بنام پرورگار یکتا

جزوه درس رایانش تکاملی

مخصوص دانشجویان کارشناسی ارشد مهندسی کامپیوتر
گرایش‌های هوش مصنوعی و نرم‌افزار
و دانشجویان مهندسی پزشکی

تالیف : دکتر رضا قائمی

فصل اول

الگوریتم‌های جستجو

1-1 انواع الگوریتم‌های جستجو

برای حل یک مسأله ابتدا بایستی آن را نمایش دهیم. نمایش مسأله (*Problem Representation*) یکی از مراحل اساسی در حل مسأله می‌باشد. استفاده از یک روش نمایش نامناسب ممکن است امکان یافتن یک راه حل مطلوب را تقریبا غیرممکن نماید.

منظور از نمایش یک مسأله، انتخاب روشی است که هر وضعیت (*State*) را در آن مسأله به روشنی بیان کند. در اینجا هر وضعیت بیانگر یک راه حل است که ممکن است بهینه یا غیر بهینه باشد. درهر حال با انتخاب یک روش برای بازنمایی هر وضعیت برای یک مسأله، عملاً فضای جستجوی آن مسأله متولد می‌شود.

تصمیم گیری برای چگونگی نمایش هر راه حل در یک مسأله، بر روی انتخاب روش مناسب برای حل آن مسأله تأثیر می‌گذارد. در صورت عدم نمایش راه حل‌های یک مسأله، هیچ گونه فضای جستجویی بوجود نخواهد آمد تا برای پویش در اختیار یک الگوریتم جستجو قرار گیرد.

عموما دو نوع بهینگی وجود دارد : محلی و سراسری. بهینگی محلی بهترین راه حل پیدا شده در یک ناحیه (همسایگی) از فضای جستجو می‌باشد، اما لزوما بهترین راه حل در همه فضای جستجو نیست. راه حل بهینه سراسری بهترین راه حل در همه فضای جستجو می‌باشد.

یافتن بهینه سراسری در بیشتر مسائل واقعی کاری دشوار است. روش‌های جستجو معمولا سعی می‌کنند که به جای یافتن پاسخ بهینه، راه حل‌های محلی، سراسری یا رضایت بخش، به استفاده از یک روش جستجو نیازخواهیم داشت.

جستجو یکی از شاخه‌های مهم در تحقیقات است. نیاز به روش‌های جستجو کارآتر، به علت بزرگ تر و پیچیده تر شدن مسائل، هر روزه بیشتر می‌شود. سه نوع روش برای الگوریتم‌های جستجو وجود دارد :

1) جستجوی تحلیلی (*Analytical Search*)
2) جستجوی ناآگاهانه یا کور (*Uniformed* (*Blind*) *Search*)
3) جستجوی آگاهانه یا مکاشفه‌ای (*Informed* (*Heuristic*) *Search*)

1-1-1 جستجوی تحلیلی

با استفاده از یک تابع ریاضی هدایت می‌شوند. برای مثال، برخی از الگوریتم‌های جستجو تحلیلی برای حل مسائل بهینه سازی، با استفاده از گرادیان و برخی با مشتق دوم هدایت می‌شوند. این الگوریتم‌ها، یافتن جواب بهینه را در صورت وجود تضمین می‌کنند.

یکی از انواع الگوریتم‌های جستجوی تحلیلی، الگوریتم نیوتن ـ رافسون می‌باشد. این الگوریتم، روشی عمومی برای یافتن ریشه‌های توابع (یعنی حل معادلاتی به شکل $f(x)=0$) بصورت زیر می باشد :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

برای یافتن یک بیشینه یا کمینه f باید x را به نحوی پیدا کرد که گرادیان صفر شود، یعنی $\nabla f(x) = 0$.

روش نیوتن ـ رافسون در فضاهایی با ابعاد بالا، پر هزینه است و به همین جهت، تقریب‌های زیادی برای آن به وجود آمده است. تنها برای مسائل جستجوی با یک فضای قابل نگاشت بوسیله توابع ریاضی مشتق‌پذیر و بسیار منظم بتواند پاسخ بهینه را بیابد. با توجه به عدم در اختیار داشتن یک چنین توابعی در بسیاری از مسائل واقعی، از الگوریتم‌های مبتنی بر جستجوی تحلیلی به ندرت استفاده می‌شود.

1-1-2 جستجوی ناآگاهانه

به آن دسته از الگوریتم‌هایی اطلاق میشود که الگوریتم هیچ اطلاعات اضافه‌ای به جز آنچه در تعریف مسائل آمده است درباره هر نقطه (راه حل) از فضای جستجو ندارد. تنها قادربه پیمایش فضای جستجوی درختی مسائل و نیز تشخیص یک حالت هدف از یک حالت غیر هدف هستند. کاربرد اصلی الگوریتم‌های جستجوی ناآگاهانه برای مسائلی است که بتوان وضعیت‌ها یا همان راه‌حل‌های موجود در فضای جستجو را به شکل یک درخت ترسیم نمود. این الگوریتم‌ها به دو دسته کامل و ناکامل قابل تقسیم هستند.

روش جستجوی کامل فضای جستجو را شمارش می‌کند (جاروب می‌کند) و با جستجوی کامل فضا، قادر است که همیشه یافتن راه‌حل‌های موجود را تضمین کند. روش جستجوی ناکامل، مجموعه‌ای از راه‌حل را (در صورت وجود) تضمین نمی‌کنند.

روش‌های جستجوی کامل ممکن است بهینه یا غیربهینه باشند. یک روش جستجوی کامل و بهینه تضمین می‌کند که همیشه راه‌حل بهینه را بیابد، در حالی که روش جستجوی کامل و غیربهینه تنها می‌تواند در مورد یافتن یک راه‌حل که شاید لزوما بهینه هم نباشد، تضمین بدهد. بدیهی است که بحث بهینگی را نمی‌توان برای یک الگوریتم جستجوی ناکامل مطرح نمود.

الگوریتم‌های جستجوی ناآگاهانه، بر اساس ترتیب گسترش حالات (گره‌ها)، از یکدیگر متمایز می‌شوند.

1-1-2-1 جستجوی اول ـ سطح

در روش جستجوی اول - سطح یا به اختصار BFS (*Breadth-First Search*)، ابتدا ریشه گسترش می‌یابد، سپس تمامی گره‌های فرزند ریشه گسترش می‌یابند. آنگاه پسین‌های فرزندان ریشه نیز گسترش می‌یابند و به همین ترتیب الی آخر. به طور کلی، باید تمامی گره‌های یک سطح از درخت جستجو گسترش داده شوند تا یک گره در سطح بعدی بتواند گسترش یابد.

این روش جستجو کامل و بهینه است و همچنین، دارای پیچیدگی فضایی و زمانی بالایی است. روش‌های مختلفی برای بهبود کارآیی این الگوریتم پیشنهاد شده‌اند که به عنوان نمونه می‌توان به روش جستجوی

همزمان سطحی از گره شروع وگره هدف اشاره نمود. این روش که به روش جستجوی دوجهته یا به اختصار BS (*Bidirectional Search*) شهرت دارد، همزمان عملیات جستجوی سطحی را از گره‌های آغاز و هدف انجام می‌دهد. بدیهی است که هزینه‌های زمانی و فضایی روش جستجوی دوجهته در مقایسه با روش جستجوی اول-سطح کمتر است.

1-1-2-2 جستجوی هزینه – یکنواخت

در روش جستجوی هزینه - یکنواخت یا به اختصار UCS (*Uniform-Cost Search*)، به جای گسترش کم عمق‌ترین گره، گره n را که کم‌ترین هزینه مسیر را دارد، گسترش می‌یابد. اگر تمامی هزینه‌های مراحل مختلف مساوی باشد، این جستجو با جستجوی اول – سطح یکسان خواهد شد.

جستجوی هزینه-یکنواخت، به تعداد مراحل یک مسیر اهمیت نمی‌دهد، بلکه تنها هزینه کامل آنها را در نظر می‌گیرد. درنتیجه، اگر گرهی را گسترش دهد که دارای فرزندی با هزینه صفر باشد و به همان حالت برگردد، این جستجو در یک حلقه بی نهایت گرفتار خواهد شد. در صورتی می‌توانیم کامل بودن را تضمین کنیم که هزینه تولید و پیمایش فرزندان گره‌ها در هر مرحله، بزرگ‌تر یا مساوی یک مقدار ثابت مثبت ε باشد. این شرط برای تضمین بهینگی نیز کافی خواهد بود. براساس این ویژگی به سادگی می‌توان نتیجه گرفت که این الگوریتم، گره‌ها را به ترتیب افزایش هزینه مسیر کسترش می‌دهد.

1-1-2-3 جستجوی اول – عمق

در روش جستجوی اول - عمق یا به اختصار DFS (*Depth-First Search*)، همیشه عمیق‌ترین گره از درخت جستجو، زودتر از گره‌های با عمق کمتر، گسترش می‌یابد. جستجو بلافاصله به عمیق‌ترین سطح درخت جستجو هدایت می‌شود که در آنجا گره‌ها هیچ پسینی ندارند. در همان حال که گره‌ها گسترش می‌یابند، از حافظه خارج می‌شوند.

این روش به حافظه نسبتا کمی نیاز دارد. تنها بایستی یک مسیر از گره ریشه تا گره برگ، همراه با گره‌های هم نیاز گسترش نیافته گره‌های آن مسیر، را نگهداری کند.

مشکل جستجوی اول-عمق این است که ممکن است انتخاب غلطی انجام دهد و در پایین رفتن از یک مسیر بسیار طولانی (و حتی نامتناهی) گرفتار شود. بنابراین روش جستجوی اول-عمق کامل نیست. اگر زیر درخت سمت چپ دارای عمق نامحدود باشد و درعین حال شامل هیچ راه‌حلی نباشد، جستجوی اول-عمق هرگز پایان نمی‌یابد.

انواع دیگری از روش های جستجوی عمقی، مانند روش جستجوی عمق-محدود یا به اختصار DLS (*Depth-Limited Search*)، که در آن تا عمق مشخص L عملیات جستجوی عمقی انجام می‌شود، ویا روش جستجوی عمیق شونده تکراری یا به اختصار IDS (*Iterative-Deepening Search*)، که در آن

الگوریتم جستجوی عمق-محدود در یک حلقه با امتحان L های مختلف، تا رسیدن به جواب نهایی، مرتبا اجرا می‌گردد، نیز وجود دارند، که محققان سعی در رفع مشکلات آنها دارند.

1-1-3 جستجوی آگاهانه

با بهره‌مندی از یک تابع تخمینی، استراتژی هوشمندانه‌تری را برای کاوش فضای جستجو درپیش می‌گیرند. روش‌های مبتنی بر جستجوی آگاهانه ویا مکاشفه‌ای می‌توانند با استفاده از دانش خاصی که مسأله در اختیار آنها قرار می‌دهد، به طور موثرتری فضای جستجو را پیموده و راه‌حل‌ها را بیابند. در ادامه دو خانواده مهم الگوریتم‌های جستجوی آگاهانه را معرفی می‌کنیم.

1-1-3-1 جستجوی اول – بهترین

روش جستجوی اول - بهترین یا به اختصار BFS (*Best-First Search*)، نمونه‌ای از الگوریتم عمومی جستجوی درختی (*Tree-Search*) یا جستجوی گرافی (*Graph-Search*) است که در آن یک گره براساس یک تابع تخمینی $f(n)$ جهت گسترش انتخاب می‌شود. به طور معمول کم هزینه‌ترین گره، برای جستجو انتخاب خواهد شد، زیرا تابع تخمینی $f(n)$، هزینه رسیدن از گره n تا گره هدف را محاسبه می‌کند.

یک جزء کلیدی این الگوریتم، تابع مکاشفه‌ای است که به شکل $h(n)$ نشان داده می‌شود. در واقع $h(n)$ برآورد کم هزینه‌ترین مسیر از گره n تا گره هدف است. توابع مکاشفه‌ای، معمول‌ترین شکل انتقال اطلاعات اضافی به الگوریتم جستجو هستند، با این محدودیت که اگر n گره هدف باشد، آنگاه $h(n)=0$ است. در اینجا دو نوع الگوریتم جستجوی آگاهانه مبتنی بر استراتژی اول-بهترین را تشریح می‌کنیم.

1-1-3-2 جستجوی حریصانه

روش جستجوی حریصانه یا به اختصار GS (*Greedy Search*)، سعی می‌کند که نزدیک‌ترین گره به هدف را گسترش دهد، به این دلیل که احتمال زیادی دارد که نزدیک‌ترین مسیر باشد. بنابراین گره‌ها را تنها با استفاده از تابع مکاشفه‌ای $f(n)=h(n)$ ارزیابی می‌کند.

جستجوی اول-بهترین حریصانه، شبیه جستجوی اول-عمق است، ازاین جهت که ترجیح می‌دهد یک مسیر را در تمام طول راه تا هدف دنبال کند. ولی اگر به بن‌بست برسد، به عقب برمی‌گردد. این الگوریتم بهینه و کامل نیست، زیرا ممکن است با آغاز یک مسیر بی انتها، هرگز هیچ امکان دیگری را امتحان نکند.

1-1-3-3 جستجوی A*

این روش جستجو که هدف اصلی در آن کمینه کردن کل هزینه‌ی برآورد راه‌حل است، شناخته شده ترین شکل جستجوی اول-بهترین است. این روش با توجه به تابع تخمینی $f(n)=g(n)+h(n)$، سعی در پیمایش درخت فضای جستجو می‌نماید. در این تابع تخمین، $g(n)$ هزینه رسیدن به گره n از گره شروع (هزینه واقعی) و $h(n)$ هزینه رسیدن از گره n به گره هدف (هزینه تخمینی) است. با توجه به جمع شدن هزینه واقعی $g(n)$ با هزینه تخمینی $h(n)$، $f(n)$ هزینه تخمینی ارزان‌ترین مسیر از گره آغاز به گره هدف و از طریق گره n است.

اگر تابع مکاشفه‌ای $h(n)$ شرایط خاصی داشته باشد، جستجوی A* هم کامل و هم بهینه است. یک مثال واضح از یک مکاشفه قابل قبول در مسأله جستجوی بهترین راه‌حل برای گذشتن از چند شهر و رسیدن به شهر مقصد، در نظر گرفتن فاصله مستقیم و هوایی بین شهرهاست. از آنجایی که کوتاه‌ترین فاصله بین دو نقطه، یک خط مستقیم است، بنابراین مکاشفه خط مستقیم نمی‌تواند یک برآورد بیش از اندازه واقعی باشد.

از نقاط ضعف جستجوی A*، زمان محاسبه زیاد و نیاز به حافظه بزرگ است، زیرا که این الگوریتم تمامی گره‌های تولید شده را در حافظه نگهداری می‌کند. به همین دلیل جستجوی A* در مورد بسیاری از مسائل بزرگ، عملی نیست.

1-1-3-4 جستجوی فرامکاشفه‌ای

در بسیاری از مسائل جستجو در دنیای واقعی، نمی‌توان فضای جستجو را به شکل یک درخت نمایش داد (مخصوصا هنگامی که این فضا بسیار بزرگ و نامنظم باشد). روش جستجوی فرامکاشفه‌ای یا به اختصار MHS (*Meta-Heuristic Search*)، نام خانواده‌ای از الگوریتم‌های جستجوی آگاهانه است که در آنها از یک پدیده طبیعی، برای کاوش فضای جستجو، الهام گرفته می‌شود.

1-2 الگوریتم‌های فرامکاشفه‌ای

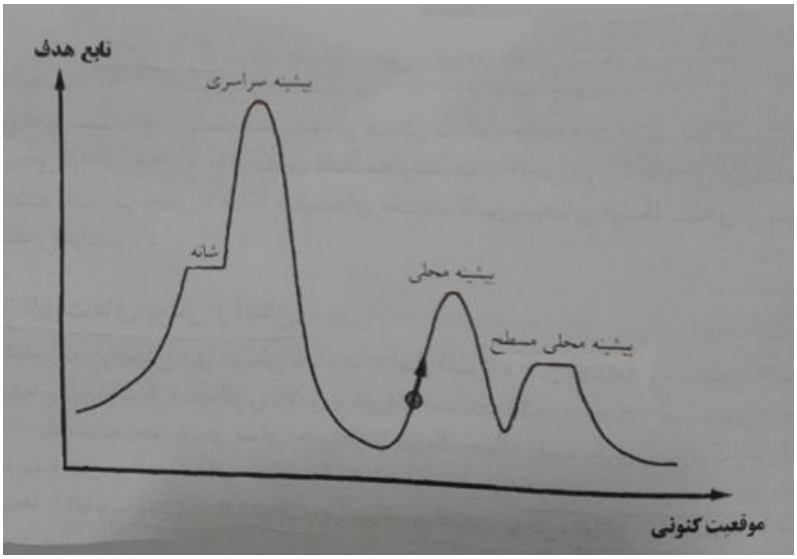
الگوریتم‌های فرامکاشفه‌ای کل فضای جستجو را به دلیل بزرگی آن پیمایش نکرده (که به همین دلیل ناکامل و البته غیربهینه هستند) و تنها به پیمایش بخشی از فضا که احتمال وجود یک پاسخ به اندازه کافی خوب در آن بیشتر است، اکتفا می‌کنند.

برخی از الهام‌های طبیعی که بر اساس آنها یک الگوریتم فرامکاشفه‌ای طراحی شده است عبارتند از: تکامل موجودات در طی نسل‌ها، فرآیند سرد شدن یا تبرید در فلزات، زندگی مورچه‌ها در یک کلونی، حرکت گروهی پرندگان و سیستم ایمنی در بدن انسان.

1-2-1 فضای جستجو و دور نمای برازش

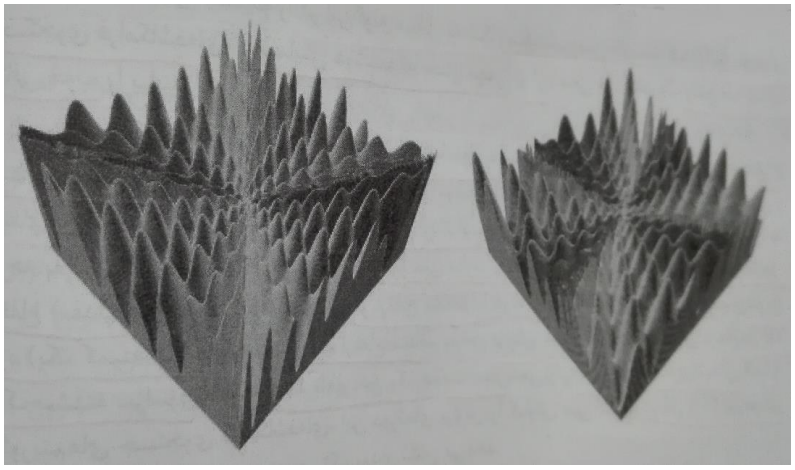
فضای جستجو در بسیاری از مسائل واقعی آن‌چنان بزرگ و نامنظم است که امکان تبیین و نمایش آن به شکل یک درخت وجود ندارد. به همین دلیل امکان استفاده از هیچکدام از روش‌های جستجوی تحلیلی و ناآگاهانه وجود ندارد. در این گونه موارد حتی نمی‌توان از روش‌های اگاهانه‌ی مبتنی بر ایده جستجو اول-بهترین نظیر جستجوی حریصانه و یا *A بهره‌برداری نمود. می‌توان فضای جستجو را در این گونه مسائل به شکل یک سرزمین دانست که الگوریتم‌های جستجوی فرامکاشفه‌ای، با پیمایش هوشمندانه بخش مهمی از آن، سعی می‌کنند یک پاسخ به اندازه کافی خوب را بیابند.

سرزمینی که به عنوان فضای جستجوی نامنظم مسائل پیچیده عنوان شد دارای پستی و بلندی‌های متعددی است که از آن در متون علمی مرتبط با نام دورنمای برازش (*Fitness Landscape*) یاد می‌شود. پستی و بلندی در دورنمای برازش توسط تابع برازش یا شایستگی (*Fitness Function*) مشخص می‌گردد. تعریف تابع برازش با توجه به اطلاعات مساله انجام می‌شود. یک دورنمای برازش شامل موقعیت (یا همان پاسخ) و نیز شامل ارتفاع (مقدار تابع شایستگی) می‌باشد. اگر ارتفاع متناظر با هزینه باشد، آنگاه هدف، یافتن عمیق‌ترین دره (یک کمینه سراسری) است و اگر ارتفاع متناظر با تابع برازش باشد، آنگاه هدف، یافتن بلندترین قله (یک بیشینه سراسری) است. لازم به یادآوری است که تنها با افزودن یک علامت منفی، می‌توان یکی را به دیگری تبدیل کرد. شکل (1-1) دور نمای برازش را برای فضای جستجوی تک بعدی نشان می‌دهد.



شکل (1-1) : دورنمای برازش برای فضای جستجوی یک بعدی

در شکل (1-2) دور نمای برازش، که در آن‌ها پستی‌ها و بلندی‌های متعددی وجود دارند، به تصویر کشیده شده‌اند. در متون علمی مرتبط از واژه چند قله‌ای یا خارپشتی (*Multimodal*) برای این گونه فضاهای جستجو استفاده می‌شود.



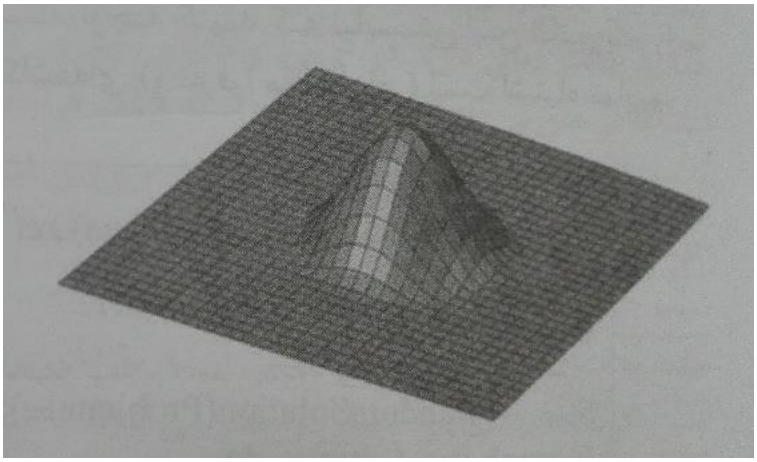
شکل (1-2) : دورنماهای خارپشتی برای دو فضای جستجوی دو بعدی

1-2-2 قابلیت‌های پویش و انتفاع

قابلیت پویش (*Exploration Capability*) و قابلیت انتفاع (*Exploitation Capability*)، دو نوع از مهمترین قابلیت ها در الگوریتم های تکاملی به شمار می روند. قابلیت پویش، به توانایی الگوریتم فرامکاشفه‌ای در جستجوی آزادانه و بدون هرگونه توجه به دستاوردهای آن در طول فرآیند جستجو و قابلیت انتفاع (*Exploitation Capability*) به میزان توجه الگوریتم به دستاوردهایش در طول فرآیند جستجو اطلاق می گردند.

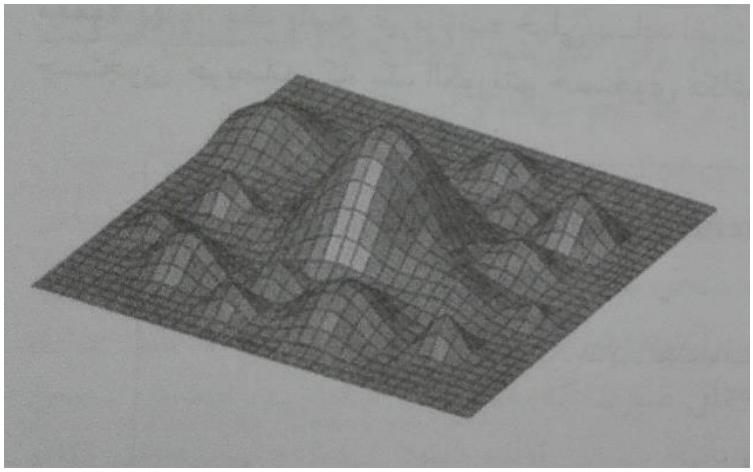
بدیهی است که به هر میزان که قابلیت پویش در یک الگوریتم جستجو بیشتر باشد، این الگوریتم رفتاری تصادفی‌تر و غیرقابل پیش‌بینی‌تر خواهد داشت. در نقطه مقابل، تقویت قابلیت انتفاع در یک الگوریتم سبب می‌شود که این الگوریتم رفتاری حساب‌شده‌تر و محتاطانه‌تر داشته باشد. از آنجا که تقریبا اکثر روش‌های جستجو فرامکاشفه‌ای دارای پارامترهای قابل تنظیمی هستند، می‌توانیم میزان قابلیت‌های پویش و انتفاع را در آنها کنترل نماییم.

برای پیمایش و کاوش موثر یک مساله جستجو بایستی قابلیت پویش و انتفاع در یک الگوریتم فرامکاشفه‌ای با توجه به ماهیت مساله مورد بررسی تنظیم شود. شکل (1-3) بیانگر یک فضای جستجوی تک قله‌ای و منظم است. برای مساله مربوط به این دورنمای برازش، بایستی قابلیت انتفاع الگوریتم را تقویت کرد.



شکل (1-3) : دورنمای فضای جستجوی تک قله‌ای

درنقطه مقابل، به منظور جستجوی موثر در فضای جستجوی مربوط به مساله در شکل (1-4)، بایستی قابلیت پویش را در الگوریتم فرامکاشفه‌ای افزایش داد.



شکل (1-4) : دورنمای فضای جستجوی چند قله‌ای

می‌توان نتیجه گرفت که هر چقدر دورنمای برازش در یک مساله جستجو منظم‌تر باشد، استفاده از یک الگوریتم فرامکاشفه‌ای با قابلیت انتفاع بهتر خواهد بود. درعین حال برای مسائل جستجو با دورنمای برازش خارپشتی، بهره برداری از یک الگوریتم فرامکاشفه‌ای با قابلیت پویش توصیه می‌شود.

در میان الگوریتم‌های مختلف جستجوی فرامکاشفه‌ای، شاید بتوان روش جستجوی تپه‌نوردی یا به اختصار HCS (*Hill-Climbing Search*) را به عنوان الگوریتمی با بیشترین قابلیت انتفاع دانست. در

نقطه مقابل، جستجوی تصادفی یا به اختصار RS (*Random Search*) را به عنوان الگوریتمی با بالاترین قابلیت پویش در میان انواع مختلف الگوریتم‌های جستجوی فرامکاشفه‌ای می‌شناسیم. جستجوی تپه‌نوردی یک روش بهینه‌سازی سراسری و فرامکاشفه‌ای می‌باشد که شبه کد آن در الگوریتم (1-1) نشان داده شده است.

Function Hill-Climbing(problem) **returns** a state that is a local maximum

Input : $Iter_{max}$, $Problem_{size}$

Output : S_{best}

$S_{best} \leftarrow$ RandomSolution($Problem_{size}$) ;

For each $iter_i \in Iter_{max}$ do

  Candidate $\leftarrow$ RandomNeighbor(S_{best}) ;

  If Fitness(Candidate) ≥ Fitness(S_{best}) then

    $S_{best} \leftarrow$ Candidate ;

  end

end

return S_{best} ;

الگوریتم (1-1) : شبه کد جستجوی تپه‌نوردی

این الگوریتم فقط از یک حلقه تشکیل شده است که به طور مداوم در جهت افزایش مقدار برازش (البته با توجه به یک مساله بیشینه‌سازی)، حرکت می‌کند، یعنی به سمت بالای تپه. الگوریتم هنگامی خاتمه می‌یابد که به قله‌ای برسد که در آنجا هیچ همسایه‌ای، مقدار برازش بیشتری را برای تابع هدف نداشته باشد. در این الگوریتم، درخت جستجو نگهداری نمی‌شود. بنابراین در ساختمان داده‌ی گره فعلی، تنها نیاز به ثبت حالت و مقدار تابع هدف آن وجود دارد. جستجوی تپه‌نوردی، فراتر از همسایه‌های مجاور پاسخ فعلی، دورنمای برازش را مورد بررسی قرار نمی‌دهد.

جستجوی تپه‌نوردی را گاهی جستجوی حریصانه‌ی محلی نیز می‌نامند، زیرا یک حالت همسایه خوب را، بدون اینکه از قبل برنامه‌ریزی کند که از آنجا به کجا خواهد رفت، انتخاب می‌نماید. با وجود حریصانه بودن این الگوریتم، جستجوی تپه‌نوردی اغلب پیشرفت سریعی به سمت راه‌حل دارد. الگوریتم جستجوی تپه‌نوردی برای مسائلی مناسب است که دارای یک دور نمای برازش ساده و با حداقل پستی و بلندی باشند. یک نمونه از چنین مسائلی را می‌توان مساله یافتن کمینه تابع $y=x(x+1)$ دانست.

متاسفانه جستجوی تپه‌نوردی اغلب به دلایل زیر در بهینه محلی گرفتار می‌شود :

- **بیشینه‌های محلی :** یک بیشینه محلی قله‌ای است که از تمامی حالت‌های همسایه‌اش بلندتر، ولی از بیشینه سراسری کوتاه‌تر است. الگوریتم جستجوی تپه‌نوردی، هنگامی که به همسایگی یک بیشینه محلی می‌رسد، رو به بالا به سمت قله کشیده می‌شود، ولی پس از آن گرفتار شده و متوقف می‌شود.
- **دماغه‌ها (Ridge) :** دماغه‌ها یک رشته بیشینه محلی هستند که گذشتن از آنها برای الگوریتم‌هایی که ماهیت جستجوی حریصانه دارند، بسیار مشکل است.
- **فلات‌ها (Plateau) :** فلات‌ها ناحیه‌ای از دورنمای برازش فضای جستجو هستند که در آن مقدار برازندگی تابع برازش، ثابت است. فلات می‌تواند یک بیشینه محلی مسطح باشد که از آن هیچ مسیر رو به بالایی وجود ندارد یا یک شانه باشد که از آن بتوان بالاتر هم رفت. یک جستجوی تپه‌نوردی ممکن است قادر نباشد که راه خروج از فلات را بیابد.

در هر کدام از این موارد، الگوریتم به حالتی می‌رسد که دیگر پیشرفتی حاصل نمی‌شود. جستجوی تپه‌نوردی در صورتی که به یک فلات برسد متوقف می‌شود.

تا کنون انواع متعددی از روش های جستجوی تپه‌نوردی ابداع گردیده است. روش جستجوی تپه‌نوردی اتفاقی (*Stochastic Hill-Climbing Search*) از میان حرکت‌های رو به بالا، به صورت تصادفی یکی را انتخاب می‌کند. احتمال این انتخاب می‌تواند براساس شیب حرکت‌های رو به بالا تغییر کند. این روش جستجو، معمولا کندتر از روش جستجوی تپه‌نوردی ساده، به نتیجه می‌رسد، اما در بعضی از دورنماهای برازش، بهترین راه‌حل را پیدا می‌کند. روش جستجوی تپه‌نوردی اولین-گزینه (*First-Choice Hill-Climbing Search*)، تپه‌نوردی اتفاقی را بکار می‌گیرد. به این صورت که به صورت تصادفی پاسخ‌های پسین تولید می‌کند. این کار را تا زمانی ادامه می‌دهد که پاسخ پسینی تولید شود که از حالت فعلی بهتر باشد. این راهبرد هنگامی مفید است که یک پاسخ دارای تعداد زیادی پاسخ پسین باشد.

الگوریتم‌های جستجوی تپه‌نوردی که تاکنون توضیح داده شدند، همگی ناکامل هستند و اغلب موفق نمی‌شوند اهداف موجود را بیابند، زیرا ممکن است که در یک بیشینه محلی گرفتار شوند. روش جستجوی تپه‌نوردی با شروع مجدد تصادفی (*Random restart hill climbing*)، یک مجموعه جستجوی تپه‌نوردی را با حالت‌های شروع تصادفی اجرا می‌کند و هنگامی که یک هدف پیدا شد، متوقف می‌شود. موفقیت تپه‌نوردی، بستگی زیادی به شکل دورنمای برازش فضای حالت دارد. اگر تعداد بیشینه محلی و فلات کم باشد، تپه‌نوردی با شروع مجدد تصادفی یک راه خوب را خیلی سریع خواهد یافت.

الگوریتم جستجوی تپه‌نوردی بیشترین قابلیت انتفاع را در میان همه روش‌های جستجوی فرامکاشفه‌ای دارا است، زیرا این الگوریتم هیچ ریسکی را در حین فرآیند پیمایشی خود نمی‌کند. در نقطه مقابل جستجوی تپه‌نوردی، روش جستجوی تصادفی قرار دارد که ماهیتی کاملا جستجوگرانه و پویشی دارد. این الگوریتم در هر قدمی که در فضای جستجو بر می‌دارد، هیچ بهره‌ای از تابع برازش نمی‌برد و پاسخی کاملا تصادفی را در فضای جستجو انتخاب می‌کند. به عبارت دیگر، جستجوی تصادفی هیچ انتفاعی از دستاوردهای فرآیند جستجو برای حرکت‌های آتی خود نمی‌برد. به همین دلیل این روش جستجو، بالاترین قابلیت پویش را در میان انواع مختلف الگوریتم‌های جستجوی فرامکاشفه‌ای دارا است.

شبه کد مربوط به جستجوی تصادفی در الگوریتم (1-2) نشان داده شده است.

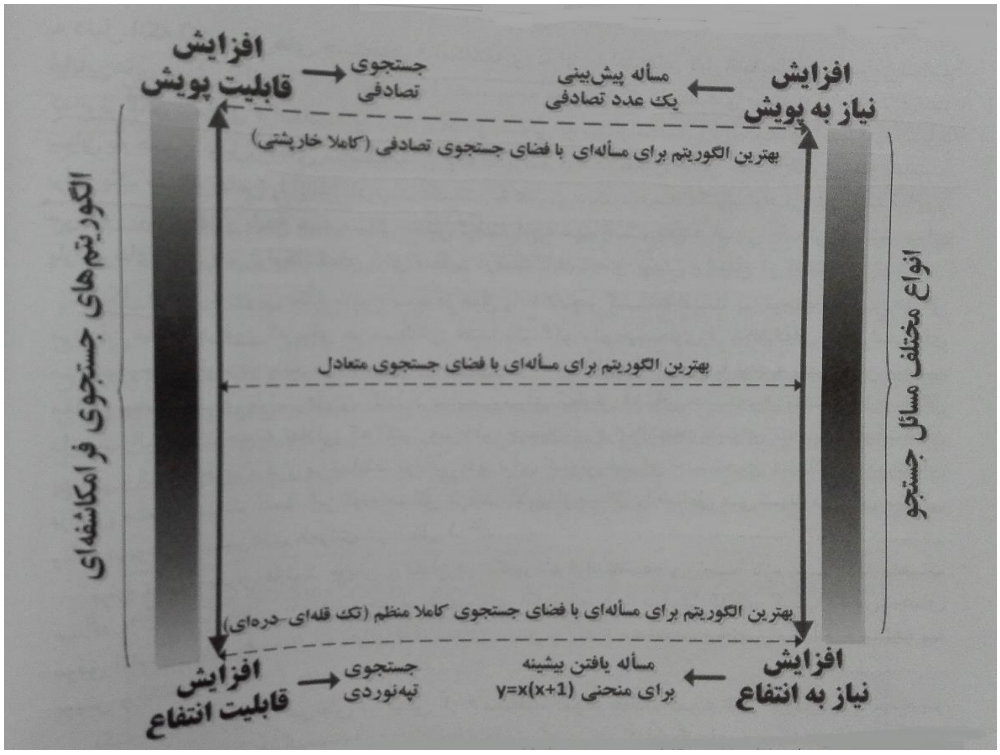
```
Function Random-Search(problem) return a state that is a local maximum
Input : NumIteration , Problemsize , SearchSpace
Output: Sbest

Sbest ← 0 ;
for each iteri ∈ NumIterations do
    candidate ← RandomSolution(Problemsize , SearchSpace) ;
    if Fitness(candidate(i) > Fitness(Sbest) then
        Best ← candidate(i) ;
    end
end
return Sbest ;
```

الگوریتم (1-2) : شبه کد جستجوی تصادفی

بسیاری از مسائل دنیای واقعی، بسیار پیچیده‌تر از آنهایی هستند که تا کنون اشاره نموده‌ایم. به عبارت دیگر، برای آن که بتوانیم پاسخ به اندازه کافی خوب را برای اکثر مسائل واقعی بیابیم، بایستی یک مصالحه (*Trade-off*)، میان قابلیت‌های پویش و انتفاع در الگوریتم‌های فرامکاشفه‌ای برقرار نماییم. قابلیت‌های پویش و انتفاع برای هر کدام از الگوریتم‌های فرامکاشفه‌ای ثابت نبوده و بستگی به مقدار تنظیمی پارامترهای آنها دارد. شاید بتوان به همین دلیل، ادعای مناسب بودن یک الگوریتم فرامکاشفه‌ای خاص جهت یافتن پاسخ مناسب برای یک مساله خاص را کاملا نادرست دانست. به عبارت دیگر، همه الگوریتم‌های فرامکاشفه‌ای را می‌توان برای یافتن پاسخ همه مسائل به کار گرفت. آنچه مهم است، یافته شدن مقدار مناسب برای پارامترهای الگوریتم فرامکاشفه‌ای برای تنظیم درست قابلیت پویش و انتفاع آن است.

شکل (1-5) به خوبی نکته مطرح شده را به تصویر نشان داده است. الگوریتم‌های فرامکاشفه‌ای به دلیل اینکه همگی دارای پارامترهایی برای تنظیم هستند، این قابلیت را دارند که برای یک مساله جستجوی خاص به صورت بهینه طراحی شوند. به همین دلیل است که این الگوریتم‌ها امروزه شهرت بالایی را در حل طیف وسیعی از مسائل بهینه‌سازی کسب نموده‌اند.



شکل (1-5) : ارتباط مفاهیم قابلیت پویش و انتفاع با یکدیگر در الگوریتم‌های جستجوی فرامکاشفه‌ای

در حل بسیاری از مسائل بهینه‌سازی، بهتر است الگوریتم‌های جستجوی فرامکاشفه‌ای در ابتدای فرآیند جستجوی خود در فضای حالت مساله قابلیت پویش بیشتری داشته باشند و هر چه که به پایان عملیات جستجو نزدیک‌تر می‌شوند، اندازه قابلیت انتفاع خود را افزایش دهند. به عبارت دیگر، بهتر است که در آغاز عملیات جستجو، برای افزایش احتمال یافتن پاسخ‌های به اندازه کافی خوب، ریسک کنند و درپایان عملیات جستجو با حفظ دستاوردهای حاصل شده بر سرعت همگرایی خود بیافزایند.

1-2-3 طبقه‌بندی الگوریتم‌های فرامکاشفه‌ای

معیارهای مختلفی می‌تواند برای طبقه‌بندی الگوریتم‌های فرامکاشفه‌ای استفاده شوند. این معیارها در ادامه توضیح داده شده اند.

- **جمعیتی و غیرجمعیتی :** الگوریتم‌های جمعیتی در حین جستجو، یک جمعیت از جواب‌ها را در نظر می‌گیرند. این در حالی است که الگوریتم‌های غیرجمعیتی (مبتنی بر یک جواب)، در حین فرآیند جستجو یک جواب را تغییر می‌دهند.
- **زیستی و غیرزیستی :** بسیاری از الگوریتم‌های فرامکاشفه‌ای از زندگی موجودات در طبیعت الهام گرفته شده‌اند و به عبارت دیگر، زیستی هستند. در این میان برخی از الگوریتم‌های

فرامکاشفه‌ای نیز از زندگی موجودات در طبیعت الهام گرفته نشده‌اند. بایستی توجه کنیم که الگوریتم‌های تکاملی نیز عضو خانواده الگوریتم‌های زیستی هستند.

- **تکاملی و غیرتکاملی :** الگوریتم‌های زیستی که به صورت مستقیم از فرامکاشفه‌ای بقاء اصلح که مبتنی بر نظریه داروین است استفاده می‌کنند، تکاملی محسوب می‌شوند. این در حالی است که سایر روش‌های جستجوی زیستی که به صورتی غیر مستقیم مبتنی بر فرامکاشفه بقاء اصلح داروین هستند، غیرتکاملی هستند.
- **باحافظه و بدون‌حافظه :** برخی از الگوریتم‌های فرامکاشفه‌ای فاقد حافظه می‌باشند، به این معنا که، این الگوریتم‌ها از اطلاعات بدست آمده در حین فرآیند جستجو استفاده نمی‌کنند، به طور نمونه الگوریتم تبرید شبیه‌سازی شده یا به اختصار SA (*Simulated Annealing*). این در حالی است که در برخی از الگوریتم‌های فرامکاشفه‌ای نظیر جستجوی ممنوعه از حافظه استفاده می‌شود. این حافظه، اطلاعات بدست آمده در حین جستجو را در خود ذخیره می‌کند.
- **احتمالی و قطعی :** در الگوریتم‌های فرامکاشفه‌ای احتمالی، نظیر تبرید شبیه‌سازی شده، یک سری قوانین احتمالی در حین جستجو مورد استفاده قرار می‌گیرد. اما یک الگوریتم فرامکاشفه‌ای قطعی نظیر جستجوی ممنوعه (*Tabu Search*)، مساله را با استفاده از تصمیمات قطعی حل می‌کند.

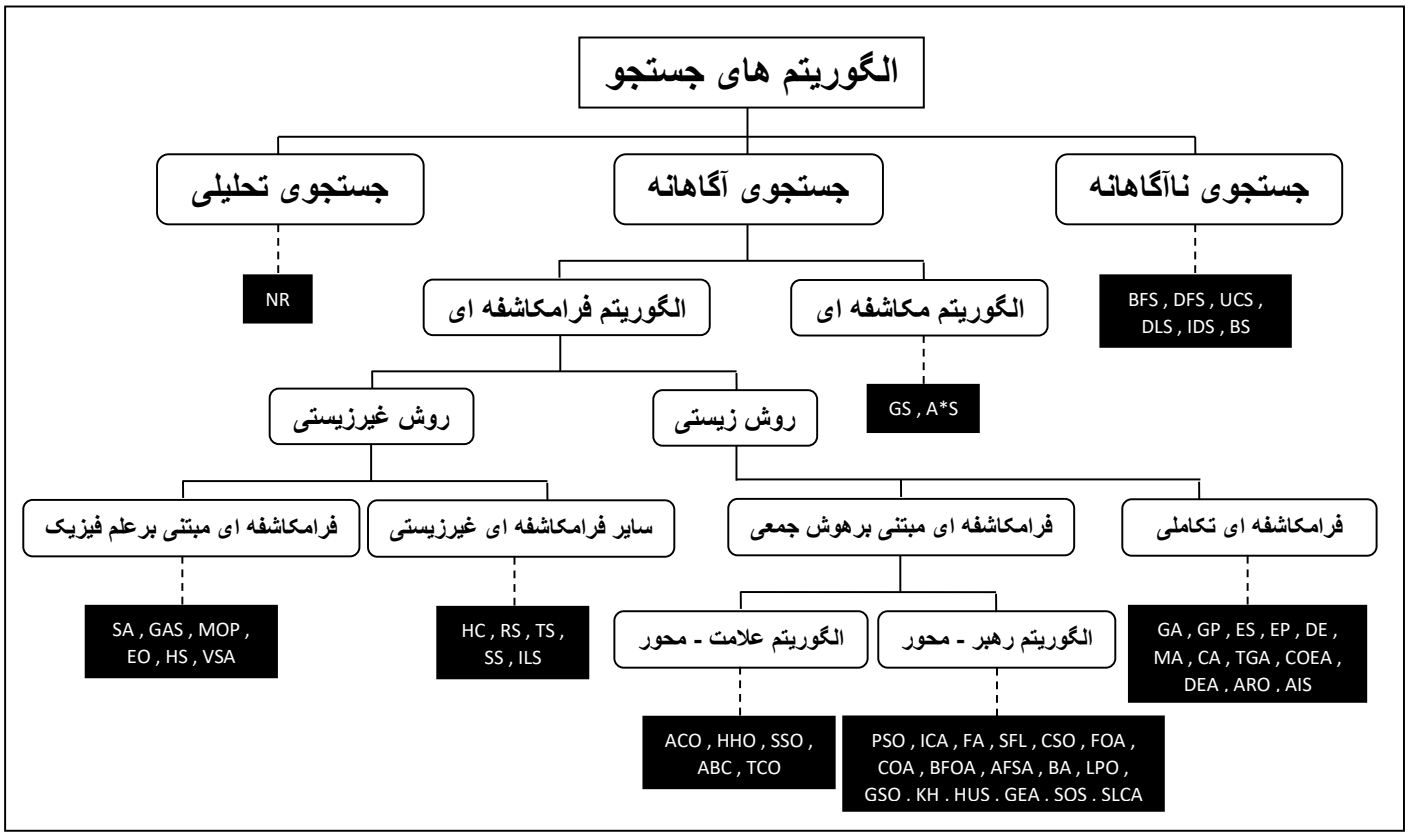
درجدول (1-1)، کلیه الگوریتم های فرامکاشفه ای قابل مشاهده است.

جدول (1-1) : معرفی الگوریتم‌های فرامکاشفه‌ای

اختصار	نام فارسی الگوریتم	ردیف	اختصار	نام فارسی الگوریتم	ردیف
TGA	الگوریتم ژنتیک تاگوچی	28	NR	الگوریتم نیتون-رافسون	1
CoEA	الگوریتم هم تکاملی	29	BFS	جستجوی اول-سطح	2
DEA	الگوریتم تکاملی دیپلوئیدی	30	DFS	جستجوی اول-عمق	3
ARO	بهینه‌سازی تولیدمثل غیرجنسی	31	UCS	جستجوی هزینه-یکنواخت	4
AIS	سیستم ایمنی مصنوعی	32	DLS	جستجوی عمق-محدود	5
ACO	بهینه‌سازی کلونی مورچگان	33	IDS	جستجوی عمیق شونده تکراری	6
HHO	بهینه‌سازی کندوی زنبورعسل	34	BS	جستجوی دو جهته	7
SSO	بهینه‌سازی جامعه علامت-محور	35	GS	جستجوی حریصانه	8
ABC	کلونی زنبور مصنوعی	36	A^*S	جستجوی A^*	9
TCO	بهینه‌سازی کلونی موریانه	37	HC	جستجوی تپه‌نوردی	10
PSO	بهینه‌سازی ازدحام ذرات	38	RS	جستجوی تصادفی	11
ICA	الگوریتم رقابت استعماری	39	TS	جستجوی ممنوعه	12
FA	الگوریتم کرم شب‌تاب	40	SS	جستجوی پراکنده	13
SFL	الگوریتم قورباغه جهنده	41	ILS	جستجوی محلی مکرر	14
CSO	بهینه‌سازی ازدحام گربه‌ها	42	SA	تبرید شبیه سازی شده	15
FOA	الگوریتم بهینه‌سازی مگس میوه	43	GAS	الگوریتم جستجوی گرانشی	16
COA	الگوریتم بهینه‌سازی فاخته	44	MOP	الگوریتم بهینه‌سازی مغناطیسی	17
BFOA	الگوریتم بهینه‌سازی غذایابی باکتری	45	EO	بهینه‌سازی افراطی	18
AFSA	الگوریتم بهینه‌سازی ازدحام ماهی‌ها	46	HS	جستجوی هارمونی	19

20	الگوریتم جستجوی گردابی	VSA	47	الگوریتم خفاش	BA
21	الگوریتم ژنتیک	GA	48	بهینه‌سازی گله شیرها	LPO
22	برنامه‌نویسی ژنتیک	GP	49	بهینه‌سازی جستجوی گروهی	GSO
23	استراتژی تکامل	ES	50	بهینه‌سازی گروه میگوها	KH
24	برنامه‌نویسی تکاملی	EP	51	جستجوی شکار	HuS
25	تکامل تفاضلی	DE	52	الگوریتم تکامل گرادیان	GEA
26	الگوریتم ممتیک	MA	53	جستجوی جانداران همزیست	SOS
27	الگوریتم فرهنگی	CA	54	الگوریتم رقابت لیگ فوتبال	SLCA

همانطور که پیش‌تر عنوان شد و در شکل (1-6) نیز مشاهده می‌شود، انواع گوناگون الگوریتم‌های جستجو به 3 دسته قابل تقسیم هستند. این دسته‌ها عبارتند از جستجوی تحلیلی، جستجوی ناآگاهانه و جستجوی آگاهانه.



شکل (1-6) : طبقه‌بندی الگوریتم‌های فرامکاشفه‌ای

فصل دوم

پردازش تکاملی

2-1 مقدمه

در این فصل، پردازش تکاملی یا به اختصار EC (*Evolutionary Computation*) را به عنوان شاخه‌ای از محاسبات هوشمند، که فرآیند انتخاب طبیعی را مدل می‌کند، توضیح خواهیم داد. در ادامه، ابتدا نظریه تکاملی داروین را تشریح نموده وسپس، مفاهیم پایه‌ای را در حوزه پردازش تکاملی ارائه خواهیم کرد. پس از آن، به توصیف مراحل یک الگوریتم تکاملی خواهیم پرداخت. بخش‌های بعدی، به تشریح مباحث مربوط به یک الگوریتم تکاملی می‌پردازند. این مباحث عبارتند از روش‌های نمایش کروموزوم (*Chromosome Encoding*)، جمعیت اولیه (*Initial Population*)، تابع برازش (*Fitness Function*)، عملگرانتخاب (*Selection Mechanism*)، عملگر تولیدمثل (*Reproduction*)، نخبه‌گرایی (*Elitism*) و شرایط توقف (*Termination Condition*). در انتهای این فصل، مفاهیمی چون کنترل قابلیت‌های پویش و انتفاع در الگوریتم‌های تکاملی، نظریه اسکیما، الگوریتم‌های تکاملی موازی و بهینه‌سازی چند-هدفه تکاملی مورد بحث و بررسی قرار خواهد گرفت.

2-2 نظریه داروین

تکامل یک فرآیند بهینه‌سازی است، که هدف آن بهبود توانایی یک ارگانیزم (یا سیستم) برای نجات در یک محیط متغیر و پویا می‌باشد. تمرکز ما بر روی تکامل زیست‌شناختی است. در این حوزه خاص نیز در تعریف تکامل زیست‌شناختی دیدگاه‌های مختلفی وجود دارد که دیدگاه لامارکی و داروینی نسبت به بقیه محبوب‌تر هستند.

نظریه تکامل لامارک، درباره انتقال موروثی (*Heredity*) می‌باشد. ایده اصلی این است که موجودات در طول زندگی، خود را با شرایط محیطی با یادگیری ویژگی‌هایی، وفق می‌دهند و آن ویژگی‌ها را به فرزندان خود انتقال می‌دهند. فرزندان سپس تطبیق‌پذیری را ادامه می‌دهند. لامارک معتقد است که در طول زمان، موجودات ویژگی‌هایی را که به آنها نیاز ندارند، از دست می‌دهند و با بکارگیری ویژگی‌هایی که مفید هستند، منجر به توسعه آنها می‌شوند. در این باره ولی نظریه داروین نگاهی کاملا متفاوت دارد. داروین سه نکته مهم را مطرح کرده است. این نکته‌ها عبارتند از :

1) حیات دیرینه است و صدها میلیون سال از آن وجود آن می‌گذرد.
2) حیات تنها با یک یا تعدادی از ارگانیسم‌های ساده، که بعدها تکامل یافته و تبدیل به میلیون‌ها گونه متفاوت امروزی شده‌اند، آغاز شده است.
3) تمامی فرآیند خلقت این گونه‌ها، ناشی از یکی از نیروهای طبیعت با نام انتخاب طبیعی (*Natural Selection*) بوده است.

نکته نخست ریشه در فسیل‌هایی داشت که داروین در زمان سفر خود به دور دنیا آنها را یافته بود. تنوع فسیل‌ها در طی زمان‌های متمادی و اینکه فسیل‌های جوان‌تر نشان‌دهنده موجود پیچیده‌تری نسبت به فسیل‌های پیرتر بودند، داروین را برای تبیین نکته دوم ترغیب نمود. در نظریه تکاملی داروین وجود نیرویی هدایت کننده در طبیعت با نام انتخاب طبیعی است. بر طبق دیدگاه داروینرین، این نیرو منجر به

از میان رفتن نمونه‌های ضعیف‌تر و زنده ماندن نمونه‌های برتر در گونه‌های مختلف جانوری می‌شود. زنده ماندن نمونه‌های برتر در گونه‌های مختلف جانوری می‌شود. زنده ماندن نمونه‌های برتر در گونه‌های مختلف جانوری، شانس آنها را برای تولید یک نسل جوان و احتمالا بهتر از والدین خویش، افزایش می‌دهد. در واقع از دیدگاه داروین، انتخاب طبیعی راز بقاء برترین‌ها در جانوران است. به عبارت دیگر، این برترین‌ها هستند که شانس بیشتری را برای انتقال ژن‌ها یا همان خصیصه‌هایشان به نسل‌های بعد، در اختیار دارند. بنابراین آینده متعلق به برترین‌هاست.

یکی از مسائلی که نقشی بی‌دلیل در تعیین برترین موجودات دارد، محیطی است که موجود در آن زندگی می‌کند. به عبارت دیگر، گونه‌های مختلفی از موجودات برای زندگی در یک محیط مشخص در طی نسل‌های طولانی تکامل می‌یابند. بنابراین، می‌توان هر موجود زنده را پاسخی دانست که توسط طبیعت و در طی سالیان متمادی برای حل مساله زندگی در یک محیط خاص یافته شده است. منظور از مساله در یک محیط خاص آن است که هر محیطی شرایطی دارد که منجر می‌شود برخی ویژگی‌های جاندار برای تسهیل ادامه حیات در آن برجسته شوند. در ذیل، به بررسی چندین محیط زندگی متفاوت جهت تبیین تاثیر نیروی انتخاب طبیعی در تکامل گونه‌های متنوع جانوری می‌پردازیم :

- محیط تاریک غار موجب تکامل سیستم شنوایی خفاش و تضعیف سیستم بینایی این پستاندار شده است.
- رنگ سفید خرس‌های قطبی برای افزایش امکان موفقیت عملیات شکار این حیوان، پاسخی است که طبیعت برای مساله شکار خرس‌ها در قطب در طی سالیان طولانی ارائه کرده است.
- به دلیل ارتفاع بالایی که عقاب حین پرواز در یک کوهستان دارد، امکان رصد کردن صید برای این پرنده، تنها با داشتن چشم‌هایی فوق پیشرفته و تیزبین میسر است.
- ویژگی استتار در بسیاری از موجودات نظیر آفتاب‌پرست‌ها، مارمولک‌ها و حتی پروانه‌ها در طی سالیان متمادی به نوعی تکامل یافته که این جانوران بتوانند برای صید کردن و یا پنهان ماندن از چشم صیاد به بهترین شکل ممکن عمل نمایند.
- دلیل زندگی گروهی در بسیاری از موجودات نظیر غزال‌ها و گورخرها، ریشه در فشاری است که در گذر زمان به حیوان برای اجتناب از زندگی فردی وارد شده است. اگر این جانوران به شکل فردی زندگی کنند، شانس شکار شدن آنها توسط صیادانی نظیر شیرها به شدت افزایش می‌یابد.
- گوزن‌های قرمز نر در فصل پاییز، برای دستیابی به ماده‌ها با یکدیگر به رقابت برمی‌خیزند. موفقیت برای تولیدمثل در این حیوانات، بستگی به قدرت جنگیدن دارد. قوی‌ترین نرها مالک تعداد مادگان بیشتری شده و بنابراین شانس بالاتری را برای انتقال ژن‌هایشان به نسل‌های آتی در اختیار خواهند داشت. هرچند جنگیدن فوائد بلقوه بسیاری دارد، ولی هزینه‌های آن نیز بسیار جدی هستند (هزینه‌هایی نظیر شکسته شدن پاها و یا کور شدن چشم‌ها). به همین دلیل گوزن‌های قرمز در طی سالیان طولانی یادگرفته‌اند که با به حداقل رسانیدن هزینه‌های جنگ، از نبرد با رقبایی که شانس پایینی برای غلبه کردن بر آنها دارند، پرهیز نمایند.
- بر طبق نظریه تکامل، شانس برترین‌ها برای انتقال ژن‌هایشان به نسل‌های آینده بالاتر است. نکته حائز اهمیت در اینجا، هدف مشترک انتقال ژن به نسل‌های آینده در تمام گونه‌های جانوری است. با توجه به آنکه انتظار می‌رود که فرزندان 50%، نوه‌ها 25% و نتیجه‌ها 12/5% ژن‌های یک

جاندار را در خود داشته باشند، این توقع کاملا معقول است که علاقه حیوانات به فرزندان، نوه‌ها و نتیجه‌هایشان بر طبق همین اعداد و درصدهای مذکور باشد. جالب آن است که بدانیم برای یک جاندار، خواهران و برادران با 50% ژن مشترک، خواهران و برادران ناتنی و نیز خواهرزادگان و برادرزادگان با 25% ژن مشترک و عموزادگان با 12/5% ژن مشترک، خویشانی هستند که لزوما از نسل آن جاندار نبوده، ولی در ژن‌هایشان مشترکاتی قابل توجه با آن جاندار دارند. این موضوع به نحو خیره‌کننده‌ای در بسیاری از موجودات نظیر سنجاب زمینی و سگ علفزار قابل مشاهده است. در این حیوانات، می‌توان رفتاری فداکارگونه را در مراقبت از فرزندان اقوامشان مشاهده نمود. رفتاری که متناسب با درصد اشتراک ژن‌ها بوده و حتی گاهی اوقات ممکن است به از دست رفتن جان حیوان منجر شود. شیرهای ماده، با توجه به محدود بودن منابع غذایی، حیوان ممکن است که هرگز بچه‌دار نشود و در عوض ترجیح دهد که با نگهداری از فرزندان خواهر قوی‌تر و برازنده‌تر خود، شانس انتقال ژن‌های خویش را به نسل‌های آتی افزایش دهد.

- شکل آواز پرندگان بستگی جالبی به زیستگاه آنها دارد. پرندگانی که در محیط‌های جنگلی با پوشش انبوه گیاهی زندگی می‌کنند، صدایی با فرکانس پایین‌تر و تحریرهای فاصله‌دارتر دارند. این در حالی است که آواز پرندگانی که در علفزارها و زیستگاه‌های باز زندگی می‌کنند، فرکانسی بالاتر داشته و با تحریرهای سریع و پی‌درپی همراه است. با توجه به آنکه به نظر می‌رسد که شنیده شدن درست و کامل صدا (جهت جذب جفت، دفع رقیب، هشدار به صیاد و غیره) هدفی مهم برای پرندگان محسوب می‌شود، به همین دلیل نوع پوشش‌های گیاهی محل زندگی پرندگان توجیه کننده ساختاری است که در آواز پرندگان وجود دارد.
- رفتارهای متقلبانه در زندگی گروهی حیوانات، در طی فرآیند تکامل، حذف شده و از میان رفته‌اند. یک نمونه از این رفتارها در زندگی شترمرغ‌ها مشاهده می‌شود. یک شترمرغ هنگامی که به صورت گروهی زندگی می‌کند، سطح هوشیاری بالاتری را در مقایسه با زندگی فردی خواهد داشت. در گروه‌های بزرگ، هوشیاری کلی در حداکثر ممکن و نزدیک به صد درصد است و برای شیر کمین کرده تقریبا غیرممکن است که بتواند در خلال سر بالا کردن‌های تصادفی شترمرغ‌ها، بدون آنکه ردیابی شود، به جلو بخزد. شاید برای یک شترمرغ به صرفه‌تر باشد که تقلب نموده و در تمام اوقات با سر رو به پایین مشغول خوردن باشد. گروه‌ها با تعداد کمتری شترمرغ متقلب شایستگی و برازش بالاتری را در مقایسه با گروه‌های با تعداد متقلبین بیشتر داشته و بر اساس نظریه بقاء اصلح، شانس بیشتری را برای زندگی داشته‌اند.
- موجودات معمولا در راستایی تکامل می‌یابند که بتوانند با وفق‌پذیری بهتر با محیط زندگی خود، فرآیند انتقال ژن‌هایشان را به نسل‌های آتی تسهیل نمایند. از آنجا که صید نشدن توسط صیاد در این راستا امری اجتناب‌ناپذیراست، بنابراین به نظر می‌رسد که حیوان بایستی با تقویت توان فرار کردن (تکامل شکل دست‌ها و پاها برای افزایش سرعت دویدن) و یا تقویت توان استتار (تکامل قابلیت تغییر رنگ بدن متناسب با محیط زندگی)، توانایی خود را برای زندگی ایمن در کنار صیاد افزایش می‌دهد. با توجه به این توضیحات، چه پاسخی می‌تواند برای این پرسش وجود داشته باشد که چرا طاووس‌های نر زیباتر و درخشان‌تر، با وجودی که امکان شناسایی شدن آنها توسط صیادان بالقوه‌ای نظیر ببرها افزایش می‌یابد، در مقایسه با طاووس‌های نر با پرهای شکسته، ناقص و رنگ پریده، توسط طاووس‌های ماده ترجیح داده می‌شوند؟ نظریه تکامل به این

پرسش چنین پاسخ می‌دهد. به دلیل درصد بالاتر ژن‌های سالم در طاووس‌های نر زیباتر، ریسک افزایش خطر شکار شدن این طاووس‌ها توسط طاووس‌های ماده پذیرفته می‌شود. به عبارت دیگر، زیبایی نشانه‌ای است که طاووس‌های ماده را از صحت سلامت ژن‌های طاووس نر آگاه می‌سازد.

شاخه پردازش تکاملی با الهام از نظریه تکامل داروین، به حل مسائل گوناگون بهینه‌سازی، جستجو و یادگیری ماشین می‌پردازد.

2-3 مفاهیم پایه در پردازش تکاملی

با کشف ساختار DNA تحولی شگرف در بررسی نظریه تکاملی داروین حاصل شد. این ساختار که می‌تواند از یک نسل به نسل دیگر منتقل شود، ساختار مارپیچی دوگانه‌ای دارد که در صورت بازکردن آن، کدهای بسط یافته‌ای حاصل می‌شوند.این کدهای بسط یافته ژن نامیده می‌شوند. انسان تقریبا 22هزار ژن در DNA خود دارد. این ژن‌ها اطلاعاتی را با خود حمل می‌کنند که مشخص کننده رنگ چشم، طول قد و هر آنچه که بیانگر خصیصه‌های فردی یک انسان است می‌باشند. یک دسته ژن ممکن است تولیدکننده یک ماهی، یک عقاب و یا یک فیل باشند. با بررسی دنباله ژنی در موجودات مختلف این امکان وجود دارد که چگونگی تغییر گونه‌های متنوع جانوری را به یکدیگر، در طی نسل‌های متمادی دنبال نماییم.

در ذیل برخی از مفاهیم مهمی که در الگوریتم‌های تکاملی به وفور از آنها یاد می‌شود تشریح شده‌اند :

- **کروموزوم (*Chromosome*) :** اطلاعات ژنی یک موجود در کروموزوم ذخیره می‌شود. هر کروموزوم از DNA تشکیل شده است. در یک الگوریتم تکاملی منظور از کروموزوم، یک رشته، گراف، درخت و یا دنباله صفر و یک است که معمولا معادل یک پاسخ برای مساله مورد بررسی می‌باشد.
- **ژن (*Gene*) :** کروموزوم‌ها به چندین قسمت تقسیم‌بندی می‌شوند که ژن نام دارند. ژن‌ها خصوصیات گونه‌ها یا همان افراد را تشکیل می‌دهند. در یک الگوریتم تکاملی، هر پاسخ از چندین بخش تشکیل شده است که به هر کدام از آن بخش‌ها یک مشخصه یا ویژگی گفته می‌شود. مشخصه یا ویژگی در هر پاسخ معادل ژن در کروموزوم است.
- **آلل (*Allele*) :** هر ژن مجموعه‌ای از مقادیر مجاز را می‌تواند اختیار کند. به هر کدام از این مقادیر یک آلل گفته می‌شود. به عنوان مثال، یک ژن مربوط به رنگ چشم است و آلل‌های ممکن برای آن عبارتند از سیاه، خاکستری، قهوه‌ای، آبی، سبز و عسلی (و نه سفید). در یک الگوریتم تکاملی، مقادیر مجاز برای مشخصه‌های هر پاسخ معادل مفهوم آلل است.
- **ژنوتایپ (*Genotype*) :** ترکیب کامل تمامی ژن‌ها برای یک فرد مشخص، ژنوتایپ (ژنوتیپ) نامیده می‌شود. دنباله ژنی مربوط به هر پاسخ در یک الگوریتم تکاملی ژنوتایپ آن پاسخ را نشان می‌دهد.

- **فنوتایپ (*Phenotype*) :** خصوصیات ظاهری یک شخص که از رمزگشایی یک ژنوتایپ حاصل می‌شود، فنوتایپ (فنوتیپ) نامیده می‌شود. به عبارت دیگر، تجلی معنایی ژنوتایپ مربوط به یک پاسخ در دنیای واقعی، گویای فنوتایپ آن پاسخ است.
- **برازش (*Fitness*) :** شایستگی یک موجود در یک جمعیت، برازش آن موجود نامیده می‌شود. برازش هر پاسخ در جمعیت، با توجه به شایستگی فنوتایپ آن پاسخ تعیین می‌گردد. در یک الگوریتم تکاملی، با استفاده از یک یا چند تابع، برازش هر پاسخ محاسبه می‌شود. تابع برازش، تنها ارتباط یک الگوریتم تکاملی با مساله مورد تحلیل می‌باشد.

2-4 مراحل یک الگوریتم تکاملی

فرآیند تکامل، از طریق انتخاب طبیعی موجودات در یک جمعیت، را می‌توان به صورت یک جستجو در فضای مقادیر ممکن کروموزوم‌ها در نظر گرفت. مراحل یک الگوریتم تکاملی مطابق شکل (2-1) عبارتند از :

مرحله 1) تولید جمعیت اولیه : یک جمعیت از کروموزوم‌ها یا همان پاسخ‌های مساله تولید می‌شود. قبل از تولید جمعیت اولیه بایستی نحوه نمایش کروموزوم‌ها، که گویای یک پاسخ برای مساله مورد بررسی است، تعیین شود. خروجی مرحله اول یک جمعیت از پاسخ‌هاست.

مرحله 2) محاسبه برازش جمعیت ورودی : برازش تک تک کروموزوم‌های جمعیت تولید شده، با توجه به تابع برازش تعیین شده، محاسبه می‌شود. خروجی این مرحله، یک جمعیت از پاسخ‌های ارزیابی شده می‌باشد.

مرحله 3) انتخاب برای تولیدمثل : آن دسته از اعضاء جمعیت ورودی که برازش بالاتری نسبت به سایر اعضاء دارند، برای تبیین قانون بقاء اصلح داروین، انتخاب می‌شوند.خروجی این مرحله، جمعیتی از والدین برازنده است.

مرحله 4) بازترکیب والدین انتخاب شده : با توجه به جمعیت والدین برازنده و با استفاده از عملگر بازترکیب، جمعیتی از فرزندان تولید می‌شوند. اعمال عملگر بازترکیب، با توجه به یک پارامتر با نام احتمال بازترکیب (P_c) انجام می‌شود. خروجی این مرحله جمعیتی از فرزندان تولید شده می‌باشد.

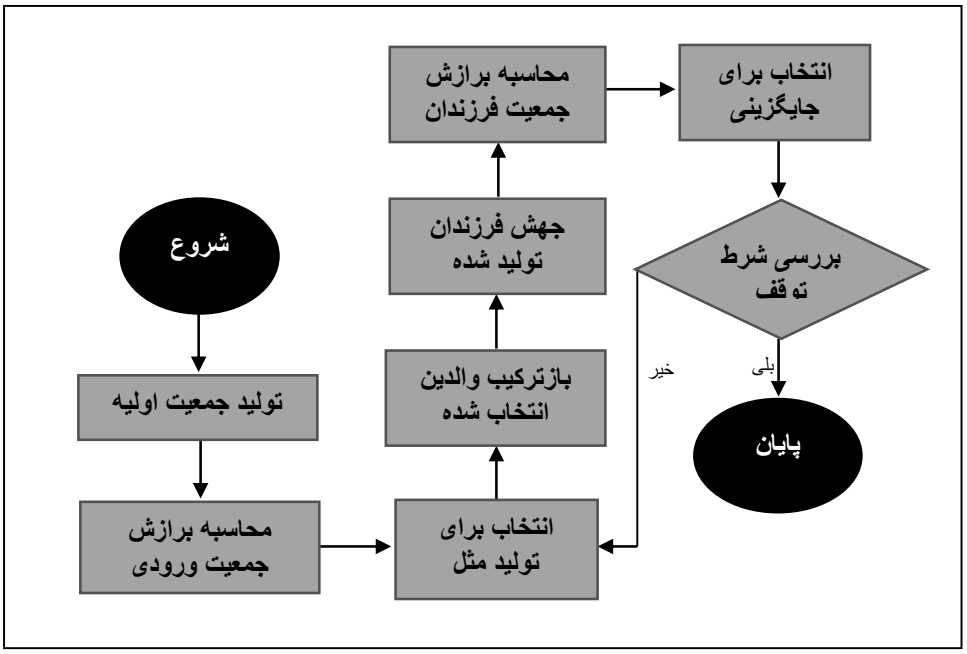
مرحله 5) جهش فرزندان تولید شده : فرزندان جدید مرحله قبل تحت عملگر جهش قرار می‌گیرند. البته عملگر جهش با یک احتمال برروی دنباله ژنی فرزندان رخ می‌دهد. این احتمال با نام احتمال جهش (P_m) شناخته می‌شود. خروجی این مرحله جمعیتی از فرزندان جهش یافته است.

مرحله 6) محاسبه برازش جمعیت فرزندان : شایستگی فرزندان جهش یافته، با استفاده از تابع برازش، محاسبه می‌شود. خروجی این مرحله، جمعیت فرزندان ارزیابی شده است.

مرحله 7) انتخاب برای جایگزینی : با توجه به جمعیت والدین (ورودی مرحله سوم) و جمعیت فرزندان ارزیابی شده (خروجی مرحله ششم) یک جمعیت جدید برای نسل بعد (هرتکرار در این الگوریتم معادل

یک نسل است.) تولید می‌شود. که خروجی این مرحله جمعیتی است که در آن بخشی از والدین نسل قبل و تعدادی از فرزندان جدید تولید شده نسل جاری وجود دارند.

مرحله 8) بررسی شرط توقف : در مورد ادامه فرآیند تکاملی الگوریتم تصمیم‌گیری می‌شود. در صورت عدم ارضاء شرط توقف، فرآیند تکاملی الگوریتم با رجوع به مرحله 3 ادامه می‌یابد. در غیر این صورت، الگوریتم متوقف شده و بهترین پاسخ در آخرین نسل به عنوان حاصل جستجوی تکاملی در خروجی ارائه می‌شود.



شکل (2-1) : روندنمای یک الگوریتم تکاملی

فرآیند جستجوی فرامکاشفه‌ای در الگوریتم‌های تکاملی تحت تاثیر مولفه‌های زیر می‌باشد :

- طریقه کد کردن راه‌حل مساله به عنوان کروموزوم.
- طراحی تابعی که با آن برازش کروموزوم‌ها (پاسخ‌ها) را ارزیابی کنیم.
- الگوریتم تولید جمعیت اولیه.
- الگوریتم مربوط به عملگرهای انتخاب، بازترکیب و جهش.
- نحوه تولید جمعیت نسل بعد.
- تعیین شرط توقف.

2-5 روش‌های نمایش کروموزوم

هر موجود در الگوریتم تکاملی، بیانگر یک راه‌حل کاندید برای مساله بهینه‌سازی است. ویژگی‌های موجودات توسط کروموزوم آنها نشان داده می‌شوند (*Chromosome Encoding*). این ویژگی‌ها مربوط به متغیرهای موجود در مسائل بهینه‌سازی می‌باشند. هرمتغیری که نیاز به بهینه شدن داشته باشد، به عنوان یک ژن مشخص می‌شود. یک ژن، کوچک‌ترین واحد اطلاعات است. یکی از مهم‌ترین مراحل در طراحی یک الگوریتم تکاملی، یافتن روش نمایش مناسب برای راه‌حل‌های کاندید (کروموزوم) می‌باشد. کارآیی و پیچیدگی الگوریتم جستجو به روش نمایش کروموزوم بسیار وابسته است. الگوریتم‌های تکاملی مختلف، از نمایش‌های گوناگونی استفاده می‌کنند. در بیشتر الگوریتم‌های تکاملی، راه‌حل‌ها به صورت برداری از یک نوع داده خاص نمایش داده می‌شوند (برنامه‌نویسی ژنتیک یک استثنا می‌باشد، زیرا که موجودات در آن به شکل درخت نمایش داده می‌شوند).

مدل نمایش سنتی برای الگوریتم‌های تکاملی، بردار دودویی با طول ثابت است. با داشتن فضای جستجوی n_x - بعدی، هر موجود دارای n_x متغیر می‌باشد که هر متغیر به صورت رشته بیتی کد شده است. برای متغیرهایی با مقادیر اسمی (*Nominal*)، هرمقدار اسمی را می‌توان مطابق شکل (2-2) به صورتی که 2^n تعداد کل مقادیر اسمی گسسته برای آن متغیر است، نشان داد.

0	1	1	0	1	0	0	0	1

شکل (2-2) : نمایش دودویی کروموزوم

برای حل مسائل بهینه سازی با متغیرهای پیوسته، روش نمایش دیگری لازم است که نمایش مبتنی براعدادحقیقی نامیده می‌شود. در این روش مطابق شکل (2-3)، هر بیت درکروموزوم‌های جمعیت تکاملی، با یک عدد حقیقی مقداردهی می‌شود.(مانند شکل 2-3)

1.12	32.6	15.1	19.4	6.18	0.12	10.5	12.3	65.1

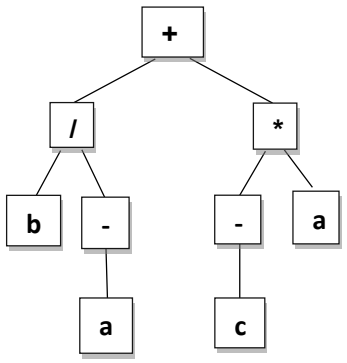
شکل (2-3) : نمایش اعداد حقیقی کروموزوم

روش دیگر نمایش کروموزوم‌ها، روش جایگشت عناصر (*Permutation of Elements*) است. هدف در این روش، یافتن بهترین دنباله‌ای ازعناصر می‌باشد، به طوری که برای هر دنباله، برازشی مشخص قابل محاسبه است. مساله فروشنده دوره‌گرد یا به اختصار TSP (*Travelling Salesman Problem*)، بهترین نمونه برای استفاده از چنین روش نمایش کروموزوم می‌باشد. دراین روش مطابق شکل (2-4)، مقادیر هیچ دو ژنی در هر کدام از والدین نبایستی تکراری باشد.

4	3	9	6	1	7	2	8	5

شکل (2-4) : نمایش جایگشت عناصر کروموزوم

نمایش درختی روش نمایش کروموزوم مشهور دیگری است که درآن مطابق شکل (2-5)، هر کدام از اعضاء جمعیت درالگوریتم تکاملی به شکل یک درخت نمایش داده می‌شوند. از این روش معمولا برای تکامل برنامه‌ها ویا روابط ریاضی بهره گرفته می‌شود. دربرنامه‌نویسی ژنتیک از این روش نمایش برای تمایش کروموزوم ها استفاده می‌شود.



شکل (2-5) : نمایش درختی کروموزوم

2-6 جمعیت اولیه

الگوریتم‌های تکاملی مبتنی بر جمعیت هستند. هرالگوریتم تکاملی، جمعیتی از راه‌حل‌های کاندید را نگهداری می‌کند. اولین مرحله در الگوریتم تکاملی برای حل مسائل بهینه‌سازی، تولید یک جمعیت اولیه (*Initial Population*) است. روش استاندارد و رایج برای تولید جمعیت اولیه، نسبت دادن مقدار تصادفی از دامنه مجاز، به هر یک از ژن‌های هر کدام از کروموزوم‌ها است. این روش با نام مقداردهی اولیه تصادفی شناخته می‌شود. هدف از مقداردهی تصادفی این است که جمعیت اولیه را به صورت یکنواخت از کل فضای جستجو بدست آورده باشیم. در صورتی که ناحیه‌ای از فضای جستجو در جمعیت اولیه پوشش داده نشود، امکان غفلت از آن قسمت‌ها در فرآیند جستجو وجود دارد.

روش دیگر مقداردهی جمعیت اولیه در الگوریتم‌های تکاملی، روش مقداردهی اولیه هوشمندانه است. در این روش، براساس اطلاعات موجود از فضای جستجوی مساله، کروموزوم‌های اولیه به گونه‌ای تولید می‌شوند که برازندگی آنها در بدو تولد بالا باشد.

اندازه جمعیت اولیه، یکی از پارامترهای مهم الگوریتم‌های تکاملی است. بایستی توجه شود که این اندازه ممکن است ثابت و یا متغیر باشد. به عنوان نمونه، در الگوریتم‌های ژنتیک این اندازه ثابت است، در حالی که در سیستم ایمنی مصنوعی اندازه جمعیت اولیه پویا می باشد.

جمعیت اولیه، در میزان پیچیدگی محاسباتی و قابلیت‌های پویش و انتفاع الگوریتم‌های تکاملی اثرگذار است. افزایش این پارامتر سبب تقویت هر دو قابلیت پویش و انتفاع می‌شود. به عبارت دیگر، هر چقدر که یک الگوریتم تکاملی توان پویشی داشته باشد، با افزایش اندازه جمعیتش، آن توان پویشی تقویت می‌شود. به همین ترتیب، هر چقدر که الگوریتم توان انتفاعی داشته باشد، با افزایش اندازه جمعیتش، توان مزبور تقویت خواهد شد.

دلیل این مدعا آن است که با افزایش اندازه جمعیت، شانس انجام عملگرهای تولیدمثل که معمولا ترکیبی از دو قابلیت پویش و انتفاع را در خود دارند، افزایش می‌یابد. هرچقدر که تعداد موجودات بیشتر شوند، میزان پیچیدگی محاسباتی هر نسل افزایش می‌یابد. هنگامی که جمعیت تکاملی کوچک باشد، قسمت کوچکی از فضای جستجو توسط الگوریتم کنکاش می‌شود. در این حالت، با وجود اینکه پیچیدگی زمانی هر نسل پایین است، اما الگوریتم تکاملی معمولا نیاز به نسل‌های بیشتری برای رسیدن به همگرایی دارد.

2-7 تابع برازش

درمدل تکاملی داروینی، موجوداتی با ویژگی‌های برتر، شانس بیشتری برای بقاء و تولیدمثل دارند. به منظور تشخیص توانایی بقاء هر کدام از پاسخ‌های موجود در الگوریتم‌های تکاملی، یک تابع برازش (*Fitness Function*) مورد استفاده قرار می‌گیرد. این تابع میزان شایستگی راه‌حل نمایش داده شده توسط کروموزوم را مشخص می‌نماید. تابع برازش f، میزان شایستگی هر کروموزوم را به یک مقدار عددی، طبق رابطه زیر نگاشت می‌کند.

$$f : \tau^{n_x} \rightarrow R$$

تابع برازش یک معیار مطلق برای اندازه‌گیری شایستگی است و راه‌حل نمایش داده شده توسط کروموزوم، مستقیما توسط تابع هدف مورد ارزیابی قرار می‌گیرد. برای برخی از کاربردها، برای نمونه یادگیری قوانین دسته‌بندی در حوزه داده کاوی، امکان یافتن یک تابع برازش قطعی وجود ندارد. درمقابل، یک معیار برازش نسبی برای سنجش کارآیی موجود در ارتباط با سایر موجودات در جمعیت استفاده می‌شود.

هر کدام از مسائل بهینه‌سازی، تاثیری متفاوت در فرموله کردن تابع برازش دارند. تابع برازش از اهمیت ویژه‌ای برخوردار می‌باشد. مهمترین کاربرد تابع برازش در عملگرانتخاب است که در آن برازنده‌ترین اعضاء در جمعیت الگوریتم تکاملی برای ارسال به مرحله تولیدمثل گزینش می‌شوند. تنها رابطه یک الگوریتم تکاملی (و درواقع هرالگوریتم فرامکاشفه‌ای) با مساله بهینه‌سازی مورد بررسی،

تابع برازش است. به جز تابع برازش، بقیه بخش‌های الگوریتم تکاملی برای بسیاری از مسائل ممکن است مشابه باشد.

2-8 عملگر انتخاب

مکانیزم انتخاب (*Selection Mechanism*) یکی از عملگرهای اصلی در الگوریتم‌های تکاملی است و مستقیما به مفهوم بقاء اصلح در نظریه داروین مربوط می‌شود. هدف اصلی در عملگر انتخاب، یافتن راه‌حل‌های برتر است. مهمترین پارامتر درعملگر انتخاب، فشارانتخاب (*Selective Pressure*) به شمار می رود که عبارت است از سرعت پرشدن جمعیت از راه‌حل‌های خوب، توسط بکارگیری عملگر انتخاب. عملگرهایی با فشار انتخاب زیاد، تنوع (*Diversity*) را در جمعیت به نسبت عملگرهایی با فشار انتخاب پایین، به سرعت کاهش می‌دهند که منجر به همگرایی زودرس الگوریتم به راه‌حل‌هایی با بهینگی محلی می‌شود. فشار انتخاب زیاد (توجه بیش از اندازه به اعضاء برازنده در جمعیت)، قابلیت پویش را در الگوریتم‌های تکاملی محدود نموده و درمقابل، قابلیت انتفاع را در این الگوریتم‌ها تقویت می‌سازد. انواع عملگرهای انتخاب عبارتند از انتخاب تصادفی، انتخاب نسبی، انتخاب رتبه‌ای، انتخاب مسابقه‌ای و انتخاب برشی.

2-8-1 انتخاب تصادفی

انتخاب تصادفی (*Random Selection*)، یکی از ساده‌ترین عملگرهای انتخاب است. در این روش انتخاب، هر موجود دارای احتمال یکسان $1 / n_s$ می‌باشد که درآن، n_s اندازجمعیت است. درانتخاب تصادفی، از هیچ اطلاعات برازشی استفاده نمی‌شود. به این معنی که بهترین و بدترین موجودات دارای احتمال کاملا یکسان برای قرار گرفتن در نسل بعدی هستند. انتخاب تصادفی دارای کمترین فشار انتخاب در میان عملگرهای انتخابی می‌باشد.

2-8-2 انتخاب نسبی

در روش انتخاب نسبی (*Proportional Selection*)، شانس انتخاب موجودات برتر بیشتر است. در این روش انتخاب، یک توزیع احتمالی متناسب با شایستگی هر پاسخ ایجاد می‌شود و موجودات از طریق نمونه‌برداری از توزیع زیر انتخاب می‌شوند.

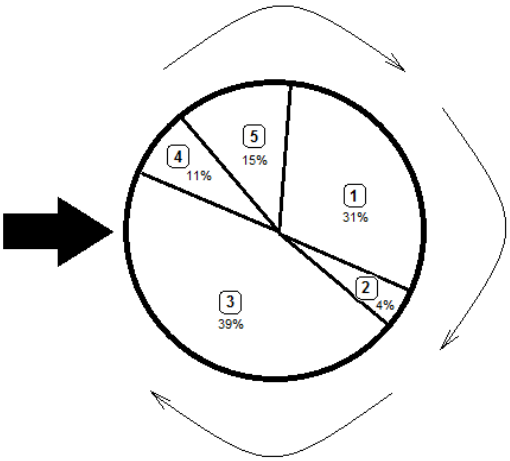
$$(x_i) = \frac{f_Y(x_i)}{\sum_{l=1}^{n_s} f_Y(x_i)} \varphi_s$$

در این رابطه، n_s تعداد کل موجودات در جمعیت، $(x_i)\varphi_s$ احتمال انتخاب عضو x_i در جمعیت توسط روش انتخاب نسبی، و نهایتا $f_Y(x_i)$ برازش عضو x_i است. برای مسائل کمینه سازی

(*Minimization*)، بایستی برازش عضو x_i را با استفاده از یک تابع به قسمی مطابق رابطه زیر تبدیل کنیم که معنای بیشینه‌سازی (*Maximization*) در آن حاصل شود.

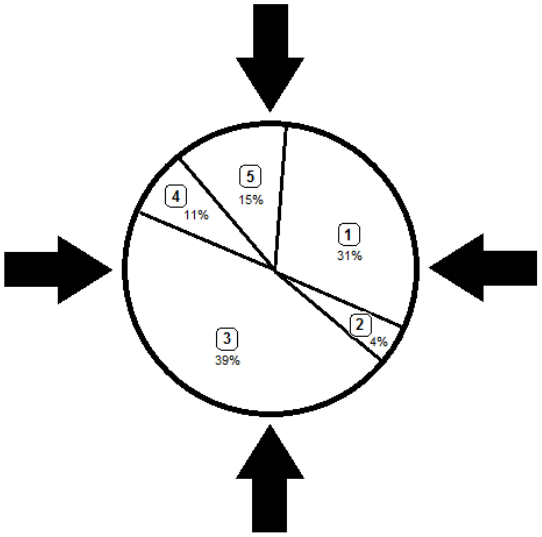
$$g_y(x_i) = \frac{1}{f_Y(x_i)}$$

به دلیل نحوه پیاده‌سازی الگوریتم انتخاب نسبی، از این روش در اغلب موارد با نام چرخ رولت (*Roulette Wheel Selection*) نیز یاد می‌شود. در این روش، توزیع احتمال به عنوان چرخ رولت در نظر گرفته می شود، به صورتی که اندازه هر قطعه متناسب با احتمال نرمال شده انتخاب هر موجود باشد. برای پیاده‌سازی این روش مطابق شکل (2-6)، ابتدا یک دایره (چرخ دوار) در نظر گرفته شده و این چرخ با توجه به تعداد کروموزوم‌ها به گونه‌ای تقسیم می‌شود که هر بخش متناظر با مقدار برازندگی کروموزوم مربوطه باشد. سپس به تعداد اعضاء در جمعیت، چرخ دوار را چرخانده و هر کجا که چرخ متوقف شود، با توجه به قطاع مشخص شده توسط اشاره‌گر چرخ، کروموزوم مربوط به آن قطاع انتخاب می‌گردد.



شکل (2-6) : انتخاب چرخ رولت با یک اشاره‌گر

در این روش به تعداد n_s (اندازه جمعیت اعضاء) چرخ رولت چرخانده شده و هر بار یک عضو انتخاب می‌شود. در این روش به اعضاء دارای برازش بالا توجه زیادی می‌شود. به همین دلیل مشکل بزرگ انتخاب چرخ رولت، فشار انتخاب بالای آن است که ممکن است استفاده از آن در الگوریتم تکاملی سبب همگرایی زودرس الگوریتم شود. چنانچه در روش انتخاب چرخ رولت مطابق شکل (2-7)، بجای یک اشاره‌گر از چندین اشاره‌گر (حداکثر تعداد اشاره‌گرها برابر با اندازه جمعیت یعنی n_s است) برای انتخاب اعضاء استفاده گردد، احتمال انتخاب اعضاء با برازش زیاد تعدیل شده و شانس انتخاب کروموزوم‌های ضعیف‌تر بیشتر می‌شود. در این روش، چرخ رولت تنها یک‌بار(به جای n_s بار) چرخانده می‌شود. از آنجا که تعداد اشاره‌گرها برابر با n_s است، کل جمعیت انتخابی با یک‌بار چرخانده شدن چرخ رولت ایجاد می‌شود.

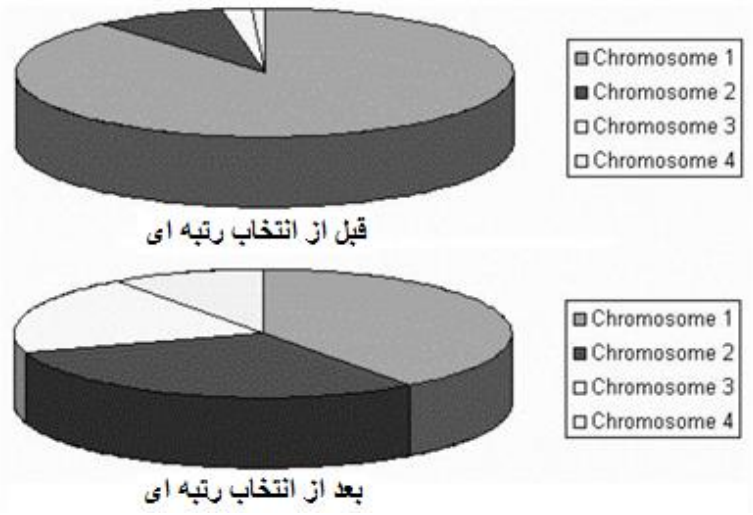


شکل (2-7) : انتخاب چرخ رولت با چند اشاره‌گر

توجه داشته باشید که در روش چرخ رولت با چند اشاره‌گر، فاصله اشاره‌گرهای در اطراف چرخ یکسان است. فشار انتخاب روش چرخ رولت بهبود یافته با چند اشاره‌گر، نسبت به نوع سنتی با یک اشاره‌گر، کم‌تر می باشد.

2-8-3 انتخاب رتبه‌ای

در روش انتخاب رتبه‌ای (*Rank-based Selection*)، بجای استفاده از مقدار مطلق برازندگی، از رتبه برازندگی اعضاء در جمعیت برای تعیین احتمال انتخاب استفاده می‌شود. در این روش، برازش برترین عضو جمعیت مساوی با n_s در نظر گرفته می‌شود. به دومین عضو برتر جمعیت برازش $n_s - 1$ نسبت داده می‌شود و این کار تا رسیدن به ضعیف‌ترین عضو جمعیت ادامه می‌یابد (بدیهی است که برازش ضعیف‌ترین عضو برابر با 1 خواهد بود). در صورت یکسان بودن برازش دو عضو در جمعیت، بایستی به صورت تصادفی به یکی برازش k و به دیگری k-1 را نسبت دهیم. در ادامه با استفاده از رابطه $\varphi_s(x_i) = \frac{f_Y(x_i)}{\sum_{l=1}^{n_s} f_Y(x_i)}$ ، اعضاء انتخابی برای معرفی به عملگرهای تولیدمثل مشخص می گردند. همانطور که درشکل (2-8) مشاهده می شود، روش انتخاب رتبه‌ای در قیاس با روش انتخاب چرخ رولت از فشار انتخاب پایین‌تری برخوردار است، چراکه شانس بالای اعضاء برازنده را در جمعیت تعدیل می‌نماید. به همین دلیل احتمال همگرایی زودرس الگوریتم تکاملی با استفاده از این روش انتخاب کاهش خواهد یافت.



شکل (2-8) : انتخاب رتبه‌ای و تعدیل شانس انتخاب اعضاء برازنده جمعیت

4-8-2 انتخاب مسابقه‌ای

درگام اول روش انتخاب مسابقه ای (*Tournament Selection*)، یک گره (به تعداد t) از موجودات را به طور تصادفی از جمعیت اعضاء انتخاب می‌کند، البته با فرض اینکه $t < n_s$ است که در آن، n_s مجموع کل افراد درون یک جمعیت می باشد. در گام دوم روش انتخاب مسابقه ای، کارآیی t عضو انتخاب شده با یکدیگر مقایسه شده و بهترین فرد از درون این گروه انتخاب می‌شود.

درصورتی که اندازه t خیلی بزرگ نباشد، این الگوریتم از انتخاب بهترین افراد جلوگیری می‌کند. بنابراین فشار انتخاب این الگوریتم کم می شود. از طرف دیگر، اگر t خیلی کم باشد، شانس انتخاب ضعیف‌ترین اعضاء افزایش می‌یابد. فشار انتخاب ارتباط مستقیمی با اندازه t دارد. اگر $t = n_s$ باشد، همیشه بهترین فرد انتخاب می‌شود و در نتیجه، فشار انتخاب بسیار زیاد خواهد شد. به علاوه، اگر $t = 1$ باشد، این الگوریتم تبدیل به انتخاب تصادفی می‌شود. دربیشتر مقالات و کتب، مقدار t معمولا برابر با مقادیر 2 و یا 3 تنظیم می‌شود.

2-8-5 انتخاب برشی

در روش انتخاب برشی (*Truncation Selection*)، ابتدا اعضای جمعیت را براساس شایستگی‌شان مرتب شده وسپس، از میان t درصد از برترین اعضاء جمعیت، n_s عضو را به صورت تصادفی انتخاب می شوند. دراین روش، هر چقدر مقدار t بزرگتر باشد، فشارانتخاب کمتر خواهد شد. در حالتی که $t = 100$ باشد، این روش انتخاب معادل با روش انتخاب تصادفی است.

2-9 عملگرهای تولیدمثل

تولید مثل یا جفت گیری (*Mating*)، فرآیند تولید فرزندان از والدین انتخاب شده توسط بکارگیری عملگرهای بازترکیب (*Recombination*) (تقاطع یا همبرش (*Crossover*)) وهمچنین، جهش (*Mutation*) می‌باشد. بازترکیب، فرآیند تولید یک و یا چند فرزند از طریق ترکیب ژن‌های تصادفی انتخاب شده از دو و یا بیشتر از دو والد می‌باشد. در عملگر بازترکیب، ایده اصلی آن است که فرزندان تولید شده ژن‌های والدین خود را به ارث برده و پاسخ‌هایی با برازش بهتر تولید کنند.

البته لزوما همیشه فرزندان تولید شده برازش بالاتری نسبت به والدین خود ندارند. در صورتی که انتخاب بر روی برترین پاسخ‌ها متمرکز باشد، ممکن است فشار انتخاب منجر به همگرایی زودرس الگوریتم شود. دلیل این رویداد آن است که استفاده از عملگر بازترکیب در طی نسل‌های طولانی منجر به کاهش تنوع در جمعیت خواهد شد. استفاده بیشتر از این عملگر، سبب تقویت قابلیت انتفاع در الگوریتم تکاملی می‌شود. عملگر بازترکیب با احتمال p_c بر روی اعضاء جمعیت انتخاب شده انجام می‌شود.

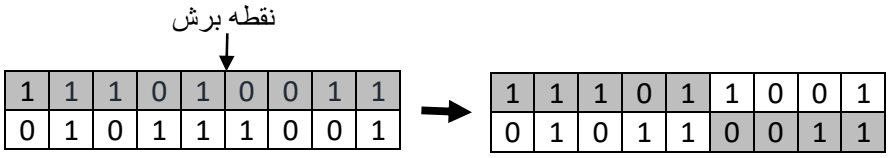
جهش، فرآیند تغییر تصادفی مقدار ژن‌ها در یک کروموزوم می‌باشد. هدف اصلی در عملگر جهش، یافتن مقادیر جدید برای ژن‌های فرزندان تولید شده است، تا تنوع ژنوتایپی در جمعیت افزایش یابد. جهش باید به گونه‌ای انجام شود که ژن‌های خوب در پاسخ‌های برتر خراب نشوند. با توجه به آنکه عملگر جهش سبب بروز یک تغییر تصادفی در کروموزوم شده و استفاده از این عملگر، احتمال یافته شدن مقادیر جدید را برای ژن‌ها افزایش می‌دهد، بهره‌برداری بیشتر از این عملگر سبب تقویت قابلیت پویش در الگوریتم تکاملی خواهد شد. عملگر جهش با احتمال p_m بر روی فرزندان تولید شده انجام می‌شود.

انواع مختلف عملگرهای بازترکیب و جهش، بستگی کامل با انواع گوناگون روش‌های نمایش کروموزوم‌ها در الگوریتم تکاملی دارد. کلیه عملگرهای بازترکیب که در این بخش معرفی می‌شوند بر روی دو والد انجام می‌شوند و دو فرزند تولید می‌کنند. عملگرهای جهش مورد بحث هم بر روی یک کروموزوم انجام می‌شوند و یک کروموزوم جهش یافته را ایجاد می‌کنند.

2-9-1 بازترکیب و جهش در نمایش دودویی

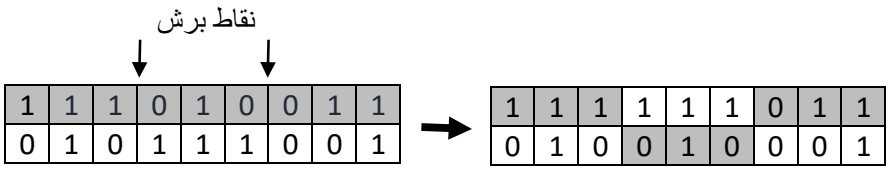
عملگرهای مشهور بازترکیب و جهش در نمایش دودویی عبارتند از عملگرهای بازترکیب همبرش تک نقطه‌ای (*Single-Point Crossover*)، چند نقطه‌ای (*Multi-Point Crossover*)، یکنواخت (*Uniform Crossover*) و عملگر جهش معکوس‌سازی بیت (*Bit-flipping Mutation*).

همانطور که در شکل (2-9) مشاهده می گردد، درعملگر بازترکیب تک نقطه‌ای، ابتدا یک نقطه تصادفی در دنباله کروموزوم‌های والدین انتخاب می‌شود و سپس، از محل انتخاب شده، کروموزوم هر دو والد برش می‌خورد. بخش اول والد اول و بخش دوم والد دوم برای تولید فرزند نخست استفاده می‌شود. فرزند دوم، شامل بخش اول والد دوم و بخش دوم والد اول خواهد بود.



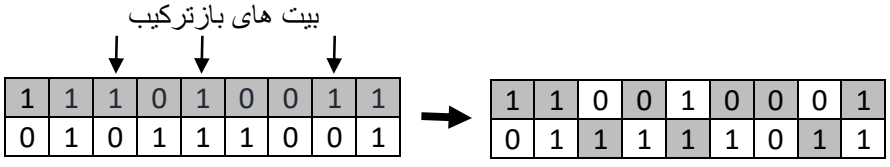
شکل (2-9) : عملگر بازترکیب هم برش تک نقطه‌ای

همانطور که در شکل (2-10) نشان داده شده است، در عملگر بازترکیب هم برش چند (دو) نقطه‌ای، چند (دو) نقطه تصادفی در دنباله کروموزوم‌های والدین انتخاب می‌شوند و سپس از محل این نقطه‌ها، کروموزوم والدین برش می‌خورد. بخش‌های اول و سوم والد اول و بخش دوم والد دوم برای تولید فرزند نخست استفاده می‌شوند. فرزند دوم، شامل بخش اول و سوم والد دوم و بخش دوم والد اول است. بدیهی است که با افزایش تعداد نقاط شکست در عملگر بازترکیب چند نقطه‌ای، شباهت فرزندان به هر کدام از والدین کمتر شده و توانایی پویشی عملگر بازترکیب تقویت خواهد شد.



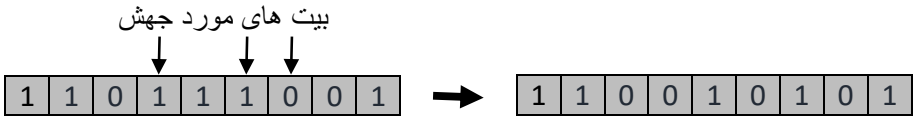
شکل (2-10) : عملگر بازترکیب هم برش دو نقطه‌ای

در عملگر بازترکیب یکنواخت، مقدار ژن فرزند با توجه به مقادیر ژن‌های متناظر هر دو والد انتخاب می‌شود. همانطور که در شکل (2-11) مشاهده می گردد، در این روش، مقادیر ژن‌های هر کدام از والدین، شانسی برابر برای حضور در ژن متناظر فرزند دارند. درعملگر بازترکیب یکنواخت، بر اساس یک توزیع تصادفی دودویی مشخص می‌شود که مقدار هر ژن فرزند از مقدار ژن متناظر کدام والد انتخاب گردد.



شکل (2-11) : عملگر بازترکیب یکنواخت

همانطور که در شکل (2-12) نشان داده شده است، در عملگر جهش معکوس سازی بیت، یک یا چند ژن به طور تصادفی انتخاب شده و مقدارش هر چه که باشد، تغییر می‌کند.

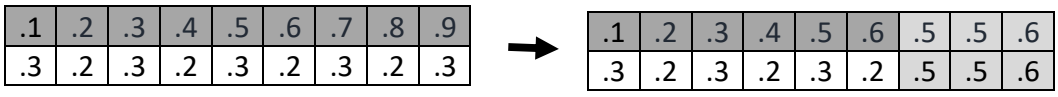


شکل (2-12) : عملگر جهش معکوس سازی بیت در نمایش دودویی

2-9-2 بازترکیب و جهش در نمایش اعداد حقیقی

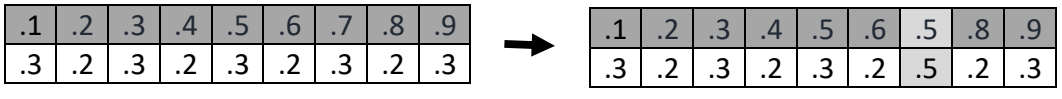
عملگرهای بازترکیب و جهش در نمایش اعداد حقیقی عبارتند از سه عملگر بازترکیب شامل عملگرهای بازترکیب ساده (*Simple Recombination*)، عملگر بازترکیب حسابی ساده (*Simple Arithmetic Recombination*)، عملگر بازترکیب حسابی کامل (*Whole Arithmetic Recombination*) و یک عملگر جهش شامل عملگر جهش مکمل (*Complement Mutation*).

همانطور که در شکل (2-13) مشاهده می گردد، در عملگر بازترکیب ساده، ابتدا یک بخش مشابه در کروموزوم‌های والدین انتخاب می‌شود. سپس بخش انتخاب شده در والد اول مستقیما به فرزند اول و بخش انتخاب شده در والد دوم نیز مستقیما به فرزند دوم منتقل می‌شود. در ادامه، مقادیر ژن‌های مربوط به بخش‌های انتخاب نشده دو والد با یکدیگر جمع شده و نتیجه در عدد a که در بازه (0,1) است ضرب می‌شود. نتیجه هر جفت ژن، به ژن معادل در فرزند اول منتقل می‌شود. همین کار برای فرزند دوم تکرار می‌شود، لیکن بجای ضرب در عدد a ، حاصل جمع ژن‌های دو والد در عدد $a - 1$ ضرب می‌شود (در شکل (2-13)، a=0.5 انتخاب شده است.). در این عملگر، به جای جمع نمودن مقادیر ژن‌ها در والدین، می‌توان از هر عملگر ریاضی دیگر و یا حتی تابع ریاضی پیچیده‌ای استفاده نمود.



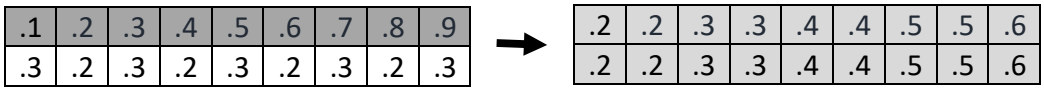
شکل (2-13) : عملگر بازترکیب ساده در نمایش اعداد حقیقی

همانطور که در شکل (2-14) مشاهده می گردد، عملگر بازترکیب حسابی ساده، مشابه با عملگر بازترکیب ساده است، با این تفاوت که تنها یک ژن در دو والد برای تولید فرزندان دست خوش تغییر می‌شود (در شکل (2-14)، a=0.5 انتخاب شده است.).



شکل (2-14) : عملگر بازترکیب حسابی ساده در نمایش اعداد حقیقی

همانطور که در شکل (2-15) نشان داده شده است، در عملگر بازترکیب حسابی کامل، همه ژن‌ها در والدین تغییر کرده و فرزندان در هیچ‌کدام از ژن‌هایشان شباهتی به هیچ یک از والدین خود ندارند (در شکل (2-15)، a=0.5 انتخاب شده است.). این عملگر بازترکیب با مقایسه با دیگر عملگرهای بازترکیبی در نمایش اعداد حقیقی، قابلیت پویش بالاتری دارد.



شکل (2-15) : عملگر بازترکیب حسابی کلی در نمایش اعداد حقیقی

همانطور که در شکل (2-16) مشاهده می گردد، عملگر جهش مکمل در نمایش اعداد حقیقی، به این صورت انجام می‌شود که مقدار مربوط به ژن‌های مورد جهش از مقدار بیشینه ممکن برای آنها کم شده و نتیجه به عنوان مقدار جدید ژن جایگزین مقدار قبلی می‌شود. به عبارت دیگر، حاصل جمع مقادیر قبلی و جدید هر ژن برابر با مقدار بیشینه ژن خواهد بود. در شکل (2-16) فرض بر آن است که مقدار بیشینه همه ژن‌ها برابر با 1 است.

شکل (2-16) : عملگر جهش مکمل در نمایش اعداد حقیقی

2-9-3 بازترکیب و جهش در نمایش جایگشت عناصر

عملگرهای بازترکیب و جهش در نمایش جایگشت عناصر عبارتند از عملگرهای بازترکیب ترتیبی (*Order Recombination*) و بازترکیب چرخشی (*Cycle Recombination*) وهمچنین، عملگرهای جهش جابجایی (*Swap Mutation*)، جهش درجی (*Insert Mutation*)، جهش درهم سازی (*Scramble Mutation*) و جهش وارونه‌سازی (*Inversion Mutation*).

درعملگر بازترکیب ترتیبی ابتدا دو نقطه برش، همانند عملگر بازترکیب دو نقطه‌ای در نمایش دودویی به صورت تصادفی تعیین می‌شود. سپس، دنباله میان این دو نقطه از والد اول به فرزند منتقل می‌گردد. در ادامه، از بعد از نقطه دوم تا آخرین ژن در فرزند تولید شده، از مقادیر ژن‌های معادل در والد دوم برای مقداردهی ژن‌های معادل در فرزند استفاده می‌شود. در صورتیکه مقدار ژنی قبلا در فرزند وجود

داشته باشد از آن مقدار صرف‌نظر خواهد شد. پس از رسیدن به آخرین ژن والد دوم به ابتدای این والد رفته و عملیات را تا رسیدن به نقطه برش اول در والد دوم ادامه می‌دهیم. شکل (2-17)، فرآیند مزبور را البته فقط برای یک فرزند نشان می‌دهد. فرزند دیگر را با تکرار همین فرآیند، ولی با عوض کردن جای دو والد تولید می‌شود.

گام اول) استفاده از والد اول

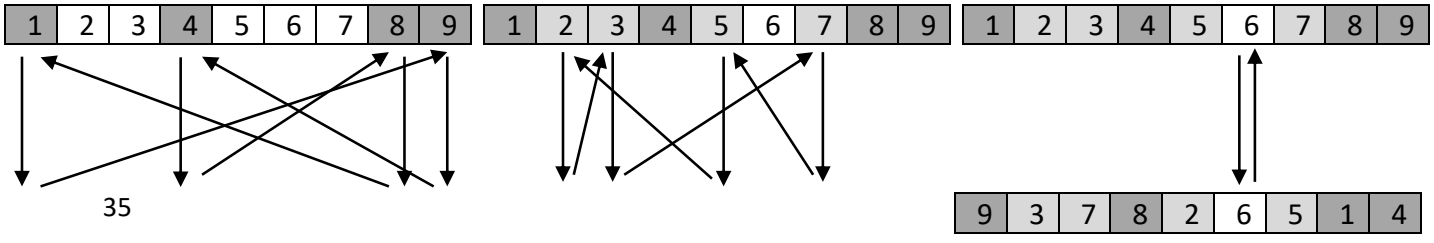
گام دوم) استفاده از والد دوم

شکل (2-17) : عملگر بازترکیب ترتیبی در نمایش جایگشت عناصر

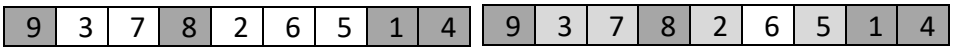
همانطور که در شکل (2-18) نشان داده شده است، عملگر بازترکیب چرخشی در نمایش جایگشت عناصر، به این صورت انجام می پذیرد که ابتدا در گام اول، کلیه دورهای موجود که توسط مقادیر ژن‌ها در دو والد نشان داده می‌شوند، شناسایی می‌شوند. این کار بدین صورت انجام می‌شود که از ژن اول والد اول به شماره ژنی در والد دوم می‌رویم که توسط ژن اول والد اول نشان داده می‌شود. سپس، همین کاربرای ژن مشخص شده در والد دوم انجام می‌شود و به ژنی در والد اول می‌رویم که توسط ژن والد دوم نشان داده می‌شود. در صورتی که دوباره به ژن اول والد نخست برسیم، یک دور ایجاد شده است. چنانچه هنوز ژنی در دو والد موجود باشد که پیمایش نشده باشد، این فرآیند مجددا از اولین ژن پیمایش نشده در والد نخست تکرار می‌گردد. به این ترتیب کلیه دورها شناسایی خواهند شد.

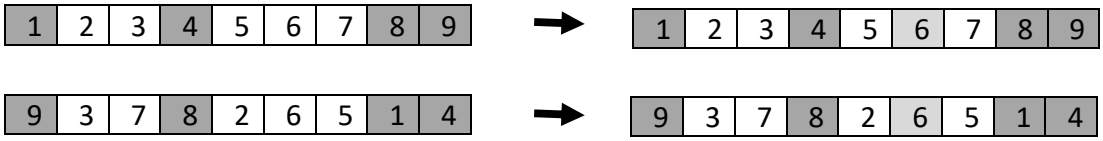
در گام دوم، ژن‌های مشخص شده در والد اول که حاصل دور اول هستند را به ژن‌های معادل فرزند اول منتقل کرده و برای ژن‌های باقیمانده این فرزند از ژن‌های مشخص شده والد دوم در دور دوم بهره می‌بریم. این کار تا تکمیل مقادیر ژن‌های فرزند نخست ادامه پیدا می کند. فرزند دوم نیز به همین شکل ساخته می‌شود، البته با این تفاوت که جای دو والد با یکدیگر تعویض می‌شوند. به عبارت دیگر، در بازترکیب چرخشی نقاط شکست برای تشکیل دو فرزند از دو والد با مشخص شدن دورهای موجود شناسایی می‌شوند.

گام اول) تعیین دورها (از چپ به راست)



گام دوم) کپی کردن دورها در فرزندان





شکل (2-18) : عملگر بازترکیب چرخشی در نمایش جایگشت عناصر

عملگر جهش جابجایی در نمایش جایگشت عناصر بسیار ساده بوده و کافی است مقادیر دو ژن تصادفی با یکدیگر جابجا شوند. یک نمونه برای درک بهتر عملکرد عملگر جهش جابجایی در شکل (2-19) نشان داده شده است.

شکل (2-19) : عملگر جهش جابجایی در نمایش جایگشت عناصر

همانطور که در شکل (2-20) مشاهده می گردد، عملگر جهش درجی، دو ژن را به طور تصادفی انتخاب کرده وسپس، مقدار ژن دوم (ژن سمت راست) را در ژن همسایه ژن نخست (همسایه سمت راست این ژن) کپی می‌کند. اکنون مقادیر ژن‌های باقیمانده به سمت راست انتقال (شیفت) داده می شوند.

شکل (2-20) : عملگر جهش درجی در نمایش جایگشت عناصر

در عملگر جهش درهم‌سازی، ابتدا دو نقطه به طور تصادفی، در کروموزوم مورد جهش، انتخاب می‌شوند. سپس مقادیر کلیه ژن‌های میان این دو نقطه، به طور تصادفی با یکدیگر جابجا می‌شوند. بدیهی است که هر چقدر فاصله دو نقطه انتخاب شده بیشتر باشد، قابلیت پویش این عملگر جهش تقویت خواهد شد. یک نمونه برای درک بهتر عملکرد عملگر جهش درهم‌سازی در شکل (2-21) نشان داده شده است.

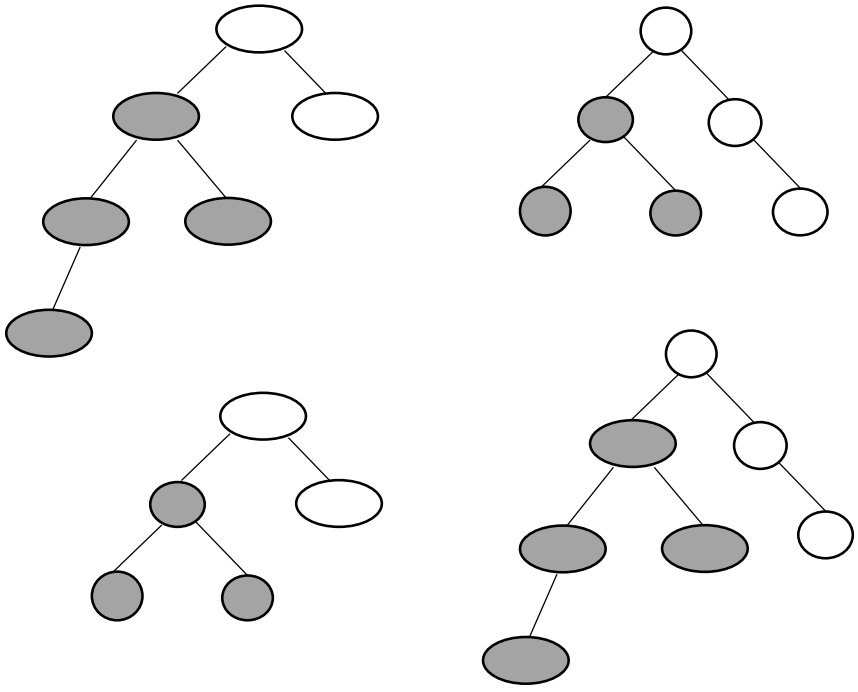
شکل (2-21) : عملگر جهش درهم سازی در نمایش جایگشت عناصر

همانطور که در شکل (2-22) مشاهده می گردد، عملگر جهش وارونه‌سازی، بسیار مشابه عملگر جهش درهم‌سازی است. تنها تفاوت این دو عملگر درآن است که پس از انتخاب دو نقطه درعملگر جهش وارونه‌سازی، مقادیر ژن‌های میان این دو نقطه، به شکلی که گویی آینه‌ای در وسط دو نقطه شکست قرار داده شده است، به جای یکدیگر کپی می‌شوند.

شکل (2-22) : عملگر جهش وارونه سازی در نمایش جایگشت عناصر

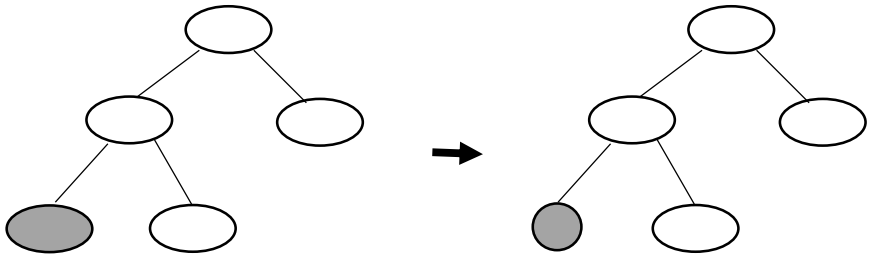
2-9-4 بازترکیب و جهش در نمایش درختی

عملگرهای بازترکیب و جهش در نمایش درختی، به گستردگی نمایش‌های دیگری که توضیح داده شدند، نیستند. در این بخش دو عملگر ساده برای بازترکیب و جهش در نمایش درختی ارائه شده است. شکل (2-23)، یک نمونه برای عملگر بازترکیب ساده را در نمایش درختی نشان می‌دهد. همانطور که در این نمونه نشان داده شده است، برای تولید دو فرزند در نمایش درختی کافی است که با در نظر گرفتن نقاط شکست در دو والد، جای زیر درخت های مربوط به آنها در یکدیگر تعویض گردند.



شکل (2-23) : عملگر بازترکیب ساده در نمایش درختی

همانطور که در شکل (2-24) نشان داده شده است، عملگر جهش در نمایش درختی، با تغییر مقدار مربوط به یک گره در درخت به مقداری تصادفی، قابل پیاده‌سازی است.



شکل (2-24) : عملگرجهش ساده در نمایش درختی

2-10 مرحله جایگزینی

در این مرحله، از مجموع جمعیت والدین و جمعیت فرزندان تولید شده، یک جمعیت جدید برای نسل بعد گزینش می‌شود. به عبارت دیگر، جمعیت والدین در نسل جاری، با جمعیت جدید جایگزین می‌گردد. دو روش مهم جایگزینی عبارتند از جایگزینی حالت پایدار (*Steady State Replacement*) و جایگزینی نسلی (*Generational Replacement*).

2-10-1 جایگزینی حالت پایدار

به منظور حفظ تنوع جمعیتی و جلوگیری از همگرا شدن الگوریتم تکاملی به پاسخ‌های بهینه محلی، بخش بزرگی از جمعیت والدین را حفظ کرده و تنها درصد کوچکی از این جمعیت با بهترین فرزندان تولید شده جایگزین می‌شوند. به همین دلیل، بافت کلی جمعیت جدید نسبت به جمعیت قبلی تغییر زیادی نکرده و گوناگونی جمعیتی حفظ می‌شود. درصد جمعیت والدین که در مرحله جایگزینی تغییر می‌کند با p_{rep} نشان داده می‌شود. هر چقدر p_{rep} عدد بزرگتری باشد (به 100 نزدیکتر باشد)، احتمال از دست دادن گوناگونی در جمعیت افزایش یافته و درنتیجه، موجب همگرایی سریعتر الگوریتم تکاملی خواهد شد.

2-10-2 جایگرینی نسلی

کل جمعیت والدین را با جمعیت فرزندان تولید شده جایگزین می‌کند. این روش موجب تسریع همگرایی الگوریتم تکاملی می‌شود. به منظور جلوگیری از نابودی برترین پاسخ‌ها در جمعیت، بهترین عضو از جمعیت والدین جایگزین ضعیف‌ترین فرزند می‌شود تا بدین ترتیب، احتمال از دست رفتن بهترین عضو

در کل جمعیت والدین و فرزندان از بین برود. به این جایگزینی (جایگزینی بهترین عضو جمعیت والدین با ضعیف‌ترین عضو جمعیت فرزندان)، نخبه سالاری (*Elitism*) اطلاق می‌شود.

دو روش جایگزینی انتخاب $(\mu + \lambda)$ و جایگزینی انتخاب (μ, λ) ، که مربوط به استراتژی‌های تکامل هستند، را می‌توان به روش جایگزینی نسلی نزدیک‌تر دانست. در روش جایگزینی انتخاب $(\mu + \lambda)$ ، تعداد μ عضو برتر از مجموع μ والد و λ فرزند، انتخاب شده و به نسل بعد انتقال داده می‌شوند. در روش جایگزینی انتخاب (μ, λ) ، تعداد μ عضو برتر از تعداد λ فرزند، گزینش شده و به نسل بعد منتقل می‌شوند. در این روش، فرض می‌شود که $\mu < \lambda$ می باشد. دلیل اینکه این دو روش جایگزینی را به روش جایگرینی نسلی نزدیک‌تر می‌دانیم آن است که در هر دوی این روش‌ها (مخصوصا روش جایگزینی انتخاب (μ, λ)) هیچ‌گونه تقیدی به حفظ جمعیت والدین، مستقل از برازش آنها، وجود ندارد. به همین دلیل، امکان از دست رفتن ناگهانی تنوع جمعیتی در هر دوی این روش‌ها وجود دارد.

2-11 شرایط توقف

مراحل انتخاب، بازترکیب، جهش و جایگزینی مکررا برای یک الگوریتم تکاملی بکار می‌روند تا زمانی که شرایط توقف مهیا شوند. برای یک الگوریتم تکاملی می‌توان شرایط زیر را به عنوان شرایط توقف بکاربرد :

- شاید بتوان بهترین زمان پایان یک الگوریتم تکاملی را هنگامی دانست که این الگوریتم پاسخ بهینه سراسری را یافته باشد. اگر بتوان برای مساله مورد بررسی، برازش بهترین پاسخ را از ابتدا تعیین نمود، بهترین شرط توقف رسیدن به آن پاسخ خواهد بود. با توجه به آنکه همیشه امکان تعیین برازش بهترین پاسخ وجود ندارد، ناگزیر به استفاده از سایر شروط توقف هستیم.
- ساده‌ترین شرط توقف، محدود کردن تعداد نسل‌هایی می‌باشد که الگوریتم تکاملی اجازه اجرای آنها را دارد (این شرط، محدودیت بر روی تعداد ارزیابی‌های تابع برازش نیز نامیده می‌شود). تعداد نسل‌های قابل انجام در الگوریتم تکاملی نباید خیلی کوچک باشد، چراکه در غیر این صورت، الگوریتم زمان کافی برای پیمایش فضای جستجو را نخواهد داشت.
- شرط دیگری که معمولا از آن برای توقف الگوریتم تکاملی استفاده می‌شود، همگرا شدن جمعیت در آخرین نسل است. همگرایی زمانی اتفاق می‌افتد که جمعیت راکد شود. زمانی که هیچ ژنوتایپ و فنوتایپی در جمعیت تغییر نمی‌کند، رکود جمعیت نامیده می‌شود. در این حالت می‌توان از یک پارامتر K، که نشان‌دهنده بیشترین تعداد نسلی که بهترین عضو در جمعیت تغییر نکرده است، برای تعیین زمان توقف الگوریتم تکاملی استفاده نمود.

اگر در زمان توقف الگوریتم تکاملی نتایج حاصل شده رضایت‌بخش نباشد، می‌توان با بکارگیری روش‌های افزایش تنوع در جمعیت (برای نمونه، افزایش احتمال جهش) قابلیت پویش را در الگوریتم تکاملی افزایش داد.

2-12 مباحث پیشرفته در الگوریتم های تکاملی

در این بخش به چند مبحث پیشرفته در رابطه با الگوریتم‌های تکاملی خواهیم پرداخت که عبارتند از کنترل قابلیت‌های پویش و انتفاع، نظریه اسکیما، الگوریتم‌های تکاملی موازی و بیهینه‌سازی چندهدفه تکاملی.

2-12-1 کنترل قابلیت‌های پویش و انتفاع

مهمترین مساله برای موفقیت یک الگوریتم تکاملی، برقراری یک مصالحه (*Trade-off*) و توازن میان قابلیت‌های پویش و انتفاع الگوریتم می‌باشد. با برقراری این مصالحه، سرعت کاهش فشار انتخاب در الگوریتم تکاملی کنترل شده و از همگرایی زودرس و نیل به پاسخ‌های بهینه محلی اجتناب خواهد شد. روش‌های متعددی برای برقراری این مصالحه وجود دارد. این روش‌ها را می‌توان در سه دسته کنترل پارامترها، استفاده از توابع مناسب انتخاب، تولیدمثل و جایگزینی، و در نهایت حفظ تنوع جمعیتی برشمرد. در ادامه به توصیف هریک بطور جداگانه پرداخته خواهد شد.

1- کنترل پارامترها

براساس ماهیت فضای جستجوی مساله بهینه‌سازی، سعی می‌شود که با کنترل پارامترهای الگوریتم تکاملی میان توان‌های پویش و انتفاع الگوریتم یک مصالحه مناسب برقرار شود. برخی از این پارامترها عبارتند از احتمال بازترکیب (p_c)، احتمال جهش (p_m)، درصد جایگزینی (p_{rep})، تعداد اعضای مورد گزینش در انتخاب مسابقه‌ای (p_{tourn}) و درصد اعضای مورد بررسی در انتخاب برشی (p_{trunc}).

- افزایش (یا کاهش) p_c باعث تقویت (یا تضعیف) قابلیت انتفاع الگوریتم تکاملی خواهد شد.
- افزایش (یا کاهش) p_m سبب تقویت (یا تضعیف) توان پویش الگوریتم تکاملی می‌شود.
- افزایش (یا کاهش) p_{rep} منجر به تقویت (یا تضعیف) توانایی انتفاع الگوریتم تکاملی خواهد شد.
- افزایش (یا کاهش) p_{tourn} باعث تقویت (یا تضعیف) قابلیت انتفاع الگوریتم تکاملی می‌شود.
- افزایش (یا کاهش) p_{trunc} سبب تقویت (یا تضعیف) قابلیت پویش الگوریتم تکاملی می‌شود.

2- استفاده ازتوابع مناسب

در این روش سعی می‌شود که با توجه به ماهیت فضای جستجوی مساله، یک تابع مناسب برای مراحل انتخاب، بازترکیب، جهش و جایگزینی گزینش شود. چگونگی تاثیر انتخاب برخی از توابع مزبور در تقویت یا تضعیف قابلیت‌های پویش و انتفاع الگوریتم‌های تکاملی در ذیل اشاره شده است :

- استفاده از تابع انتخاب جستجوی چرخ رولت ساده، منجر به تقویت قابلیت انتفاع الگوریتم تکاملی می‌شود.
- بکارگیری تابع انتخاب جستجوی چرخ رولت با چند اشاره‌گر، سبب تقویت قابلیت پویش الگوریتم تکاملی خواهد شد.
- گزینش روش بازترکیب تک نقطه‌ای به جای روش‌های بازترکیب چند نقطه‌ای، قابلیت انتفاع الگوریتم تکاملی را به میزان بیشتری افزایش خواهد داد.
- در میان انواع روش‌های بازترکیب، روش بازترکیب یکنواخت بهترین انتخاب برای تقویت توان پویش الگوریتم تکاملی است.
- استفاده از روش‌های جهشی که تخریب بیشتری را در ساختار کروموزوم‌ها ایجاد می‌کنند، برای تقویت قابلیت پویش الگوریتم‌های تکاملی، اجتناب‌ناپذیر است.
- چنانچه تقویت قابلیت پویش در الگوریتم تکاملی در اولویت باشد، بهره‌مندی از روش جایگزینی حالت پایدار به جای روش جایگزینی نسلی، توصیه می‌شود.

3- حفظ تنوع جمعیتی

در صورتی که سعی می‌شود گوناگونی اعضاء در جمعیت مورد تکامل کنترل شده و از کاهش سریع تنوع جمعیتی جلوگیری به عمل آید، هدف برقراری مصالحه میان قابلیت های پویش و انتفاع، حاصل خواهد شد. در واقع، کاهش سرعت همگرایی الگوریتم با حفظ گوناگونی در جمعیت، کمک به قابلیت پویش در الگوریتم‌های تکاملی محسوب می‌شود. روش‌های مهم حفظ تنوع جمعیتی عبارتند از کرانه سازی (*Niching*) و گونه سازی (*Speciation*).

- **کرانه‌سازی**

ایده اصلی در کرانه‌سازی از تنوع موجودات در طبیعت گرفته شده است. در حقیقت انواع مختلف موجودات به طور موازی در حال تکامل برای هر کدام از محیطها و شرایط زندگی موجودات یافته است. اگر قله‌های موجود در دورنمای برازش مربوط به یک مساله بیشینه‌سازی را به یک پاسخ مناسب برای شرایطی مشخص منتسب بدانیم، کاملا منطقی خواهد بود که با جلوگیری از رقابت ناعادلانه پاسخ‌های مربوط به هر قله یا کرانه (*Niche*)، تنوع جمعیتی را در فرآیند تکاملی الگوریتم حفظ نماییم. استفاده از این ایده سبب می‌شود که گوناگونی موجود در جمعیت اولیه محافظت شده و با رویکردی تدریجی، متوسط برازش پاسخ‌ها در جمعیت بهبود یابد. دو روش مشهور کرانه سازی شامل مشترک سازی برازش (Fitness Sharing) و انبوه سازی (Crowding) در ادامه تشریح شده‌اند.

1. مشترک سازی برازش : این روش که معروف‌ترین رویکرد برای کرانه سازی محسوب می‌شود، بر اساس راهکار به اشتراک گذاری منابع محدود در طبیعت، توسط موجوداتی که در یک منطقه زندگی می‌کنند، بنیان گذاشته شده است. در این روش یک برازش کاذب، با هدف ترغیب الگوریتم تکاملی به پویش بیشتر فضای جستجو و جلوگیری از همگرایی زودرس الگوریتم، برای هر عضو

جمعیت در نظر گرفته می‌شود. اگر برازش واقعی عضو i ام جمعیت را با $f(i)$ نشان دهیم، برازش کاذب این عضو با توجه به روابط ذیل قابل محاسبه است :

$$f(i) = \frac{f(i)}{\sum_{j=1}^{n} sh(d(i,j))}$$

$$sh(d(i,j)) = \begin{cases} 1 - (\frac{d(i,j)}{\sigma_{share}})^{\alpha} & d(i,j) < \sigma_{share} \\ 0 & \text{otherwise} \end{cases}$$

در روابط فوق، $d(i,j)$ فاصله ژنوتیپی (ویا فنوتیپی) میان اعضای i ام و j ام در جمعیت مورد تکامل می‌باشد. همچنین، مقدار α ثابت و معمولا برابر 1 است. برای اعضای یکسان در جمعیت، مقدار $sh(d(i,j))$ برابر با 1 بوده و در صورت آنکه اختلاف فاصله اعضاء در جمعیت از آستانه σ_{share} بیشتر باشد، مقدار $sh(d(i,j))$ برابر 0 خواهد بود. درنظر گرفتن مقادیر بزرگتر برای آستانه σ_{share} سبب تقویت قابلیت پویش الگوریتم تکاملی می‌شود.

2. انبوه سازی : ایده اصلی در انبوه‌سازی این است که اعضاء جدید جایگزین اعضاء مشابه خود در جمعیت شوند. با این کار، گوناگونی در جمعیت ابتدایی به صورت یک سرمایه حفظ شده و الگوریتم به صورت تدریجی به پاسخ نهایی همگرا خواهد شد. انبوه‌سازی به روش‌های مختلفی قابل پیاده‌سازی است. درنسخه اصلی انبوه‌سازی که توسط دی‌جانگ در سال 1975 ارائه شد، با استفاده از رویکرد جایگزینی حالت پایدار، تنها به بخشی از اعضاء در جمعیت اجازه تولیدمثل و مرگ داده شد. در روش دی‌جانگ، انبوه‌سازی به این شکل انجام می‌شود که ابتدا، یک بخش از اعضاء در جمعیت نسل جاری (در حدود 10% از جمعیت) توسط روش انتخاب چرخ رولت برای انجام عملیات بازترکیب و جهش گزینش می‌شوند. سپس برای هرفرزند جدید تولید شده، تعداد CF (*Crowding Factor*) که معمولا دربازه [2,5] انتخاب می‌شود، از اعضاء در جمعیت به صورت تصادفی انتخاب شده و با فرزند جدید مذکور مقایسه می‌شوند. فرزند جدید تولید شده، جایگزین شبیه‌ترین عضو در میان CF پاسخ انتخابی خواهد شد.

یک روش دیگر برای انبوه‌سازی، که توسط هاریک در سال 1995 ارائه شد، از یک روش انتخاب مسابقه‌ای محدود شده (*Restricted Tournament Selection*) بهره می‌گیرد. در این روش، از رقابت یک پاسخ در جمعیت با پاسخ‌های بسیار متفاوت با آن جلوگیری به عمل می‌آید. در روش هاریک، ابتدا دو پاسخ (به عنوان نمونه A و B) از جمعیت به صورت تصادفی انتخاب می‌شوند. سپس، با استفاده از عملگرهای بازترکیب و جهش، دو پاسخ جدید (به عنوان نمونه A′ و B′) ایجاد می‌شوند. در ادامه، به ازای هر کدام از پاسخ‌های جدید، تعداد W از اعضاء جمعیت به صورت تصادفی بررسی شده و از میان آنها شبیه‌ترین پاسخ به هرکدام از دو پاسخ جدید گزینش می‌شوند (به عنوان نمونه A″ و B″). پاسخ A′ با پاسخ A″ رقابت می‌کند و اگر بهتر بود، جایگزین آن در جمعیت می‌شود. به طور مشابه، همین عملیات برای B′ و B″ نیز انجام می‌شود. لازم به ذکر است که در

روش انبوه‌سازی هاریک، همانند روش دی‌جانگ، از رویکرد جایگزینی حالت پایدار بهره‌برداری می‌شود.

- **گونه سازی**

امکان یافتن پاسخ‌های متفاوت و متنوع با بکارگیری روش گونه سازی میسر است. به دسته‌ای از اعضاء که در مقایسه با سایر اعضاء در جمعیت شباهت بیشتری با یکدیگر دارند، یک گونه گفته می‌شود. ایده اصلی در روش گونه‌سازی آن است که تنها به اعضاء مشابه در جمعیت تکاملی اجازه تولیدمثل داده شود. با این کار عملا به اعضاء هر گونه، اجازه تولیدمثل با اعضاء همان گونه داده شده است. این موضوع در واقع کاملا عادی و بدیهی است، چرا که در طبیعت شیرها با فیل جفت‌گیری نمی‌کنند.

در روش کرانه‌سازی، اعضاء در جمعیت تکاملی به سمت کرانه‌ها (قله‌ها در مسائل بیشینه‌سازی) پراکنده می‌شوند. در این روش، از بازترکیب اعضاء کرانه‌های متفاوت جلوگیری به عمل نمی‌آید. بازترکیب اعضاء عضو کرانه‌های متفاوت، معمولا منجر به تولید پاسخ‌های مهلک که برازش ضعیفی دارند، می‌شود. این مشکل با استفاده از روش‌های گونه‌سازی قابل حل است. چند نمونه از انواع روش‌های گونه‌سازی در زیر ارائه شده‌اند :

- ✓ دب و گلدبرگ دو روش جفت گیری محدود شده، براساس برازش‌های ژنوتایپی و فنوتایپی، ارائه نمودند. ایده آنها بر اساس پارامتری با نام δ_{mating} است. برای انجام عملگر بازترکیب یک عضو از جمعیت با نام m انتخاب می‌گردد. حال از میان اعضاء موجود در جمعیت یک عضو را به صورت تصادفی انتخاب شده و فاصله آن با عضو m محاسبه می‌شود. اگر این فاصله از δ_{mating} کمتر بود، عملگر بازترکیب میان دو عضو انتخابی رخ خواهد داد. درغیر این صورت، عضو دیگری از جمعیت انتخاب می‌گردد.
- ✓ روش مشهور دیگر برای گونه‌سازی، بیت‌های نشانه (*Tag bit*) نام دارد و توسط اسپیرز در سال 1994 ارائه شده است. در روش بیت‌های نشانه، هر کروموزوم با یک یا چند بیت نشانه برچسب زده می‌شود. این بیت‌ها نشان‌گر عضویت کروموزوم به یک گونه مشخص هستند. به عنوان نمونه، اگر بخواهیم که دو گونه کروموزومی ایجاد کنیم، به تنها یک بیت نشانه نیاز داریم. در این حالت، مقدار صفر برای این بیت نشان دهنده عضویت کروموزوم به گونه اول و مقدار یک برای این بیت بیان‌گر عضویت کروموزوم به گونه دوم است. بدیهی است که برای ایجاد گونه‌های بیشتری به تعداد بالاتری بیت نشانه نیاز داریم. اگر فرض کنیم که برای یک جمعیت، k گونه مختلف داشته باشیم، بنابراین به تعداد k مجموعه به شکل $\{ S_0, \dots, S_{k-1} \}$ خواهیم داشت که هر یک شامل تعدادی از اعضاء با گونه یکسان (بیت‌های نشانه با مقدار مشابه) می‌باشند.

2-12-2 نظریه اسکیما

هالند برای اینکه نشان دهد یک الگوریتم تکاملی درست کار می‌کند، نظریه اسکیما را مطرح نمود. این نظریه از یک الگوریتم تکاملی با روش کدگذاری دودویی، روش انتخاب چرخ رولت، عملگر بازترکیب تک نقطه‌ای و عملگر جهش معکوس‌سازی بیتی استفاده می‌شود.

در این بخش، کروموزوم‌ها را با (*String*) *S* و اسکیماها را با (*Hyper Plane*) *H* نشان داده می شوند. یک اسکیما (*Schema*) بیانگر مجموعه‌ای از رشته‌هاست که دارای شباهت‌هایی در برخی مکان‌هایشان هستند. در واقع، اسکیما یک قالب برای نمایش رشته‌های مشابه می‌باشد. برای نمایش اسکیما در مورد رشته‌های دودویی، از یک سه‌تایی به صورت زیر استفاده می‌شود :

$S \in \{0,1\}^l$

$H \in \{0,1,*\}^l$

علامت $*$، بیانگر 0 یا 1 است (don`t care). بنابراین تعداد رشته‌های دودویی به طول *L* برابر با 2^L و تعداد اسکیماهای به طول *L* برابر با 3^L خواهد بود.

تعداد بیت‌های تعریف شده (غیر $*$) در اسکیما، مرتبه اسکیما (*Schema Order*) گفته می‌شود. به علاوه، فاصله بین اولین و آخرین بیت تعریف شده در اسکیما، طول اسکیما (*Schema Defining Length*) می‌باشد.

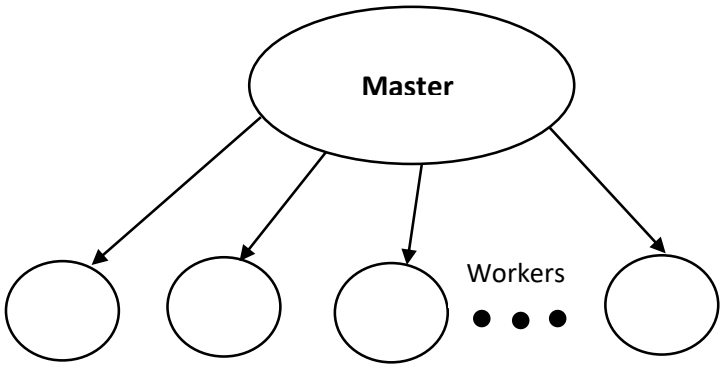
تعداد کل اسکیماهای *H* به طول *L* و مرتبه O(*H*) برابر است با $2^{O(H)}\binom{L}{O(H)}$. به عنوان نمونه، تعداد 12 اسکیما به طول 3 و مرتبه 2 وجود دارد. به همین ترتیب تعداد اسکیماهای به طول *L* برابر خواهد بود با :

$$\sum_{O(H)=0}^{L} 2^{O(H)}\binom{L}{O(H)} = 3^L$$

2-12-3 الگوریتم‌های تکاملی موازی

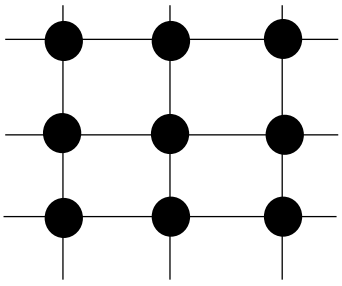
الگوریتم‌های تکاملی به دلیل ماهیت جمعیتی که دارند، دارای عملکرد بسیار کندی هستند. الگوریتم‌های تکاملی موازی، گونه‌ای از الگوریتم‌های تکاملی هستند که علاوه بر داشتن نقاط قوت روش‌های تکاملی، سرعت اجرای بسیار بالایی را در مقایسه با الگوریتم‌های تکاملی غیرموازی دارا می‌باشند. انواع مشهور الگوریتم‌های تکاملی موازی عبارتند از الگوریتم تکاملی ارباب-برده (*Master-Slave Parallel EA*)، الگوریتم تکاملی ریزدانه‌ای (*Fine-Grained Parallel EA*) و الگوریتم تکاملی چندجمعیتی (*Multiple-Population Parallel EA*). لازم به ذکر است که الگوریتم‌های ژنتیک جزیره‌ای (*Island Genetic Algorithms*) عنوانی است که گاها برای نسخه ژنتیکی روش‌های تکاملی موازی بکار برده می‌شود.

طرح کلی الگوریتم موازی ارباب-برده در شکل (2-25) نشان داده شده است. در این الگوریتم، کل جمعیت مورد تکامل در پردازنده ارباب نگهداری می‌شود. همچنین عملیات انتخاب، تولیدمثل و جایگزینی نیز در پردازنده ارباب اجرا می‌شوند. پردازنده ارباب برای ارزیابی برازش اعضاء در جمعیت، آنها را میان پردازنده‌های برده تقسیم می‌نماید. پردازنده‌های برده، پس از محاسبه برازش اعضاء ارسال شده به آنها، نتیجه را به پردازنده ارباب عودت می‌دهند. در آن دسته ازمسائل که محاسبات مربوط به تابع برازش سنگین می‌باشد (مانند داده کاوی) استفاده از الگوریتم تکاملی موازی ارباب-برده اکیدا توصیه می‌شود. از آنجا که در این الگوریتم، عملگرهای انتخاب و تولیدمثل بر روی کل جمعیت انجام می‌شوند، در برخی مقالات و کتب از این الگوریتم با نام الگوریتم تکاملی موازی سراسری (*Global Parallel EA*) نیز یاد می‌شود.



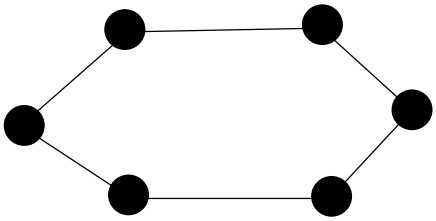
شکل (2-25) : طرح کلی الگوریتم تکاملی موازی ارباب – برده

شکل (2-26)، طرح کلی الگوریتم تکاملی موازی ریزدانه‌ای را نشان می‌دهد. در این روش، کل جمعیت به تعداد زیادی زیر جمعیت شکسته شده و سپس، زیرجمعیت‌های تولید شده به تعداد زیادی از پردازنده‌های توزیع شده ارسال می‌شوند. در هر پردازنده c_i، عملیات انتخاب و تولیدمثل به صورت محدود شده و بر روی زیرجمعیت پردازنده c_i و نیز بخشی از زیر جمعیت پردازنده‌های مجاور پردازنده c_i رخ می‌دهد. در واقع در این روش، بخشی از زیرجمعیت مورد تکامل، در پردازنده‌های مجاور مشترک است. به همین دلیل تغییرات رخ داده شده در زیرجمعیت یک پردازنده در فرآیند تکاملی الگوریتم به بخش‌های دیگر جمعیت منتقل می‌شود.



شکل (2-26) : طرح کلی الگوریتم تکاملی موازی ریزدانه‌ای

طرح کلی الگوریتم تکاملی موازی چندجمعیتی در شکل (2-27) نشان داده شده است. در این روش، که نسبت به دو روش دیگر پیچیده‌تر است، جمعیت اصلی به تعدادی زیرجمعیت شکسته می‌شود که البته تعداد این زیرجمعیت‌ها در قیاس با روش ریزدانه‌ای بسیار کمتر است و به همین دلیل هم روش چندجمعیتی، در قیاس با روش ریزدانه‌ای از لحاظ سخت‌افزاری ارزان‌تر است. همانند روش ریزدانه‌ای زیرجمعیت‌های تولید شده به چندین پردازنده ارسال می‌شوند.



شکل (2-27) : طرح کلی الگوریتم تکاملی موازی چند جمعیتی

برخلاف روش ریزدانه‌ای، در روش چندجمعیتی زیرجمعیت مربوط به هر پردازنده فقط توسط همان پردازنده قابل دیدن است و به عبارت دیگر، پردازنده‌ها هیچ‌گونه اشتراکی در زیر جمعیت‌های خود ندارند. در روش چندجمعیتی نیز عملیات انتخاب و تولیدمثل بر روی زیرجمعیت هر پردازنده به صورت جداگانه و مستقل انجام می‌شوند. به منظور برقراری ارتباط میان فرآیند تکاملی در پردازنده‌های مختلف، هرچند نسل یک‌بار برخی از اعضاء برازنده در میان پردازنده‌های مجاور مبادله می‌شوند. به این عملیات مبادله‌ای در الگوریتم‌های تکاملی موازی چندجمعیتی، مهاجرت (*Migration*) گفته می‌شود. نبایستی تعداد نسل‌های میان مهاجرت‌های فرزندان عددی کوچک باشد، در غیر این صورت الگوریتم تکاملی موازی بسیار کند خواهد شد و در این حالت اساس فلسفه موازی‌سازی که تسریع فرآیند تکامل است خدشه‌دار خواهد شد. شاید بتوان موفق‌ترین نوع الگوریتم‌های تکاملی موازی را روش‌های چندجمعیتی دانست. به این دلیل که درکنار سرعت اجرای بالاتر و نیز جستجوی موثرتر فضای جستجو، ارزان تر نیز هستند.

2-12-4 بهینه‌سازی چند-هدفه تکاملی

فرض کنید که قصد خرید یک خودرو را داشته باشید. یک خودروی خوب از نظر شما احتمالا بایستی پرقدرت و در حین حال ارزان باشد. در این حالت به هر خودرو (هرپاسخ در الگوریتم تکاملی)، دو عدد نسبت داده می‌شود وهدف، بیشینه‌سازی عدد مربوط به قدرت و کمینه‌سازی عدد مربوط به قیمت می‌باشد. در صورتی که بخواهیم یک مساله با چند هدف را حل کنیم به جای یک تابع برازش، چند تابع برازش خواهیم داشت. چنانچه در مساله بهینه‌سازی مورد بررسی، بیش از یک تابع برازش وجود داشته باشد، نمی‌توانیم از الگوریتم‌های تکاملی سنتی تک‌هدفه استفاده کنیم.

روش‌های بهینه‌سازی چند-هدفه تکاملی یا EMOO (*Evolutionary Multi-Optimization*)، مشتمل بر سه دسته می‌باشند که عبارتند از توابع انبوهشی (*Aggregating Functions*)، رویکردهای جمعیتی (*Population-based Approaches*) ورویکردهای مبتنی برپرتو (*Pareto-based Approaches*). توابع انبوهشی، ساده‌ترین روش در بهینه‌سازی چند-هدفه تکاملی هستند. کلیه اهداف موجود با استفاده از عملیات جمع، ضرب و یا هر ترکیبی از عملگرهای ریاضی به یک هدف نگاشت می‌شود. رابطه زیر، یک تابع انبوهشی نمونه را برای K هدف نشان می‌دهد :

$$F(x) = \sum_{i=1}^{K} w_i f_i(x)$$

در رابطه بالا، w_i نشان دهنده وزنی است که برای هدف $f_i(x)$ در نظر گرفته شده است. برای اهداف بیشینه‌سازی، مقدار w_i مثبت و برای اهداف کمینه‌سازی، مقدار w_i منفی است. همچنین رابطه زیر، میان وزن‌های اهداف در تابع انبوهشی برقرار است :

$$\sum_{i=1}^{K} |w_i| = 1$$

رابطه بالا، نشان می‌دهد که مثلا برای یک مساله بهینه‌سازی با دو هدف با ارزش برابر داریم $w_1=w_2=0.5$.

روش دیگر برای بهینه‌سازی چند-هدفه تکاملی، استفاده از رویکردهای جمعیتی می‌باشد. در این روش، جمعیت مورد تکامل با توجه به تعداد اهداف مساله به زیرجمعیت‌هایی تقسیم می‌شود. تعداد اعضاء هر کدام از زیرجمعیت‌ها، بستگی به اهمیت اهداف مساله دارد. به عنوان نمونه، اگر دو هدف با ارزش یکسان داشته باشیم، جمعیت به دو بخش مساوی تقسیم می‌شود. در مرحله انتخاب برای تولیدمثل، اعضاء مربوط به هر کدام از زیرجمعیت‌ها را با توجه به تنها یکی از اهداف مساله برگزیده و جمعیت جدید را می‌سازیم. به عنوان نمونه، برای مساله بهینه‌سازی با دو هدف هم ارزش، 50% جمعیت (زیرجمعیت اول) را با توجه به هدف اول و 50% باقیمانده جمعیت (زیرجمعیت دوم) را با توجه به هدف دوم انتخاب می‌کنیم. با توجه به آنکه در هر نسل زیرجمعیت‌ها مجددا ساخته می‌شوند، بنابراین پس از طی چندین نسل، هر پاسخ در جمعیت با توجه به همه اهداف مساله تکامل خواهد یافت. مهمترین کاربرد روش‌های مبتنی بر جمعیت برای مسائلی است که تعداد اهداف در آنها زیاد می‌باشد. بزرگترین نقطه ضعف روش‌های مزبور آن است که پاسخ‌هایی که برازش آنها در همه توابع برازش اهداف مساله متوسط است، با وجود آنکه ارزش بالایی دارند، ممکن است در فرآیند تکاملی خذف شوند.

سومین و مهمترین رویکرد در بهینه‌سازی چند-هدفه تکاملی، روش‌های مبتنی بر پرتو هستند. این روش‌ها پاسخ‌های غیرمغلوب را در جمعیت شناسایی نموده و سعی در حفظ آنها می‌کنند. روش‌های چند-هدفه تکاملی مبتنی بر رویکرد پرتو بسیار متنوع هستند. در اکثر این روش‌ها، برای محاسبه برازش هر پاسخ i ، تعداد پاسخ‌هایی که پاسخ i را مغلوب می‌کنند، به عنوان یک عدد مهم در نظر گرفته می‌شود. در برخی از روش‌های مبتنی بر رویکرد پرتو، تعداد پاسخ‌هایی که توسط پاسخ i مغلوب می‌شوند نیز به عنوان یک عدد مهم دیگر مورد توجه قرار می‌گیرد. این روش‌ها معمولا جمعیت کوچکی از پاسخ‌های غیرمغلوب را در فرآیند جستجوی الگوریتم تکاملی، همراه با جمعیت اصلی، تکامل می‌دهند. به همین

دلیل، برخلاف دیگر روش‌های بهینه‌سازی چند-هدفه تکاملی، در پایان عملیات جستجوی الگوریتم، یک مجموعه پاسخ برتر (غیرمغلوب) را به جای تنها یک پاسخ به اندازه کافی خوب خواهند یافت. از روش‌های بهینه‌سازی چند-هدفه تکاملی مبتنی بر رویکرد پرتو، هنگامی استفاده می‌شود که ارزش همه اهداف مساله یکسان بوده و به جای یک پاسخ، یافتن مجموعه‌ای از پاسخ‌های مطلوب مورد توجه باشد. در هر حال چنانچه بتوان اهمیت هر کدام از اهداف را در آغاز فرآیند تکاملی الگوریتم مشخص نمود، بهتر است از همان روش ساده توابع انبوهشی برای یافتن یک پاسخ مناسب بهره برد. زیرا مهمترین مشکل روش‌های مبتنی بر پرتو، پیچیدگی زمانی آنهاست که آن هم به دلیل رویکرد این روش‌ها در یافتن چندین پاسخ غیرمغلوب (به جای تمرکز بر یافتن تنها یک پاسخ به اندازه کافی خوب) است.

فصل سوم
فرامکاشفه‌های تکاملی

3-1 مقدمه

در این فصل چندین الگوریتم فرامکاشفه ای تکاملی توضیح داده خواهد شد. این الگوریتم های فرا مکاشفه‌ای عبارتند از الگوریتم ژنتیک، برنامه‌نویسی ژنتیک، استراتژی تکامل، برنامه‌نویسی تکاملی، تکامل تفاضلی، الگوریتم ممتیک، الگوریتم فرهنگی، الگوریتم ژنتیک تاگوچی، الگوریتم هم‌تکاملی، الگوریتم تکاملی دیپلوئیدی، بهینه‌سازی تولیدمثل غیرجنسی و سیستم ایمنی مصنوعی که در ادامه، به توصیف هریک به طور جداگانه پرداخته خواهد شد.

3-2 الگوریتم ژنتیک

الگوریتم‌های ژنتیک یا GA (*Genetic Algorithm*)، احتمالا اولین مدل الگوریتمی برای شبیه سازی سیستم‌های مبتنی بر ژن بوده‌اند. الگوریتم‌های ژنتیک، تکامل ژنی را مدل می‌کنند، به صورتی که در آنها برای نشان دادن ویژگی‌های موجودات از ژنوتایپ‌ها استفاده می‌شود. عملگرهای اصلی در الگوریتم‌های ژنتیک عبارتند از انتخاب (برای مدل کردن قانون بقاء اصلح) و تولیدمثل از طریق عملگرهای بازترکیب و جهش (برای مدل کردن تولیدمثل).

الگوریتم ژنتیک، برای حل گسترده وسیعی از مسائل دنیای واقعی بکار رفته است. مسائلی از قبیل جستجو، بهینه‌سازی پیوسته و ترکیباتی، یادگیری ماشین، مهندسی کنترل، طراحی، زمانبندی کارها، برنامه‌ریزی حرکت روبات، پردازش سیگنال و مسائل بازی‌های گوناگون.

3-2-1 شبه کد الگوریتم ژنتیک استاندارد

خصوصیات مهم الگوریتم ژنتیک استاندارد یا CGA (*Canonical Genetic Algorithm*) به طور خلاصه به شرح ذیل می باشند :

- استفاده از نمایش رشته بیتی.
- طول ثابت و یکسان برای هر کروموزوم.
- بکارگیری یک جمعیت با تعداد اعضاء ثابت.
- بهره‌مندی از عملگر انتخاب نسبی برای انتخاب والدین.
- بکارگیری عملگر بازترکیب تک نقطه‌ای (عملگر بازترکیب در CGA عملگر اصلی است).
- استفاده از جهش معکوس‌سازی بیت (عملگر جهش در CGA عملگر فرعی است).
- تنظیم احتمال بازترکیب p_c به مقادیربزرگ (بزرگتر از 0.95).

- تنظیم احتمال جهش p_m به مقادیر کوچک (مثلا برابر با $\frac{1}{L}$ ، L طول کروموزوم).

شبه کد الگوریتم ژنتیک استاندارد در شکل (3-1) نشان داده شده است.

Function GA (Problem) return a state that is a local optimum

Input : $Population_{size}$, $p_{crossover}$, $p_{mutation}$

Output : S_{best}

Population ← InitilizePopulation ($Population_{size}$, $Problem_{size}$) ;

EvaluatePopulation (Population) ;

S_{best} ← GetBestSolution (Population) ;

While−StopCondition() do

  Parents ← SelecParents (Population , $Population_{size}$) ;

  Children ← ∅ ;

  for each $Parent_1$, $Parent_2$ ∈ Parents do

    $Child_1$, $Child_2$ ← Crossover ($Parent_1$, $Parent_2$,$P_{crossover}$) ;

    Children ← Mutate ($Child_1$, $P_{mutation}$) ;

    Children ← Mutate ($Child_2$, $P_{mutation}$) ;

  end

  EvaluatePopulation (Children) ;

  Population ← Replace (Population , Children) ;

  S_{best} ← GetBestSolution (Population) ;

end

return S_{best} ;

شکل (3-1) : شبه کد الگوریتم ژنتیک استاندارد

3-2-2 مباحث پیشرفته الگوریتم های ژنتیک

پس از CGA، مدل‌های مختلفی برای الگوریتم‌های ژنتیک ارائه شده است که در طریقه نمایش، عملگرهای بازترکیب، جهش، پارامترهای کنترلی و غیره متفاوت هستند. در این بخش مروری بر برخی از مباحث پیشرفته در حوزه الگوریتم‌های ژنتیک خواهیم داشت.

3-2-2-1 عملگر بازترکیب

عملگر بازترکیب را می‌توان بر اساس تعداد والدینی که در این عملگر استفاده می‌شوند، به سه دسته تقسیم کرد. این سه دسته عبارتند از :

- **غیرجنسی**، به صورتی که فرزند از یک والد تولید شده است.
- **جنسی**، به صورتی که از دو والد برای تولید یک یا دو فرزند استفاده می‌شود.
- **چندترکیبی** (*Multi-Recombinarion*)، به صورتی که بیش از دو والد برای تولید یک یا بیش از یک فرزند بکار گرفته می‌شوند.

علاوه بر این، عملگر بازترکیب بر اساس مدل نمایش مورد استفاده نیز دسته‌بندی می‌شود. برای نمونه، عملگرهای مخصوص نمایش دودویی و عملگرهای مخصوص نمایش اعداد حقیقی (ممیزشناور).

والدین بر اساس یکی از روش‌های انتخاب معرفی و توصیف شده، انتخاب می‌شوند. تولیدمثل به صورت احتمالی بکار برده می‌شود. هر جفت (یا گروه) از والدین دارای احتمال p_c، برای تولید فرزندان هستند. معمولا از یک احتمال بازترکیب بزرگ استفاده می‌شود.

در انتخاب والدین، مسائل زیر باید در نظر گرفته شوند :

- ✓ به علت احتمالی بودن انتخاب، امکان دارد که یک موجود به عنوان هر دو والد انتخاب شود. در این صورت فرزندان تولید شده یک کپی از والدین خواهند بود. فرآیند انتخاب والدین باید دارای آزمونی برای جلوگیری از این عملیات غیرضروری باشد.
- ✓ همچنین امکان دارد که یک موجود در بیش از یک عملیات بازترکیب شرکت کند. این حالت زمانی که از مدل ارزیابی انتخاب نسبی استفاده می‌شود، ایجاد مشکل می‌کند (همگرایی سریع جمعیت به یک جواب برتر).
- ✓ به علاوه، درعملگر بازترکیب می‌توان به گونه‌ای عمل شود که تنها در صورتی که فرزندان از والدین خود بهتر هستند، آنها را جایگزین والدین‌شان نماییم.

3-2-2-1-1 نمایش دودویی

بیشترعملگرهای بازترکیب برای نمایش دودویی جنسی هستند و بر روی دو والد انتخاب شده اعمال می‌شوند. چندین عملگر بازترکیب با ماسک‌های مختلف وجود دارند که عبارتند از بازترکیب تک نقطه‌ای، بازترکیب دو نقطه‌ای و بازترکیب یکنواخت.

بریمرمن، اولین عملگرهای بازترکیب چند والدینی را برای نمایش دودویی پیشنهاد نمود. با داشتن بردار n_μ والدین $x_1(t), \dots, x_{n_\mu}(t)$ ، با استفاده از رابطه زیر یک فرزند تولید می‌شود، به صورتی که $\acute{n}_\mu$ تعداد والدین با $x_{ij}(t) = 0$ است :

$$\tilde{x}_{ij}(t) = \begin{cases} 0 & if \quad \acute{n}_\mu \geq \frac{n_\mu}{2}, l = 1, \dots, n_\mu \\ 1 & otherwise \end{cases}$$

همچنین روش بازترکیب n - نقطه‌ای چند والدینی نیز توسط بریمرمن پیشنهاد شده است. به صورتی که $n_\mu - 1$ نقطه بازترکیب یکسان در n_μ والد انتخاب می‌شود. یک فرزند توسط انتخاب یک بخش از هر والد تولید می‌شود.

جونزعملگر بازترکیب تپه‌نوردی را که می‌توان برای هر نمایشی بکار برد، ارائه نموده است. بازترکیب تپه‌نوردی با دو والد شروع می‌شود و فرزندان گوناگونی از ترکیب این دو والد ایجاد می‌شوند. این فرآیند تا زمانی که بیشترین ترکیبات ممکن از آن دو برسد و یا جفت فرزندی یافته شود که یکی از آنها دارای برازندگی بیشتر از بهترین والد باشد ادامه می‌یابد. بازترکیب تپه‌نوردی، سپس با این دو فرزند جدید به عنوان والدین جدید به ادامه جستجو می‌پردازد. در صورتی که والدینی بهتر، در طول دوره زمانی خاصی یافته نشدند، بدترین والد با یک والد تصادفی انتخاب شده جایگزین می‌شود.

3-2-2-1-2 نمایش ممیز شناور

همه عملگرهای بازترکیبی که تا کنون معرفی شدند، به جز عملگر بازترکیب چند والدینی، را می‌توانیم برای نمایش ممیز شناور بکار ببریم. برخلاف عملگرهای گسسته (در نمایش دودویی)، که اطلاعات بین والدین جابه‌جا می‌شوند، عملگرهای مخصوص نمایش ممیز شناور به گونه‌ای طراحی شده‌اند که مولفه‌های والدین انتخاب شده با یکدیگر ترکیب شوند.

یکی از اولین انواع عملگرهای بازترکیب ممیزشناور، عملگر خطی می‌باشد که توسط رایت ارائه شده است. از والدین $x_1(t)$ و $x_2(t)$ سه فرزند کاندید همانند $(x_1(t) + x_2(t))$ ، $(1.5\ x_1(t) + 0.5\ x_2(t))$ و $(0.5\ x_1(t) + 1.5\ x_2(t))$ تولید می‌شوند. دو تا از بهترین راه‌حل‌ها به عنوان فرزندان انتخاب می‌شوند. رایت همچنین، عملگر بازترکیب مکاشفه‌ای هدایت شده را پیشنهاد نمود، به گونه که یک فرزند از دو والد با استفاده از رابطه زیر تولید می‌شود :

$$\tilde{x}_{ij}(t) = \mathrm{U}(0,1)(x_{2j}(t) - x_{1j}(t) + x_{2j}(t))$$

البته با توجه به این محدودیت که والد $x_2(t)$ نمی‌تواند بدتر از والد $x_1(t)$ باشد. لازم به ذکر است که در رابطه بالا، U(0,1) بیانگر یک عدد تصادفی در بازه 0 و 1 می باشد.

مایکل‌ویز بازترکیب حسابی (*Arithmetic Crossover*) را ابداع نمود، که یک استراتژی تولیدمثل چند والدینی می‌باشد. در این عملگر از میانگین وزن‌دار دو یا بیش از دو والد استفاده می‌کند. یک فرزند با استفاده از رابطه زیر تولید می‌شود :

$$\tilde{\mathrm{x}}_{ij}(t) = \sum_{l=1}^{n_\mu} y_l x_{ij}(t)$$

با $\sum_{l=1}^{n_\mu} Y_l = 1$. حالت خاصی از عملگر بازترکیب حسابی برای $n_\mu = 2$ بدست می‌آید که در این صورت خواهیم داشت :

$$\tilde{x}_{ij}(t) = (1-y)x_{1j}(t) + yx_{2j}(t)$$

به طوری که $y \in [0,1]$. اگر $y = 0.5$ باشد، تاثیر آن این است که مولفه فرزند به سادگی با میانگین‌گیری از مولفه‌های متناظر در والدینش حاصل می‌شود.

اشلمن و شیفر تغییری بر میانگین وزن‌دار رابطه قبل داده‌اند که به هم‌برش مخلوط (*Blend Crossover*) و یا به اختصار BLX-α معروف می باشد و به صورت زیر می باشد :

$$\tilde{x}_{ij}(t) = \big(1 - y_j\big)x_{1j}(t) + y_j x_{2j}(t)$$

به طوری که $y_j = (1+2\alpha)\mathrm{U}(0,1) - \alpha$. عملگر BLX-α به صورت تصادفی برای هر مولفه یک مقدار تصادفی در دامنه زیر انتخاب می‌کند.

$$[x_{1j}(t) - \alpha\left(x_{2j}(t) - x_{1j}(t)\right), x_{2j}(t) + \alpha\left(x_{2j}(t) - x_{1j}(t)\right)]$$

همچنین، عملگر BLX-α فرض می‌کند که $x_{1j}(t) < x_{2j}(t)$ باشد. اشلیمن و شیفر دریافته‌اند که $\alpha = 0.5$ عملکردی مناسب دارد. عملگر BLX- α دارای این ویژگی است که موقعیت فرزند بستگی به فاصله والدین از یکدیگر دارد. در صورتی که این فاصله بزرگ باشد، پس فاصله بین فرزندان و والدینشان بزرگ خواهد بود.

مایکل‌ویز بازترکیب هندسی دو والده را برای تولید یک فرزند به صورت زیر پیشنهاد نموده است :

$$\tilde{x}_{ij}(t) = (x_{1j}x_{2j})^{0.5}$$

بازترکیب هندسی را می‌توان به تولیدمثل چند والده مانند زیر تعمیم داد :

$$\tilde{x}_{ij}(t) = (x_{1\,j}^{a_1} x_{2\,j}^{a_2} \dots x_{n_\mu j}^{a_{n_\mu}})$$

به طوری که n_μ تعداد والدین بوده و داریم $\sum_{l=1}^{n_\mu} \alpha_l = 1$.

سوتسوی، گلدبرگ و رندرز برسینی عملگر هم‌برش ساده ویا به اختصار SPX (*Simplex Crossover*) را به عنوان یکی دیگر از روش‌های هم‌برش برای نمایش ممیزشناور (اعداد حقیقی) معرفی کردند. آنها تعداد والدین را بزرگتر از دو در نظر گرفتند. در این روش بازترکیب، ابتدا بهترین و بدترین والدها تعیین شده و به ترتیب $x_1(t)$ و $x_2(t)$ نامیده می‌شوند. سپس مرکز والدین انتخاب شده، بدون در نظر گرفتن $x_2(t)$، محاسبه شده و با $\bar{x}(t)$ نام‌گذاری می‌شود. در نهایت، یک فرزند با استفاده از رابطه زیر تولید می‌شود :

$$\tilde{x}(t) = \tilde{x}(t) + (x_1(t) - x_2(t))$$

3-2-2-2 عملگر جهش

در این بخش، عملگرهای جهش برای نمایش دودویی و ممیز شناور توضیح داده شده و در انتها جهش بزرگ معرفی می‌گردد.

3-2-2-2-1 نمایش دودویی

برای الگوریتم ژنتیک در نمایش دودویی، به طور خلاصه عملگرهای جهش زیر ارائه شده‌اند :

- **جهش یکنواخت (تصادفی)** : که موقعیت بیت‌ها به صورت تصادفی انتخاب می‌شود و مقدار بیت متناظر تغییر می‌کند.
- **جهش Inorder** : که دو نقطه جهش به صورت تصادفی انتخاب می‌شود و تنها بیت‌های بین این دو نقطه، جهش تصادفی را انجام می‌دهند.
- **جهش گوسین** : برای نمایش دودویی با متغییر ممیز شناور تصمیم (*Decision*)، هینتردینگ پیشنهاد نموده است که رشته بیتی که نمایش‌دهنده متغیر Decision می‌باشد را به مقدار اعشاری برگردانده و با نویز گوسین آن را جهش داد. برای هر کروموزوم، یک مقدار تصادفی از توزیع پواسون برای تعیین میزان جهش یافتن هر ژن، تولید می‌شود. در ادامه، رشته‌های بیتی این ژن‌ها به اعداد حقیقی با ممیز شناور تبدیل می‌شوند. برای هر کدام از مقدارهای ممیز شناور اندازه گام $N(0,\sigma_j)$ اضافه می‌شود، که σ_j، برابر 0.1 از دامنه متغییر Decision است. مقدار ممیز شناور جهش یافته سپس به رشته بیتی تبدیل می‌شود.

برای رشته‌های بیتی با ابعاد بالا، ممکن است که جهش هزینه محاسباتی قابل ملاحظه‌ای به الگوریتم ژنتیک اضافه کند. به منظور کاهش این پیچیدگی محاسباتی، بیریو پیشنهاد نموده است که رشته بیتی به چندین قسمت کوچک تقسیم شود. احتمال جهش برای هر کدام از قسمت‌ها بکار می‌رود و اگر باید آن قسمت جهش یابد، یکی از بیت‌های آن به صورت تصادفی انتخاب شده و تغییر می‌کند.

3-2-2-2-2 نمایش ممیز شناور

همان‌طور که مطرح شد، زمانی که متغییر Decision دارای مقداری با ممیز شناور است، کارآیی بهتر با استفاده از یک عملگر جهش مخصوص نمایش ممیز شناور بدست می‌آید (در مقایسه با حالتی که آن را به نمایش دودویی تبدیل کنیم). درهمین راستا، یکی از اولین پیشنهادها جهش یکنواخت است که در رابطه زیر ارائه شده است :

$$\acute{x}_{ij}(t) = \begin{cases} \tilde{x}_{ij}(t) + \Delta\left(t, x_{max,j} - \tilde{x}_{ij}(t)\right) & \text{if a random digit is 0} \\ \tilde{x}_{ij}(t) + \Delta\left(t, \tilde{x}_{i,j}(t) - x_{min,ij}(t)\right) & \text{if arandom digit is 1} \end{cases}$$

در این رابطه $\Delta(t,x)$ مقدار تصادفی از دامنه [0,x] می باشد.

3-2-2-2-3 جهش بزرگ

عملگر جهش بزرگ (*Headless Chicken*) یک فرزند را از طریق بازترکیب مجدد یک والد با یک پاسخ تصادفی تولید شده به وجود می‌آورد. با وجود آنکه در اینجا از عملگر بازترکیب برای به هم پیوستن یک موجود با موجودی تصادفی تولید شده دیگر استفاده می‌شود، اما این فرآیند را نمی‌توانیم بازترکیب بنامیم، چرا که مفهوم ارث‌بری در آن وجود ندارد. این عملگر را به‌خاطر تولد ژن‌های جدید، یک عملگر جهش می‌دانند.

3-2-2-3 پارامترهای کنترلی

علاوه بر اندازه جمعیت، کارایی الگوریتم ژنتیک تحت تاثیر نرخ جهش، p_{m} و نرخ بازترکیب، p_{c} است. برای حل بیشتر مسائل با استفاده از الگوریتم ژنتیک، مقدار کم p_{m} و مقدار نسبتا بالای p_{c} بکار می‌رود. همچنین، معمولا مقادیر p_{c} و p_{m} ثابت نگه داشته می‌شوند. این پارامترها تاثیر بسیاری بر روی کارآیی الگوریتم دارند و تنظیم مناسب آنها می‌تواند کارآیی را بسیار افزایش دهد. اغلب، بدست آوردن چنین تنظیماتی بسیار زمان‌گیر می‌باشد. یک راه‌حل برای یافتن بهترین مقدار برای این پارامترهای کنترلی، تغییر پویای پارامترها است.

با استفاده از نرخ جهش پویا می‌توان کارآیی را به طور قابل ملاحظه‌ای افزایش داد و از رابطه زیر استفاده نمود که در آن نرخ جهش به صورت نمایی با شماره نسل کاهش می‌یابد :

$$p_{\mathrm{m}}(t) = \frac{1}{240} + \frac{0.11375}{2^t}$$

به علاوه، برای نمایش دودویی استفاده از دنباله نرخ جهش برای هر بیت، $j = 1,\ldots, n_b$ ، به صورت رابطه زیر پیشنهاد شده است که در این دنباله، n_b کم‌ارزش‌ترین بیت را مشخص می‌کند.

$$P_m(j) = \frac{0.3528}{2^{j-1}}$$

دو روش بالا با یکدیگر ترکیب شده‌اند و رابطه زیر بدست آمده است :

$$P_m(j,t) = \frac{28}{1905 \times 2^{j-1}} + \frac{0.4026}{2^{t+j-1}}$$

اندازه زیاد نرخ جهش در مراحل اولیه جستجو، قابلیت پویش را افزایش می‌دهد و با کاهش نرخ آن با افزایش تعداد نسل‌ها، قابلیت انتفاع افزایش می‌یابد. از روش‌های دیگر نیز می‌توان برای کاهش نرخ جهش استفاده نمود. روش بالا منجر به کاهش نمایی نرخ جهش می‌شود. در این جا می‌توان از کاهش خطی نیز استفاده نمود که p_{m} را با سرعت کمتری کاهش می‌دهد و امکان پویش بیشتر را فراهم می‌سازد. در هر حال ممکن است که کاهش آهسته برای راه‌حل‌های خوب تاکنون یافته شده، مخرب باشد. یک راه خوب برای تنظیم نرخ جهش، توجه به میزان برازندگی موجودات است، هر چقدر که موجود برازنده‌تر باشد، احتمال جهش ژن‌هایشان بایستی کمتر باشد.

به همین ترتیب، برای نمایش ممیز شناور نیز کارآیی تحت تاثیر اندازه گام جهش است. یک استراتژی مناسب، شروع با اندازه گام بزرگ جهش است تا امکان جهش‌های تصادفی بزرگ‌تر در فضای جستجو فراهم آید. اندازه گام بزرگ به مرور زمان کاهش می‌یابد. بنابراین، تغییرات بسیار کوچکی در انتهای فرآیند جستجو انجام می‌پذیرد. اندازه گام همچنین می‌تواند متناسب با برازندگی موجودات باشد. هر چقدر که موجود برازنده‌تر باشد اندازه گام کوچک‌تری برای آن در نظر گرفته می‌شود.

نرخ بازترکیب p_c نیز تاثیر بسیاری بر روی کارایی می‌گذارد. مقدار مناسب این نرخ، بستگی به مساله مورد نظر دارد. از استراتژی‌های مشابه تنظیم p_{m} می‌توانیم برای تنظیم پویای p_c نیز استفاده کنیم.

علاوه بر p_{m} (اندازه گام جهش برای نمایش ممیز شناور) و p_c ، انتخاب درست نوع عملگر تکاملی، وابستگی کاملی به مساله دارد. ترکیب مناسب عملگرهای بازترکیب، جهش و انتخاب با مقادیر مناسب برای پارامترهای کنترلی کار دشواری است. برخی روش‌های پویا برای این منظور پیشنهاد شده‌اند که بر اساس پیشرفت جستجو به صورت پویا عملگرها را تعویض می‌کنند. در کل یافتن بهترین عملگرها و کنترل مقدار پارامترها خود یک مساله بهینه‌سازی چند هدفه می‌باشد.

3-2-2-3 انواع الگوریتم‌های ژنتیک

در این بخش، چند نمونه از انواع الگوریتم‌های ژنتیک که براساس بکارگیری استراتژی‌های متنوع جایگزینی و نیز روش‌های خاص نمایش کروموزوم‌ها می‌باشد، مورد بررسی قرار می‌گیرند.

3-2-2-3-1 روش‌های شکاف نسلی

دو دسته مهم از الگوریتم‌های ژنتیک، مبتنی بر نوع استراتژی جایگزینی بکار رفته در آنها می‌باشند و به روش‌های شکاف نسلی (*Generation Gap Methods*) معروف می باشند. این دسته‌ها عبارتند از الگوریتم‌های ژنتیک نسلی یا GGA (*Generational Genetic Algorithm*) و الگوریتم‌های ژنتیک پایا یا SSGA (*Steady State Genetic Algorithm*).

برای الگوریتم‌های ژنتیک نسلی، استرانژی جایگزینی همه والدین را با فرزندان آنها جایگزین می‌کند. این جایگزینی منجر به این می‌شود که هیچ همپوشانی بین نسل کنونی و نسل جدید (فرض کنید که نخبه‌گرایی استفاده نشود) وجود نداشته باشد. در الگوریتم‌های ژنتیک پایا، تصمیمی فورا بعد از تولید فرزند و جهش یافتن آن صورت می‌گیرد، که از میان فرزند و والد آن، کدام یک به نسل بعدی منتقل شوند. بنابراین، همپوشانی بین جمعیت کنونی و بعدی وجود خواهد داشت.

میزان همپوشانی بین نسل کنونی و نسل بعدی با عنوان شکاف نسلی معروف است. الگوریتم‌های ژنتیک نسلی دارای شکاف نسلی صفر هستند، در حالی که الگوریتم‌های ژنتیک پایا دارای شکاف نسلی زیادی می‌باشند.

انواع مختلفی از استراتژی‌های جایگزینی برای الگوریتم‌های ژنتیک پایا وجود دارند که عبارتند از :

- **جایگزینی بدترین** : فرزند با بدترین موجود در جمعیت کنونی جایگزین می‌شود.
- **جایگزینی تصادفی** : فرزند به صورت تصادفی با یک موجود در جمعیت کنونی جایگزین می‌شود.
- **حذف رقابتی** : مجموعه‌ای از موجودات زنده به صورت تصادفی انتخاب شده و سپس، بدترین موجود این گروه با فرزند جایگزین می‌شود. روش دیگر انتخاب دو موجود و جایگزینی بدترین موجود با احتمال $0.5 \leq p_r \leq 1$ می‌باشد.
- **جایگزینی قدیمی‌ترین** : از استراتژی کسی که زودتر وارد شده است زودتر خارج می‌شود که به FIFO معروف است، استفاده می‌شود. این استراتژی دارای یک احتمال زیاد برای جایگزینی یکی از بهترین موجودات است.
- **انتخاب محافظه‌کار** : این روش، استراتژی جایگزینی FIFO را با انتخاب مسابقه‌ای دودویی قطعی ترکیب می‌کند. در انتخاب محافظه‌کار، دو موجود انتخاب می‌شوند که همیشه یکی از آنها پیرترین موجود در جمعیت کنونی است. بدترین این دو با فرزند جایگزین می‌شود. این روش تضمین می‌کند که پیرترین موجود در صورت خوب بودن آن از بین نمی‌رود.
- **نخبه‌گرایی** : این روش تعمیم یافته استراتژی جایگزینی انتخاب محافظه‌کار می‌باشد و از آن هنگامی که بهترین موجود در فرآیند جایگزینی حذف شده است، استفاده می‌شود.
- **رقابت والد-فرزند** : در این روش برای جایگزینی فرزند با یکی از والدین خودش تصمیم‌گیری می‌شود.

3-2-2-3-2 الگوریتم ژنتیک آشفته

در الگوریتم ژنتیک استاندارد از جمعیتی استفاده می‌شود که هر کدام دارای طول کروموزوم یکسان هستند. برای یک فضای جستجوی n_x - بعدی، الگوریتم ژنتیک استاندارد یک راه‌حل را از طریق

عملگرهای تکاملی بر روی موجودات کامل n_x - بعدی می‌یابد. در این صورت، موجودات برازنده‌ای یافته می‌شوند، اما امکان دارد که برخی از ژن‌های آنها بهینه نباشند. یافتن مقادیر مناسب برای ژن‌ها از طریق عملیات بازترکیب و جهش بر روی کل موجود کار دشواری است. امکان از دست دادن برخی از ژن‌های بهینه، یا گروهی از آنها از طریق بازترکیب وجود دارد.

گلدبرگ برای حل این مشکل، الگوریتم ژنتیک آشفته یا MGA (*Messy Genetic Algorithm*) را معرفی نمود. در یک الگوریتم ژنتیک آشفته، راه‌حل‌ها از طریق تکامل بلوک‌های سازنده و ترکیب آنها یافته می‌شوند. در اینجا منظور از بلوک سازنده گروهی از ژن‌ها می‌باشد. در الگوریتم ژنتیک آشفته، موجودات دارای اندازه‌های گوناگون هستند و به صورت جفت مقدار- موقعیت مشخص می‌شوند. موقعیت مشخص کننده مکان ژن در کروموزوم است. این جفت را ژن آشفته می‌نامند. به عنوان نمونه، موجود ((1,1)(4,0)(3,1)(1,0)) نمایشگر موجود 10*0 می‌باشد.

نمایش آشفته ممکن است منجر شود که ژن‌ها دو بار تعیین مقدار گردند و یا اصلا مقداری نداشته باشند. در مثال بالا هر دو نمونه را مشاهده می‌کنید. ژن 1 دو بار اتفاق افتاده است، در حالی که ژن 2 اصلا اتفاق نیفتاده است و دارای هیچ مقداری نیست. ارزیابی موجودات آشفته نیاز به استراتژی دارد که با این شرایط سازگار باشد. در مورد تعیین مجدد مقدار یک ژن، اولین مقدار مشخص شده به آن تعلق می‌گیرد. همچنین، برای ژن‌هایی که مقدار آنها مشخص نیست از یک الگوی رقابتی استفاده می‌شود. الگوی رقابتی یک راه‌حل بهینه محلی است. به عنوان نمونه، اگر 1101 یک الگو باشد، برازش 10*0 را همانند برازش 0101 در نظر می‌گیریم.

هدف در الگوریتم ژنتیک آشفته، تکامل بلوک‌های سازنده و سپس، ترکیب بلوک‌های سازنده بهینه برای شکل دادن یک راه‌حل بهینه است. مراحل یک الگوریتم ژنتیک آشفته عبارتند از :

1. **مقداردهی اولیه** : برای ایجاد بلوک‌های سازنده با طول مشخص شده n_m.
2. **مرحله اصلی** : هدف آن تولید بلوک‌های سازنده بهتر می‌باشد.
3. **الحاق** : برای ترکیب نمودن بلوک‌های سازنده.

حلقه بیرونی مشخص کننده اندازه بلوک سازنده است که با یک مقدار کوچک شروع شده و مقدار آن به مرور افزایش می‌یابد تا به بیشترین مقدار برسد و یا راه‌حلی قابل قبول یافته شود. در حلقه بیرونی نیز بهترین راه‌حل از مرحله الحاق به عنوان الگوی رقابتی برای نسل بعدی مشخص می‌شود.

3-2-2-3-3 برنامه‌نویسی توصیف ژن

برنامه‌نویسی توصیف ژن یا GEP (*Gene Expression Programming*) بر اساس مفاهیم تکرار و توصیف مولکول DNA در سطح ژن بنیان نهاده شده است. توصیف یک ژن شامل نسخه‌برداری از DNA آن به RNA می‌شود. فرآیندی که در ادامه منجر به شکل‌گیری آمینواسیدها شده و در نهایت، پروتئین‌ها را در سطح فنوتایپی یک موجود تولید می‌کند. در واقع، ایده اصلی در برنامه‌نویسی توصیف

ژن آن است که یک پاسخ با ماهیت فنوتایپی درختی (پروتئین) را به شکل یک دنباله ژنی ساده آرایه‌ای (DNA) نمایش دهیم و عملگرهای رشته‌ای تولیدمثل را در الگوریتم‌های ژنتیک برای آن بکار ببریم.

منطق الگوریتمی در برنامه‌نویسی توصیف ژن کاملا مشابه با الگوریتم ژنتیک است. تنها تفاوت موجود در نحوه محاسبه برازش و نیز مدیریت عملگرهای تولیدمثل است. برای محاسبه برازش بایستی که ابتدا فنوتایپ معادل هر توصیف ژنی در یک کروموزوم ساخته شده و برنامه مربوط به آن توصیف اجرا گردد. بر اساس خروجی حاصل شده، می‌توان در مورد میزان شایستگی آن برنامه که فنوتایپی درخت گونه دارد، تصمیم گیری نمود. عملگرهای تولیدمثل نیز بایستی با این فرض که طول کروموزوم‌های موجود در جمعیت، به دلیل ماهیت درختی فنوتایپ آنها، با یکدیگر متفاوت است، طراحی شوند. در کل بایستی عنوان نمود که در مواقعی که فنوتایپ مربوط به هر پاسخ شکلی درختی دارد، اغلب از برنامه‌نویسی ژنتیک استفاده می‌شود. در این گونه موارد، هنگامی از برنامه نویسی توصیف ژن بهره‌برده می‌شود که عملگرهای تولیدمثل در برنامه‌نویسی ژنتیک، نتواند پویش موثر فضای جستجو را تضمین نمایند.

3-3 برنامه‌نویسی ژنتیک

برنامه‌نویسی ژنتیک یا GP (*Genetic Programming*) از دیدگاه بسیاری از محققان به عنوان یکی از انواع الگوریتم‌های ژنتیک شناخته می‌شود. همانند الگوریتم ژنتیک، برنامه‌نویسی ژنتیک به تکامل ژنوتایپی توجه دارد. تفاوت اصلی بین این دو روش، طریقه نمایش مورد استفاده برای پاسخ‌ها (موجودات) می‌باشد. الگوریتم ژنتیک از نمایش رشته‌ای (یا برداری) استفاده می‌کند، در حالی که در برنامه‌نویسی ژنتیک از نمایش درختی استفاده می‌شود.

برنامه‌نویسی ژنتیک، در هر نسل، هر برنامه (موجود) اجرا می‌شود تا میزان کارایی آن در فضای جستجوی مساله مشخص شود. نتایج بدست آمده از اجرای برنامه‌های کامپیوتری در حال تکامل، برای تعیین برازندگی آن برنامه‌ها مورد استفاده قرار می‌گیرد. برنامه‌نویسی ژنتیک برای تکامل برنامه‌های کامپیوتری توسعه یافته است. برنامه‌ها برای دامنه وسیعی از مسائل به کار می‌روند که شامل عبارات بولی، برنامه‌ریزی، کشف تجربی، حل معادلات سیستم‌ها، شکل گیری مفهوم، برنامه نویسی خودکار، شناسایی الگو و طراحی شبکه‌های عصبی می‌باشد. حوزه‌های دیگری که در آنها از برنامه‌نویسی ژنتیک استفاده شده است عبارتند از درخت تصمیم، مسائل بازی، بیوانفورماتیک، داده کاوی و روباتیک.

3-3-1 شبه کد برنامه نویسی ژنتیک

خصوصیات مهم برنامه نویسی ژنتیک به طور خلاصه به شرح ذیل می باشند :

- استفاده از نمایش درختی.
- طول متغیر برای کروموزوم‌ها (به عنوان تنها الگوریتم تکاملی که دارای این ویژگی است).
- بکارگیری یک جمعیت با تعداد اعضاء ثابت.
- عملگر بازترکیب در برنامه‌نویسی ژنتیک عملگر اصلی است.

- عملگر جهش در برنامه‌نویسی ژنتیک عملگر فرعی است.
- تنظیم احتمال بازترکیب p_c به مقادیر بزرگ (بزرگتر از 0.9).
- در عملگر بازترکیب احتمال برش شاخه‌های غیرپایانی، زیاد (بزرگـتر از 0.9) است.
- در عملگر بازترکیب احتمال برش شاخه‌های پایانی، کم (کوچـکـتر از 0.01) است.
- تنظیم احتمال جهش p_{m} به مقادیر کوچک (کوچـکـتر از 0.01)
- عملگرهای اصلاح معماری، تنها منوط به تکثیر و یا حذف زیردرخت‌ها نمی‌باشند.

شبه کد برنامه نویسی ژنتیک در شکل (3-2) نشان داده شده است.

Function GP(Problem) returns a state that is a local optimum

Input : $\mathrm{Population_{size}}$,$\mathrm{nodes_{func}}$, $\mathrm{nodes_{sterm}}$, $\mathrm{P_{crossover}}$, $\mathrm{P_{mutation}}$, $\mathrm{P_{alteration}}$

Output : $\mathrm{S_{best}}$

Population ← InitializePopulation ($\mathrm{Population_{size}}$, $\mathrm{nodes_{func}}$, $\mathrm{nodes_{sterm}}$) ;

EvaluatePopulation (Population) ;

← GetBestSolution (Population) ;$\mathrm{S_{best}}$

While-StopCondition() do

Children ← ∅ ;

while size(Children) < $\mathrm{Population_{size}}$ do

Operator ← SelectGeneticOperator ($\mathrm{P_{crossover}}$, $\mathrm{P_{mutation}}$, $\mathrm{P_{alteration}}$) ;

if Operator ≡ CrossoverOperator then

$\mathrm{Parent_1}$, $\mathrm{Parent_2}$ ← SelectParents (Population, $\mathrm{Population_{size}}$) ;

$\mathrm{Child_1}$,$\mathrm{Child_2}$ ← Crossover ($\mathrm{Parent_1}$,$\mathrm{Parent_2}$) ;

Children ← $\mathrm{Child_1}$;

Children ← $\mathrm{Child_2}$;

else if Operator ≡ MutationOperator then

$\mathrm{Parent_1}$ ← SelectParents (Population, $\mathrm{Population_{size}}$) ;

$\mathrm{Child_1}$ ← Mutate ($\mathrm{Parent_1}$) ;

Children ← $\mathrm{Child_1}$;

else if Operator ≡ AlterationOperator then

```
    Parent₁ ← SelectParents (Population, Population_size) ;
    Child₁ ← AlterArchitecture (Parent₁) ;
    Children ← Child₁ ;
   end
 end
 evaluatePopulation (Children) ;
 Population ← Replace (Population , Children) ;
 S_best ← GetBestSolution (Population) ;
end
return S_best ;
```

شکل (3-2) : شبه کد برنامه نویسی ژنتیک

3-3-2 نمایش مبتنی بر درخت

برنامه‌نویسی ژنتیک برای تکامل برنامه‌های کامپوتری توسعه یافت. هر موجود یا کروموزوم، نمایانگر یک برنامه کامپیوتری است که با استفاده از ساختار درختی نمایش داده می‌شود. بر خلاف الگوریتم‌های ژنتیک که اندازه موجودات معمولا ثابت بود، در جمعیت برنامه‌نویسی ژنتیک معمولا موجودات در اندازه، شکل و پیچیدگی‌های مختلفی هستند. در اینجا منظور از اندازه عمق درخت می‌باشد و شکل مربوط به عدد انشعاب گره‌های درخت می‌باشد. اندازه و شکل یک موجود خاص نیز ثابت نمانده و ممکن است که از طریق عملگرهای تولیدمثل تغییر کند.

تعریف گرامر وابسته به دامنه در نمایش درختی، بسیار مهم است. در این گرامر، ماهیت مساله مورد نظر به دقت مشخص می‌شود. باید امکان نمایش هر راه‌حل ممکنی با استفاده از گرامر تعریف شده وجود داشته باشد. در گرامر، مجموعه پایانی، مجموعه تابع و قوانین معنایی باید تعریف شوند. مجموعه پایانی همه متغیرها و ثابت‌ها را مشخص می‌کند. مجموع تابع شامل همه توابعی می‌باشد که می‌توان آنها را برای عناصر مجموعه پایانی بکار برد. این توابع می‌توانند شامل توابع ریاضی، حسابی و یا دودویی باشند. ساختارهای تصمیم همانند if-then-else و حلقه نیز می‌توانند در مجموعه تابع بکار روند. عناصر مجموعه پایانی گره‌های برگ در درخت تکاملی را شکل می‌دهند و عناصر مجموعه تابع، گره‌های غیربرگ را می‌سازند. برای یک مساله خاص، فضای جستجو شامل مجموعه تمام درخت‌های ممکن می‌باشد که می‌توان آنها را با گرامر تعریف شده ساخت.

3-3-3 جمعیت اولیه

به صورت تصادفی و با توجه به محدودیت‌های بیشترین عمق و محدودیت معنایی بیان شده، توسط گرامر تولید می‌شود. برای هر موجود، یک ریشه به صورت تصادفی از عناصر مجموعه تابع انتخاب می‌شود. عدد انشعاب (تعداد فرزندان) یک ریشه و هر گره غیرپایانی دیگر، توسط تابع انتخابی تعیین می‌شود. الگوریتم برای مقداردهی اولیه هر گره غیر ریشه، عنصری را از مجموعه پایانی و یا از مجموعه تابع به صورت تصادفی انتخاب می‌کند. زمانی که عنصری از مجموعه پایانی انتخاب می‌شود، گره مربوطه یک گره برگ محسوب شده و دیگر توسعه پیدا نمی‌کند.

3-3-4 تابع برازش

تابع برازش مورد استفاده برای برنامه‌نویسی ژنتیک بستگی به مساله دارد. از آنجا که موجودات معمولا نماینده یک برنامه هستند، محاسبه برازندگی برنامه توسط یک مجموعه آزمون انجام می‌پذیرد. اندازه‌گیری برازندگی موجودات با توجه به کارآیی برنامه (موجود) بر روی مجموعه آزمون انجام می‌شود. برنامه‌نویسی ژنتیک را برای تکامل درخت تصمیم نیز می‌توان بکار برد. برای این کاربرد، هر موجود نمایانگر یک درخت تصمیم است. برازندگی موجودات، از طریق محاسبه دقت دسته‌بندی درخت تصمیم محاسبه می‌شود. در صورتی که هدف تکامل، استراتژی بازی یک برنامه کامپیوتری باشد، برازش یک موجود می‌تواند برابر با نسبت تعداد دفعاتی که موجود بازی را برنده شده به تعداد کل دفعاتی که بازی کرده است، باشد. با کمک تابع برازش می‌توانیم به ویژگی‌های غیر دلخواه امتیاز منفی بدهیم تا از پیشرفت آنها جلوگیری کنیم. برای نمونه، به جای استفاده از یک عمق مشخص شده از قبل برای درخت، می‌توان عمق درخت را به صورت یک امتیاز منفی در تابع برازش در نظر بگیریم. به طور مشابه می‌توان برای درخت‌های پرپشت (حالتی که در آن عدد انشعاب زیاد است) و همچنین، موجوداتی که از نظر معنایی صحیح نیستند، امتیاز منفی در نظر گرفت.

3-3-5 عملگرهای بازترکیب

در برنامه‌نویسی ژنتیک، دو روش برای تولید فرزندان وجود دارد که هر کدام در تعداد فرزندان تولید شده با یکدیگر متفاوت هستند و عبارتند از :

1. **تولید یک فرزند** : یک گره تصادفی در هر کدام از والدین انتخاب می‌شود. سپس، عملگر بازترکیب از طریق جایگزینی زیر درخت متناظر در یک والد توسط والد دیگر انجام می‌پذیرد.
2. **تولید دو فرزند** : یک گره تصادفی در هر کدام از والدین انتخاب می‌شود. در این مورد، زیر درخت‌های متناظر برای ایجاد دو فرزند با یکدیگر تعویض می‌شوند.

3-3-6 عملگرهای جهش

عملگرهای جهش برای کاربردهای خاص توسعه یافته‌اند. در هر حال، بسیاری از عملگرهای جهش در برنامه‌نویسی ژنتیک، در حالت کلی نیز کاربرد دارند. عملگرهای جهش زیر را می‌توان برای برنامه‌نویسی ژنتیک بکار برد :

- **جهش گره تابع** : یک گره غیر پایانی یا گره تابع، به صورت تصادفی انتخاب می‌شود و با عنصری از مجوعه تابع که به صورت تصادفی انتخاب شده است جایگزین می‌شود.
- **جهش گره پایانی** : یک گره برگ یا گره پایانی، به صورت تصادفی انتخاب می‌شود و با یک گره پایانی جدید که به صورت تصادفی از مجموعه پایانی انتخاب شده است، جایگزین می‌شود.
- **جهش تعویضی** : یک گره تابع به شکل تصادفی انتخاب شده و پارامترهای آن تعویض می‌شوند.
- **جهش رشدی** : یک گره به صورت تصادفی انتخاب می‌شود و با یک زیر درخت تصادفی انتخاب شده جایگزین می‌شود. زیر درخت جدید دارای عمق محدود و از قبل مشخص شده است.
- **جهش گوسین** : یک گره پایانی که دارای مقداری ثابت است به صورت تصادفی انتخاب شده و با اضافه کردن یک مقدار تصادفی گوسین به آن مقدار ثابت جهش می‌یابد.
- **جهش برشی** : یک گره تابع به صورت تصادفی انتخاب می‌شود و توسط یک گره پایانی تصادفی جایگزین می‌شود. این عملگر جهش، درخت را هرس می‌کند.

موجوداتی که باید جهش یابند مطابق با نرخ جهش p_m انتخاب می‌شوند. علاوه بر احتمال جهش، گره‌ها در درخت انتخاب شده طبق همین احتمال، جهش می‌یابند. هر چقدر که احتمال p_m بزرگتر باشد، تغییرات بیشتری در درخت صورت می‌پذیرد. از طرف دیگر، هر چقدر که احتمال جهش p_m بیشتر باشد، موجودات بیشتری جهش می‌یابند. همه عملگرهای جهش را می‌توان پیاده‌سازی نمود و یا تنها از بخشی از آنها استفاده کرد. اگر بیش از یک عملگر جهش پیاده‌سازی شود، یک عملگر می‌تواند از میان آنها به صورت تصادفی انتخاب شده و یا عملگرها به ترتیب بکار گرفته شوند.

3-3-7 عملگرهای اصلاح معماری

علاوه بر عملگرهای جهش مطرح شده، عملگرهای جهش غیرجنسی دیگری نیز معرفی شده‌اند که از آنها با عنوان عملگرهای اصلاح معماری نیز یاد می‌شود.

- **عملگر جایگشت** : این عملگر شبیه به جهش تعویضی است. اگر تابعی دارای n پارامتر باشد، عملگر جایگشت، جایگشتی تصادفی $n!$ جایگشت ممکن از پارامترها تولید می‌کند. در ادامه، پارامترهای تابع مطابق با جایگشت تصادفی تولید شده، جهش می‌یابند.
- **عملگر ویرایشی** : این عملگر برای ساخت موجودات مطابق با یک قانون از قبل تعریف شده عمل می‌کند. به عنوان نمونه، یک زیر درخت که نمایشگر عبارات بولی x AND x می‌باشد، با یک گره شامل تک عنصر x جایگزین می‌شود. عملگرهای ویرایشی همچنین برای تاکید بر قوانین معنایی نیز می‌توانند بکار بروند.
- **عملگرهای بلوک سازنده** : هدف در عملگر بلوک سازنده تشخیص خودکار بلوک‌های سازنده مفید می‌باشد. یک گره تابع جدید برای بلوک سازنده شناسایی شده تعریف می‌شود و برای

جایگزینی با زیر درخت نمایش داده شده توسط بلوک سازنده استفاده می‌گردد. مزیت این عملگر این است که بلوک‌های سازنده خوب توسط عملیات تولیدمثل تغییر داده نمی‌شوند.

3-4 استراتژی تکامل

ریچنبرگ ثابت نمود، از انجایی که فرایندهای زیست شناختی توسط تکامل بهینه می شوند و تکامل خود یک فرایند زیست شناختی است، پس باید بتوان تکامل را نیز بهینه نمود. استراتژی تکامل (ES) ، در سال 1960 توسط ریچنبرگ پیشنهاد شده است و تحقیقات مبتنی بر مفهوم تکامل ، توسط اسکوفل انجام شده است. در استراتژی تکامل در حالی که هم به تکامل فنوتایپی توجه داریم و هم تکامل ژنوتایپی، تأکید بیشتر بر تکامل رفتار ژنوتایپی موجود می باشد. هرموجود توسط بلوک های سازنده ژنی خود و مجموعه‌ای از پارامترهای استراتژی که رفتار موجودات در محیط شان را مدل می کنند، نمایش داده می‌شود. ویژگی های ژنی و پارامترهای استراتژی با هم تکامل می یابند، در حالی که تکامل ویژگی‌های ژنی توسط پارامترهای استراتژی کنترل می شود.

از اولین کاربردهای استراتژی تکامل ، بهینه سازی ازمایشگاهی برای مسأله هیدرودینامیکی بوده است. از آن پس ، استراتژی تکامل برای حل مسائل بهینه سازی تابعی و بسیاری از مسائل دنیای واقعی همچون طراحی کنترل شده طراحی ترانسفورماتور و سیستم های قدرت بکار برده شده است. الگوریتم و شبه کد استراتژی تکامل در شکل (3-3) نشان داده شده است. خصوصیات مهم استراتژی تکامل عبارتند از :

1. استفاده از نمایش اعداد حقیقی (ممیز شناور).
2. بکارگیری یک جمعیت با تعداد اعضاء ثابت.
3. عملگر جهش در استراتژی تکامل عملگر اصلی است (در برخی از روش های مبتنی بر استراتژی تکامل از عملگر بازترکیب نیز استفاده شده است . لیکن استفاده از این عملگر موجب دورتر شدن از ماهیت استراتژی تکامل می شود.
4. الگوریتم عملگر جهش و پارامترهای آن طی فرایند استراتژی تکامل ، تکامل می یابند.
5. پارامترهای استراتژی ، همراه هر کدام از موجودات می باشند. این پارامترهای استراتژی خود تطبیق می باشند تا بهترین جهت جستجو و بیشترین اندازه گام هر بعد را تعیین می کنند.
6. نرخ جهش با توجه به برازش پاسخ ها به صورت تطبیقی تنظیم می شود . برای پاسخ های برازنده نرخ جهش کم و بری پاسخ های با برازش پایین این نرخ بالا در نظر گرفته می شود. نرخ جهش بایستی منجر به موفقیت در بهبود برازش پاسخ در 20% موارد شود .
7. در صورتیکه درصد موفقیت بیشتر از 20% بود نرخ جهش افزایش می یابد و در غیر این صورت این نرخ کاهش خواهد یافت .
8. احتمال جهش با توجه به یک تابع توزیع احتمال (مثلا گوسین) محاسبه می شود.
9. منظور از استراتژی تکامل $ES-(\mu/\rho,\lambda)$ آن است که از μ والد λ فرزند تولید می شود .در ضمن در مرحله تولید فرزند از هر ρ والد ، یک فرزند تولید می شود و در جایگزینی انتخاب

$(\mu + \lambda)$ از مجموع جمعیت والدین و فرزندان $(\lambda + \mu)$ تعداد μ پاسخ با توجه به یک استراتژی انتخاب ، گزینش می شوند.

10. در جایگزینی انتخاب (μ, λ) تعداد μ عضو برتر از تعداد λ فرزند ، گزینش شده و به نسل بعد منتقل می شوند.
11. همیشه رابطه $1 \leq \mu \leq \lambda$ برقرار است.

Function ES (problem) returns a state that is a local optimum

Input : μ , λ , ProblemSize

Output : S_{best}

$Population_1 \leftarrow$ InitializePopulation (μ , ProblemSize) ;

EvaluatePopulation (Population) ;

$S_{best} \leftarrow$ GetBestSolution (Population) ;

while -Stop Condition () do

  Children ← Ø ;

  for i = 0 to λ do

    $P_i \leftarrow$ GetParent (Population , i) ;

    $S_i \leftarrow$ Ø ;

    $S_i$problem ← Mutate ($P_i$problem , $P_i$strategy) ;

    $S_i$strategy ← Mutate ($P_i$strategy) ;

    Children ← S_i ;

  end

  EvaluatePopulation (Children) ;

  Population ← Replace (Population , Children , μ) ;

  $S_{best} \leftarrow$ GetBestSolution (Population) ;

end

return S_{best} ;

شکل (3-3) : شبه کد استراتژی تکامل

3-5 برنامه نویسی تکاملی

برنامه نویسی تکاملی یا EP (*Evolutionary Programming*)، از تحقیقات فوگل در سال 1962 در استفاده از تکامل شبیه‌سازی شده برای توسعه هوش‌مصنوعی و مدل‌سازی هوش انسان سرچشمه می گیرد.از نظر فوگل، هوش را می‌توان ویژگی دانست که به یک سیستم اجازه می‌دهد تا رفتار خود را برای رسیدن به هدفی خاص در محیط‌های مختلف تطبیق دهد. این مدل، از تکامل خصیصه‌های رفتاری تقلید می‌کند. فرآیند تکامل، ابتدا برای تکامل ماشین‌های حالت متناهی، توسعه یافته است که شامل یافتن یک مجموعه از رفتارهای بهینه از فضای رفتارهای مشاهده شده می‌باشد. بنابراین تابع برازش، خطای رفتاری یک موجود را با توجه به محیطی که آن موجود در آن قرار دارد، اندازه گیری می‌کند.

برای تکامل ماشین‌های حالت متناهی با استفاده از برنامه‌نویسی تکاملی، از نمایش دنباله‌های ترتیبی برای جمعیت موجودات استفاده می شود، که کاملا با روش نمایش رشته بیتی در الگوریتم ژنتیک متفاوت است. البته برنامه‌نویسی تکاملی تنها محدود به نمایش دنباله ترتیبی نمی‌شود. فوگل برنامه‌نویسی تکاملی را با نمایش بردار اعداد حقیقی با کاربردهایی در بهینه‌سازی توابع پیوسته تعمیم داده است.

برنامه‌نویسی تکاملی همانند دیگر فرامکاشفه های تکاملی، سعی در تقلید از فرآیند تکامل طبیعی دارد، اما با این تفاوت اساسی که در آن به توسعه مدل‌های رفتاری (و نه به مدل‌های ژنی همانند GA و GP) اهمیت داده می‌شود. برنامه‌نویسی تکاملی از شبیه‌سازی رفتار تطبیقی در تکامل ناشی شده است. به همین علت در برنامه‌نویسی تکاملی به تکامل فنوتایپی توجه می‌شود. برنامه‌نویسی تکاملی برای حل بسیاری از مسائل دنیای واقعی همچون زمان‌بندی و مسیریابی، طراحی کنترل کننده، روباتیک، پردازش تصویر و سیستم‌های قدرت به‌کار برده شده است.

خصوصیات مهم برنامه‌نویسی تکاملی که الگوریتم و شبه‌کد آن در شکل (3-4) نشان داده شده است، عبارتند از :

1. استفاده از نمایش نمادی یا سمبلی برای نشان دادن حالات در ماشین‌های حالت متناهی (البته در برنامه‌نویسی تکاملی از سایر روش‌های نمایشی نیز استفاده شده است).
2. به‌کارگیری یک جمعیت با تعداد اعضاء ثابت.
3. عملگر جهش در استراتژی تکامل عملگر اصلی است.
4. عملکرد عملگر جهش و پارامترهای آن طی فرآیند الگوریتم، تکامل می‌یابد (استفاده از حالات جدید و نیز احتمالات قابل تکامل برای هرحالت در مسأله ماشین‌های حالت متناهی).

5. پارامترهای جهش (گام و نرخ جهش در مسائل بهینه‌سازی پیوسته) همراه هر کدام از موجودات هستند. این پارامترها خود تطبیق می‌باشند تا بهترین جهت جستجو و بیشترین اندازه گام برای هر کروموزوم در هر بعد تعیین شود.
6. پارامترهای جهش با توجه به برازش پاسخ‌ها به صورت تطبیقی تنظیم می‌شوند. برای پاسخ‌های برازنده، گام جهش کوتاه و نرخ جهش کم و برای پاسخ‌های با برازش پایین، گام جهش بلند و نرخ جهش بالا درنظر گرفته می‌شود.
7. احتمال جهش با توجه به یک تابع توزیع احتمال (مانند گوسین) محاسبه می‌شود.
8. از تابع هدف نسبی برای سنجش کارایی با توجه به گروهی از موجودات تصادفی انتخاب شده، استفاده می‌شود (تفاوت با استراتژی کامل).
9. با استفاده از روش انتخاب برای جایگزینی رقابتی، والدین و فرزندان برای بقاء با یکدیگر رقابت می‌کنند (تفاوت دیگر با استراتژی تکامل به دلیل عدم بهره‌مندی از روش‌های جایگزینی نسلی).
10. هر والد یک فرزند را ازطریق عملگر جهش ایجاد می‌کند (تفاوت دیگر با استراتژی تکامل).

Function EP (problem) returns a state that is a local maximum

Input : Populationsize , ProblemSize , BoutSize

Output : S_{best}

Population ← InitializePopulation (Populationsize , ProblemSize) ;

EvaluatePopulation (Population) ;

S_{best} ← GetBestSolution (Population) ;

while -StopCondition() do

  Children ← Ø ;

  for each Parent in Population do

    P_i ← GetParent (Population , i) ;

    S_i ← Ø ;

    $S_i$problem ← Mutate ($P_i$problem , $P_i$strategy) ;

```
    Si.strategy ← Mutate (Pi.strategy) ;
    Children ← Si ;
  end
  EvaluatePopulation (Children) ;
  Union ← Population + Children ;
  for each Si ϵ Union do
    for 1 to BoutSize do
      Sj ← RandomSelection (Union) ;
      if Fitness(Si) > Fitness(Sj) then
        Si.wins ← Si.wins + 1 ;
      end
    end
  end
  Population ← SelectBestByWins (Union , Populationsize) ;
  Sbest ← GetBestSolution (Population) ;
end
return Sbest ;
```

شکل (3-4) : شبه کد برنامه‌نویسی تکاملی

3-6 تکامل تفاضلی

تکامل تفاضلی یک استراتژی جستجوی تصادفی، مبتنی‌بر جمعیت می‌باشد که توسط استورن پرایس در سال 1995 پیشنهاد شده است. این الگوریتم دارای شباهت‌های بسیاری با سایر الگوریتم‌های تکاملی بوده، اما به‌طور اساسی ازجهت استفاده از اطلاعات جمعیت کنونی برای هدایت فرآیند جستجو با کلیه

روش‌های تکاملی متفاوت است. چراکه تکامل تفاضلی از عملگر انتخاب، به‌عنوان راه‌کار اعمال قانون بقاء اصلح داروین بهره نمی‌برد، بنابراین نمی‌توان ذات نسخه اولیه این الگوریتم را تکاملی دانست. لذا به‌دلیل بهره‌مندی تفاضل تکاملی از یک عملگر منحصربه‌فرد بازترکیب که مبتنی‌بر تفاوت پاسخ‌ها در جمعیت است و البته امکان بهره‌برداری از عملگر انتخاب، بدون نیازبه اعمال تغییرات بنیادین در تفاضل تکاملی استاندارد، این الگوریتم به‌عنوان یک روش تکاملی مورد بررسی قرار می‌گیرد.

با توجه به اهمیتی که در تکامل تفاضلی به تفاوت میان پاسخ‌ها در جمعیت داده می‌شود (از این تفاوت برانجام عملگر بازترکیب استفاده می‌شود)، می توان رفتار این الگوریتم را از این لحاظ نزدیک به روش بهینه‌سازی ازدحام ذرات به‌عنوان سرگروه الگوریتم‌های فرامکاشفه‌ای تقلید محور دانست.

تکامل تفاضلی بیشتر برای بهینه‌سازی توابع در فضاهای پیوسته به‌کار رفته است. به‌علاوه، برای آموزش شبکه‌های عصبی از تفاضل تکاملی استفاده شده است. تکامل تفاضلی برای حل بسیاری از مسائل دنیای واقعی همچون خوشه بندی، طراحی کنترل کننده، برنامه ریزی، تحلیل تصویر و طراحی فیلتر به‌کار برده شده است.

شبه‌کد الگوریتم تکامل تفاضلی در شکل (3-5) نشان داده شده است. خصوصیات مهم الگوریتم تکامل تفاضلی عبارتند از :

1. تکامل تفاضلی به‌دلیل انجام عملگر بازترکیب برروی کلیه اعضاء و نه فقط بهترین اعضاء، در مقایسه با سایر الگوریتم‌های تکاملی توانایی پویش بیشتری دارد.
2. برخلاف استراتژی‌های تکامل و برنامه‌نویسی تکاملی، در تکامل تفاضلی اندازه‌های گام از یک توزیع شناخته شده از قبل نمونه‌گیری نمی‌شوند.
3. اندازه‌های گام‌ها تحت تأثیر تفاوت بین موجودات جمعیت کنونی است.
4. میزان فاصله اولیه بین موجودات تحت تأثیر اندازه جمعیت است. هرچقدر که موجودات در یک جمعیت بیشتر باشند، فاصله بین آنها کمتر می‌باشد.
5. اندازه جمعیت دارای تأثیر مستقیم برروی قابلیت پویش تکامل تفاضلی است. هرچقدر که تعداد موجودات در جمعیت بیشتر باشند، بردار تفاضلی بیشتری دردسترس می‌باشد و جهت‌های بیشتری قابل جستجو هستند. درهرحال باید توجه داشت که پیچیدگی محاسباتی هر نسل با اندازه جمعیت افزایش می یابد.

6. فاصله بین موجودات، نمایش‌دهنده تنوع درجمعیت کنونی و اندازه‌های گامی است که باید برای رسیدن به یک نقطه برداشته شود. درصورتی‌که بین موجودات فاصله زیادی باشد، موجودات باید گام‌های بزرگی را برای پویش بهتر فضای جستجو بردارند. ازطرفی دیگر، اگر فاصله موجودات کم باشد، اندازه گام‌ها باید کوچک باشند تا قابلیت انتقاع در جستجوی محلی افزایش یابد.

Function DE (problem) returns a state that is a local maximum

Input : Populationsize , Problemsize , Weighting factor , Crossoverrate

Output : S_{best}

Population ← InitializePopulation (Populationsize , Problemsize) ;

EvaluatePopulation (Population) ;

S_{best} ← GetBestSolution (Population) ;

while - StopCondition () do

  NewPopulation ← Ø ;

  for each Pi Population do

    S_{ir} NewSample (P_i , Population , Problemsize , Weightingfactor , Crossoverrate) ;

    if Fitness(S_i) > Fitness(P_i) then

      NewPopulation ← S_i ;

    else

      NewPopulation ← P_i ;

    end

  end

  Population ← NewPopulation ;

  EvaluatePopulation (Population) ;

S_{best} ← GetBestSolution (Population) ;

end

return S_{best} ;

شکل (3-5) : شبه‌کد الگوریتم تکامل تفاضلی

در شبه‌کد الگوریتم شکل (3-6)، هرچقدر که مقدار β کوچک‌تر باشد، اندازه‌های گام جهش کوچک‌تر می‌شود و زمان بیشتری برای همگرایی الگوریتم مورد نیاز است. مقدار بزرگتر β قابلیت پویش را افزایش می‌دهد اما ممکن است که باعث شود الگوریتم از پاسخ به اندازه کافی خوب خارج شود. مقدار مناسب β باید به اندازه‌ای کوچک باشد که امکان پویش قله‌های تیز (دره‌های تنگ) را فراهم کند و به اندازه‌ای بزرگ باشد که تنوع جمعیتی ایجاد نماید. زمانی‌که اندازه جمعیت افزایش می‌یابد، بایستی مقدار β افزایش یابد.

هرچقدر که تعداد موجودات در جمعیت بیشتر باشند، مقدار بردارهای تفاضلی کوچک‌تر می‌شود و موجودات به‌یکدیگر نزدیکتر خواهند شد. بنابراین، می‌توان از گام‌های کوچک‌تری برای پویش مکان‌های محلی استفاده نمود. تعدد موجودات در جمعیت، نیاز به گام‌های جهش بزرگ‌تر را کاهش می‌دهد. نتایج تجربی نشان می‌دهد که مقادیر بزرگ برای β, n_s در شبه‌کد الگوریتم شکل (3-6)، اغلب منجربه همگرایی زودرس می‌شود. مقدار $\beta = 0.5$ معمولا کارایی خوبی دارد. لازم به توجه است که همواره $\beta \in (0, \infty)$ می‌باشد.

در شبه‌کد الگوریتم شکل (3-6)، احتمال تولیدمثل p_r ، دارای اثرمستقیم برتنوع در تکامل تفاضلی است. این پارامتر تعداد عناصری از والد $x_i(t)$ ، را که تغییر خواهند کرد، کنترل می‌کند. هرچقدر که احتمال تولیدمثل بیشتر باشد، تنوع بیشتری در جمعیت جدید ایجاد می‌شود که منجربه افزایش قابلیت پویش خواهد شد. افزایش p_r ، اغلب منجربه همگرایی سریع‌تر می‌شود، درحالی‌که کاهش آن قدرت جستجو را افزایش می‌دهد.

پارامترهای تفاضل تکاملی می توانند ثابت نبوده و در فرآیند جستجویی الگوریتم به صورت تطبیقی تغییر نمایند. به علاوه، امکان بهره‌برداری از جفت پاسخ‌های بیشتر درهرنسل (به‌جای تنها دو پاسخ P_{2i}, P_{1i} در الگوریتم برای پویش جهت‌های بیشتر در فضای جستجو وجود دارد.

Function NewSample () returns a new solution

Input : Po , Population , n_s , ß , Pr

Output : S

repeat

  $P_1 \leftarrow$ RandomMember (Population) ;

until $P_i + P_o$;

repeat

  $P_2 \leftarrow$ RandomMember (Population) ;

until $(P_2 \neq P_o)$ V $(P_2 \neq P_i)$;

repeat

  $P_3 \leftarrow$ RandomMember (Population) ;

until $(P_3 \neq P_o)$ V $(P_3 \neq P)$ V $(P_3 \neq P_2)$;

CutPoint $\leftarrow$ RandomPosition (n_s) ;

$S \leftarrow 0$;

for i to n_s do

  if i = CutPoint A Rand () < P_r then

    $S_i \leftarrow P_{3i} + \beta \times (P1_i - P2_i)$;

  else

    $S_i \leftarrow P_{0i}$;

  end

end

return S ;

شکل (3-6) : شبه کد تابع تولید پاسخ جدید

3-7 الگوریتم‌های ممتیک

همان‌طورکه در بخش‌های قبلی اشاره شد، ژن بخشی از اطلاعات زیست شناختی است که از یک نسل به نسل دیگر منتقل می‌شود. ژن‌ها مخشص‌کننده ویژگی‌های فیزیکی موجودات همچون چهره، شکل اندام و کلیه خصیصه‌هایی است که از والدین خود به ارث برده‌اند. مم (*Mem*)، با یک مقایسه ژنی در زمینه تکامل فرهنگی، توسط شخصی به نام داوکینز در سال 1979 معرفی گردید. مم یک عنصر فرهنگی یا رفتاری است که به‌وسیله عوامل غیرژنی از نسلی به نسل دیگر منتقل می‌شود. درحقیقت، مم دربرگیرنده هرخصیصه‌ای است که درطول زندگی یک موجود ازطریق تجربه و تقلید فراگرفته شده و در بین موجودات تکثیر می‌شود. تکثیری که هرگز ماهیتی ژنی نداشته و عملگرهای تولیدمثل برآن بی‌تأثیرند.

در تعریف لغوی، مم به بخشی از تمدن که ژن‌ها در به ارث رسیدنشان به نسل بعد نقشی ندارند، اطلاق می‌گردد. همان‌طور که زیست‌شناسان مم را واحد انتقال خصوصیات فیزیولوژیکی از قبیل رنگ چشم، رنگ مو و غیره از والدین به فرزندان می‌دانند، روانشناسان نیز مم را واحد انتقال خصوصیات رفتاری از قبیل تندخویی، سنت‌گرایی و غیره از والدین به فرزندان می‌شناسند. براساس دیدگاه روانشناسان، فردی که در یک خانواده بی‌سواد متولد می‌شود، لزومی ندارد که تا آخر عمر بی‌سواد باقی بماند و می-تواند با کسب برخی مهارت‌ها از محیط اطرافش ترقی پیدا کند، حال‌آن‌که زیست‌شناسان ژن‌های کروموزوم را از لحظه تولد تا مرگ ثابت و بدون تغییر می‌دانندi; پایه و اساس الگوریتم ممتیک یا MA (*Memetic Algorithm*) بر این اصل استوار است. برخلاف الگوریتم ژنتیک که کروموزوم را از لحظه تولد تا آخر عمر (حضور در فرآیند تولیدمثل برای نسل بعد) ثابت و بدون تغییر می‌داند، یک کروموزوم در الگوریتم ممتیک می‌تواند میزان شایستگی خود را در یک نسل با رفتاری موسوم به تقلید ارتقاء بخشد.

به‌عنوان نمونه‌هایی از مم می‌توان به الف) خصیصه‌ها و هنجارهای مثبت و منفی در یک جامعه که ریشه در حقیقتی فرهنگی دارد و در انتقال‌شان از مفاهیم ژنی بهره‌ای نمی‌برند، ب) مد لباس که آخرین مدهای لباس نتیجه ایده‌های طراحان مد هستند، ج) علوم که دانشمندان نظریات خود را بایکدیگر درمیان گذاشته و دانش را توسعه می‌دهند، د) ادبیات در به‌چاپ رسیدن داستان‌های‌های نویسندگان و اشعار

شاعران که موجب تقلیدی از یک سبک و یا اثر ادبی می‌شود و ی) موسیقی که علاوه‌بر موسیقیدانان، حتی پرندگان نیز از نوای موسیقی یکدیگر تقلید می‌کنند.

مهم‌ترین شباهت‌های بین ژن و مم این است که ژن‌ها به‌وسیله تولیدمثل از کروموزوم دیگر انتقال می‌یابند و ممها نیز به‌وسیله تقلید کردن از مغزی به مغز دیگر انتقال پیدا می‌کنند. همچنین در دوران تکامل ژنی، بهترین ژن‌ها و در سال‌های زندگی بهترین ممها باقی می‌مانند.

مهم‌ترین تفاوت‌های بین ژن‌ها و ممها را می توان در این موارد دانست که اولا مقادیر ژن‌ها از پیش مشخص‌شده هستند، به‌عنوان نمونه والدین سفیدپوست فرزند سفیدپوست خواهند داشت. این درحالی است که مقادیر ممها به‌هیچ عنوان تا قبل از زندگی موجود، قابل تخمین نخواهند بود. دوما، ژن‌ها به‌طور معمول در طی نسل‌های مختلف ثابت می‌مانند، درحالی‌که ممها در طول یک دوره زندگی مرتبا در حال تغییرند. سوما، ممها بهینگی و اصلاح را در نسل حاضر امکان‌پذیر می‌سازند، درحالی‌که این امکان در ژن‌ها درطی نسل‌های طولانی فراهم می‌شود، چراکه معجزه تکامل تنها با طی یک زمان بسیار طولانی رخ می دهد.

داوکینز معتقد بود که اصالت وجودی ممها از ژن‌ها به‌مراتب بیشتر است، زیرا ژن‌های یک موجود پس از مرگش ازبین می‌روند، به‌طورخاص اگر فرزند و یا خویشاوندی نداشته باشد. این درحالی است که مم-ها مرگی ندارند و از یک موجود به موجود دیگر همچون ویروس منتقل می‌شوند. درواقع، موجود میزبان حجم بزرگی از ممهای مثبت و منفی است. ممهایی که تجمیع آنها، به‌عنوان نمونه در یک انسان موجب فرهیختگی، متانت و وقار، خودخواهی و غرور، و یا سبک‌سری و خیره‌سری وی می‌شوند. در طول زندگی انسان، ممهایی توسط فرد یاد گرفته شده و یا تقلید می‌شوند. درهرحال کلیه این ممها توسط این شخص تکثیر شده و با مرگ وی نابود نمی‌شوند. دقیقا به همین دلیل است که با وجود مرگ بسیاری از انسان‌های فداکار، هنوز فداکاری وجود دارد و یا این‌که علی‌رغم مرگ بسیاری از مجرمان، هنوز جرم و جنایت ازجامعه بشری رخت برنبسته است.

دو چالش از مهم‌ترین مشکلات الگوریتم ژنتیک باعث شده است که ایده بهره‌مندی از مفهوم مم در کنار ژن مطرح شود. اولین چالش، این‌که الگوریتم‌های ژنتیک مانند بسیاری دیگر از فرامکاشفه‌های تکاملی در گام‌های نخست اجرا، ناحیه‌هایی از فضای حالت مسأله که بهینه‌های سراسری و محلی در آن واقع شده‌اند را به‌خوبی شناسایی می‌کنند، ولی در ادامه مسیرشان به سمت بهینه سراسری بسیار کند عمل می-

کنند. دومین چالش عمده‌ای که الگوریتم‌های ژنتیک با آن مواجه هستند، عدم پایداری آن‌ها می‌باشد. به‌این معنی که کیفیت جواب‌هایی که از اجراهای مختلف الگوریتم حاصل می‌گردند، ممکن است تفاوت‌های بسیاری داشته و حتی غیر قابل‌اعتماد باشند.

ازبین راه‌کارهای متعددی که برای برطرف ساختن مشکلات الگوریتم‌های جستجوی فرامکاشفه‌ای ارائه شده است، استراتژی ترکیب جایگاه ویژه‌ای را به خود اختصاص داده است که درآن، از به‌کارگیری روش‌های جستجوی سراسری و محلی در فرآیند حل مسأله نتیجه مطلوبی حاصل می‌گردد. الگوریتم‌های ممتیک، مشهورترین عضو این خانواده به‌شمار می‌آیند که از ترکیب الگوریتم‌های ژنتیک با یک مکاشفه جستجوی محلی حاصل می‌شوند.

ایده مربوط به استفاده از مفهوم مم برای طراحی یک الگوریتم فرامکاشفه‌ای، اولین بار توسط موسکاتو مطرح شد، به‌طوری‌که پیشنهاد نمود با استفاده از یک عملگر جستجوی محلی در بدنه الگوریتم ژنتیک و بعد از عملگر جهش، به هر موجود فرزند پس از تولدش، فرصتی برای زندگی داده شود. الگوریتم ممتیک مشابهت‌های بسیاری با الگوریتم ژنتیک دارد. تفاوت الگوریتم‌های ممتیک و ژنتیک در بهینه-سازی جمعیت هر نسل، پس از انجام عملگرهای بازترکیب و جهش می‌باشد. برای این منظور، به‌ازای هر فردی که در جامعه تولید می‌گردد، یک جستجوی محلی با شعاع همسایگی از پیش تعیین‌شده، حول کروموزوم مربوط به فرزند تولید شده، در فضای حالت مسأله انجام می‌پذیرد.

الگوریتم ممتیک همانند الگوریتم ژنتیک، برای حل بسیاری از مسائل بهینه‌سازی پیوسته و گسسته و نیز طیف وسیعی از مسائل دنیای واقعی به‌کار برده شده است. مهم‌ترین خصیصه الگوریتم ممتیک این است که به‌منظور افزایش سرعت و کارایی، جستجوی محلی برروی کلیه اعضاء در جمعیت انجام نمی‌شود. به‌همین دلیل معمولا جمعیت کوچکی از فرزندان برازنده، پس از عملگر جهش به عملگر جستجوی محلی معرفی می‌گردند.

الگوریتم‌های ممتیک به دو دسته مهم لامارکی و بالدوینی تقسیم می‌شوند. با فرض این‌که درحال حل یک مسأله بیشینه‌سازی باشیم، اگر کروموزوم Y نتیجه عملگر جستجوی محلی بر روی کروموزوم X باشد و داشته باشیم $\text{fitness}(Y)>\text{fitness}(X)$. در الگوریتم‌های ممتیک لامارکی، کروموزوم Y جایگزین کروموزوم X می‌شود، این درحالی است که درهمین شرایط، در الگوریتم‌های ممتیک بالدوینی، تنها برازش کروموزوم Y جایگزین برازش کروموزوم X می‌شود. درواقع، در الگوریتم ممتیک بالدوینی تنها

به عملگرهای تولیدمثل اجازه تغییر کروموزوم‌ها داده می‌شود. به‌علاوه، سرعت همگرایی در الگوریتم-های ممتیک لامارکی بیشتر از الگوریتم‌های ممتیک بالدوینی است، این درحالی است که قابلیت پویش الگوریتم‌های ممتیک بالدوینی بسیار بیشتر از نوع لامارکی است .

ویژگی خاص الگوریتم‌های ممتیک بالدوینی، عدم تغییر ژنوتایپ کروموزوم‌ها در مرحله زندگی (جستجوی محلی) الگوریتم است. این ویژگی باعث می‌شود که برازش نقاط همسایه بهینه محلی بهبود یابد و باعث اعمال تغییر در دورنمای برازش در این نقاط می‌شود. با این عمل، بخش‌هایی از برازش که ماهیتی خارپشتی دارند، هموارشده و پیمایش فضای جستجو برای الگوریتم ساده‌تر خواهد شد. در بسیاری از مقالات و کتب به ساده و هموارسازی دورنمای برازش توسط جستجوگر محلی مبتنی‌بر نظریه بالدوین، اثر بالدوین اطلاق می‌شود. می‌توان به‌جای جایگزینی پاسخ جدید در جستجوگر محلی طبق نظریه لامارک، و یا عدم جایگزینی پاسخ جدید طبق نظریه بالدوین، با یک احتمال ρ_l این جایگزینی را انجام داد. در این صورت درهنگامی که $\rho_l = 1$، الگوریتم ممتیک لامارکی و درهنگامی که $\rho_l = 0$، الگوریتم ممتیک بالدوینی حاصل می‌شود.

الگوریتم‌های ممتیک تطبیقی، به‌جای استفاده از تنها یک جستجوگر محلی ثابت، از مجموعه‌ای از روش-های جستجو بهره می‌گیرند. در این الگوریتم‌ها، با اعطای یک شایستگی به هر روش جستجوی محلی براساس میزان موفقیت آن در بهبود برازش کروموزوم‌ها، می‌توان با توجه به فضای حالت مسأله و نیاز الگوریتم در هر نسل، از یک روش مناسب جستجوی محلی بهره برد. در الگوریتم‌های ممتیک تطبیقی امکان یادگیری جستجوگرهای محلی پیشرفته نیز وجود دارد.

شبه‌کد الگوریتم ممتیک در شکل (3-7) نشان داده شده است.

Function MA (problem) returns a state that is a local optimum

Input : Populationsize , Problemsize , $P_{crossover}$, $P_{mutation}$, MemePopsize

Output : S_{best}

Population ← InitializePopulation (Populationsize , Problemsize) ;

EvaluatePopulation (Population) ;

S_{best} ← GetBestSolution (Population) ;

```
while –StopCondition () do
  Parents ← SelectParents (Population , Populationsize) ;
  Children ← Ø ;
  for each Parent1 , Parent2 ∈ Parents do
    Child1 , Child2 ← Crossover (Parent1 , Parent2 , Pcrossover) ;
    Children ← Mutate (Child1 , Pmutation) ;
    Children ← Mutate (Child2 , Pmutation) ;
  end
  EvaluatePopulation (Children) ;
  MemeticPopulation – SelectMemeticPopulation (Children , MemePopsize) ;
  for each Si ∈ Memetic Population do
    Si ← LocalSearch (Si) ;
  end
  Population ← Replace (Population , Children) ;
  Sbest ← GetBestSolution (Population) ;
end
return Sbest ;
```

شکل (3-7) : شبه‌کد الگوریتم ممتیک

3-8 الگوریتم‌های فرهنگی

الگوریتم‌های تکاملی پایه شامل GA، GP، ES و EP، درحل مسائل متنوع و پیچیده جستجو و بهینه‌سازی موفق عمل کرده‌اند. فرآیند جستجوی مورد استفاده دراین نوع الگوریتم‌ها، اغلب از دامنه اطلاعات مسأله

و یا دانش کسب‌شده حین فرآیند تکامل استفاده نمی‌کند. این کاملا منطقی است که کارایی الگوریتم‌های تکاملی پایه توسط به‌کارگیری اطلاعاتی درباره دورنمای برازش مسأله بهبود یابد. در واقع، به‌کارگیری دامنه اطلاعات مسأله و دانش کسب‌شده حین فرآیند تکامل می‌تواند باعث حرکت هوشمندانه‌تر الگوریتم در فضای جستجو دورنمای برازش شود.

تکامل فرهنگی، براساس ایده تکامل اجتماعی انسان توسط رینولدز در اوایل 1990 توسعه یافته است که درآن، فرآیند جستجو با استفاده از دانش اولیه درباره دامنه مسأله به‌همراه اطلاعاتی که در فرآیند تکاملی بدست می‌آید، هدایت می‌شود. در محاسبات تکاملی، از تکامل زیست‌شناختی تقلید می‌شود که براساس ایده وراثت ژنی می‌باشد. درسیستم‌های طبیعی، تکامل ژنی فرآیند کندی است. از طرفی دیگر، تکامل فرهنگی جوامع را توانا می‌سازد تا خود را با تغییرات محیط با سرعتی بیشتر از تکامل زیست شناختی، هماهنگ سازند. تعریف‌های گوناگونی برای فرهنگ می تواند وجود داشته باشند که به‌عنوان نمونه، به برخی از آن‌ها در ادامه اشاره شده است.

- سیستمی از پدیده‌های مفهومی و نمادی است که به‌صورت اجتماعی و تاریخی میان گروه‌های اجتماعی منتقل می‌شود.
- به ذخایری از دانش‌ها، تجارب، باورها، ارزش‌ها، افکار، مفاهیم، رهبران، مذاهب، نقش‌ها، روابط زمانی و فضایی، تدابیر جهان و اشیاء مادی و دارایی‌های بدست آمده از گروهی از مردم در طول یک نسل ازطریق تلاش گروهی و فردی، اطلاق می‌شود.
- کلیه رفتارهای یادگرفته‌شده گروهی ازمردم است که عموما به‌عنوان سنت آن مردم تلقی شده و از نسلی به نسل دیگر منتقل می‌شود.
- یک برنامه‌ریزی اجتماعی از تمایلات فکری است که اعضای یک گروه و یا دسته از مردم را از سایرین جدا می‌کند.

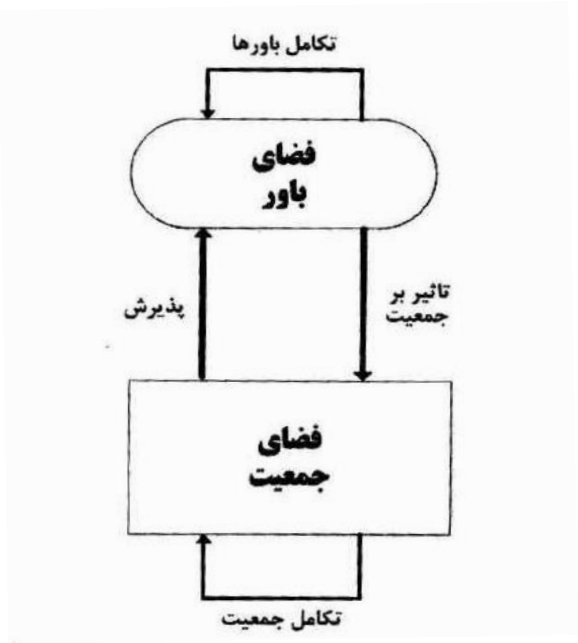
در محاسبات تکاملی، فرهنگ به‌عنوان منبعی از داده بوده که رفتار کلیه افراد دریک جمعیت را تحت تأثیر خود قرار می‌دهد. الگوریتم فرهنگی CA (*Cultural Algorithm*) با استراتژی تکامل و برنامه-نویسی ژنتیک درجایی‌که ویژگی‌های رفتاری هرموجود با استفاده از فنوتایپ آن موجود (تنها برای همان موجود و فقط در نسل کنونی) مدل می شود، متفاوت است. در الگوریتم فرهنگی، فرهنگ ویژگی‌های رفتار عمومی جمعیت را ذخیره می‌کند. اطلاعات فرهنگی برای کلیه موجودات جمعیت و در کلیه نسل‌ها

در دسترس می‌باشند. وجه مشترک اعضاء برازنده مبین پدیده‌های فرهنگی مثبت و وجه مشترک پاسخ-های کم‌ارزش، بیان‌گر حقایق فرهنگی منفی است.

یک الگوریتم فرهنگی، یک سیستم دو وراثتی بوده که دارای دو فضای جستجو می‌باشد شامل فضای جمعیت (برای نمایش مؤلفه‌های ژنی مبتنی‌بر ایده داروین) و فضای باور (برای نمایش مؤلفه‌های فرهنگی). فضای باور الگوریتم‌های فرهنگی را از سایر الگوریتم‌های تکاملی متمایز می‌سازد. فضای باور، اطلاعات فرهنگی درباره جمعیت را مدل می‌کند، درحالی‌که فضای جمعیت موجودات را در سطح ژنوتایپی و یا فنوتایپی نمایش می‌دهد. هر دو فضای جمعیت و باور به‌صورت موازی تکامل می‌یابند، درحالی‌که هر دو بریکدیگر اثر می‌گذارند. بنابراین، پروتکل‌های ارتباطی بخش جدایی ناپذیر الگوریتم-های تکاملی هستند. چنین پروتکل‌هایی دو مجرای ارتباطی را تعریف می‌کنند، یکی برای انتخاب گروهی از موجودات برای تطبیق‌پذیری مجموعه باورها و دیگری برای تعریف راه‌هایی که باورها بر روی کلیه موجودات در فضای جمعیت اثر می گذارند.

همان‌طور که بیان شد، الگوریتم‌های فرهنگی از یک فضای باور برای نشان‌دادن فرهنگ درحال تکامل موجودات در فضای جمعیت استفاده می‌کنند. فضای باور همانند یک انبار دانش عمل می‌کند که رفتارهای ذخیره‌شده (یا باورها) موجودات را در جمعیت ذخیره می‌کند. این فضا به استخر مم نیز معروف است. مم‌ها، واحد های اطلاعاتی منتقل‌شونده توسط مفاهیم رفتاری است. فضای باور به‌عنوان یک انبار دانش از ویژگی‌ها رفتاری به‌کار می‌رود. مم‌ها در فضای باور، تعمیم تجارب افراد در فضای جمعیت بوده که در طول چندین نسل جمع‌آوری شده و شکل می‌گیرند و تنها برای یک نسل نیستند.

در شکل (3-8)، فضاهای باور و جمعیت الگوریتم‌های فرهنگی به‌همراه چگونگی ارتباطشان نشان داده شده است. درهر تکرار (نسل)، موجودات ابتدا با استفاده از تابع برازندگی در سطح جمعیت ارزیابی می‌شوند. سپس، تابع پذیرش برای تعیین موجوداتی از جمعیت کنونی که بر روی باورهای کنونی تاثیر می‌گذارند، به‌کار گرفته می‌شود (اغلب برازنده‌ترین موجودات برای تغییر فضای باور استفاده می‌شوند). در پایان، تجربه موجودات پذیرفته شده جهت تنظیم باورها (برای شبیه‌سازی تکامل در فرهنگ) مورد استفاده قرار می‌گیرند.



شکل (3-8) : ارتباط فضای‌های باور و جمعیت در الگوریتم‌های فرهنگی

از باورهای تنظیم‌شده برای تاثیر برروی جمعیت بهره‌برداری می‌شود. عملگرهای تولیدمثل (بازترکیب و جهش) از باورها برای کنترل تغییرات موجودات استفاده می‌کنند. این کار اغلب ازطریق توجه به برخی مقادیر برای ژن‌ها (مقادیر شایع در اعضاء برتر جمعیت تکاملی که بیان‌گر فرهنگ مثبت هستند) و با تنظیم تطبیقی پارامتهای کنترلی، همانند استراتژی‌های تکاملی و برنامه‌نویسی ژنتیک، حاصل می‌شود. در ادامه، در مورد چگونگی تاثیر فرهنگ بر روی شکل‌گیری اعضاء جدید (فرزندان) در جمعیت تکاملی بحث خواهد شد. برخی از انواع پروتکل‌های ارتباطی برای انتقال اطلاعات بین فضاهای باور و جمعیت به‌کار می‌روند.

این پروتکل‌های ارتباطی، عملگرهایی که اثر موجودات بر روی ساختار فضای باور را کنترل می‌کنند، تعیین می‌کنند. درعین‌حال، پروتکل‌های ارتباطی وظیفه تعیین عملگرهایی را دارند که تاثیر فضای باور بر فرآیند تکامل را در جمعیت اعضاء مدیریت می‌کنند. از اطلاعات فرهنگی ذخیره‌شده در فضای باور، برای هدایت تکامل جمعیت به‌سمت مکان‌های بهتر در فضای جستجو (دورنمای برازش) استفاده می‌شود. نشان داده شده است که استفاده از فضای باور به‌طور اساسی، بار محاسباتی را به شکل قابل‌توجهی کاهش می‌دهد.

الگوریتم‌های فرهنگی برای حل شماری از مسائل به‌کار گرفته شده‌اند که برخی از آن‌ها عبارتند از یادگیری مفاهیم، ساخت درخت‌های تصمیم، بهینه‌سازی توابع با مقادیرحقیقی، بهینه‌سازی شبکه‌های معنایی، آزمون نرم‌افزار، ارزیابی کیفیت برنامه‌نویسی ژنتیک، داده‌کاوی، بخش‌بندی تصویر و روباتیک. شبکه‌کد الگوریتم فرهنگی در شکل (3-9) نشان داده شده است. خصوصیات مهم الگوریتم‌های فرهنگی عبارتند از :

1. برای انتخاب گروهی از موجودات برای تطبیق‌پذیری مجموعه باورها از تابع AcceptBelief استفاده می‌شود و برای اعمال تاثیر باورها بر روی کلیه موجودات در فضای جمعیت از تابع Reproduce WithInfluence بهره‌برداری می‌گردد.
2. انواع مختلفی از الگوریتم‌های فرهنگی توسعه یافته است که در ساختمان داده‌های مورد استفاده در مدل کردن فضای باور با یکدیگر متفاوت هستند.
3. فضای باور شامل مجموعه‌های باور است. مجموعه باور شامل تعدادی مولفه دانش می‌باشد تا الگوهای رفتاری موجودات در فضای جمعیت را نشان دهد. نوع مولفه‌های دانش و ساختمان داده-های مورد استفاده برای نمایش دانش، بستگی به مسأله مورد بررسی دارد.
4. مولفه دانش موقعیتی، شامل بهترین راه‌حل‌های پیداشده در فرآیند تکاملی و یا وجه مشترک پاسخ-های برازنده می‌باشد.
5. مولفه‌های دانش هنجار،استانداردهایی برای رفتار موجودات فراهم کرده که برای هدایت تنظیمات جهشی مانند گام جهش، احتمال جهش و غیره، در بخش تکامل جمعیت الگوریتم استفاده می‌شود. درمورد مسأله بهینه‌سازی تابع، مولفه دانش هنجار مجموعه‌ای از فواصل را نگهداری می‌کند که هر کدام مربوط به بخش‌های متفاوتی از دورنمای برازش هستند.
6. بدیهی است که احتمال جهش برای مکان‌هایی از دورنمای برازش که خارپشتی‌تر هستند، باید بیشتر از فضاهای هموارتر تکامل یابد. این فواصل گستره آن‌چه‌که گمان می‌رود برای جستجو در هر دامنه مناسب‌تر است را مشخص می‌کنند.
7. اگر در مجموعه باور تنها از مولفه‌های دانش موقعیتی و هنجار استفاده شود، فضای باور به-صورت دوتایی $B(t) = (S(t),N(t))$ نشان داده می‌شود، به‌طوری‌که $S(t)$ نشان‌دهنده مولفه دانش موقعیتی و $N(t)$ بیان‌گر مولفه دانش هنجار می‌باشد.

Function CA (problem) returns a state that is a local optimum

Input : Problemsize , Populationnum

Output : S_{best}

Population ← InitializePopulation (Problemsize , Populationnum) ;

BeliefSpace ← InitializeBeliefSpace (Problemsize , Populationnum) ;

S_{best} ← GetBestSolution (Population) ;

While -StopCondition (0) do

  EvaluatePopulation (Population) ;

  Children ← ReproduceWithInfluence (Population , BeliefSpace) ;

  Population ← Select (Children , Population) ;

  $Belief_{candidate}$ ← AcceptBelief (Population) ;

  UpdateBeliefSpace (BeliefSpace , Beliefcandidate) ;

  S_{best} ← GetBestSolution (Population) ;

end

return S_{best} ;

شکل (3-9) : شبه‌کد الگوریتم فرهنگی

با توجه‌به نیاز مسأله، در مجموعه باور امکان استفاده از مولفه‌های دانش دیگری نیز وجود دارد، به‌عنوان نمونه، در مسائلی که ممکن است دورنمای برازش طی زمان تغییر کند (مسائل بهینه‌سازی با دورنمای برازش پویا)، از یک مولفه دانش به‌نام مولف دانش سابقه استفاده می‌شود. این مولفه، اطلاعاتی درباره دنباله تغییرات محیط را نگهداری می‌کند. از این مولفه برای پیش‌بینی تغییرات آینده در فضای جستجوی مسأله جهت اخذ بهترین تصمیم استفاده می‌شود. هر پاسخ یا موجود در جمعیت یک تصمیم است. نوع مولفه‌های دانش و روشی که دانش نمایش داده می‌شود، بر روی توابع تاثیر و پذیرش، تاثیر می‌گذارد.

جمعیت کنونی، برای شکل‌گیری باورها استفاده خواهد شد. روش‌های ایستا، از رتبه‌بندی مطلق مبتنی‌بر مقدار برازش استفاده می‌کنند و $n\%$ از موجودات برتر را انتخاب می‌کنند. در این‌جا می‌توان ازهر کدام از مکانیزم‌های انتخاب در الگوریتم‌های تکاملی استفاده نمود، به‌عنوان نمونه انتخاب مسابقه‌ای یا انتخاب چرخ رولت که در آن‌ها تعداد موجودات انتخاب‌شده برابر با موجودات اولیه می‌باشد. روش‌های پویا تعداد ثابتی موجود برای تنظیم فضای باور ندارند و تعداد موجودات ممکن است که از نسلی به نسلی

دیگر تغییر کند. در این‌جا می‌توان از رتبه‌بندی نسبی استفاده کرد، به‌عنوان نمونه در آن موجوداتی که دارای کارایی بیشتر از مقدار میانگین هستند، انتخاب شوند.

تعداد موجوداتی که برای تنظیم فضای باور مورد استفاده قرار می‌گیرند، در ابتدا بزرگ بوده و در طول زمان به‌صورت نمایی کاهش می‌یابد. روش‌های تطبیق‌ناپذیر از اطلاعات فضای جستجو و فرآیند تکاملی الگوریتم برای تنظیم تعداد موجوداتی که باید انتخاب شوند، استفاده می‌کنند. رینولدز و چانگ، یک تابع پذیرش فازی را برای تعیین تعداد موجودات که مبتنی‌بر شماره نسل و نرخ موفقیت موجودات بود، پیشنهاد نمودند.

هنگامی‌که الگوریتم در تنظیم مولفه دانش هنجار، گام‌های جهش را کوچک‌تر می‌کند، یک شیوه محافظه-کارانه به‌کار می‌رود. بدین ترتیب، قابلیت پویش در الگوریتم تضعیف شده و انتفاع در آن تقویت می-گردد. باورها برای تنظیم موجودات در فضای جمعیت برای تطبیق بیشتر با عقاید عمومی استفاده می-شوند. تنظیمات از طریق تابع تاثیر (*Reproduce WithInfluence*) در شبه‌کد الگوریتم شکل (3-9) تحقق می‌پذیرد. برای درک این فرآیند، فرض کنید که EP به‌عنوان فرآیند جستجو در فضای جمعیت مورد استفاده قرار می‌گیرد. الگوریتم حاصل CAEP نامیده می‌شود. فضای باور برای تعیین اندازه‌های گام جهش در EP و جهت تغییرات (یعنی اندازه‌گیری گام اضافه یا کم می‌شود) استفاده می‌شود.

3-9 الگوریتم ژنتیک تاگوچی

یکی از مشکلات الگوریتم ژنتیک استاندارد، عدم ثابت بودن نتایج این الگوریتم در اجراهای مختلف آن می‌باشد و به‌هیچ‌عنوان نتایج پایداری را به‌دست نمی‌دهد. یکی از راه‌کارهایی که برای بهبود این مشکل الگوریتم ژنتیک استاندارد ارائه شده است، استفاده از جستجوگر محلی در بدنه این الگوریتم می‌باشد (الگوریتم ممتیک). استفاده از روش تاگوچی راه‌کار دیگری است که موجب پایدارتر شدن الگوریتم ژنتیک استاندارد را فراهم می‌سازد. در روش تاگوچی سعی می‌شود که تاثیر هرکدام از ژن‌ها درنتیجه برازش نهایی یک کروموزوم سنجیده شود. در این روش، مقدار مناسب برای یک ژن از میان مقادیر ژن‌های دو یا چند کروموزوم پایه یافته شده و از آن در تولید فرزندان بهتر استفاده می‌شود. این کار به ازای کلیه ژن‌های کروموزوم‌ها تکرارشده و ژنوتایپ فرزندان ساخته نمی‌گردد.

در روش تاگوچی از مفهوم آرایه متعامد برای تولید مجموعه کروموزوم‌های پایه استفاده می‌شود. جداول (3-1) و (3-2)، یک آرایه متعامد دوبعدی را به ترتیب با 7 و 15 درایه ورودی نشان می‌دهند. جدول (3-3) بیان‌گر یک آرایه متعامد سه‌بعدی و با 4 درایه ورودی است. لازم به ذکر است که آرایه‌های متعامد، مجموعه‌ای از کروموزوم‌هایی را نشان می‌دهند (هرستون در آرایه متعامد بیان‌گر یک کروموزوم است) که به‌طور یکنواخت در یک دورنمای برازش چندبعدی پراکنده شده‌اند. بنابراین، به نوعی یک نمونه‌برداری مناسب از فضای جستجو توسط این کروموزوم‌ها حاصل می‌شود. با داشتن برازش این کروموزوم‌های پایه می‌توان درمورد اثر مقادیر ژن‌های مختلف در ایجاد یک برازش بهتر نتیجه‌گیری نمود. این به‌طور دقیق همان کاری است که روش تاگوچی با انجام آن باعث تقویت الگوریتم ژنتیک استاندارد می‌شود.

جدول (3-1) : آرایه متعامد دوبعدی با 7 ورودی

تعداد آزمایش	فاکتورها						
	A	B	C	D	E	F	G
	تعداد ستون						
	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

جدول (3-2) : آرایه متعامد دوبعدی با 15 ورودی

تعداد آزمایش	تعداد ستون														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
3	1	1	1	2	2	2	2	1	1	1	1	2	2	2	2

4	1	1	1	2	2	2	2	2	2	2	2	1	1	1	1
5	1	2	2	1	1	2	2	1	1	2	2	1	1	2	2
6	1	2	2	1	1	2	2	2	2	1	1	2	2	1	1
7	1	2	2	2	2	1	1	1	1	2	2	2	2	1	1
8	1	2	2	2	2	1	1	2	2	1	1	1	1	2	2
9	2	1	2	1	2	1	2	1	2	1	2	2	1	1	2
10	2	1	2	1	2	1	2	2	1	2	1	1	2	2	1
11	2	1	2	2	1	2	1	1	2	1	2	2	1	2	1
12	2	1	2	2	1	2	1	2	1	2	1	1	2	1	2
13	2	2	1	1	2	2	1	1	2	2	1	1	2	2	1
14	2	2	1	1	2	2	1	2	1	1	2	2	1	1	2
15	2	2	1	2	1	1	2	1	2	2	1	2	1	1	2
16	2	2	1	2	1	1	2	2	1	1	2	1	2	2	1

جدول (3-3) : آرایه متعامد سه‌بعدی با 4 ورودی

تعداد آزمایش	**تعداد ستون**			
	1	**2**	**3**	**4**
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

شبه‌کد الگوریتم ژنتیک تاگوچی (*TGA*) در شکل (3-10) ارائه شده است.

Function TGA (problem) returns a state that is a local optimum

Input : Populationsize , Problemsize , $P_{crossover}$, $P_{mutation}$

Output : S_{best}

Population ← InitializePopulation (Populationsize , Problemsize) ;

```
EvaluatePopulation (Population) ;
S_best ← GetBestSolution (Population) ;
While-StopCondition () do
  Parents ← SelectParents (Population , Populationsize) ;
  Children ← Ø ;
  for each Parent_1 , Parent_2 ∈ Parents do
    Child_1 , Child_2 ← Crossover (Parent_1 , Parent_2 , P_crossover) ;
    Child_3 ← Taguchi-Method (Child_1 , Child_2) ;
    Children ← Mutate (Child_3 , P_mutation) ;
  end
  EvaluatePopulation (Children) ;
  Population ← Replace (Population , Children) ;
  S_best ← GetBestSolution (Population) ;
end
return S_best ;
```

شکل (3-10) : شبه‌کد الگوریتم ژنتیک تاگوچی

همان‌طور که در شبه‌کد الگوریتم ژنتیک تاگوچی شکل (3-10) نشان داده شده است، از روش تاگوچی بین دو عملگر بازترکیب و جهش استفاده می‌شود. از هر دو فرزند تولیدشده، یک فرزند سوم به‌وجود می‌آید. درادامه، فرزند سوم به عملگر جهش ارائه می‌شود. نمونه مثال زیر، برای درک بهتر نحوه تولید فرزند سوم را نشان می‌دهد. فرض کنید دو فرزند تولید شده توسط عملگر بازترکیب به ترتیب C_1 و C_2 باشند. کروموزوم‌های این فرزندان در زیر نشان داده شده‌اند.

Choromosome C_1 : 1.0 , 1.0 , 1.0 , 1.0 ,0.0 , 0.0 , 0.0

Choromosome C_2 : 0.0 , 0.0 , 0.0 , 0.0 , 1.0 , 1.0 , 1.0

حال اگر فرض شود که هدف کمینه‌سازی تابع y_i باشد، بنابراین سعی خواهد شد که تابع $\eta_i = 1/y_i$ مقدار بیشینه داشته باشد. با توجه به یک آرایه متعامد دوبعدی با 7 ورودی (براساس تعداد ژن‌های کروموزوم های C_1 و C_2 چگونگی تولید ژن‌های فرزند سوم (کلیه ژن های این فرزند برابر با 0.0 می-باشند) در شکل (3-11) نشان داده شده است. مشاهده می‌شود که برای تعیین مقدار هرکدام از ژن‌ها در فرزند سوم، به ژن مشابه در فرزند اول و دوم رجوع می‌گردد. از میان دو مقدار موجود در دو فرزند

اول و دوم مقداری انتخاب می‌شود که حاصل جمع برازش تابع η_i برای آن مقدار بیشتر باشد. بنابراین، همان‌طورکه در شکل (3-11) مشخص شده است، مقدار چهار ژن نخست فرزند سوم از C_2 گرفته شده و مقدار سه ژن آخر این فرزند هم از C_1 کپی شده است.

Experiment Number (i)	Factors (f) A	B	C	D	E	F	G	Function value	η_i
	Column number								
	1	2	3	4	5	6	7		
1	1.0	1.0	1.0	1.0	0.0	0.0	0.0	4.0	1/4.0
2	1.0	1.0	1.0	0.0	1.0	1.0	1.0	6.0	1/6.0
3	1.0	0.0	0.0	1.0	0.0	1.0	1.0	4.0	1/4.0
4	1.0	0.0	0.0	0.0	1.0	0.0	0.0	2.0	1/2.0
5	0.0	1.0	0.0	1.0	1.0	0.0	1.0	4.0	1/4.0
6	0.0	1.0	0.0	0.0	0.0	1.0	0.0	2.0	1/2.0
7	0.0	0.0	1.0	1.0	1.0	1.0	0.0	4.0	1/4.0
8	0.0	0.0	1.0	0.0	0.0	0.0	1.0	2.0	1/2.0
E'_{f1}	1.167*	1.167*	1.167	1	1.5	1.5	1.5		
E_{f2}	1.5*	1.5*	1.5	1.667	1.167	1.167	1.167		
Optimal level	2*	2*	2	2	1	1	1		
Optimal chromosome	0.0	0.0	0.0	0.0	0.0	0.0	0.0		

شکل (3-11) : چگونگی تولید فرزند سوم از دو فرزند محصول عملگر بازترکیب

الگوریتم ژنتیک تاگوچی در مقایسه با الگوریتم ژنتیک استاندارد مقاوم‌تر بوده و در اجراهای متعدد، واریانس بهترین پاسخ آن از الگوریتم ژنتیک استاندارد به مراتب کمتر است. بهترین پاسخ در الگوریتم ژنتیک تاگوچی با چندین اجرا تفاوت زیادی نمی‌کند و شامل مجموعه کوچکی از پاسخ‌های برازنده می‌شود.

3-10 الگوریتم هم‌تکاملی

در الگوریتم‌های تکاملی پایه، تکامل جمعیت اغلب در یک محیط ثابت فیزیکی درنظر گرفته می‌شود. از آن‌جاکه در تکامل طبیعی، یک محیط فیزیکی تحت تاثیر سایر جمعیت‌های زیست‌شناختی است که مستقل عمل می‌کنند، به‌همین دلیل در الگوریتم هم‌تکاملی (*CoEA*) تکامل به‌صورت محلی در یک جمعیت نمی-

باشد، بلکه در پاسخ به تغییرات محیط که توسط سایر جمعیت‌ها ایجاد می‌شود، صورت می‌پذیرد. تفاوت دیگر بین الگوریتم‌های تکاملی پایه و هم‌تکاملی این است که در الگوریتم‌های تکاملی، مفهوم بهینگی از طریق یک تابع برازش مطلق تعریف می‌شود، این‌درحالی است که در الگوریتم‌های هم‌تکاملی برای تعریف بهینگی از یک تابع برازش نسبی استفاده می‌شود. درواقع، الگوریتم‌های هم‌تکاملی سعی دارند که گونه‌های بهینه را با توجه به تاثیر موجودات بر یکدیگر تکامل دهند.

هم‌تکاملی درواقع تکامل مکمل گونه‌های وابسته به یکدیگر است. به تکاملی بین دو گونه با نمونه زیبایی از هالند، که درباره تعامل رقابتی بین یک گیاه و حشرات است، توضیح داده می‌شود. یک نوع خاصی از یک گیاه را درنظر بگیرید که در محیطی قرار دارد که حشراتی از آن تغذیه می‌کنند. بازی بقاء دارای دو قسمت می باشد. 1- برای نجات، گیاه نیاز به تکامل مکانیزمی برای دفاع از خود در مقابل حشرات دارد، 2- حشرات نیاز به منبع غذایی برای بقاء دارند. گیاه و حشره هر دو تکامل می‌یابند تا ویژگی‌های موردنیاز برای بقاء را به‌دست آورند. به‌عنوان نمونه، ممکن است سطح خارجی گیاه ضخیم-تر شود، اما از طرفی آرواره‌های حشره نیز قوی‌تر می‌شود. ممکن است گیاه دارای سمی شود که حشره را بکشد. در نسل بعدی، حشره آنزیمی تولید خواهد کرد که این سم را هضم می‌کند. تاثیر این فرآیند هم-تکاملی در هرنسل، بهترشدن گیاه و حشره هم از نظر تدافعی و هم تهاجمی می‌باشد. در نسل بعدی، هر حشره با توجه به عکس‌العمل سایر حشرات در نسل قبلی تغییر می‌کند.

مثال زیست‌شناختی توضیح داده شده در بالا، نمونه‌ای از هم‌تکاملی رقابتی (شکارچی - شکار) می‌باشد، به‌گونه‌ای که رابطه برازش معکوس بین این دو گونه وجود دارد. برنده شدن برای یک موجود به معنای بازنده شدن دیگری است. برای بقاء موجودات زنده خود را با موجود برنده تطبیق می‌دهند تا برنده شوند. در طول این فرآیند، پیچیدگی شکار و شکارچی هر دو افزایش می‌یابد.

نوع دیگری از هم‌تکاملی، همزیستی می‌باشد، به‌طوری‌که گونه‌های مختلف موجودات به‌جای رقابت، با یکدیگر همکاری می‌کنند. دراین مورد موفقیت یک موجود قدرت بقاء سایر موجودات را افزایش می‌دهد. هم‌تکاملی همزیستی، از طریق بازخورد مثبت میان گونه‌هایی که در فرآیند همکاری شرکت کرده‌اند، حاصل می‌شود. بنابراین، دو دسته از شیوه‌های هم‌تکاملی شامل هم‌تکاملی رقابتی و هم‌تکاملی همزیستی وجود داشته که برای هر کدام از این دسته‌ها، فیوکودا و کیوباتا زیردسته‌هایی را تعریف نموده‌اند. برای الگوریتم‌های هم‌تکاملی رقابتی، زیردسته‌های زیر را می‌توان اشاره نمود.

- رقابت دوطرفه : ازهر دو موجود جلوگیری می‌شود، به‌علت وجود رابطه معکوس بین برازندگی دو موجود، موفقیت در یک گونه به‌عنوان شکست دیگری محسوب می‌شود (رقابت برد-باخت).
- رقابت یک‌طرفه : یک گونه جلوگیری می‌شود و سایر موجودات تاثیر نمی‌پذیرند (رقابت برد-مساوی).

برای الگوریتم‌های هم‌تکاملی هم‌زیستی یا همکاری، زیردسته‌های زیر را می‌توان اشاره نمود.

- همیاری : هر دو گونه سود می‌برند. تعامل برازندگی مثبت منجربه بهبود یک موجود می‌شود، هرزمانی‌که سایر موجودها بهبود یابند (همکاری برد-برد).
- همیاری یک‌طرفه : تنها یک موجود سود می‌برد و سایر موجودات متاثر نمی‌شوند (همکاری برد-مساوی).
- انگلی : تنها یک موجود (انگل) سود می‌برد، درحالی‌که سایر گونه‌ها (میزبان) زیان می‌بینند (همکاری برد-باخت).

درادامه، بیشتر بر هم‌تکاملی رقابتی دوطرفه (رقابت برد-باخت) و همیاری (همکاری برد-برد) تمرکز خواهد شد.

3-10-1 هم‌تکاملی رقابتی

یک الگوریتم هم‌تکاملی رقابتی اغلب دو جمعیت را به‌صورت همزمان تکامل می‌دهد. موجودات در یک جمعیت راه‌حل‌های مسأله را نشان می‌دهند، درحالی‌که در جمعیت دیگر نمونه‌های آزمایشی (مجموعه آزمون) نشان داده می‌شوند. موجودات در جمعیت راه‌حل، برای حل تا حدامکان تعداد بیشتری نمونه آزمایشی، تکامل می‌یابند، درحالی‌که موجودات در مجموعه آزمون، در جهت افزایش سطح پیچیدگی موجودات راه‌حل، تکامل پیدا می‌کنند. برازندگی موجودات در جمعیت راه حل متناسب با تعداد نمونه‌های آزمایشی حل‌شده توسط راه‌حل‌ها می‌باشد. برازندگی هر موجود در جمعیت آزمایش نسبت معکوس با تعداد پاسخ‌های حل‌کننده آن موجود دارد.

هم‌تکاملی رقابتی با دو جمعیت توسط هیلیس ارائه شده است. هالند هم‌تکاملی رقابتی را با الگوریتم ژنتیک تک جمعیتی ترکیب نمود، به‌صورتی‌که موجودات با یکدیگر در یک جمعیت یکسان رقابت می-

کنند. در هم‌تکاملی رقابتی، امکان استفاده از دو جمعیت راه‌حل رقیب نیز وجود دارد. دراین صورت، موجودات در یک جمعیت به‌عنوان موجودات آزمایشی در جمعیت دیگر درنظر گرفته می‌شوند. دراین‌جا راه‌حل برگردانده شده توسط الگوریتم، بهترین موجودات در هر دو جمعیت خواهد بود.

در الگوریتم‌های تکاملی پایه، از تابع برازندگی مطلق تعریف شده توسط کاربر استفاده می‌شود تا کیفیت راه‌حل‌ها سنجیده شود. این تابع برازش مطلق مستقیم مسأله بهینه‌سازی را نمایش می‌دهد. برازندگی هر موجود با استفاده از این تابع برازش به‌طور مستقل از سایر موجودات یا هر جمعیت دیگری ارزیابی می‌شود. همان‌طورکه بیان شد، فرآیند تکامل در الگوریتم‌های هم‌تکاملی ازطریق تابع برازش نسبی هدایت می‌شود، به‌طوری‌که که کارایی موجودات در یک جمعیت با توجه به سایر موجودات در جمعیت-های دیگر محاسبه می‌گردد. برای محاسبه برازش نسبی در هم‌تکاملی رقابتی، تعداد رقیبان شکست‌خورده توسط موجود، به‌عنوان امتیاز محسوب می‌شود. درواقع، دراین‌جا هیچ‌گونه تابع برازشی مورد استفاده قرار نمی‌گیرد که شایستگی مطلق هر پاسخ را توصیف کند و تنها موجوداتی که بهتر عمل کنند، نسبت‌به سایرین برازنده‌تر هستند.

به‌منظور محاسبه برازش نسبی موجودات در روش هم‌تکاملی رقابتی، دو جنبه مهم زیر وجود دارد :

1. نمونه‌گیری برازش : کدام‌یک از موجودات از جعیت رقیب مورد استفاده قرار می‌گیرند؟
2. ارزیابی برازش نسبی : به‌طوردقیق چگونه این موجودات رقیب به‌کار گرفته می‌شوند تا برازش نسبی را محاسبه کنند؟

درادامه، ابتدا انواع روش‌های نمونه‌گیری برازش و ارزیابی برازش نسبی در الگوریتم‌های هم‌تکاملی رقابتی تشریح شده وسپس، به بررسی انواع الگوریتم‌های هم‌تکاملی رقابتی پرداخته می‌شود.

- **نمونه گیری برازش**

برازش نسبی موجودات، ازطریق نمونه‌ای از موجودات جمعیت رقیب ارزیابی می‌شود. روش‌های نمونه‌گیری عبارتند از :

1. نمونه‌گیری همه درمقابل همه : هرموجود درمقابل همه موجودات سایرجمعیت‌ها ارزیابی می‌شود.

2. نمونه‌گیری تصادفی : برازندگی هرموجود با گروهی از موجودات تصادفی انتخاب‌شده از سایر جمعیت‌ها ارزیابی می‌شود. گروه نمونه‌گیری شده می‌تواند شامل یک یا بیش از یک موجود باشد. نمونه‌گیری تصادفی ازنظر محاسباتی پیچیدگی کمتری نسبت‌به نمونه‌گیری همه موجودات دارد.
3. نمونه گیری مسابقه‌ای : از معیارهای برازندگی که نسبی است، برای انتخاب بهترین موجودات رقیب استفاده می‌کنند.
4. همه درمقابل نمونه‌گیری بهترین : همه موجودات درمقابل بهترین موجودات سایر جمعیت‌ها ارزیابی می‌شوند.
5. نمونه‌گیری اشتراکی : نمونه‌هایی انتخاب می‌شوند که تعداد بیشتری از موجودات جمعیت رقیب را شکست می‌دهند.

- **ارزیابی برازش نسبی**

فرض کنید که دو جمعیت C_1 و C_2 هم‌تکامل می‌باشند وهدف، محاسبه برازش نسبی هرموجود X_1 در جمعیت C_1 می‌باشد. روش‌های زیر می‌تواند برای اندازه‌گیری برازش نسبی هرکدام ازموجودات درجمعیت به‌کار رود.

1. برازش ساده : ابتدا از جمعیت C_2 نمونه‌گیری می‌شود و تعداد موجوداتی که موجود X_i و C_1 در جمعیت نمونه‌گیری شده C_2 برآن‌ها غلبه می‌کند، شمارش می‌شود. برازش نسبی موجود X_i در C_1 به‌سادگی با محاسبه مجموع پیروزی‌های آن به‌دست می‌آید.
2. اشتراک برازش : از یک تابع اشتراک برای درنظرگرفتن شباهت‌ها درمیان موجودات جمعیت C_1 استفاده می‌شود. درتابع اشتراک، برازش ساده موجود X_i در C_1 بر مجموع شباهت‌هایش با کلیه موجودات دیگر درجمعیت تقسیم می‌شود. شباهت را می‌توان تعداد موجوداتی از جمعیت C_1 در نظر گرفت که بر کلیه نمونه موجوداتی در جمعیت C_2، که موجود X_i در C_1 برآن‌ها غلبه می‌کند، غلبه نمایند.
3. برازش مسابقه‌ای : عددی که مربوط به یک مسابقه تک حذفی و دودویی است، برای تعیین برازش نسبی که مبتنی‌بر رتبه موجود درجمعیت است، محاسبه می‌شود. برازش مسابقه‌ای درخت مسابقه را تولید می‌کند که در ریشه آن بهترین موجود قرار دارد. برای هرسطح درخت، دو رقیب به‌صورت تصادفی از آن سطح انتخاب می‌شوند و بهترین آن‌ها به سطح بعدی پیشروی می‌کند (مشابه

مسابقات جام حذفی)، درصورتی‌که تعداد رقیبان فرد باشد، یک موجود ازسطح کنونی به سطح بعدی می‌رود. بعد از رتبه‌بندی مسابقه‌ای، هرعملگر انتخاب استانداری را می‌توان برای انتخاب والدین بکار برد.

کارایی روش‌های هم‌تکاملی رقابتی را می‌توان با افزایش تنوع در دو جمعیت بهبود داد. استفاده از روش‌های نمونه‌گیری اشتراکی و اشتراک برازش رسیدن به این هدف را ممکن خواهد نمود. از هم-تکاملی رقابتی برای حل مسائل دنیای واقعی همچون نقشه‌کشی تاکتیکی نظامی، طراحی کنترل‌کننده، طراحی دارو و خودروی بدون سرنشین استفاده شده است.

شبه‌کد الگوریتم هم‌تکاملی رقابتی استاندارد، در شکل (3-12) نشان داده شده است. در شبه‌کد مذکور، دو جمعیت C_1 به‌عنوان جمعیت راه‌حل و C_2 به‌عنوان جمعیت آزمایشی با یکدیگر رقابت می‌کنند. به‌علاوه، شبه‌کد الگوریتم هم‌تکاملی رقابتی تک جمعیتی، در شکل (3-13) نشان داده شده است. برای تکامل جمعیت‌ها در هریک از شبه‌کدهای الگوریتم‌های شکل‌های (3-12) و (3-13)، هرکدام از الگوریتم‌های تکاملی پایه شامل GA، GP، ES، EP و یا حتی هرکدام از الگوریتم‌های فرامکاشفه‌ای زیستی مانند PSO را می‌توان به‌کار برد.

Function CoEC-2Pop (problem) returns a state that is a local optimum

Input : Populationsize , Problemsize , $P_{crossover}$, $P_{mutation}$

Output : S_{best}

Initialize two populations , C and Cz ;

while stopping condition(s) not true do

  for each $C_1.X_i$, i = 1, ..., $C_1.n_s$, do

    Select a sample of opponents from Cz ;

    Evaluate the relative fitness of $C_1 . X_i$ with respect to this sample ;

  end

  for each $C_2. X_i$, i = 1, ..., $C_z. n_s$ do

    Select a sample of opponents from C_1 ;

    Evaluate the relative fitness of $C_2 . X_i$ with respect to this sample ;

  end

  Evolve population G_1 for one generation ;

  Evolve population C_2 for one generation ;

end

Select the best individual from the solution population S_{best}

return S_{best} ;

شکل (3-12) : شبه‌کد الگوریتم هم‌تکاملی رقابتی با دو جمعیت (روش هیلیس)

Function CoEC-1 Pop(problem) returns a state that is a local optimum

Input : Populationsize , Problemsize , $P_{crossover}$, $P_{mutation}$

Output : S_{best}

Initialize one population, C ;

while stopping condition(s) not true do

  for each $C . X_i$, i = 1,...,$C.n_s$ do

    Select a sample of opponents from C to exclude $C . X_i$;

    Evaluate the relative fitness of $C . X_i$ with respect to the sample ;

  end

  Evolve population C for one generation ;

end

Select the best individual from C as solution S_{best} ;

return S_{best} ;

شکل (3-13) : شبه‌کد الگوریتم هم‌تکاملی رقابتی تک جمعیتی (روش هالند)

3-10-2 هم‌تکاملی همکاری

همان‌طور که بیان شد، سه نوع مختلف از هم‌تکاملی همکاری وجود دارد که درآن، موجودات از گونه-های مختلف (یا زیرجمعیت‌ها) باید با یکدیگر درحل یک وظیفه عمومی همکاری کنند. در همیاری، برازش موجودات وابسته به توانایی موجودات در همکاری با موجودات از گونه‌های دیگر می‌باشد. یکی از مشکلات اساسی درعملکرد الگوریتم‌های هم‌تکاملی همکاری، واگذاری نحوه محاسبه برازش نهایی می‌باشد که چگونه برازندگی نهایی توسط تکامل جمعی کلیه گونه‌ها به‌دست می‌آید.

دی جانگ و پوتر یک چهارچوب برای تکامل راه‌حل‌های پیچیده توسط الحاق زیرمولفه‌ها که به‌طور مستقل از یکدیگر می‌باشند، پیشنهاد نمودند. یک جمعیت مجزا برای تکامل هر زیرمولفه با استفاده از

تعدادی الگوریتم تکاملی استفاده شده وسپس، نمایش‌های مختلف ازهر زیرمولفه با یکدیگر ترکیب می‌شود تا راه‌حل کامل را شکل دهند، که برای تعیین برازش سراسری ارزیابی می‌شود. مبتنی‌بر این برازش سراسری، اعتباری (برازشی) به هرکدام از زیرمولفه‌ها داده می‌شود که میزان همکاری مولفه‌ها با یکدیگر را نشان می‌دهد. درواقع، آن‌ها از این روش برای بهینه‌سازی تابع استفاده کردند. برای یک مسأله n_x بعدی، n_x زیرجمعیت استفاده شده است، یکی برای هرکدام از ابعاد مسأله. بنابراین، هر زیرجمعیت مسئول تکامل یکی از پارامترهای مسأله می‌باشد و هیچ زیرجمعیتی نمی‌تواند خودش یک راه‌حل کامل را شکل دهد، به‌طوری‌که همکاری ازطریق ترکیب نمایش هر زیرجمعیت حاصل می شود. میزان کارایی این همکاری به این صورت تقریب زده می‌شود که به‌عنوان نمونه، برای J امین موجود در زیرجمعیت C_j، محاسبه برازش هرموجود X_i، از زیرجمعیت C_j، برازش مربوط به پاسخ (راه‌حل کامل) تشکیل‌شده از همکاری X_i، C_j با بهترین موجود از هرکدام از زیرجمعیت‌های دیگر محاسبه می‌شود. برازش حاصل‌شده برای راه‌حل کامل، به‌عنوان برازش X_i ، C_j درنظر گرفته می‌شود.

تبیین واقعی چارچوب پیشنهادی پوتر و دی جانگ را می‌توان به‌این شکل عنوان نمود که مسیرهای تکامل چشم‌ها، گوش‌ها، دست‌ها، پاها و دیگر اعضای بدن یک موجود (مانند انسان) مستقل از یکدیگر می‌باشند، ازآن‌جاکه برازش هر مولفه m مانند چشم، مستقل از دیگر مولفه‌ها قابل محاسبه است، به‌عنوان نمونه میزان موفقیت آن انسان در زندگی، تولید شود وسپس، برازندگی آن موجود کامل به‌عنوان برازندگی مولفه m درنظر گرفته شود.

علاوه‌بر بهینه‌سازی تابع، پوتر و دی جانگ از چارچوب پیشنهادی خود در تکامل شبکه‌های عصبی و یادگیری روبات استفاده نمودند. سایر کاربردهای هم‌تکاملی همکاری شامل تکامل توابع عضویت فازی، کنترل کننده روبات، پیش‌بینی سری‌های زمانی و آموزش شبکه‌های عصبی می‌باشد.

3-11 الگوریتم تکاملی دیپلوئیدی

الگوریتم دیپلوئیدی (*DEA*) نوعی الگوریتم تکاملی است که نمایش دیپلوئیدی دارد. نمایش دیپلوئیدی، ایده‌ای برگرفته از ژنتیک موجودات دیپلوئیدی است. در این نوع نمایش، هرموجود با دو کروموزوم مدل می‌شود و فنوتایپ آن، با یک تابع نگاشت ژنوتایپ به فنوتایپ به‌دست می‌آید.

به‌منظور درک عملکرد الگوریتم‌های تکاملی دیپلوئیدی، ابتدا باید با سلول‌های دیپلوئیدی و ساختار آن‌ها آشنا شد. سلول‌های دیپلوئیدی، سلول‌هایی هستند که دو مجموعه کروموزومی دارند که یک مجموعه را از پدر و یک مجموعه را از مادر به ارث برده‌اند. در این سلول‌ها، کروموزوم به‌صورت جفت‌جفت با یکدیگر مشابه هستند. هرجفت کروموزوم از دو کروموزوم همتا تشکیل شده است. کروموزوم‌های همتا، کروموزوم‌هایی هستند که اندازه، شکل و محتوای ژنتیک آن‌ها مشابه است. تعداد کروموزوم‌های یک سلول دیپلوئیدی را به‌صورت $2n$ نشان می‌دهند و به‌آن عدد دیپلوئید می‌گویند. درمقابل، سلول‌های هاپلوئیدی فقط یک مجموعه کروموزوم دارند و تعداد کروموزوم‌های یک سلول هاپلوئیدی را به‌صورت n نشان می‌دهند و آن را عدد هاپلوئید می‌نامند. سلول‌های اسنادیپلوئیدی هستند و از 46 کروموزوم شامل دو مجموعه 23 کروموزومی، تشکیل شده‌اند.

الگوریتم‌های تکاملی استاندارد، از مدل‌های ساده هاپلوئیدی استفاده می‌کنند، به این معنی که در آن‌ها هر فرد فقط یک ژنوتایپ دارد. در این الگوریتم‌ها، ژنوتایپ هر فرد یک راه‌حل برای مسأله بهینه‌سازی موردنظر می‌باشد. یکی از بزرگترین مشکلاتی که این الگوریتم‌ها با آن مواجه هستند، مشکل همگرایی زودرس و عدم توانایی آن‌ها در حفظ تنوع جمعیت در طول نسل‌هاست. این مشکل سبب افت کارایی الگوریتم تکاملی در مواجهه با برخی مسائل ازجمله مسائل بهینه‌سازی پویا، مسائل بهینه‌سازی چندهدفه و آن دسته از مسائل بهینه‌سازی که بیش از یک جواب دارند، می‌شود. یکی از عواملی که در طبیعت، قابلیت حفظ تنوع جمعیت در طول نسل‌ها را در گروهی از موجودات زنده بوجود می‌آورد، دیپلوئیدی بودن سلول آن موجودات است.

در الگوریتم‌های تکاملی دیپلوئیدی، هر فرد با دو کروموزوم مدل شده و درنتیجه، هر ویژگی با دو مقدار (آلل) مشخص می‌شود. میزان به‌روز هریک ازآن مقادیر در فنوتایپ مربوط به آن ویژگی متفاوت است. این مسأله باعث بوجود آمدن روابط غالب و مغلوبی بین آلل می‌شود. اگر برای یک ویژگی، هر دو آلل یکسان باشند، آن‌گاه فنوتایپ به‌روزیافته آن ویژگی بدون هیچ بحثی مشخص است و در اصطلاح، فرد با توجه به این ویژگی، همگن است. اما اگر آلل‌های یک ویژگی متفاوت باشند، یا به عبارت دیگر، اگر فرد در یک ویژگی ناهمگن باشد، فنوتایپ به‌روزیافته در آن ویژگی با توجه به سازوکار غلبه حاکم بر آن آلل‌ها مشخص می‌شود. در ساده‌ترین حالت این سازوکار، همواره یکی از آلل‌ها مغلوب و دیگری غالب فرض شده و آلل غالب در موجود به‌روز می‌شود. لازم به ذکر است که سازوکار غلبه در الگوریتم‌های تکاملی دیپلوئیدی استاندارد قطعی بوده و همیشه آلل غالب در فنوتایپ مربوط به یک ویژگی به‌روز می-

شود. البته این سازوکار غلبه ساده، با آن‌چه‌که به‌طورواقعی در طبیعت رخ می‌دهد، تفاوت زیادی داشته و درنتیجه، در حفظ تنوع جمعیت چندان خوب عمل نمی‌کند. مطالعات نشان داده‌اند که هرچه این سازوکار، مدل نزدیک‌تری به سازوکار طبیعی ارائه دهد، در حفظ تنوع جمعیت بهتر عمل می‌کند.

با توجه به آن‌چه بیان شد، برای تعریف یک الگوریتم تکاملی دیپلوئیدی جدید، باید مراحل به‌ترتیب زیر انجام شوند :

1. تعیین روشی برای بازنمایی موجودات به‌صورت دیپلوئیدی.
2. تعیین مجموعه آلل‌هایی که هر ویژگی می‌تواند داشته باشد.
3. تعریف سازوکار غلبه با توجه به کاربرد و اهداف الگوریتم.
4. تعریف عملگرهای جهش و بازترکیب با توجه به بازنمایی موجودات.
5. تعیین روشی برای ارزیابی شایستگی موجودات.

شبه‌کد الگوریتم تکاملی دیپلوئیدی در شکل (3-14) نشان داده شده است. در این شبه‌کد، C_1، C_2 و *Child*، به‌ترتیب کروموزوم‌های اول، دوم و فرزند تولیدشده می‌باشند. برای درک بهتر شبه‌کد الگوریتم تکاملی دیپلوئیدی در شکل (3-14)، باید تفاوت الگوریتم تکاملی دیپلوئیدی را با الگوریتم‌های تکاملی پایه درک نمائیم. الگوریتم‌های تکاملی دیپلوئیدی در دو زمینه کلی شامل نمایش و ارزیابی موجودات، و عملگر تولیدمثل، با الگوریتم‌های تکاملی ساده متفاوت هستند. درادامه، به تشریح این دو زمینه پرداخته می‌شود.

Function DEA (problem) returns a state that is a local optimum

Input : Populationsize , Problemsize , $P_{crossover}$, $P_{mutation}$

Output : S_{best}

Population ← InitializePopulation (Populationsize , Problemsize) ;

EvaluatePopulation (Population) ;

S_{best} ← GetBestSolution (Population) ;

While –StopCondition () do

  Parents ← SelectParents (Population , Populationsize) ;

  PreBornChildren ← ProducePBC (Parents , Populationsize) ;

  Children ← Ø ;

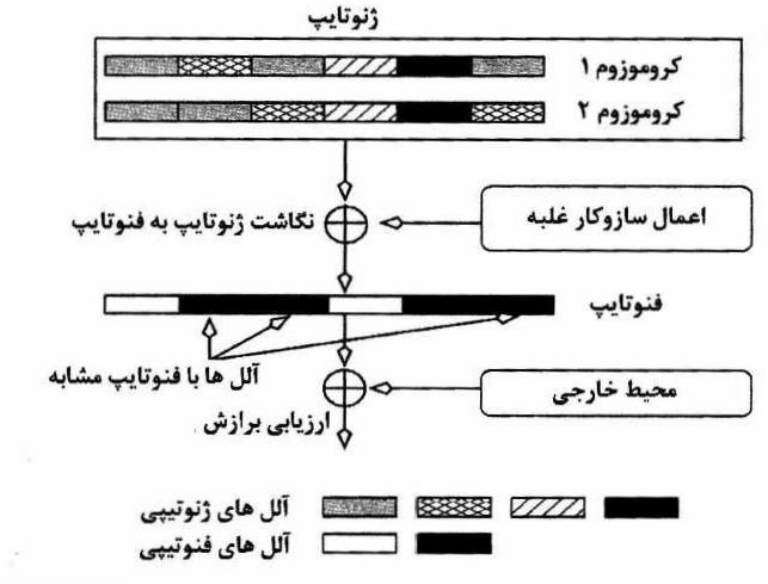
  for each Child ∈ PreBornChildren do

```
    Child1 , Child2 ← Crossover (Child.C1 , Child.C2 , Pcrossover) ;
    Children ← Mutate (Childi , Pmutation) ;
    Children ← Mutate (Child2 , Pmutation) ;
  end
  EvaluatePopulation (Children) ;
  if dominance change used and condition holds then
    change dominance scheme
  end
  Population ← Replace (Population , Children) ;
  Sbest← GetBestSolution (Population) ;
end
return Sbest ;
```

شکل (3-14) : شبه‌کد الگوریتم تکاملی دیپلوئیدی

- **نمایش و ارزیابی موجودات**

در الگوریتم تکاملی دیپلوئیدی، هر فرد با دو کروموزوم مدل می‌شود. برای ارزیابی شایستگی هر فرد، یک راه متداول این است که ابتدا، فنوتایپ فرد با استفاده از سازوکار، غلبه استخراج شود وسپس، این فنوتایپ با توجه به محیط خارجی ارزیابی گردد. شکل (3-15) نمایش و ارزیابی افراد را در این الگوریتم‌ها نشان می‌دهد. دراین شکل، یک سازوکار غلبه قطعی برای استخراج فنوتایپ استفاده شده است و در ژنوتایپ آن، هر ژن می‌تواند چند نوع آلل مختلف داشته باشد، اما در فنوتایپ، هر ژن می‌تواند آلل صفر یا یک را بگیرد. البته کلیه الگوریتم‌های تکاملی دیپلوئیدی این‌گونه عمل نمی‌کنند و شکل (3-15) تنها نمونه‌ای را برای تعیین روش نمایش ژنوتایپ و چگونگی استخراج فنوتایپ ازآن نشان می‌دهد.



شکل (3-15) : نمونه‌ای از روش نمایش ژنوتایپ و چگونگی استخراج فنوتایپ ازآن

- **عملگرهای تولید مثل**

در الگوریتم تکاملی دیپلوئیدی، تولیدمثل در سه مرحله انجام می‌شود. درمرحله اول، از هر دو والد فرزند تولید می شود، به‌گونه‌ای که هریک از فرزندان، یک کروموزوم از هر والد را به ارث ببرند (تابع Produce PBC در شبه‌کد الگوریتم شکل (3-14)). این‌که هرفرزند، کدام یک از کروموزوم‌های هر والد را بگیرد، می‌تواند تصادفی انتخاب شود. در این مرحله، ساختار ابتدایی دیپلوئیدی فرزندان شکل می‌گیرد و صفات موجودات یک نسل، به نسل بعدی منتقل می‌شوند. درمرحله دوم، دو کرروموزم هرفرزند، با احتمال P_c وارد مرحله بازترکیب می‌شوند. فرآیند مذکور در این‌جا به‌طوردقیق به همان شکلی که در الگوریتم تکاملی ساد وجود دارد، انجام می‌شود. نکته قابل توجه این است که دراین مرحله، فنوتایپ تغییری نمی‌کند و تنها دلیل برای اعمال این روی افراد جمعیت ترکیب کروموزوم‌های والدین، برای انتقال به نسل بعد است. درمرحله سوم، روی هرکدام از کروموزوم‌های هر فرزند، به‌صورت مستقل، با احتمال P_m جهش اعمال می‌شود. با اعمال جهش روی ژنوتایپ یک فرد، ممکن است در فنوتایپ فرد هیچ تغییری ایجاد نشود. در اصلاح زیست‌شناسی به این نوع جهش، جهش خنثی می‌گویند.

همان‌طورکه بیان شد، هر ویژگی در موجودات دیپلوئیدی، به وسیله دو آلل نشان داده می‌شود که این آلل-ها در جایگاه یکسانی روی دو کروموزوم همتا قرار گرفته‌اند. براساس والدین مدل، اگر صفتی دو آلل

یکسان داشته باشد، آن دو آلل در فنوتایپ هم به‌روز می‌شود و چنان‌چه دو آلل متفاوت باشند، آللی به‌روز می‌شود که غالب باشد. بنابراین از بین دو آلل مربوط به یک ویژگی، فقط آن‌که غالب است در فنوتایپ ظاهر می‌شود، اما آلل دیگر ازبین نمی‌رود. این آلل تا زمانی‌که مفید واقع شود، در ژنوتایپ پنهان می‌ماند.

ایده استفاده از بازنمایی دیپلوئیدی و سازوکار غلبه، اولین بار توسط هولشتاین در سال 1971 در یک الگوریتم ژنتیک به‌کار گرفته شد. در الگوریتم تکاملی دیپلوئیدی می توان با الهام از قوانین مندل در مورد غلبه آلل‌ها بریکدیگر، از سازوکارهای غلبه بین آلل‌ها استفاده نمود. این سازوکار غلبه می‌تواند به‌صورت ثابت یا متغیر تعریف شود . درحالتی‌که سازوکار غلبه ثابت باشد، در تمام طول اجرای الگوریتم، یک آلل بر آلل دیگر غلبه دارد. درمقابل، درحالتی‌که سازوکار غلبه متغیر باشد، میزان غلبه آلل‌ها بریکدیگر می‌تواند در طول زمان تغییر کند (انتهای شبه‌کد الگوریتم شکل (3-16)). به‌عنوان نمونه، دراین حالت می‌توان هرچند وقت یک‌بار از محیط بازخورد گرفت و میزان شایستگی نسبی آلل‌ها را سنجید و میزان غلبه آن‌ها را براساس این شایستگی، کم یا زیاد نمود. همچنین، در این الگوریتم‌ها می‌توان قانون غلبه مندل را پایه و اساس مدل ژنی الگوریتم قرار نداد و مدل ژنی الگوریتم را براساس چگونگی وراثت در صفاتی که از قانون غلبه پیروی نمی‌کنند، ساخت. به‌عنوان نمونه، صفاتی مانند قد و وزن انسان و یا صفات رنگ مو صفاتی هستند که قوانین غلبه را نقص کرده و وراثت آن‌ها به شکل دیگری صورت می‌گیرد.

تابه‌حال مطالعات زیادی براستفاده از ژنوتایپ‌های دیپلوئیدی و عملگرهای غلبه در الگوریتم‌های تکاملی انجام شده است که بیشتر آن‌ها بر الگوریتم‌های ژنتیک دیپلوئیدی تمرکز دارند. از الگوریتم‌های ژنتیک دیپلوئیدی برای حل بسیاری از مسائل بهینه‌سازی، به‌طورخاص در محیط‌های پویا استفاده شده است.

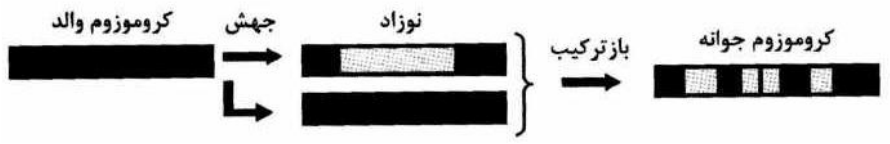
3-12 بهینه‌سازی تولیدمثل غیرجنسی

بهینه‌سازی تولیدمثل غیرجنسی (*ARO*)، ازجمله الگوریتم‌های جدید ارائه شده درحوزه روش‌های جستجوی فرامکاشفه‌ای می‌باشد. دراین روش جستجو، ابتدا یک عضو تکی (کروموزوم والد) به‌صورت

تصادفی تولیدشده وسپس، مورد ارزیابی واقع می‌شود. درادامه، عملیات زیر به‌صورت تکراری انجام می‌شوند. ابتدا، زیررشته‌ای در کروموزوم والد به‌صورت تصادفی انتخاب‌شده و مورد جهش قرار می-گیرد. طول زیررشته انتخابی (g) نیز تصادفی بوده و دربازه $[1,L]$ می‌باشد (L طول کروموزوم والد است). نتیجه این جهش به‌نام نوزاد شناخته می‌شود. سپس باتوجه‌به یک احتمال P_c کروموزوم والد و نوزاد تحت عملگرد بازترکیب یکنواخت قرار می‌گیرند. به دلیل بازترکیب والد با نوزادش، مفهوم غیرجنسی بودن یا عدم نیاز به دو عضو غیرهم‌ریشه، در نام‌گذاری این الگوریتم وجود دارد). احتمال P_c نیز به‌طورکامل تصادفی بوده و طبق رابطه زیر تعیین می شود.

$$p_c = \frac{1}{1 + lng}$$

نتیجه بازترکیب به‌نام کروموزوم جوانه شناخته می‌شود. شکل (3-16)، مراحل مربوط به عملیات تولیدمثل را به بهینه‌سازی تولیدمثل غیرجنسی نشان می‌دهد. درادامه، کروموزوم جوانه ارزیابی شده و درصورتی‌که برازش آن از کروموزوم والد بهتر باشد، جایگزین آن می‌شود.



شکل (3-16) : نمونه‌ای برای تبیین عملکرد عملگرهای تولیدمثل در بهینه‌سازی تولیدمثال غیرجنسی

بهینه‌سازی تولیدمثل غیرجنسی توسط فراست پیشنهاد شد. شبه‌کد الگوریتم بهینه‌سازی تولیدمثل غیرجنسی در شکل (3-17) نشان داده شده است. خصوصیت‌های مهم الگوریتم بهینه‌سازی تولیدمثل غیرجنسی عبارتند از :

- استفاده از نمایش رشته بیتی.
- طول ثابت و یکسان برای هر کروموزوم.
- به‌کارگیری یک عضو تکی به‌جای جمعیتی از اعضاء.
- در بهینه‌سازی تولیدمثل غیرجنسی، دیدگاهی حریصانه برای حرکت در دورنمای برازش مسأله وجود دارد. با توجه به این دیدگاه، فرزند تولیدشده تنها زمانی جایگزین والد خود می شود که از او برازنده‌تر باشد.

- به‌کارگیری عملگر بازترکیب یکنواخت.
- استفاده از جهش معکوس سازی بیت.
- تنظیم احتمال بازترکیب P_c به‌صورت کاملا تصادفی.
- برای طول زیررشته تصادفی مورد جهش (g) در کروموزوم والد، رابطه $G \sim \text{uniform1}, L$ همواره برقرار است.
- سرعت بالا و عدم وجود هرگونه پارامتر در این الگوریتم، از ویژگی‌های مثبت بهینه‌سازی تولیدمثل غیرجنسی به‌شمار می‌رود.
- ویژگی منفی بهینه‌سازی تولیدمثل غیرجنسی رفتار شدید تصادفی این الگوریتم است که متاسفانه ماهیت این روش را به الگوریتم جستجوی تصادفی بسیار نزدیک کرده است.

```
Function ARO (problem) returns a state that is a local maximum
Input : Problemsize
Output : Sbest
Parent ← InitializeParent (Problemsize) ;
EvaluateSolution (Parent) ;
While -StopCondition() do
  Larva ← Mutate (Parent) ;
  Bud ← Crossover (Parent , Larva) ;
  EvaluateSolution (Bud) ;
  if Fitness (Bud) > Fitness (Parent) then
    Parent ← Bud ;
  end
end
Sbest ← Parent ;
return Sbest ;
```

شکل (3-17) : شبه‌کد الگوریتم بهینه‌سازی توایدمثل غیرجنسی

3-13 سیستم ایمنی مصنوعی

هدف اصلی سیستم ایمنی طبیعی در بدن انسان، تمایز بین بافت خودی و عامل خارجی (غیرخودی یا آنتی‌ژن) می‌باشد. سیستم ایمنی بدن، به مواد خارجی یا مواد بیماری زا معروف به آنتی‌ژن عکس‌العمل

نشان می‌دهد. درحین این واکنش، سیستم ایمنی برای تشخیص بهتر آنتی‌ژن دیده شده، تطبیق یافته و حافظه‌ای برای ثبت آنتی‌ژن‌های رایج ایجاد می‌کند. حافظه ایجادشده باعث بهبود و نیز سرعت بخشیدن به واکنش سیستم ایمنی تطبیق‌پذیر در برخوردهای آینده با همان آنتی‌ژن خواهد شد. شناسایی آنتی‌ژن‌ها، منجربه تولید سلول‌های خاصی می‌شود که آنتی‌ژن‌ها را غیرفعال و یا نابود می‌کنند. سیستم ایمنی بدن همانند یک سیستم تشخیص الگو عمل می‌کند، برای تشخیص الگوهای غیرخودی از الگوهای خودی.

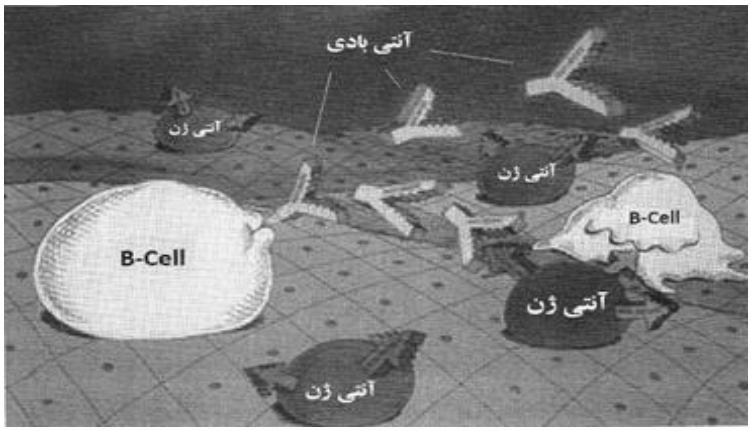
در سیستم ایمنی طبیعی، آنتی‌ژن‌ها موادی هستند که می‌توانند پاسخ ایمنی را ایجاد کنند. پاسخ ایمنی واکنش بدن به آنتی‌ژن می‌باشد. بنابراین، از صدمه زدن آنتی ژن به بدن جلوگیری می‌شود. آنتی‌ژن‌ها می‌توانند باکتری، قارچ، انگل و یا ویروس باشند. یک آنتی‌ژن باید به‌عنوان یک عامل خارجی غیرخودی تشخیص داده شود. وظیفه تشخیص آنتی‌ژن‌ها برعهده نوعی گلبول سفید به‌نام لنفوسیت می‌باشد. دو نوع لنفوسیت وجود دارند شامل T-Cell و B-Cell که هر دو در مغز استخوان تولید می‌شوند. لازم به ذکر است که B-Cellها در تماس با آنتی‌ژن‌ها، آنتی‌بادی‌هایی تولید کرده که درمقابل آنتی‌ژن‌ها مؤثر می‌باشد. آنتی‌بادی‌ها پروتئین‌های شیمیایی هستند و دارای شکل *Y* مانند هستند.

آنتی‌بادی‌ها دارای گیرنده‌های خاص برای تشخیص آنتی‌ژن‌ها هستند. زمانی‌که تماس بین آنتی‌بادی یک B-Cell و آنتی‌ژن برقرار می‌شود، تکثیر کلونی در سطح B-Cell اتفاق می‌افتد و به‌کمک T-Cell تقویت می‌شود. درصورت آن‌که یک آنتی‌بادی به‌جای تشخیص آنتی‌ژن، بافت خودی را به‌عنوان عامل خارجی تشخیص داد، نابود می‌شود.

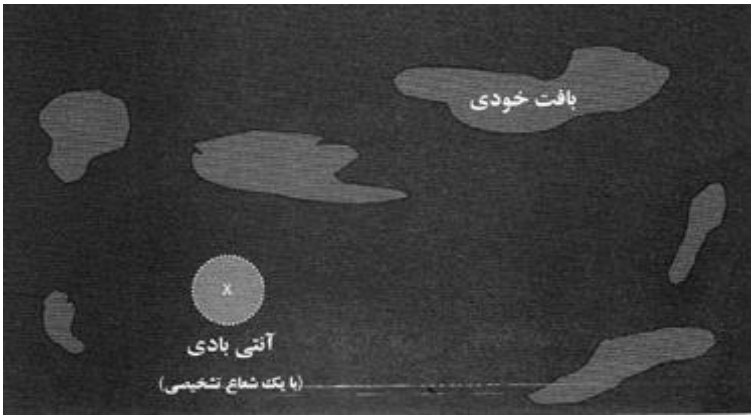
درحین تکثیر کلونی، دو نوع از سلول شکل می‌گیرد شامل سلول‌های پلاسما و سلول‌های حافظه. وظیفه سلول‌های حافظه، تکثیر سلول‌های پلاسما برای واکنش سریع‌تر درمواجهه تکراری با آنتی‌ژن‌ها و تولید آنتی‌بادی برای آن‌ها می‌باشد. سلول پلاسما، یک B-Cell است که آنتی‌بادی تولید می‌کند.

بخش کوچکی ازیک آنتی‌ژن، اپیتپ و بخش کوچکی از آنتی‌بادی‌ها، پاراتپ نامیده می‌شوند. اپیتپ‌ها پاسخ ایمنی خاصی را طلب می‌کنند و پاراتپ‌ها در آنتی‌بادی‌ها می‌توانند به اپیتپ با یک توان پیوند مشخص، پیوند بخورد. بعداز برقراری پیوند بین پاراتپ یک آنتی‌بادی و اپیتپ یک آنتی‌ژن، ترکیب آنتی‌بادی – آنتی‌ژن تشکیل می‌شود که منجربه ازکارافتادن آنتی‌ژن می‌شود. برای درک بهتر مفاهیم بحث‌شده در سیستم ایمنی بدن، شکل (3-18) تولید آنتی‌بادی توسط B-Cell جهت تشخیص آنتی‌ژن‌ها، شکل (3-19) چگونگی جستجو جهت تشخیص و انهدام آنتی‌ژن توسط آنتی‌بادی، شکل (3-20)

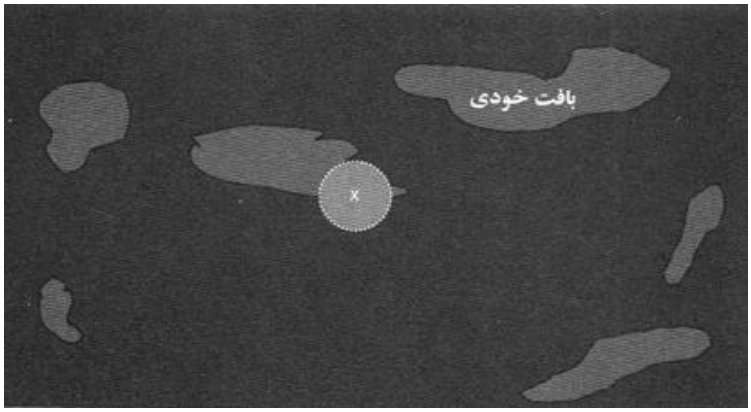
چگونگی تشخیص اشتباه بافت خودی و انهدام آنتی‌ژن توسط آنتی‌بادی، شکل (3-21) چگونگی تشخیص و انهدام آنتی‌ژن توسط آنتی‌بادی و نابودشدن آنتی‌بادی غیرشایسته، شکل (3-22) چگونگی تشخیص و انهدام آنتی‌ژن توسط آنتی‌بادی و تشخیص آنتی‌ژن، و شکل (3-23) چگونگی تشخیص و انهدام آنتی‌ژن توسط تکثیر آنتی‌بادی را نشان می‌دهند.



شکل (3-18) : تولید آنتی‌بادی توسط B-Cell جهت تشخیص آنتی‌ژن‌ها



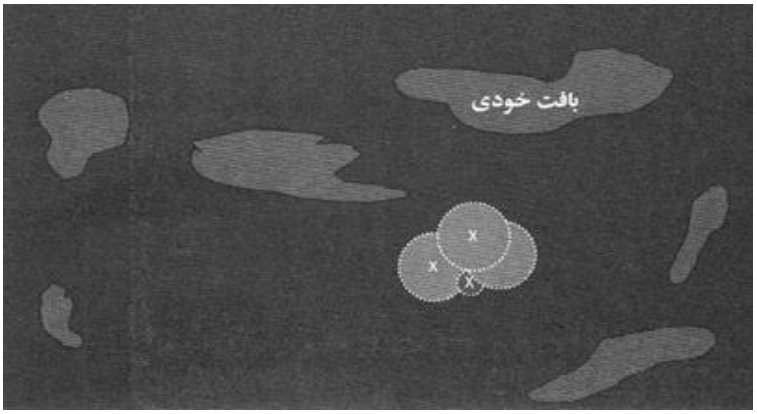
شکل (3-19) : چگونگی جستجو جهت تشخیص و انهدام آنتی‌ژن توسط آنتی‌بادی



شکل (3-20) : چگونگی تشخیص اشتباه بافت خودی و انهدام آنتی‌ژن توسط آنتی‌بادی

شکل (3-21) : چگونگی تشخیص و انهدام آنتی‌ژن توسط آنتی‌بادی و نابودشدن آنتی‌بادی غیرشایسته

شکل (3-22) : چگونگی تشخیص و انهدام آنتی‌ژن توسط آنتی‌بادی و تشخیص آنتی‌ژن



شکل (3-23) : چگونگی تشخیص و انهدام آنتی‌ژن توسط تکثیر آنتی‌بادی

از عملکرد سیستم ایمنی بدن در تشخیص و انهدام عوامل خارجی برای طراحی الگوریتم‌های متنوعی در بهینه‌سازی، یادگیری ماشین و شناسایی الگو بهره‌برداری شده است. باتوجه‌به آن‌که دراین‌جا الگوریتم-هایی مورد بحث قرار می‌گیرند که بتوانند برای هرمسأله‌ای دارای یک دورنمای برازش، به‌کار روند. از مشهورترین الگوریتم‌های جستجوی فرامکاشفه‌ای مبتنی‌بر سیستم ایمنی طبیعی، سیستم ایمنی مصنوعی (AIS) می‌باشد که اولین بار توسط دی‌کاسترو و زوبن تحت نام الگوریتم انتخاب کلونی (CLONALG) ارائه شد.

ایده اصلی در سیستم ایمنی مصنوعی، برگرفته از فرآیند تکثیر سلولی پس‌از تشخیص عامل خارجی در سیستم ایمنی طبیعی می‌باشد. این فرآیند در سیستم ایمنی مصنوعی به‌نام انتخاب کلونی شناخته شده و شامل سه مرحله است. درمرحله اول، گروهی از بهترین سلول‌ها که آنتی‌بادی مربوط به آن‌ها بیشترین تطابق را با آنتی‌ژن‌ها داشته باشد، انتخاب می‌شوند. لازم به ذکر است که به‌طورعملی هر سلول معادل یک پاسخ مسأله، مشابه کروموزم در الگوریتم ژنتیک، می‌باشد. در مرحله دوم، سلول‌های انتخاب‌شده تحت عملیات کلونی قرارگرفته و با توجه به یک پارامتر به‌نام نرخ تکثیر، مورد تکثیر واقع می‌شوند. تعداد کپی‌هایی که از هرسلول تولید می‌شود، بستگی به برازش آن سلول دارد. هرچقدر برازش یک سلول بیشتر باشد، تعداد کپی‌های بیشتری از آن ایجاد خواهد شد. درمرحله سوم، سلول‌های تکثیرشده تحت عملگری به‌نام فراجهش قرار می‌گیرند. این عملگر که مشابه عملگر جهش در الگوریتم ژنتیک است، با توجه به پارامتری به‌نام P_m که نرخ جهش می‌باشد، تغییر کوچکی را به هر سلول تکثیرشده اعمال می‌کند. تفاوتی که عملگر فراجهش با عملگر جهش در الگوریتم ژنتیک دارد، آن است که در

فراجهش، اندازه تغییراتی که در هرسلول ایجاد می‌شود، بستگی به برازش آن سلول دارد. هر چقدر یک سلول برازنده‌تر باشد، تغییرات کمتری به آن اعمال خواهد شد.

سیستم ایمنی مصنوعی به‌طور موفقیت‌آمیزی برای دامنه وسیعی از مسائل به‌کار رفته است. از مسائل تشخیص نفوذ به شبکه تا مدل‌های دسته بندی داده، یادگیری مفهوم، خوشه‌بندی داده، روباتیک، شناسایی الگو و داده کاوی، و همچنین برای مقداردهی اولیه وزن‌های شبکه عصبی پیشرو و بهینه‌سازی تابع چندوجهی نیز استفاده شده است. درادامه، خصوصیت‌های مهم سیستم ایمنی مصنوعی به‌طور خلاصه توصیف شده است :

- استفاده از نمایش رشته بیتی.
- طول ثابت و یکسان برای هر سلول (هرعضو جمعیت).
- به‌کارگیری یک جمعیت با تعداد اعضاء ثابت. لازم به دکراست که در سیستم ایمنی مصنوعی می‌توان با تعریف طول عمر برای سلول‌های جمعیت، از یک جمعیت با تعداد اعضاء متغیر بهره برد.
- در شبه‌کد الگوریتم سیستم ایمنی مصنوعی در شکل (3-24)، Selectionsize نشانگر تعداد سلول‌هایی است که برای انجام عملیات تکثیر، از جمعیت دورجاری درحلقه While انتخاب می‌شوند که همان مرحله اول در انتخاب کلونی است. به‌علت وجود همین مرحله در سیستم ایمنی مصنوعی، ماهیتی مشابه انتخاب داروینی در این فرامکاشفه وجود دارد و در این فصل به‌عنوان یک روش تکاملی مورد بحث و بررسی قرار گرفته است.
- برای تعیین عدد تکثیر سلولی برای یک سلول ایمنی انتخاب‌شده در تابع Clone شکل (3-24)، از رابطه $N_c = \mathrm{round}(\beta . N . R)$ استفاده می‌شود که درآن، N_c بیانگر تعداد کپی‌های سلول، β مبین نرخ تکثیر (Clonerate)، N نشانگر تعداد سلول‌های جمعیت (Populationsize) و R بیانگر برازش مبتنی‌بر رتبه سلول مورد تکثیر می‌باشد.
- استفاده از جهش معکوس‌سازی بیت در عملگر فراجهش (عملگر جهش در *AIS* عملگر اصلی می‌باشد.
- برخلاف الگوریتم‌های تکاملی که اغلب احتمال جهش مقداری ثابت می‌باشد، احتمال فراجهش (P_{hm}) در سیستم ایمنی مصنوعی مقداری متغیر داشته و اندازه آن بستگی‌به برازش سلول ایمنی

(عضو جمعیت) دارد. برای محاسبه احتمال فراجهش از رابطه $P_{hm} = \exp(-P_m . f)$ استفاده می‌شود، به‌طوری‌که P_{hm} نرخ جهش بوده و f برازش سلول ایمنی مورد جهش می‌باشد.

```
Function AIS(problem) returns a state that is a local optimum
Input : Populationsize , Selectionsize , Problemsize , Clonerate , P_mutation
Output : S_best
Population ← InitializePopulation (Populationsize , Problemsize) ;
EvaluatePopulation (Population) ;
S_best ← GetBestSolution (Population) ;
While-StopCondition () do
  Populationselect ← Select (Population , Selectionsize) ;
  Populationclones ← Ø ;
  for each p_i ε Populationselect do
    Populationclones ← Clone (p_i, Clonerate) ;
  end
  for each p_i ε Populationclones do
    Populationnew ← Hypermutate (p_i , P_mutation) ;
  end
  EvaluatePopulation (Populationnew) ;
  Population ←Replace (Population , Populationnew) ;
  S_best ← GetBestSolution (Population) ;
end
return S_best ;
```

شکل (3-24) : شبه‌کد الگوریتم سیستم ایمنی مصنوعی