R. (1997). Differential evolution: Numerical optimization made easy. *Dr. Dobb's Journal*, pages 18-24.

Price, K., Storn, R., and Lampinen, J. (2005). *Differential Evolution.* Springer. PSC (2012). Particle Swarm Central, www.particleswarm.info.

Päun, G. (2003). Membrane computing. In Lingas, A. and Nilsson, B., editors, *Fundamentals of Computation Theory*, pages 177-220. Springer.

Pyke, G. (1978). Optimal foraging in bumblebees and coevolution with their plants. *Oecologia*, 36(3):281-293.

Qin, A., Huang, V., and Suganthan, P. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 13(2):39-417.

Qing, A. (2009). *Differential Evolution: Fundamentals and Applications in Electrical Engineering.* John Wiley &; Sons.

Quammen, D. (1997). *The Song of the Dodo: Island Biogeography in an Age of Extinction.* Scribner.

Quijano, N., Passino, K., and Andrews, B. (2006). Foraging theory for multizone temperature control. *IEEE Computational Intelligence Magazine*, 1(4): 18-27.

Rabanal, P., Rodriguez, I., and Rubio, F. (2007). Using river formation dynamics to design heuristic algorithms. In Akl, S., Calude, C , Dinneen, M., Rozenberg, G., and Wareham, H., editors, *Unconventional Computation*, pages 163-177. Springer.

Radcliffe, N. and Surry, P. (1995). Fundamental limitations on search algorithms: Evolutionary computing in perspective. In Van Leeuwen, J., editor, *Computer Science*

Today: Recent Trends and Developments (Lecture Notes in Computer Science, No. 1000), pages 275-291. Springer-Verlag.

Rahnamayan, S., Tizhoosh, H., and Salama, M. (2008). Opposition-based differential evolution. *IEEE Transactions on Evolutionary Computation*, 12(l):64-79.

Rao, R. and Patel, V. (2012). An elitist teaching-learning-based optimization algorithm for solving complex constrained optimization problems. *International Journal of Industrial Engineering Computations*, 3(4):535-560.

Rao, R. and Savsani, V. (2012). *Mechanical Design Optimization Using Advanced Optimization Techniques.* Springer.

Rao, R., Savsani, V., and Vakharia, D. (2011). Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, 43(3):303-315.

Rao, R., Savsani, V., and Vakharia, D. (2012). Teaching-learning-based optimization: A novel optimization method for continuous non-linear large scale problems. *Information Sciences*, 183(1): 1-15.

Rashedi, E., Nezamabadi-pour, H., and Saryazdi, S. (2009). GSA: A gravitational search algorithm. *Information Sciences*, 179(13):2232-2248.

Rashedi, E., Nezamabadi-pour, H., and Saryazdi, S. (2010). BGS A: Binary gravitational search algorithm. *Natural Computing*, 9(3):727-745.

Rasheed, K. and Hirsh, H. (2000). Informed operators: Speeding up genetic algorithmbased design optimization using reduced models. *Genetic and Evolutionary Computation Conference*, Las Vegas, Nevada, pages 628-635.

Rashid, M. and Baig, A. (2010). Improved opposition-based PSO for feedforward neural network training. *International Conference on Information Science and Applications*, Seoul, Korea, pages 1-6.

Rastrigin, L. (1974). *Extremal Control Systems.* Nauka. In Russian.

Rawlins, G., editor (1991). *Foundations of Genetic Algorithms.* Morgan Kaufmann Publishers.

Ray, T., Isaacs, A., and Smith, W. (2009a). A memetic algorithm for dynamic multiobjective optimization. In Goh, C.-K., Ong, Y.-S., and Tan, K., editors, *Multi-Objective Memetic Algorithms*, pages 353-367. Springer.

Ray, T. and Liew, K. (2003). Society and civilization: An optimization algorithm based on the simulation of social behavior. *IEEE Transactions on Evolutionary Computation*, 7(4):386-396.

Ray, T., Singh, H., Isaacs, A., and Smith, W. (2009b). Infeasibility driven evolutionary algorithm for constrained optimization. In Mezura-Montes, E., editor, *Constraint-Handling in Evolutionary Optimization*, pages 145-165. Springer.

Rechenberg, I. (1973). *Evolutionsstrategie - Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution.* Frommann-Holzboog. The English translation of

the title is *Evolution strategy - Optimization of Technical Systems according to Principles of Biological Evolution.*

Rechenberg, I. (1998). Cybernetic solution path of an experimental problem. In Fogel, D., editor, *Evolutionary Computation: The Fossil Record,* pages 301-310. Wiley-IEEE Press. First published in 1964.

Reeves, C. (1993). *Modern Heuristic Techniques for Combinatorial Problems.* John Wiley & Sons.

Reeves, C. and Rowe, J. (2003). *Genetic Algorithms: Principles and Perspectives.* Kluwer Academic Publishers.

Reinelt, G. (2008). TSPLIB. http://comopt.ifi.uni-heidelberg.de/software/ TSPLIB95.

Reynolds, R. (1994). An introduction to cultural algorithms. *Third Annual Conference on Evolutionary Computing,* Madison, Wisconsin, pages 131-139.

Reynolds, R. (1999). Cultural algorithms: Theory and applications. In Corne, D., Dorigo, M., Glover, F., Dasgupta, D., Moscato, P., Poli, R., and Price, K., editors, *New Ideas in Optimization,* pages 367-378. McGraw-Hill.

Reynolds, R., Ashlock, D., Yannakakis, G., Togelius, J., and Preuss, M. (2011). Tutorials: Cultural algorithms: Incorporating social intelligence into virtual worlds. *IEEE Conference on Computational Intelligence and Games,* Seoul, South Korea, pages J1-J5.

Reynolds, R. and Chung, C. (1997). Knowledge-based self-adaptation in evolutionary programming using cultural algorithms. *IEEE International Conference on Evolutionary Computation,* Indianapolis, Indiana, pages 71-76.

Ritscher, T., Helwig, S., and Wanka, R. (2010). Design and experimental evaluation of multiple adaptation layers in self-optimizing particle swarm optimization. *IEEE Congress on Evolutionary Computation,* Barcelona, Spain, pages 1-8.

Ritzel, B., Eheart, J., and Ranjithan, S. (1995). Using genetic algorithms to solve a multiple objective groundwater pollution containment problem. *Water Resources Research,* 30(5): 1589-1603.

Robert, C. and Casella, G. (2010). *Monte Carlo Statistical Methods.* Springer.

Ronald, S. (1998). Duplicate genotypes in a genetic algorithm. *IEEE World Congress on Computational Intelligence,* Anchorage, Alaska, pages 793-798.

Ros, R. and Hansen, N. (2008). A simple modification in CMA-ES achieving linear time and space complexity. In Rudolp, G., Jansen, T., Lucas, S., Poloni, C , and Beume, N., editors, *Parallel Problem Solving from Nature - PPSN X,* pages 296-305. Springer.

Rosca, J. (1997). Analysis of complexity drift in genetic programming. *Second Annual Conference on Genetic Programming,* Palo Alto, California, pages 286-294.

Rosenberg, R. (1967). *Simulation of genetic populations with biochemical properties.* PhD thesis, University of Michigan.

Rosenbrock, H. (1960). An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3): 175-184.

Rosenkrantz, D., Steams, R., and Lewis, P. (1977). An analysis of several heuristics for the traveling salesman problem. *SI AM Journal on Computing*, 6(3):563-581.

Ross, T. (2010). *Fuzzy Logic with Engineering Applications.* John Wiley *h* Sons, 3rd edition.

Rossi, F., van Beek, P., and Walsh, T. (2006). *Handbook of Constraint Programming.* Elsevier.

Rothlauf, F. and Goldberg, D. (2003). Redundant representations in evolutionary computation. *Evolutionary Computation,* 11(4):381-415.

Rubinstein, A. (1986). Finite automata play the repeated prisoner's dilemma. *Journal of Economic Theory*, 39(l):83-96.

Rudlof, S. and Koppen, M. (1996). Stochastic hill climbing by vectors of normal distributions. *First Online Workshop on Soft Computing,* Nagoya, Japan, pages 60-70.

Rudolph, G. (1992). Parallel approaches to stochastic global optimization. In Joosen, W. and Milgrom, E., editors, *Parallel Computing: From Theory to Sound Practice*, pages 256-267. IOS Press.

Rudolph, G. and Agapie, A. (2000). Convergence properties of some multi-objective evolutionary algorithms. *IEEE Congress on Evolutionary Computation,* San Diego, California, pages 1010-1016.

Rudolph, G. and Schwefel, H.-P. (2008). Simulated evolution under multiple criteria conditions revisited. *IEEE World Congress on Computational Intelligence,* Hong Kong, pages 249-261.

Runarsson, T. and Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation,* 4(3):284-294.

Sakawa, M. (2002). *Genetic Algorithms and Fuzzy Multiobjective Optimization.* Springer.

Salkind, N. (2007). *Statistics for People Who (Think They) Hate Statistics.* Sage Publications.

Salomon, R. (1996). Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions. *BioSystems,* 39(3):263-278.

Salustowicz, R. and Schmidhuber, J. (1997). Probabilistic incremental program evolution. *Evolutionary Computation,* 5(2)-.123-141.

Sano, Y. and Kita, H. (2002). Optimization of noisy fitness functions by means of genetic algorithms using history of search with test of estimation. *IEEE Congress on Evolutionary Computation,* Honolulu, Hawaii, pages 360-365.

Santana, R. (1998). Estimation of distribution algorithms with Kikuchi approximations. *Evolutionary Computation,* 13(l):67-97.

Santana, R. (2003). A Markov network based factorized distribution algorithm for optimization. *14th European Conference on Machine Learning,* Cavtat, Croatia, pages 337-348.

Santana, R. and Echegoyen, C. (2012). Matlab Toolbox for Estimation of Distribution Algorithms (MATEDA-2.0). www.sc.ehu.es/ccwbayes/members/rsantana/softwaxe/ matlab/MATEDA.html.

Santana, R., Larranaga, P., and Lozano, J. (2008). Adaptive estimation of distribution algorithms. In Cotta, C , Sevaux, M., and Sörensen, K., editors, *Adaptive and Multilevel Metaheuristics*, pages 177-197. Springer.

Santana-Quintero, L., Montano, A., and Coello Coello, C. (2010). A review of techniques for handling expensive functions in evolutionary multi-objective optimization. In Tenne, Y. and Goh, C.-K., editors, *Computational Intelligence in Expensive Optimization Problems*, pages 29-60. Springer.

Sareni, B. and Krähenbühl, L. (1998). Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation*, 2(3):97-106.

Sastry, K. and Goldberg, D. (2000). On extended compact genetic algorithm. *Genetic and Evolutionary Computation Conference*, Las Vegas, Nevada, pages 352-359.

Sastry, K., Goldberg, D., and Pelikan, M. (2001). Don't evaluate, inherit. *Genetic and Evolutionary Computation Conference*, San Francisco, California, pages 551-558.

Savicky, P. and Robnik-Sikonja, M. (2008). Learning random numbers: A Matlab anomaly. *Applied Artificial Intelligence*, 22(3):254-265.

Savla, K., Frazzoli, E., and Bullo, F. (2008). Traveling salesperson problems for the Dubins vehicle. *IEEE Transactions on Automatic Control*, 53(6): 1378-1391.

Sayadi, M., Ramezanian, R., and Ghaffari-Nasab, N. (2010). A discrete firefly = metaheuristic with local search for makespan minimization in permutation flow shop scheduling problems. *International Journal of Industrial Engineering Computations*, 1(1):1-10.

Schaffer, C. (1994). A conservation law for generalization performance. *11th International Conference on Machine Learning*, Boca Raton, Florida, pages 259-265.

Schaffer, J. (1985). Multiple objective optimization with vector evaluated genetic algorithms. *International Conference on Genetic Algorithms and Their Application*, Pittsburgh, Pennsylvania, pages 93-100.

Schervish, M. (1996). P values: What they are and what they are not. *The American Statistician*, 50(3):203-206.

Schmidhuber, J. (1987). *Evolutionary principles in s elf-referential learning, or on learning how to learn: The meta-meta-... hook.* PhD thesis, Technische Universität München.

Schoenauer, M. and Michalewicz, Z. (1996). Evolutionary computation at the edge of feasibility. In Ebeling, W., Rechenberg, I., Schwefel, H.-P., and Voigt, H.-M., editors, *Parallel Problem Solving from Nature - PPSN IV*, pages 245-254. Springer.

Schoenauer, M., Sebag, M., Jouve, F., Lamy, B., and Maitournam, H. (1996). Evolutionary identification of macro-mechanical models. In Angeline, P. and Kinnear, K., editors, *Advances in Genetic Programming*, volume 2, pages 467-488. MIT Press.

Schoenauer, M. and Xanthakis, S. (1993). Constrained GA optimization. *International Conference on Genetic Algorithms*, Urbana-Champaign, Illinois, pages 573-580.

Schrijver, A. (2005). On the history of combinatorial optimization (till 1960). In Aardal, K., Nemhauser, G., and Weismantel, R., editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, pages 1-68. Elsevier.

Schultz, T. (1999). Ants, plants and antibiotics. *Nature*, 398(6730):747-748.

Schultz, T. (2000). In search of ant ancestors. *Proceedings of the National Academy of Sciences*, 97(26): 14028-14029.

Schumacher, C., Vose, M., and Whitley, L. (2001). The no free lunch and problem description length. *Genetic and Evolutionary Computation Conference*, San Francisco, California, pages 565-570.

Schütze, O., Lara, A., and Coello Coello, C. (2011). On the influence of the number of objectives on the hardness of a multiobjective optimization problem. *IEEE Transactions on Evolutionary Computation*, 15(4):444-454.

Schwefel, H.-P. (1977). *Numerische Optimierung von Computer-Modellen*. Birkhauser. The English translation of the title is *Evolutionary Strategy and Numerical Optimization.*

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models.* John Wiley & Sons. Translation of [Schwefel, 1977] along with some additional material.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking.* John Wiley & Sons. Expanded version of [Schwefel, 1981].

Schwefel, H.-P. and Mendes, M. (2010). 45 years of evolution strategies. *SIGEVOlution*, 4(2):2-8.

Sebag, M. and Ducoulombier, A. (1998). Extending population-based incremental learning to continuous search spaces. In Eiben, A., Back, T., Schoenauer, M., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN V*, pages 418-427. Springer.

Sefrioui, M. and Périaux, J. (2000). A hierarchical genetic algorithm using multiple models for optimization. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 879-888. Springer.

Selvakumar, A. and Thanushkodi, K. (2007). A new particle swarm optimization solution to nonconvex economic dispatch problems. *IEEE Transactions on Power Systems*, 22(1):42-51.

Seneta, E. (1966). Markov and the birth of chain dependence theory. *International Statistical Review*, 64(3):255-263.

Settles, B. (2010). Active learning literature survey. Technical report, University of Wisconsin-Madison, www.cs.emu.edu/~bsettles/pub/settles.activelearning.pdf.

Shah-Hosseini, H. (2007). Problem solving by intelligent water drops. *IEEE Congress on Evolutionary Computation*, Singapore, pages 3226-3231.

Shakya, S. and Santana, R., editors (2012). *Markov Networks in Evolutionary Computation.* Springer.

Shi, L. and Rasheed, K. (2010). A survey of fitness approximation methods applied in evolutionary algorithms. In Tenne, Y. and Goh, C.-K., editors, *Computational Intelligence in Expensive Optimization Problems*, pages 3-28. Springer.

Shi, Y. and Eberhart, R. (1999). Empirical study of particle swarm optimization. *IEEE Congress on Evolutionary Computation*, Washington, District of Columbia, pages 1945-1950.

Shibani, Y., Yasuno, S., and Ishiguro, I. (2001). Effects of global information feedback on diversity. *Advances in Genetic Programming*, 45(l):80-96.

Simöes, A. (2011). *Evolutionary Algorithms in Dynamic Optimization Problems.* Lambert Academic Publishing.

Simon, D. (2005). Research in the balance. *IEEE Potentials*, 24(2):17-21.

Simon, D. (2006). *Optimal State Estimation.* John Wiley & Sons.

Simon, D. (2008). Biogeography-based optimization. *IEEE Transactions on Evolutionary Computation*, 12(6):702-713.

Simon, D. (2011a). A dynamic system model of biogeography-based optimization. *Applied Soft Computing*, 11 (8):5652-5661.

Simon, D. (2011b). A probabilistic analysis of a simplified biogeography-based optimization algorithm. *Evolutionary Computation*, 19(2): 167-188.

Simon, D. (2012). Biogeography-Based Optimization Web Site. http://embeddedlab. csuohio.edu/BBO.

Simon, D., Ergezer, M., and Du, D. (2009). Population distributions in biogeographybased optimization algorithms with elitism. *IEEE Conference on Systems, Man, and Cybernetics*, San Antonio, Texas, pages 1017-1022.

Simon, D., Ergezer, M., Du, D., and Rarick, R. (2011a). Markov models for biogeographybased optimization. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, 41(l):299-306.

Simon, D., Rarick, R., Ergezer, M., and Du, D. (2011b). Analytical and numerical comparisons of biogeography-based optimization and genetic algorithms. *Information Sciences*, 181(7):1224-1248.

Singh, H., Ray, T., and Smith, W. (2010). Surrogate assisted simulated annealing (SASA) for constrained multi-objective optimization. *IEEE Congress on Evolutionary Computation*, Barcelona, Spain, pages 1-8.

Smith, A. and Täte, D. (1993). Genetic optimization using a penalty function. *International Conference on Genetic Algorithms*, Urbana-Champaign, Illinois, pages 499-505.

Smith, R., Dike, B., and Stegmann, S. (1995). Fitness inheritance in genetic algorithms. *Symposium on Applied Computing*, Nashville, Tennessee, pages 345-350.

Smith, S. (1980). *A Learning System Based on Genetic Adaptive Algorithms.* PhD thesis, University of Pittsburgh.

Sobotnik, J., Hanus, R., Kalinovâ, B., Piskorski, R., Cvacka, J., Bourguignon, T., and Roisin, Y. (2008). (E,E)-a-Farnesene, an alarm pheromone of the termite Prorhinotermes canalifrons. *Journal of Chemical Ecology*, 34(4):478-486.

Socha, K. and Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3): 1155-1173.

Solnon, C. (2010). *Ant Colony Optimization and Constraint Programming*. John Wiley & Sons.

Spears, W. and De Jong, K. (1997). Analyzing G As using Markov models with semantically ordered and lumped states. In Belew, R. and Vose, M., editors, *Foundations of Genetic Algorithms*, volume 4, pages 85-100. Morgan Kaufmann.

Srinivas, N. and Deb, K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221-248.

Stanley, K. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99-127.

Stephens, D. and Krebs, J. (1986). *Foraging Theory*. Princeton University Press.

Sterne, J. and Smith, G. (2001). Sifting the evidence - what's wrong with significance tests? *Physical Therapy*, 81 (8): 1464-1469.

Stone, L. (2009). *Zebras*. Lerner Publications Company.

Storn, R. (1996a). Differential evolution design of an IIR-filter. *IEEE Conference on Evolutionary Computation*, Nagoya, Japan, pages 268-273.

Storn, R. (1996b). On the usage of differential evolution for function optimization. *Conference of the North American Fuzzy Information Processing Society*, Berkeley, California, pages 519-523.

Storn, R. and Price, K. (1996). Minimizing the real functions of the ICEC'96 contest by differential evolution. *IEEE Conference on Evolutionary Computation*, Nagoya, Japan, pages 842-844.

Storn, R. and Price, K. (1997). Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341-359.

Stroud, P. (2001). Kalman-extended genetic algorithm for search in nonstationary environments with noisy fitness evaluations. *IEEE Transactions on Evolutionary Computation*, 5(l):66-77.

Stiitzle, T. and Hoos, H. (2000). MAX-MIN ant system. *Future Generation Computer Systems*, 16(8):889-914.

Su, C. and Lee, C. (2003). Network reconfiguration of distribution systems using improved mixed-integer hybrid differential evolution. *IEEE Transactions on Power Delivery*, 18(3):1022-1027.

Suganthan, P., Hansen, N., Liang, J., Deb, K., Chen, Y.-P., Auger, A., and Tiwari, S. (2005). Problem definitions and evaluation criteria for the CEC 2005 special session on real-

parameter optimization. Technical report, www.iitk.ac.in/kangal/papers/ k2005005. pdf, www. n t u . edu. sg/home/EPNSugan/index_f iles/cec-benchmarking. htm.

Sun, J., Lai, C.-H., and Wu, X.-J. (2011). *Particle Swarm Optimisation: Classical and Quantum Perspectives.* CRC Press.

Sverdlik, W. and Reynolds, R. (1993). Incorporating domain specific knowledge into version space search. *Fifth International Conference on Tools with Artificial Intelligence,* Boston, Massachusetts, pages 216-223.

Syberfeldt, A., Ng, A., John, R., and Moore, P. (2010). Evolutionary optimisation of noisy multi-objective problems using confidence-based dynamic resampling. *European Journal of Operational Research,* 204(3):533-544.

Syswerda, G. (1991). Schedule optimization using genetic algorithms. In Davis, L., editor, *Handbook of Genetic Algorithms,* pages 332-349. Van Nostrand Reinhold.

Syswerda, G. (2010). Differential evolution research - trends and open questions. In Chakraborty, U., editor, *Advances in Differential Evolution,* pages 1-32. Springer.

Szu, H. and Hartley, R. (1987). Fast simulated annealing. *Physics Letters A,* 122(3- 4):157-162.

Takahama, T. and Sakai, S. (2009). Solving difficult constrained optimziation problems by the e constrained differential evolution with gradient-based mutation. In Mezura- Montes, E., editor, *Constraint-Handling in Evolutionary Optimization,* pages 51-72. Springer.

Tan, K., Khor, E., and Lee, T. (2010). *Multiobjective Evolutionary Algorithms and Applications.* Springer.

Tanaka, M. and Tanino, T. (1992). Global optimization by the genetic algorithm in a multiobjective decision support system. *International Conference on Multiple Criteria Decision Making,* Taipei, Taiwan, pages 261-270.

Tao, G. and Michalewicz, Z. (1998). Inver-over operator for the TSP. In Eiben, A., Back, T., Schoenauer, M., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN V,* pages 803-812. Springer.

Taylor, J. (1997). *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements.* University Science Books, 2nd edition.

Tenne, Y. and Goh, C.-K., editors (2010). *Computational Intelligence in Expensive Optimization Problems.* Springer.

Teodorovic, D. (2003). Transport modeling by multi-agent systems: A swarm intelligence approach. *Transportation Planning and Technology,* 26(4):289-312.

Tereshko, V. (2000). Reaction-diffusion model of a honeybee colony's foraging behaviour. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN VI,* pages 807-816. Springer.

Tessema, B. and Yen, G. (2006). A self adaptive penalty function based algorithm for constrained optimization. *IEEE Congress on Evolutionary Computation,* Vancouver, Canada, pages 246-253.

Thiele, L., Miettinen, K., Korhonen, P., and Molina, J. (2009). A preference-based evolutionary algorithm for multi-objective optimization. *Evolutionary Computation,* 17(3) :411-436.

Thomson, I. (2010). *Culture Wars and Enduring American Dilemmas.* The University of Michigan Press.

Tilman, D., May, R., Lehman, C., and Nowak, M. (1994). Habitat destruction and the extinction debt. *Nature,* 371(3):65-66.

Tinoco, J. and Coello Coello, C. (2013). hypDE: A hyper-heuristic based on differential evolution for solving constrained optimization problems. In Schütze, 0., Coello Coello, C, Tantar, A.-A., Tantar, E., Bouvry, P., Mora, P. D., and Legrand, P., editors, *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II,* pages 267-282. Springer.

Tinos, R. and Yang, S. (2005). Genetic algorithms with self-organized criticality for dynamic optimziation problems. *IEEE Congress on Evolutionary Computation,* Edinburgh, United Kingdom, pages 2816-2823.

Tizhoosh, H. (2005). Opposition-based learning: A new scheme for machine intelligence. *International Conference on Computational Intelligence for Modelling, Control and Automation,* Vienna, Austria, pages 695-701.

Tizhoosh, H., Ventresca, M., and Rahnamayan, S. (2008). Opposition-based computing. In Tizhoosh, H. and Ventresca, M., editors, *Oppositional Concepts in Computational Intelligence,* pages 11-28. Springer.

Tomassini, M. and Vanneschi, L. (2009). Introduction: Special issue on parallel and distributed evolutionary algorithms, Part I. *Genetic Programming and Evolvable Machines,* 10(4):339-341.

Tomassini, M. and Vanneschi, L. (2010). Guest editorial: Special issue on parallel and distributed evolutionary algorithms, Part II. *Genetic Programming and Evolvable Machines,* 11 (2): 129-130.

Torregosa, R. and Kanok-Nukulchai, W. (2002). Weight optimization of steel frames using genetic algorithm. *Advances in Structural Engineering,* 5(2):99-lll.

Toth, P. and Vigo, D., editors (2002). *The Vehicle Routing Problem.* The Society for Industrial and Applied Mathematics.

Tripathi, P., Bandyopadhyay, S., and Pal, S. (2007). Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients. *Information Sciences,* 177(22) :5033-5049.

Tsutsui, S. (2004). Ant colony optimisation for continuous domains with aggregation pheromones metaphor. *Fifth International Conference on Recent Advances in Soft Computing (RASC-04),* Nottingham, United Kingdom, pages 207-212.

Turing, A. (1950). Computing machinery and intelligence. *Mind,* 59(236):433-460.

Turner, G. and Pitcher, T. (1986). Attack abatement: A model for group protection by combined avoidance and dilution. *The American Naturalist,* 128(2):228-240.

Twain, M. (2010). *Autobiography of Mark Twain, Volume 1.* University of California Press.

Tylor, E. (2009). *Primitive Culture: Researches into the Development of Mythology, Philosophy, Religion, Art and Custom,* volume 1. Cornell University Library. Originally published in 1871.

Tylor, E. (2011). *Researches into the Early History of Mankind and the Development of Civilization.* University of California Libraries. Originally published in 1878.

Ufuktepe, U. and Bacak, G. (2005). Applications of graph coloring. *International Conference on Computational Science and Applications,* Singapore, pages 465-477.

van Laarhoven, P. and Aarts, E. (2010). *Simulated Annealing: Theory and Applications.* Springer.

Van Veldhuizen, D. and Lamont, G. (2000). Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation*, 8(2): 125-147.

Vavak, F. and Fogarty, T. (1996). Comparison of steady state and generational genetic algorithms for use in nonstationary environments. *IEEE Conference on Evolutionary Computation,* Nagoya, Japan, pages 192-195.

Ventresca, M. and Tizhoosh, H. (2007). Simulated annealing with opposite neighbors. *IEEE Symposium on Foundations of Computational Intelligence,* Honolulu, Hawaii, pages 186-192.

Volk, T. (1997). *Gaia's Body: Toward a Physiology of Earth.* Springer.

Vorwerk, K., Kennings, A., and Greene, J. (2009). Improving simulated annealing-based FPGA placement with directed moves. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 28(2):179-192.

Vose, M. (1990). Formalizing genetic algorithms. *IEEE Workshop on Genetic Algorithms, Neural Networks, and Simulated Annealing Applied to Signal and Image Processing,* Glasgow, Scotland.

Vose, M. (1999). *The Simple Genetic Algorithm: Foundations and Theory.* MIT Press.

Vose, M. and Liepins, G. (1991). Punctuated equilibria in genetic search. *Complex Systems,* 5(l):31-44.

Vose, M. and Wright, A. (1998a). The simple genetic algorithm and the Walsh transform: Part I, Theory. *Evolutionary Computation,* 6(3):253-273.

Vose, M. and Wright, A. (1998b). The simple genetic algorithm and the Walsh transform: Part II, The inverse. *Evolutionary Computation,* 6(3):275-289.

Waghmare, G. (2013). Comments on "A note on teaching-learning-based optimization algorithm". *Information Sciences,* in print.

Wallace, A. (2006). *The Geographical Distribution of Animals (two volumes).* Adamant Media Corporation. First published in 1876.

Wang, H., Yang, S., Ip, W., and Wang, D. (2009). Adaptive primal-dual genetic algorithms in dynamic environments. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics,* 39(6): 1348-1361.

Wedde, H., Farooq, M., and Zhang, Y. (2004). Beehive: An efficient fault-tolerant routing algorithm inspired by honey bee behavior. In Dorigo, M., Birattari, M., Blum, C, Gambardella, L., Mondada, F., and Stiitzle, T., editors, *Ant Colony Optimization and Swarm Intelligence: 4th International Workshop, ANTS 2004,* pages 83-94. Springer.

Weiland, M. (2009). *Sand: The Never-Ending Story.* University of California Press.

Welsch, R. and Endicott, K. (2005). *Taking Sides: Clashing Views in Cultural Anthropology.* McGraw-Hill, 2nd edition.

Wesche, T., Goertler, G., and Hubert, W. (1987). Modified habitat suitability index model for brown trout in southeastern Wyoming. *North American Journal of Fisheries Management,* 7(2):232-237.

Wetzel, C. and Insko, C. (1982). The similarity-attraction relationship: Is there an ideal one? *Journal of Experimental Social Psychology,* 18(3):253-76.

Whigham, P. (1995). A schema theorem for context-free grammars. *IEEE Conference on Evolutionary Computation,* Perth, Western Australia, pages 178-181.

Whitley, D. (1989). The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. *International Conference on Genetic Algorithms,* Fairfax, Virginia, pages 116-121.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing,* 4(2):65-85.

Whitley, D. (1999). A free lunch proof for gray versus binary encodings. *Genetic and Evolutionary Computation Conference,* Orlando, Florida, pages 726-733.

Whitley, D. (2001). An overview of evolutionary algorithms: Practical issues and common pitfalls. *Information and Software Technology,* 43(14):817-831.

Whitley, D., Rana, S., Dzubera, J., and Mathias, K. (1996). Evaluating evolutionary algorithms. *Artificial Intelligence,* 85(l-2):245-276.

Whitley, D., Rana, S., and Heckendorn, R. (1998). The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology,* 7(l):33-47.

Whitley, D. and Watson, J. (2005). Complexity theory and the no free lunch theorem. In Burke, E. and Kendall, G., editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques,* pages Chapter 11, 317-339. Springer.

Whittaker, R. and Bush, M. (1993). Dispersal and establishment of tropical forst = assemblages, Krakatoa, Indonesia. In Miles, J. and Walton, D., editors, *Primary Succession on Land,* pages 147-160. Blackwell Science.

Wienke, D., Lucasius, C , and Kateman, G. (1992). Multicriteria target vector optimization of analytical procedures using a genetic algorithm: Part I. Theory, numerical simulations and application to atomic emission spectroscopy. *Analytica Chimica Act,* 265(2):211-225.

Wilson, P. and Macleod, M. (1993). Low implementation cost IIR digital filter design using genetic algorithms. *IEE/IEEE Workshop on Natural Algorithms in Signal Processing,* Chelmsford, England, pages 4/1-4/8.

Winchester, S. (2008). *The Day the World Exploded.* Collins.

Winston, P. and Horn, B. (1989). *Lisp.* Addison Wesley, 3rd edition.

Woldesenbet, Y. and Yen, G. (2009). Dynamic evolutionary algorithm with variable relocation. *IEEE Transactions on Evolutionary Computation,* 13(3):500-513.

Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation,* l(l):67-82.

Wolpert, D. and Macready, W. (2005). Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation,* 9(6): 721-735.

Worden, L. and Levin, S. (2007). Evolutionary escape from the prisoner's dilemma. *Journal of Theoretical Biology,* 245(3):411-422.

Wright, A., Poli, R., Stephens, C., Langdon, W., and Pulavarty, W. (2004). An estimation of distribution algorithm based on maximum entropy. *Genetic and Evolutionary Computation Conference,* Seattle, Washington, pages 343-354.

Wright, S. (1987). *Primal-Dual Interior-Point Methods.* Society for Industrial Mathematics. Wu, J. and Vankat, J. (1995). Island biogeography theory and applications. In Nierenberg, W., editor, *Encyclopedia of Environmental Biology,* pages 317-379. Academic Press.

Wu, S. and Chow, T. (2007). Self-organizing and self-evolving neurons: A new neural network for optimization. *IEEE Transactions on Neural Networks,* 18(2):385-396.

Wyatt, T. (2003). *Pheromones and Animal Behaviour: Communication by Smell and Taste.* Cambridge University Press.

Xinchao, Z. (2010). A perturbed particle swarm algorithm for numerical optimization. *Applied Soft Computing,* 10(1):119-124.

Yang, C. and Simon, D. (2005). A new particle swarm optimization technique. *International Conference on Systems Engineering,* Las Vegas, Nevada, pages 164-169.

Yang, C.-H., Tsai, S.-W., Chuang, L.-Y., and Yang, C.-H. (2011). A modified particle swarm optimization for global optimization. *International Journal of Advancements in Computing Technology,* 3(7): 169-189.

Yang, S. (2003a). Non-stationary problem optimization using the primal-dual genetic algorithm. *IEEE Congress on Evolutionary Computation,* Canberra, Australia, pages 2246-2253.

Yang, S. (2003b). PDG A: The primal-dual genetic algorithm. *International Conference on Hybrid Intelligent Systems,* Melbourne, Australia, pages 214-223.

Yang, S. (2008a). Genetic algorithms with memory- and elitism-based immigrants in dynamic environments. *Evolutionary Computation,* 16(3):385-416.

Yang, S., Ong, Y.-S., and Jin, Y., editors (2010). *Evolutionary Computation in Dynamic and Uncertain Environments.* Springer.

Yang, S. and Yao, X. (2005). Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing,* 9(ll):815-834.

Yang, S. and Yao, X. (2008a). Population-based incremental learning with associative memory for dynamic environments. *IEEE Transactions on Evolutionary Computation*, 12(5):542-561.

Yang, S. and Yao, X. (2008b). Population-based incremental learning with associative memory for dynamic environments. *IEEE Transactions on Evolutionary Computation*, 12(5):542-561.

Yang, X.-S., editor (2008b). *Nature-Inspired Metaheuristic Algorithms.* Luniver Press.

Yang, X.-S. (2009a). Cuckoo search via Levy flights. *World Congress on Nature & Biologically Inspired Computing*, Coimbatore, India, pages 210-214.

Yang, X.-S. (2009b). Firefly algorithm, Levy flights and global optimization. In Ellis, R. and Petridis, M., editors, *Research and Development in Intelligent Systems XXVI*, pages 209-218. Springer.

Yang, X.-S. (2010a). Firefly algorithm, stochastic test functions and design optimisation. *International Journal of Bio-Inspired Computation*, 2(2):78-84.

Yang, X.-S. (2010b). Firefly algorithms for multimodal optimization. In Watanabe, O. and Zeugmann, T., editors, *Stochastic Algorithms: Foundations and Applications*, pages 169-178. Springer.

Yang, X.-S. (2010c). A new metaheuristic bat-inspired algorithm. In Gonzalez, J., Pelta, D., Cruz, C , Terrazas, G., and Krasnogor, N., editors, *Nature Inspired Cooperative Strategies for Optimization*, pages 65-74. Springer.

Yang, Z., Tang, K., and Yao, X. (2008). Large scale evolutionary optimization using

ooperative coevolution. *Information Sciences*, 178(15) :2985-2999.

Yao, X. and Liu, Y. (1997). Fast evolution strategies. In Angeline, P., Reynolds, R., McDonnell, J., and Eberhart, R., editors, *Evolutionary Programming VI*, pages 151-161. Springer.

Yao, X., Liu, Y., and Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary Programming*, 3(2):82-102.

Yen, G. (2009). An adaptive penalty function for handling constraint in multi-objective evolutionary optimization. In Mezura-Montes, E., editor, *Constraint-Handling in Evolutionary Optimization*, pages 121-143. Springer.

Yoshida, H., Kawata, K., and Fukuyama, Y. (2001). A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *IEEE Transactions on Power Systems*, 15(4): 1232-1239.

Yu, X., Tang, K., Chen, T., and Yao, X. (2009). Empirical analysis of evolutionary algorithms with immigrants schemes for dynamic optimization. *Memetic Computing*, l(l):3-24.

Yuan, B., Orlowska, M., and Sadiz, S. (2007). On the optimal robot routing problem in wireless sensor networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(9):1251-1261.

Zaharie, D. (2002). Critical values for the control parameters of differential evolution algorithms. *International Conference on Soft Computing*, Brno, Czech Republic, pages 62-67.

Zavala, A., Aguirre, A., and Diharce, E. (2009). Continuous constrained optimization with dynamic tolerance using the COPSO algorithm. In Mezura-Montes, E., editor, *Constraint-Handling in Evolutionary Optimization*, pages 1-23. Springer.

Zhan, Z.-H., Zhang, J., Li, Y., and Chung, H. (2009). Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 39(6):1362-1381.

Zhang, J. and Sanderson, A., editors (2009). *Adaptive Differential Evolution*. Springer.

Zhang, Q., Sun, J., and Tsang, E. (2005). An evolutionary algorithm with guided mutation for the maximum clique problem. *IEEE Transactions on Evolutionary Computation*, 9(2): 192-200.

Zhang, Q., Zhou, A., Zhao, S., Suganthan, P., Liu, W., and Tiwari, S. (2009). Multiobjective optimization test instances for the CEC 2009 special session and competition. Technical report, www.ntu.edu.sg/home/EPNSugan/index_files/cec-benchmarking.htm.

Zhu, Y., Yang, Z., and Song, J. (2006). A genetic algorithm with age and sexual features. *International Conference on Intelligent Computing*, Kunming, China, pages 634-640.

Zimmerman, A. and Lynch, J. (2009). A parallel simulated annealing architecture for model updating in wireless sensor networks. *IEEE Sensors Journal*, 9(11):1503-1510.

Zitzler, E., Deb, K., and Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2): 173-195.

Zitzler, E., Laumanns, M., and Bleuler, S. (2004). A tutorial on evolutionary multiobjective optimization. In Gandibleux, X., Sevaux, M., Sörensen, K., and T'Kindt, V., editors, *Metaheuristics for Multiobjective Optimisation*, pages 3-38. Springer.

Zitzler, E., Laumanns, M., and Thiele, L. (2001). SPEA2: Improving the strength Pareto evolutionary algorithm. *EUROGEN 2001: Evolutionary Methods for Design, Optimisation and Control with Applications to Industrial Problems*, Athens, Greece, pages 95-100.

Zitzler, E. and Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257-271.

Zitzler, E., Thiele, L., and Bader, J. (2010). On set-based multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 14(l):58-79.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C , and da Fonseca, V. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2): 117-132.

Zlochin, M., Birattari, M., Meuleau, N., and Dorigo, M. (2004). Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research*, 131(1-4):373-395.