موضوع پروژه:

الگوریتم ژنتیک (Genetic Algorithm - GA)

چکیده

الگوریتم ژنتیک (Genetic Algorithm - GA) تکنیک جستجویی در علم رایانه برای یافتن راه‌حل تقریبی برای بهینه‌سازی و مسائل جستجو است. الگوریتم ژنتیک نوع خاصی از الگوریتم‌های تکامل است که از تکنیک‌های زیست‌شناسی فرگشتی مانند وراثت و جهش استفاده می‌کند.

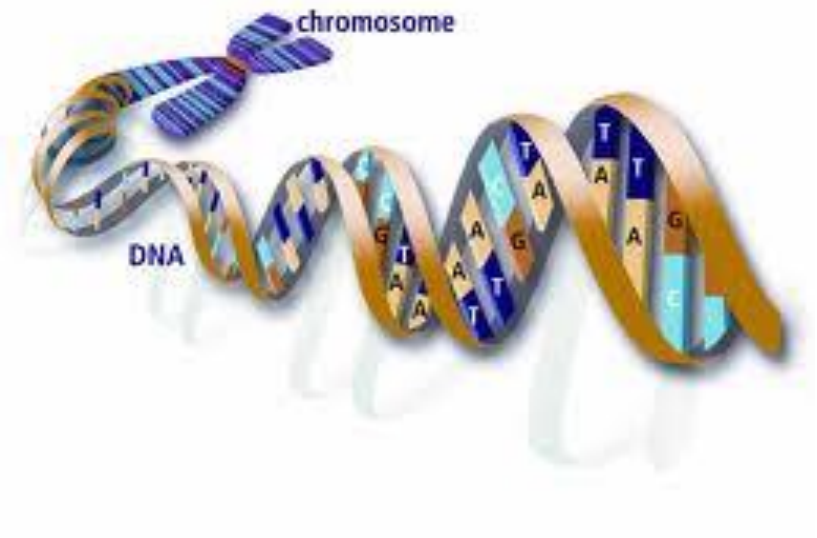
در واقع الگوریتم‌های ژنتیک از اصول انتخاب طبیعی داروین برای یافتن فرمول بهینه جهت پیش‌بینی یا تطبیق الگو استفاده می‌کنند. الگوریتم‌های ژنتیک اغلب گزینه خوبی برای تکنیک‌های پیش‌بینی بر مبنای تصادف هستند. مختصراً گفته می‌شود که الگوریتم ژنتیک (یا GA) یک تکنیک برنامه‌نویسی است که از تکامل ژنتیکی به عنوان یک الگوی حل مسئله استفاده می‌کند. مسأله‌ای که باید حل شود ورودی است و راه‌حل‌ها طبق یک الگو کد گذاری می‌شوند که تابع fitness نام دارد هر راه حل کاندید را ارزیابی می‌کند که اکثر آنها به صورت تصادفی انتخاب می‌شوند.

کلاً این الگوریتم‌ها از بخش های زیر تشکیل می‌شوند: تابع برازش، نمایش، انتخاب، تغییر.

فهرست مطالب

فصل اول:

مقدّمات



۱-۱- مقدمه

امروزه یکی از مهم‌ترین زمینه‌های تحقیق و پژوهش، توسعهٔ روش‌های جستجو بر مبنای اصول تکامل طبیعی می‌باشد. در محاسبات تکاملی به صورت انتزاعی از مفاهیم اساسی تکامل طبیعی در راستای جستجو برای یافتن راه حلّ بهینه برای مسائل مختلف الهام گرفته شده است. در همین راستا مطالبی که در این فصل پیش روی شما پژوهندهٔ گرامی قرار خواهد گرفت مفاهیمی دربارهٔ علم کامپیوتر و علم ژنتیک مانند: الگوریتم و انواع آن، جستجو، هیوریستیک، تاریخچه الگوریتم ژنتیک و علم ژنتیک، ژن، کروموزوم، ارث بری و... می باشد، و یا به بیانی خلاصه‌تر می‌توان گفت: در این فصل به بیان مقدّمات خواهیم پرداخت.

انشاءالله مطالعهٔ این فصل مفهومی ساده و روشن از موضوعِ این نوشتار را برای شما خوانندهٔ محترم به تصویر خواهد کشید و شما را در درک آسان و سریع فصول بعدی یاری خواهد رساند.

۱-۲- به دنبال تکامل...

بسیاری از دانشمندان و اندیشمندان، میل به تکامل را مهترین عامل پیشرفت دستگاه آفرینش و انسان می‌دانند. از این دیدگاه هر پدیده‌ای را که بنگرید، یک مسأله جستجوست. انسان همواره می‌کوشد تا به تکامل برسد، از این رو می‌اندیشد، می‌پژوهد، می‌کاود، می‌سازد، می‌نگارد و همواره می‌کوشد تا باقی بماند. حتی می‌توان گفت که میل به زادن فرزند، گامی در برآوردن این نیاز و البته دیگر جانداران است. می‌توان این تلاش در راه رسیدن به تکامل را یک مسألهٔ جستجو تعبیر کرد.

کوشش یک مؤسسه اقتصادی یا تولیدی –که تابعی برای تبدیل داده‌ها به ستادهاست– برای کمینه کردن هزینه‌ها و بیشینه کردن سود، یک مسألهٔ جستجو است. تلاش یک سپاه در حال جنگ، برای وارد کرد بیشترین خسارات بر دشمن با از دست دادن کمترین نیرو و جنگ‌افزار، یا کوشش یک دانش‌آموز برای دست یافتن به بالاترین نمره، سعی یک موسیقیدان یا نگارگر برای خلق زیباترین اثر هنری، تلاش یک کاندیدا برای به دست آوردن بیشترین رأی، طراحی یک نجّار برای ساختن راحت‌ترین صندلی، تلاش و نقشه چینی ورزشکاران و مربّیان برای یافتن راه‌های پیروزی بر حریف و... همگی جستجویی در فضای یک مسأله برای یافتن نقاط یا ناحیه بهینگی (بیشینه یا کمینه) هستند و همین امر موجب پیشرفت تمدن و آفرینش شده است.

در دانش کامپیوتر و فناوری اطلاعات هم «جستجو» یکی از مهمترین مسائل است. تنها کافیست که حجم اطلاعات قرار گرفته بر حافظه‌های گوناگون و اینترنت را در نظر بگیریم تا جایگاه ویژه آن را دریابیم.

تاکنون روشهای بسیاری توسط طراحان الگوریتم‌ها برای انجام جستجو بر داده‌های دیجیتالی ارائه شده است. روش‌هایی به نام جستجوی سریع[۱] و جستجوی دودویی[۲]، از ساده‌ترین

[۱] Quick Search

[۲] Binary Search

الگوریتم‌هایی هستند که دانشجویان گرایش‌های مهندسی کامپیوتر در نخستین سال‌های دوره کارشناسی فرا می‌گیرند، امّا این الگوریتم‌ها شاید، هنگامی که با حجمی گسترده از داده‌ها روبرو شوند، کارایی ندارند و حتی الگوریتم‌های پیشرفته‌تر مانند جستجوی بازپخت شبیه‌سازی شده[۱] و الگوریتم عمیق‌شوندهٔ تکراری[۲] نیز در هنگام رویارویی با مسائل ابرفضا[۳] از یافتن راه‌حل یا ناحیه‌های دلخواه در می‌مانند. در این میان یک روش جادویی وجود وجود دارد که مسائل بزرگ را به سادگی و به گونه‌ای شگفت‌انگیز حل می‌کند و آن «الگوریتم ژنتیک»[۴] است. ناگفته پیداست که واژهٔ «الگوریتم ژنتیک» از دو واژهٔ «الگوریتم» و «ژنتیک» تشکیل شده است که خود مبیّن این مطلب است که این روش از دو علم ریاضی و زیست‌شناسی برای حل مسائل کمک می‌گیرد.

الگوریتم‌ژنتیک بر خلاف دیگر روش‌های جستجو، که توسط طراحان نگاشته می‌شوند، در حقیقت به دست دستگاه آفرینش پدید آمده، و پس از شناخت نسبی دانشمندان از این روش به صورت مسأله‌ای ریاضی فرموله شده و وارد دانش مهندسی کامپیوتر و دیگر علوم مرتبط گردیده است. در یکی دو دهه گذشته که این الگوریتم در علوم مهندسی بکار گرفته شده، ناباورانه چنان دست‌آوردها و نتایج شگفت‌انگیزی داشته که نگاه بسیاری از دانش‌پژوهان علوم گوناگون فنی‌مهندسی را به خود جلب کرده است.[1]

۱-۳- ایدهٔ اصلی استفاده از الگوریتم ژنتیک

در دهه ۷۰ میلادی دانشمندی از دانشگاه میشیگان به نام «جان هلند»[۵] ایده استفاده از الگوریتم ژنتیک را در بهینه‌سازی‌های مهندسی مطرح کرد. ایده اساسی این الگوریتم انتقال خصوصیات موروثی توسط ژن‌هاست. (ژنها قطعاتی از یک کروموزوم هستند که اطلاعات مورد نیاز

[۱] Simulated Annealing
[۲] Iterative Deepening
[۳] Hyper Space
[۴] Genetic Algorithm - GA
[۵] John Holland

برای یک مولکول DNA یا یک پلی پپتید را دارند. علاوه بر ژنها، انواع مختلفی از توالی‌های مختلف تنظیمی در روی کروموزوم‌ها وجود دارد که در همانندسازی، رونویسی و... شرکت دارند.)[1]. فرض کنید مجموعه خصوصیات انسان توسط کروموزوم‌های او به نسل بعدی منتقل می‌شوند. هر ژن در این کروموزوم‌ها نماینده یک خصوصیت است. بعنوان مثال ژن ۱ می‌تواند رنگ چشم باشد، ژن ۲ طول قد، ژن ۳ رنگ مو و الی آخر. حال اگر این کروموزوم به تمامی، به نسل بعد انتقال یابد، تمامی خصوصیات نسل بعدی شبیه به خصوصیات نسل قبل خواهد بود. بَدیهیست که در عمل چنین اتفاقی رخ نمی‌دهد. در واقع بصورت همزمان دو اتفاق برای کروموزوم‌ها می‌افتد. اتّفاق اول موتاسیون(جهش)[2] است. موتاسیون به این صورت است که بعضی ژن‌ها بصورت کاملاً تصادفی تغییر می‌کنند. البته تعداد اینگونه ژن‌ها بسیار کم می‌باشد اما در هر حال این تغییر تصادفی همانگونه که پیشتر دیدیم بسیار مهم است. مثلاً ژن رنگ چشم می‌تواند بصورت تصادفی باعث شود تا در نسل بعدی یک نفر دارای چشمان سبز باشد، در حالی که تمامی نسل قبل دارای چشم قهوه‌ای بوده‌اند. علاوه بر موتاسیون اتفاق دیگری که می‌افتد و البته این اتفاق به تعداد بسیار بیشتری نسبت به موتاسیون رخ می‌دهد چسبیدن ابتدای یک کروموزوم به انتهای یک کروموزوم دیگر است.[3] این همان چیزیست که مثلاً باعث می‌شود تا فرزند تعدادی از خصوصیات پدر و تعدادی از خصوصیات مادر را با هم به ارث ببرد و از شبیه شدن تام فرزند به تنها یکی از والدین جلوگیری می‌کند.[10]

حال می‌توانیم اینگونه بیان کنیم که: الگوریتم ژنتیک ابزاری می‌باشد که توسط آن ماشین می‌تواند مکانیزم انتخاب طبیعی را شبیه سازی نماید. این عمل با جستجو در فضای مسأله جهت یافتن جواب برتر و نه الزاماً بهینه صورت می‌پذیرد.[13] الگوریتم ژنتیک را می‌توان یک روش جستجوی کلّی نامید که از قوانین تکامل بیولوژیک طبیعی تقلید می کند.[3] در واقع الگوریتم‌های

---

[1] ژن/»، به معنای زاینده است، و واژه «زن» و مصدر «زایش» و «زندگی» Jan واژه «ژن» در زبآنهای اروپائی، برگرفته از واژه اوستائی «
در فارسی امروزی نیز از همان ریشه است. در گویش کردی هنوز «زن» به همان صورت باستانی «ژن» خوانده می‌شود.

[2] Mutation

[3] این مسأله با نام Crossover شناخته می‌شود.

ژنتیک از اصول انتخاب طبیعی داروین برای یافتن فرمول بهینه جهت پیش‌بینی یا تطبیق الگو استفاده می‌کنند. الگوریتم‌های ژنتیک اغلب گزینه خوبی برای تکنیک‌های پیش‌بینی بر مبنای رگرسیون[۱] هستند.[10]

۱-۴- درباره علم ژنتیک

ژنتیک یا ژن‌شناسی[۲] بخشی از دانش زیست‌شناسی است که به وراثت و تفاوت‌های جانداران می‌پردازد. بوسیله قوانین و مفاهیم موجود در این علم می‌توانیم به تشابه یا عدم تشابه دو موجود نسبت به یکدیگر پی ببریم و بدانیم که چطور و چرا چنین تشابه و یا عدم تشابه در داخل یک جامعه گیاهی و یا جامعه جانوری، بوجود آمده‌است. علم ژنتیک علم انتقال اطلاعات زیست‌شناسی از یک سلول به سلول دیگر، از والد به نوزاد و بنابراین از یک نسل به نسل بعد است. ژنتیک با چگونگی این انتقالات که مبنای اختلالات و تشابهات موجود در ارگانیسم‌هاست، سروکار دارد. علم ژنتیک در مورد سرشت فیزیکی و شیمیایی این اطلاعات نیز صحبت می‌کند.

علم زیست‌شناسی، هرچند به صورت توصیفی از قدیمی‌ترین علومی بوده که بشر به آن توجه داشته‌است اما از حدود یک قرن پیش این علم وارد مرحله جدیدی شد که بعداً آن را ژنتیک نامیده‌اند و این امر انقلابی در علم زیست‌شناسی بوجود آورد. در قرن هجدهم، عده‌ای از پژوهشگران بر آن شدند که نحوه انتقال صفات ارثی را از نسلی به نسل دیگر بررسی کنند. ولی به دو دلیل مهم که یکی عدم انتخاب صفات مناسب و دیگری نداشتن اطلاعات کافی در زمینه ریاضیات بود، به نتیجه‌ای نرسیدند.[12]

۱-۵- تاریخچهٔ علم ژنتیک

---

[۱] Regression

رگرسیون در فرهنگ لغت به معنی بازگشت است، و اغلب جهت رساندن مفهوم بازگشت به یک مقدار متوسط یا میانگین به کار می‌رود، بدین معنی که برخی پدیده‌ها به مرور زمان از نظر کمّی به یک مقدار متوسط میل می‌کنند.

[۲] واژهٔ مصوب فرهنگستان زبان و ادب فارسی، دفتر نخست تا چهارم، ۱۳۷۶ تا ۸۵

اوّلین کسی که توانست قوانین حاکم بر انتقال صفات ارثی را شناسایی کند، کشیشی اتریشی به نام «گریگور مندل»[۱] بود که در سال ۱۸۶۵ این قوانین را که حاصل آزمایشاتش روی گیاه نخودفرنگی بود، ارائه کرد.[6,12] وی با ترکیب نژادهای گوناگون، نتایجی در مورد اثر متقابل خصوصیات به دست آورد. به عنوان مثال وقتی که گیاهان بلند را با گیاهان کوتاه ترکیب می‌کرد، بدون توجّه به اینکه کدامیک، گرده را اهداء کرده، فرزندان همه بلند می‌شدند. «مندل» نتیجه گرفت که خاصیت گیاه بلند (یا همان ژن که بعدها شناخته شد) پیروز شده و خاصیت گیاه کوتاه کنار گذاشته شده است.[۲] [3] اما متأسفانه جامعهٔ علمی آن دوران به دیدگاه‌ها و کشفیات او اهمیّت چندانی نداد و نتایج کارهای «مندل» به دست فراموشی سپرده شد. در سال ۱۹۰۰ میلادی کشف مجدّدِ قوانین ارائه شده از سوی «مندل»، توسط «درویس»، «شرماک» و «کورنز» باعث شد که نظریات او مورد توجه و قبول قرار گرفته و «مندل» به عنوان پدر علم ژنتیک شناخته شود.

در سال ۱۹۵۳ با کشف ساختمان جایگاه ژن‌ها از سوی «جیمز واتسون»[۳] و «فرانسیس کریک»[۴]، رشته‌ای جدید در علم زیست‌شناسی بوجود آمد که زیست‌شناسی مولکولی نام گرفت. با حدود گذشت یک قرن از کشفیات «مندل» در خلال سال‌های ۱۹۷۱ و ۱۹۷۳ در رشته زیست‌شناسی مولکولی و ژنتیک که اوّلی به برّرسی ساختمان و مکانیسم عمل ژن‌ها و دوّمی به برّرسی بیماری‌های ژنتیک و پیدا کردن درمانی برای آنها می‌پرداخت، ادغام شدند و رشته‌ای به

---

[۱] Gergor Mandel

[۲] شنیدن این مطلب خالی از لطف نیست:

امام محمدباقر(علیه السلام) نقل فرموده‌اند: یکی از انصار، خدمت پیامبر خدا رسید و عرض کرد: فرزند تازه متولد شده‌ام مویش فرفری، سوراخ‌های بینی‌اش بزرگ و بینی‌اش پهن است. با تمام ایمانی که به پاکدامنی همسرم دارم ولی شبیه او را نه در دایی‌هایش می‌شناسم و نه در نیاکانم. پیامبر خدا بعد از گفتگویی با همسر آن مرد فرمود: «ای مرد! هیچکس نیست مگر اینکه بین او و آدم، ۹۹ رگ و ریشه وجود دارد،که همگی در نسب نقش دارند. وقتی نطفه در رحم قرار می‌گیرد، این رگ‌ها به حرکت در می‌آیند و از خداوند می‌خواهند که فرزند به آنها شبیه باشد(رقابت ژنی). لذا این، از رگ‌هایی است که نه نیاکان تو، نه نیاکانِ نیاکان تو آن را درک کرده‌اند.»[3]

[۳] James Watson

[۴] Francis Crick

نام «مهندسی‌ژنتیک»[۱] را بوجود آوردند که طی اندک زمانی توانست رشته‌های مختلفی اعم از پزشکی، صنعت و کشاورزی را تحت‌الشّعاع خود قرار دهد.[12]

۱-۶- تکامل طبیعی (قانون انتخاب طبیعی داروین[۲])

هنگامی که لغت تنازع بقا به کار می‌رود اغلب بار ارزشی منفی آن به ذهن می‌آید. شاید همزمان قانون جنگل به ذهن برسد و حکم بقای قوی‌ترها !

البته همیشه هم قوی‌ترین‌ها برنده نبوده‌اند. مثلاً دایناسورها با وجود جثّه عظیم و قوی‌تر بودن در طی روندی کاملاً طبیعی بازیِ بقا و ادامه نسل را واگذار کردند در حالی که موجوداتی بسیار ضعیف‌تر از آنها حیات خویش را ادامه دادند. ظاهراً طبیعت، بهترین‌ها را تنها بر اساس هیکل انتخاب نمی‌کند! در واقع درست‌تر آنست که بگوییم طبیعت مناسب‌ترین‌ها را انتخاب می‌کند نه بهترین‌ها.

قانون انتخاب طبیعی[۳] بدین صورت است که تنها گونه‌هایی از یک جمعیت ادامه نسل می‌دهند که بهترین خصوصیات را داشته باشند و آنهایی که این خصوصیات را نداشته باشند به تدریج و در طی زمان از بین می‌روند.[10]

برای نمونه می‌توان گفت که: در میان یک گلّه گرگ با افرادِ دارای ژن‌های گوناگون، گرگی احتمال بقای بالاتر را دارد که قوی‌تر از دیگران است، چراکه هم امکان بیشتری برای جفت‌گیری

---

[۱] Genetic Engineering

[۲] مقوله ژنتیک با انتشار کتاب مهم و جنجالی چارلز داروین(Charles Darwin) انگلیسی که در آن فرضیه تکاملی خود را مطرح کرده بود در تاریخ ۲۹نوامبر سال ۱۸۵۹ میلادی به طور جدی مطرح شد. او در کتاب خود که در فارسی با نام «بنیاد انواع» شهرت پیدا کرده است، مبانی فکری و فرضیه خود دال بر سیر تکامل پیش رونده و رو به جلوی «خلقت تدریجی» را ارائه کرد. ناگفته پیداست که این مسأله با نظریات کلیسای آن زمان که به آفرینش آنی و «خلقت دفعی» انسان معتقد بود، قرار داشت.[13,6]

[۳] Natural Selection

و گسترش ژن خود را دارد و هم بیش از گرگ‌های ضعیف‌تر به شکار دست یافته، زنده می‌ماند. در یک گلّه گوزن نیز وضع به همین شکل است: گوزن ضعیف‌تر هم کمتر امکان تولید مثل می‌یابد و هم زودتر توسّط درندگان گرفتار می‌آید.[1] [1]

و یا در مثالی دیگر، فرض کنید گونهٔ خاصی از افراد، هوش بیشتری از بقیه افرادِ یک جامعه دارند. در شرایط کاملاً طبیعی، این افراد پیشرفت بهتری خواهند کرد و رفاه نسبتاً بالاتری خواهند داشت و این رفاه، خود باعث طول عمر بیشتر و باروری بهتر خواهد بود (توجه کنید شرایط، طبیعیست نه در یک جامعه سطح بالا با ملاحظات امروزی؛ یعنی طول عمر بیشتر در این جامعهٔ نمونه با زاد و ولد بیشتر همراه است). حال اگر این خصوصیّت (هوش) ارثی باشد بالطّبع در نسل بعدی همان جامعه تعداد افراد باهوش به دلیل زاد و ولد بیشترِ این‌گونه افراد، بیشتر خواهد بود. اگر همین روند را ادامه دهید خواهید دید که در طی نسل‌های متوالی دائماً جامعهٔ نمونهٔ ما باهوش و باهوش‌تر می‌شود. بدین ترتیب یک مکانیزم ساده طبیعی توانسته است در طی چند نسل عملاً افراد کم‌هوش را از جامعه حذف کند علاوه بر اینکه میزان هوش متوسط جامعه نیز دائماً در حال افزایش است.

بدین ترتیب می‌توان دید که طبیعت با بهره‌گیری از یک روش بسیار ساده (حذف تدریجی گونه‌های نامناسب و در عین حال تکثیر بالاتر گونه‌های بهینه)، توانسته است دائماً هر نسل را از لحاظ خصوصیّات مختلف ارتقاء بخشد.

البتّه آنچه در بالا ذکر شد به تنهایی توصیف کنندهٔ آنچه واقعاً در قالب تکامل در طبیعت اتفاق می‌افتد نیست. بهینه‌سازی و تکامل تدریجی به خودی خود نمی‌تواند طبیعت را در دسترسی به بهترین نمونه‌ها یاری دهد. اجازه دهید تا این مسأله را با یک مثال شرح دهیم:

---

[1] این مسألهٔ ساده راز تکامل در آفرینش است. اما در جوامع انسانی امری نکوهیده و به دور از اخلاق به شمار می‌آید. مانند این است که ما افراد معلول و کم‌توان را کنار بگذاریم تا قوی‌ترها رشد کنند.خوشبختانه وجود عاملی به نام اندیشه و خرد در وجود انسان، نیاز به زور و سلطه برای بقا را کم‌رنگ می‌کند و عاملی دیگر به نام فرهنگ و شرافت انسانی مانع ضعیف‌کشی می‌شود.

پس از اختراع اتومبیل به تدریج و در طیّ سال‌ها اتومبیل‌های بهتری با سرعت‌های بالاتر و قابلیت‌های بیشتر نسبت به نمونه‌های اولیه تولید شدند. طبیعیست که این نمونه‌های متأخّر حاصل تلاش مهندسانِ طرّاح جهت بهینه‌سازی طرّاحی‌های قبلی بوده‌اند. اما دقّت کنید که بهینه‌سازی یک اتومبیل، تنها یک «اتومبیل بهتر» را نتیجه می‌دهد.

اما آیا می‌توان گفت اختراع هواپیما نتیجه همین تلاش بوده است؟ یا فرضاً می‌توان گفت فضاپیماها حاصل بهینه‌سازی طرح اولیه هواپیماها بوده‌اند؟

پاسخ اینست که گرچه اختراع هواپیما قطعاً تحت تأثیر دستاورهای صنعت اتومبیل بوده است؛ اما به‌هیچ‌وجه نمی‌توان گفت که هواپیما صرفاً حاصل بهینه‌سازی اتومبیل و یا فضاپیما حاصل بهینه‌سازی هواپیماست. در طبیعت هم عیناً همین روند حکم‌فرماست. گونه‌های متکامل‌تری وجود دارند که نمی‌توان گفت صرفاً حاصل تکامل تدریجی گونه قبلی هستند.

در این میان آنچه شاید بتواند تا حدودی ما را در فهم این مسأله یاری کند مفهومیست به نام تصادف یا جهش.

به عبارتی طرح هواپیما نسبت به طرح اتومبیل یک جهش بود و نه یک حرکت تدریجی. در طبیعت نیز به همینگونه است. در هر نسل جدید بعضی از خصوصیات به صورتی کاملاً تصادفی تغییر می‌یابند سپس بر اثر تکامل تدریجی که پیشتر توضیح دادیم در صورتی که این خصوصیت تصادفی شرایط طبیعت را ارضاء کند حفظ می‌شود در غیر این‌صورت به شکل اتوماتیک از چرخه طبیعت حذف می‌گردد.

در واقع می‌توان تکامل طبیعی را به این‌صورت خلاصه کرد: جست‌وجوی کورکورانه (تصادف)[1] + بقای قوی‌تر.[10]

[1] Blind Search

۱-۷- رابطه تکامل طبیعی با روش‌های هوش مصنوعی

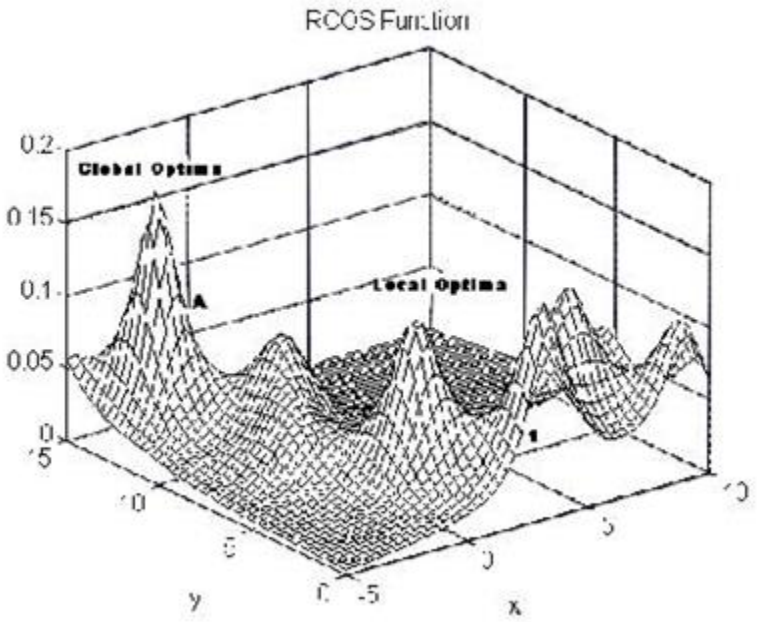
حال ببینیم که رابطه تکامل طبیعی با روش‌های هوش مصنوعی چیست. هدف اصلی روش‌های هوشمندِ به کار گرفته شده در هوش مصنوعی، یافتن پاسخ بهینه مسائل مهندسی است. بعنوان مثال اینکه چگونه یک موتور را طراحی کنیم تا بهترین بازدهی را داشته باشد یا چگونه بازوهای یک ربات را متحرک کنیم تا کوتاه‌ترین مسیر را تا مقصد طی کند (دقت کنید که در صورت وجود مانع یافتن کوتاه‌ترین مسیر دیگر به سادگی کشیدن یک خط راست بین مبدأ و مقصد نیست) همگی مسائل بهینه‌سازی هستند.

روش‌های کلاسیک ریاضیات دارای دو اشکال اساسی هستند. اغلب این روش‌ها نقطه بهینه محلی[۱] را بعنوان نقطه بهینه کلی[۲] در نظر می‌گیرند و نیز هر یک از این روش‌ها تنها برای مسأله خاصی کاربرد دارند. این دو نکته را با مثال‌های ساده‌ای روشن می‌کنیم.

به شکل زیر توجه کنید. این منحنی دارای دو نقطه ماکزیمم می‌باشد. که یکی از آنها تنها ماکزیمم محلی است. حال اگر از روش‌های بهینه‌سازی ریاضی استفاده کنیم مجبوریم تا در یک بازه بسیار کوچک مقدار ماکزیمم تابع را بیابیم. مثلاً از نقطه ۱ شروع کنیم و تابع را ماکزیمم کنیم. بدیهی است اگر از نقطه ۱ شروع کنیم تنها به مقدار ماکزیمم محلی دست خواهیم یافت و الگوریتم ما پس از آن متوقف خواهد شد. اما در روش‌های هوشمند، به ویژه الگوریتم ژنتیک بدلیل خصلت تصادفی آنها حتی اگر هم از نقطه ۱ شروع کنیم باز ممکن است در میان راه نقطه A به صورت تصادفی انتخاب شود که در این صورت ما شانس دست‌یابی به نقطه بهینه کلی را خواهیم داشت.

[۱] Local Optimal

[۲] Global Optimal



شکل ۱-۱-نقاط بهینهٔ محلی و بهینهٔ کلی

در مورد نکته دوم باید بگوییم که روش‌های ریاضی بهینه‌سازی اغلب منجر به یک فرمول یا دستورالعمل خاص برای حل هر مسأله می‌شوند(تنها برای مسأله خاصی کاربرد دارند)، در حالی که روش‌های هوشمند دستورالعمل‌هایی هستند که به صورت کلی می‌توانند در حل هر مسأله‌ای به کار گرفته شوند.[10] این نکته را پس از آشنایی با خود الگوریتم بیشتر و بهتر خواهید دید.

۱-۸- الگوریتم

در علوم کامپیوتر و ریاضیات، یک الگوریتم جستجو، الگوریتمی است که یک مسأله را به عنوان ورودی می‌گیرد و بعد از ارزیابی کردن راه‌حل‌های ممکن، یک راه‌حل برای آن مسأله برمی‌گرداند. هنگامی که مسأله‌ای را حل می‌کنیم معمولاً دنبال آن هستیم که بهترین راه‌حل و یا به بیان دیگر به یک حلّ بهینه از بین حل‌های ممکن برای مسأله برسیم.[9] به محدوده‌ای که جواب‌های مسأله قابل قبول می‌باشند به طوری که جواب بهینه هم یکی از زیرمجموعه‌های این

محدوده است «فضای جستجو»[1] نامیده می‌شود. هر نقطه از محدودهٔ فضای جستجو نشان دهندهٔ یکی از روش‌های حلّ مسأله می‌باشد.[5] و یا به بیانی ساده‌تر می‌توان گفت: مجموعهٔ راه‌حل‌های ممکن برای یک مسأله را فضای جستجو می‌نامند.

مهمترین عامل در حل هر مسأله، جستجو به دنبال پاسخ‌های احتمالی مسئله است.[15] به طور کلّی با دو دسته از الگوریتم‌ها مواجه هستیم؛ بعضی از الگوریتم‌ها که با عنوان الگوریتم‌های ناآگاهانه شناخته می‌شوند الگوریتم‌هایی هستند که از روش‌های ساده‌ای برای جستجوی فضای نمونه استفاده می‌کنند. در حالی که الگوریتم‌های آگاهانه با استفاده روش‌هایی مبتنی بر دانش در بارهٔ ساختار فضای جستجو، می‌کوشند تا زمان جستجو را کاهش دهند.

در کتاب «راسل» این الگوریتم‌ها به شکل زیر رده‌بندی شده‌اند:

- الگوریتم‌های ناآگاهانه
- الگوریتم‌های آگاهانه

۱-۸-۱- الگوریتم‌های جستجوی ناآگاهانه

یک الگوریتم جستجوی ناآگاهانه الگوریتمی است که به ماهیّت مسأله کاری ندارد. از این‌رو می‌توانند به طور عمومی طراحی شوند و از همان طراحی برای محدودهٔ عظیمی از مسائل استفاده کنند، این امر نیاز به طراحی انتزاعی دارد. از جمله مشکلاتی که این چنین الگوریتم‌هایی دارند این است که اغلب فضای جستجو بسیار بزرگ است و نیازمند زمان زیادی (حتی برای نمونه‌های کوچک) می‌باشد. از این‌رو برای بالا بردن سرعت پردازش غالبا از الگوریتم‌های آگاهانه استفاده می‌کنند.[9]

[1] Search Space

۱-۸-۱-الف- جستجوی لیست

الگوریتم‌های جستجویِ لیست شاید از ابتدایی‌ترین انواع الگوریتم‌های جستجو باشند. هدف آن پیدا کردن یک عنصر از مجموعه‌ای از کلیدهاست(ممکن است شامل اطلاعات دیگری مرتبط با آن کلید نیز باشد). ساده‌ترین این الگوریتم‌ها، جستجوی خطّی است که هر عنصر از لیست را با عنصر مورد نظر مقایسه می‌کند. زمان اجرای این الگوریتم از O(n) است وقتی که n تعداد عناصر در لیست باشد. اما می‌توان از روش دیگری استفاده کرد که نیازی به جستجوی تمام لیست نباشد. جستجوی دودویی اندکی از جستجوی خطّی بهتر است. زمان اجرای آن از O(log n) است. این روش برای لیستی با تعداد دادهٔ زیاد بسیار کارآمدتر از روش جستجوی خطّی است. اما در این روش لیست باید قبل از جستجو مرتب شده باشد. «جستجو با میان‌یابی» برای داده‌های مرتب شده با تعداد زیاد و توزیع یکنواخت، مناسب‌تر از جستجوی دودویی است. زمان اجرای آن به طور متوسّط O(log(log n)) است ولی بدترین زمان اجرای آن O(n) می‌باشد. الگوریتم «گراور»[۱] الگوریتم پلّه‌ای است که برای لیست‌های مرتب نشده استفاده می‌شود. الگوریتم Table Hash نیز برای جستجوی لیست به کار می‌رود. به طور متوسط زمان اجرای ثابتی دارد. اما نیاز به فضای اضافه داشته و بدترین زمان اجرای آن از O(n) است.[9]

۱-۸-۱-ب- جستجوی درختی

الگوریتم‌های جستجوی درختی، قلب شیوه‌های جستجو برای داده‌های ساخت یافته هستند. مبنای اصلی جستجوی درختی، گره‌هایی است که از یک ساختمان داده گرفته شده‌اند. هر عنصر که بخواهد اضافه شود با داده‌های موجود در گره‌های درخت مقایسه می‌شود و به ساختار درخت

[۱] Graver

اضافه می‌شود. با تغییر ترتیب داده‌ها و قرار دادن آنها در درخت، درخت با شیوه‌های مختلفی جستجو می‌شود. برای مثال سطح به سطح (جستجوی نخست-پهنا) یا پیمایش معکوس درخت (جستجوی نخست-ژرفا). از مثال‌های دیگر جستجوهای درختی می‌توان به جستجوی عمقی تکرار شونده، جستجوی عمقی محدود شده، جستجوی دوطرفه و جستجوی هزینه یکنواخت اشاره کرد.[9]

۱-۸-۱-پ- جستجوی گراف

بسیاری از مسائل در نظریهٔ گراف می‌تواند با الگوریتم‌های پیمایش درخت حل شوند، مثل الگوریتم دیکسترا[۱]، الگوریتم کروسکال[۲]، الگوریتم نزدیک‌ترین‌همسایه[۳] و الگوریتم پریم[۴]. می‌توان این الگوریتم‌ها را توسعه یافتهٔ الگوریتم‌های جستجوی درختی دانست.[9]

۱-۸-۲- الگوریتم‌های جستجوی آگاهانه

در یک جستجوی آگاهانه، از نوع خاصی از مسائل به عنوان راهنما استفاده می‌شود. یک گونهٔ خوب، یک جستجوی آگاهانه با کارایی قابل توجّهی نسبت به جستجوی ناآگاهانه به وجود می‌آورد. الگوریتم‌های برجستهٔ کمی از جستجوی آگاهانهٔ یک لیست وجود دارد. یکی از این الگوریتم‌ها Hash Table با یک تابع Hash که برمبنای نوع مسأله‌ای که دردست است می‌باشد. بیشتر الگوریتم‌های جستجوی آگاهانه، بسطی از درخت‌ها هستند. همانند الگوریتم‌های ناآگاهانه، این الگوریتم‌ها برای گراف‌ها نیز می‌توانند به کار روند.[9]

---

[۱] Dijkstra

[۲] Kruskal

[۳] Nearest Neighbor

[۴] Prim

دلیل نیاز به روش‌های جستجوی آگاهانه، نیاز به کاهش هزینهٔ زمانی مورد نیاز برای حلّ مسأله است. در واقع به این دلیل که ما تمایل داریم مسائل را در زمان کمتری حل کرده و از بررسی تمام حالات ممکن اجتناب کنیم، می‌بایست روشی برای تشخیص کیفیت مسیر (حتی به شکل نسبی) داشته باشیم.[15]

۱-۸-۲-الف- جستجوی خصمانه

در یک بازی مثل شطرنج، یک درخت بازی شامل تمام حرکات ممکن توسط هر دو بازیکن و نتایج حاصل از ترکیب این حرکات وجود دارد، و ما می‌توانیم این درخت را جستجو کرده و مؤثرترین استراتژی برای بازی را بیابیم. این چنین مسائلی دارای مشخصهٔ منحصر به فردی هستند. برنامه‌های بازی‌های رایانه‌ای، و همچنین فرم‌های هوش مصنوعی مثل برنامه‌ریزی ماشین‌ها، اغلب از الگوریتم‌های جستجو مثل الگوریتم مین‌ماکس[1] (می‌نیمیم مجموعه‌ای از ماکزیمم‌ها)، هرس کردن درخت جستجو و هرس کردن آلفا-بتا استفاده می‌کنند.[9]

۱-۹- مسائل NP-Hard

نمونه‌ای از مسائلی که نمی‌توان آنها را به روش سنتی حل کرد مسائل NP هستند.[13] مجموعه «ان‌پی-سخت» شامل چندهزار مسألهٔ مختلف با کاربردهای فراوان است که تاکنون برای آنها راه‌حلّ سریع و قابل انجام در زمان معقول پیدا نشده است و به احتمال زیاد در آینده نیز یافت نخواهد شد. این که راه‌حلّ سریعی برای آنها وجود ندارد هم اثبات شده‌است. البته ثابت شده است که اگر فقط برای یکی از این مسأله‌ها راه‌حل سریعی پیدا شود، این راه‌حل موجب حلّ سریع بقیهٔ مسأله‌ها

[1] Min-Max

خواهد شد. البته احتمال پیدا شدن چنین الگوریتمی ضعیف است. منظور از راه‌حلّ سریع آن است که زمان اجرای آن با اندازهٔ ورودی مسأله به صورت چندجمله‌ای رابطه داشته باشد.[17]

روش‌های مختلفی برای حلّ سریع ولی نزدیک به بهینه برای یک مسألهٔ NP-Hard وجود دارد:

- راه حلّ تقریبی قابل اثبات(الگوریتم‌های تقریبی): که در آن یک الگوریتم سریع برای حلّ مسأله ارائه می‌شود ولی اثبات می‌شود که اندازهٔ خروجی ضریبی از اندازهٔ خروجی بهینهٔ مسأله است.
- الگوریتم‌های مکاشفه‌ای: با این که الگوریتم‌هایی سریع هستند و به صورت تقریبی جواب را به دست می‌آورند، اما در مورد ضریب تقریب یا میزان خوبی الگوریتم اثباتی وجود ندارد. بسیاری از این الگوریتم‌ها به صورت تجربی آزمایش می‌شوند. برخی از این الگوریتم‌ها از «روش حریصانه» برای حل استفاده می‌کنند.

راه‌های معمول مقابله با چنین مسائلی عبارتند از:

- طراحی الگوریتم‌هایی برای پیدا کردن جواب‌های دقیق که استفاده از آنها فقط برای مسائل با اندازه کوچک صورت می‌گیرد.
- استفاده از «الگوریتم‌های مکاشفه‌ای»[1] که جواب‌هایی به‌دست می‌دهد که احتمالاً درست هستند.
- پیدا کردن زیرمسأله‌هایی از مسأله، یعنی تقسیم مسأله به مسأله‌های کوچکتر.

دو مسألهٔ زیر جزءِ مسائل NP-Hard می‌باشند:

- مسئله فروشنده دوره‌گرد

[1] Heuristic

- مسئله بزرگترین خوشه (پیدا کردن بزرگترین زیرگراف کامل)[1] [17]

اما مسائل مهم زیادی نیز وجود دارند که یافتن راه‌حل در آنها بسیار دشوار است. اما اگر راه‌حل را داشته باشیم، بررسی آن آسان می شود. این واقعیت منجر به مسائل NP-Complete problems شد. NP معرف Nondeterministic (چندجمله‌ای های غیرجبری) و به این معناست که امکان این وجود که راه‌حل را حدس زد و سپس آن را بررسی کرد.

برای سهولت کار، بررسی مسائل NP-Complete ، محدود به مسائلی است که پاسخ می‌تواند بله یا خیر باشد. به دلیل وجود کارهایی با نتایج پیچیده، دسته دیگری از مسائل با نام NP-Hard معرفی شده‌اند. این دسته مانند مسائل NP-Complete محدود نیستند.

یکی از ویژگی‌های مسائل NP آن است که یک الگوریتم ساده را (که ممکن است در نگاه اول بدیهی به نظر برسد) می‌توان برای یافتن راه‌حل‌های مفید به کار برد. اما بطور کلی، این روش، روش‌های ممکن زیادی را فراهم می‌کند و بررسی کردن تمام راه‌حل‌ها، فرآیند بسیار کندی خواهد بود.

امروزه، هیچکس نمی‌داند که آیا الگوریتم سریعتری برای یافتن جواب دقیق در مسائل NP وجود دارد یا خیر. و یافتن چنین الگوریتمی وظیفه مهمی است که به عهده محققان می‌باشد. امروزه اکثر مردم تصور می‌کنند که چنین الگوریتمی وجود ندارد و بنابراین به دنبال روش دیگری (جایگزین) هستند. و نمونه‌ای از روش جایگزین، الگوریتم ژنتیکی است. [13]

۱-۱۰- هیوریستیک[2]

[1] Maximum Clique Problem

[2] Heuristic

سیستم‌های پیچیدهٔ اجتماعی، تعداد زیادی از مسائلِ دارایِ طبیعتِ ترکیباتی را پیش روی ما قرار می‌دهند. مسیر کامیون‌های حمل‌ونقل باید تعیین شود، انبارها یا نقاط فروش محصولات باید جایابی شوند، شبکه‌های ارتباطی باید طراحی شوند، کانتینرها باید بارگیری شوند، رابط‌های رادیویی می‌بایست دارای فرکانس مناسب باشند، مواد اولیه چوب، فلز، شیشه و چرم باید به اندازه‌های لازم بریده شوند؛ از این دست مسائل بی‌شمارند. تئوری پیچیدگی به ما می‌گوید که مسائلِ ترکیباتی اغلب چندجمله‌ای[1] نیستند. این مسائل در اندازه‌های کاربردی و عملی خود به قدری بزرگ هستند که نمی‌توان جواب بهینهٔ آنها را در مدّت زمان قابل پذیرش به دست آورد. با این وجود، این مسائل باید حل شوند و بنابراین چاره‌ای نیست که به جواب‌های زیر بهینه بسنده نمود؛ به گونه‌ای که دارای کیفیّت قابل پذیرش بوده و در مدّت زمان قابل پذیرش به دست آیند. چندین رویکرد برای طراحی جواب‌های با کیفیّت قابل پذیرش تحت محدودیّت زمانی قابل پذیرش پیشنهاد شده است. الگوریتم‌هایی وجود دارند که می‌توانند یافتن جواب‌های خوب در فاصله مشخصی از جواب بهینه را تضمین کنند که به آنها الگوریتم‌های تقریبی می‌گویند. الگوریتم‌های دیگری هستند که تضمین می‌دهند با احتمال بالا جواب نزدیک بهینه تولید کنند که به آنها الگوریتم‌های احتمالی گفته می‌شود. جدای از این دو دسته، می‌توان الگوریتم‌هایی را پذیرفت که هیچ تضمینی در ارائه جواب ندارند اما بر اساس شواهد و سوابق نتایج آنها، به طور متوسط بهترین تقابل کیفیت و زمان حل برای مسأله مورد بررسی را به همراه داشته‌اند؛ به این الگوریتم‌ها، الگوریتم‌های هیوریستیک گفته می‌شود.

هیوریستیک‌ها عبارتند از معیارها، روشها یا اصولی برای تصمیم‌گیری بین چندین خط‌مشی و انتخاب اثربخش‌ترین برای دستیابی به اهداف موردنظر. هیوریستیک‌ها نتیجهٔ برقراری اعتدال بین دو نیاز هستند: نیاز به ساخت معیارهای ساده و در همان زمان توانایی تمایز درست بین انتخاب‌های خوب و بد.

[1] Polynomial

یک هیوریستیک می‌تواند حسابی سرانگشتی باشد که برای هدایت یک دسته از اقدامات به کار می‌رود. برای مثال، یک روش مشهور برای انتخاب طالبی رسیده عبارتست از فشار دادن محل اتصال به ریشه از یک طالبی نامزدِ انتخاب و سپس بو کردن آن محل؛ اگر بوی آن محل مانند بوی داخل طالبی باشد آن طالبی به احتمال زیاد رسیده است. این قاعده سرانگشتی نه تضمین می‌کند که تنها طالبی‌های رسیده به عنوان نامزد انتخاب شوند و نه تضمین می‌کند که طالبی‌های رسیده آزمایش‌شده، رسیده تشخیص داده شوند اما به هر حال این روش، اثربخش‌ترین روش شناخته شده است.

به عنوان مثالی دیگر از استفاده هیوریستیک‌ها، یک استاد بزرگ شطرنج را در نظر بگیرید که با انتخاب بین چندین حرکت ممکن روبرو شده است. وی ممکن است تصمیم بگیرد که یک حرکت خاص، اثربخش‌ترین حرکت خواهد بود زیرا موقعیتی فراهم می‌آورد که به نظر می‌رسد بهتر از موقعیت‌های حاصل از حرکت‌های دیگر باشد. به کارگیری معیار به نظر می‌رسد خیلی ساده‌تر از تعیین دقیق حرکت یا حرکاتی خواهد بود که حریف را مجبور به مات کند. این واقعیت که اساتید بزرگ شطرنج همواره پیروز بازی نخواهند بود نشان دهنده این است که هیوریستیک‌های آنها انتخاب اثربخش‌ترین حرکت را تضمین نمی‌کنند. نهایتا وقتی از آنها خواسته می‌شود که هیوریستیک خود را تشریح نمایند آنها فقط توصیفی ناقص از قواعدی ارائه می‌دهند و به نظر خود آنها، انجام آن قواعد از توصیف آنان ساده‌تر است.

خاصیت هیوریستیک‌های خوب این است که ابزار ساده‌ای برای تشخیص خطّمشی‌های بهتر ارائه دهند و در حالی که به صورت شرطی لازم، تشخیص خطّمشی‌های اثربخش را تضمین نمی‌کنند اما اغلب به صورت شرط کافی این تضمین را فراهم آورند. بیشتر مسائل پیچیده نیازمند ارزیابی تعداد انبوهی از حالت‌های ممکن برای تعیین یک جواب دقیق می‌باشند. زمان لازم برای یافتن یک جواب دقیق اغلب بیشتر از یک طول عمر است. هیوریستیک‌ها با استفاده از روش‌هایی که نیازمند

ارزیابی‌های کمتر هستند و جوابهایی در محدودیت‌های زمانی قابل قبول ارایه می‌نمایند، دارای نقشی اثربخش در حل چنین مسائلی خواهند بود.[Pearl, J. 1984.] [8,13,14,16]

۱-۱۰-۱- انواع الگوریتم‌های هیوریستیک

در حالت کلی سه دسته از الگوریتم‌های هیوریستیک قابل تشخیص است:

۱- الگوریتم‌هایی که بر ویژگی‌های ساختاری مسأله و ساختار جواب متمرکز می‌شوند و با استفاده از آنها الگوریتم‌های سازنده یا جستجوی محلی تعریف می‌کنند.

۲- الگوریتم‌هایی که بر هدایت هیوریستیک یک الگوریتم سازنده یا جستجوی محلی متمرکز می‌شوند به گونه‌ای که آن الگوریتم بتواند بر شرایط حساس (مانند فرار از بهینه محلی) غلبه کند. به این الگوریتم‌ها، متاهیوریستیک[۱] گفته می‌شود.

۳- الگوریتم‌هایی که بر ترکیب یک چارچوب یا مفهوم هیوریستیک با گونه‌هایی از برنامه‌ریزی ریاضی (معمولا روشهای دقیق) متمرکز می‌شوند.

هیوریستیک‌های نوع اول می‌توانند خیلی خوب عمل کنند (گاهی اوقات تا حد بهینگی) اما ممکن است در جواب‌های دارای کیفیت پایین گیر کنند. همانطور که اشاره شد یکی از مشکلات مهمی که این الگوریتم‌ها با آن روبرو می‌شوند افتادن در بهینه‌های محلی است، بدون اینکه هیچ شانسی برای فرار از آنها داشته باشند. برای بهبود این الگوریتم‌ها از اواسط دهۀ ۷۰، موج تازه‌ای از رویکردها آغاز گردید. این رویکردها شامل الگوریتم‌هایی است که صریحاً یا به صورت ضمنی تقابل بین ایجاد تنوع جستجو (وقتی علائمی وجود دارد که جستجو به سمت مناطق بد فضای جستجو می‌رود) و تشدید جستجو (با این هدف که بهترین جواب در منطقه

[۱] Meta Heuristic

مورد بررسی را پیدا کند) را مدیریت می‌کنند. این الگوریتم‌ها متاهیوریستیک نامیده می‌شوند.

از بین این الگوریتم‌ها می‌توان به موارد زیر اشاره کرد:

۱- بازپخت شبیه‌سازی شده.

۲- جستجوی ممنوع.

۳- الگوریتم‌های ژنتیک.

۴- شبکه‌های عصبی مصنوعی.

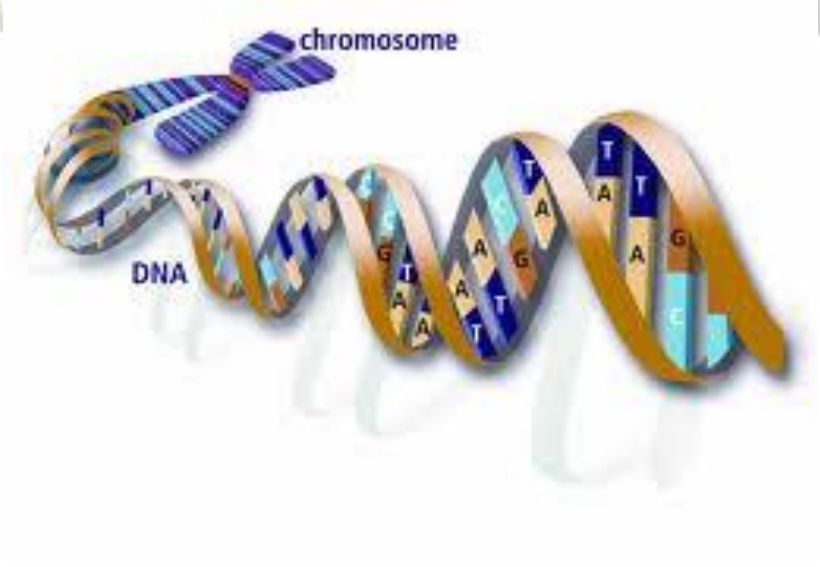
۵- بهینه‌سازی مورچه‌ای یا الگوریتم‌های مورچه.

[8,13,14,15]

که در این بین الگوریتم‌های ژنتیک از شهرت بیشتری نسبت به دیگر الگوریتم‌ها برخوردار است.

فصل دوم:

الگوریتم ژنتیک



۲-۱- مقدمه

الگوریتم‌های ژنتیک یکی از اعضای خانوادهٔ مدل‌های محاسباتی الهام گرفته شده از روند تکامل است. این الگوریتم‌ها راه‌حل‌های بالقوّهٔ یک مسأله را در قالب کروموزوم‌های ساده‌ای کد می‌کنند و سپس عملگرهای ترکیبی را بر روی این ساختارها اِعمال می‌کنند. الگوریتم‌های ژنتیک اغلب به عنوان روشی برای بهینه‌سازی توابع شناخته می‌شوند که البته دامنهٔ استفاده از این روش‌ها بسیار گسترده‌تر از این است.

در این فصل درباره کلّیات و جزئیات مربوط به الگوریتم ژنتیک صحبت خواهیم کرد، عملگرها و توابع مورد استفاده، به همراه بعضی تکنیک‌ها برای پیاده‌سازی آنها شرح داده خواهند شد. و در ادامه انواع و محدودیت‌های الگوریتم‌های ژنتیک را خواهیم گفت.

۲-۲- الگوریتم ژنتیک

الگوریتم ژنتیک که روش بهینه‌سازی الهام گرفته از طبیعت جاندار(موجودات زنده) است که می‌توان در طبقه‌بندی‌ها، از آن به عنوان یک روش عددی، جستجوی مستقیم و تصادفی یاد کرد. این الگوریتم، الگوریتمی مبتنی بر تکرار است و اصول اولیهٔ آن همانطور که پیشتر اشاره شد از علم ژنتیک اقتباس گردیده است و با تقلید از تعدادی از فرآیندهای مشاهده شده در تکامل طبیعی اختراع شده است و به طور موثّری از معرفت قدیمی موجود در یک جمعیت استفاده می‌کند، تا حل‌های جدید و بهبود یافته را ایجاد کند. این الگوریتم در مسائل متنوعی نظیر بهینه‌سازی، شناسایی و کنترل سیستم، پردازش تصویر و مسایل ترکیبی، تعین توپولوژی و آموزش شبکه‌های عصبی مصنوعی و سیستم‌های مبتنی بر تصمیم و قاعده به کار می‌رود.

در اینجا لازم می‌بینم که برای ورود به موضوع اصلی این نوشتار (الگوریتم ژنتیک) مروری بر مطالبی که پیشتر در مورد مباحث «ژنتیک» و «الگوریتم» ارائه شد داشته باشیم.

چنانکه قبل‌تر اشاره شد، علم ژنتیک، علمی است که دربارهٔ چگونگی توارث و انتقال صفحات بیولوژیکی از نسلی به نسل بعد صحبت می‌کند. عامل اصلی انتقال صفحات بیولوژیکی در موجودات

زنده کروموزوم‌ها[۱] و ژن‌ها[۲] می‌باشد و نحوه عملکرد آنها به گونه‌ای است که در نهایت ژن‌ها و کروموزوم‌های برتر و قوی مانده و ژن‌های ضعیف‌تر از بین می‌روند. به عبارت دیگر نتیجهٔ عملیات متقابل ژن‌ها و کروموزوم‌ها باقی ماندن موجودات اَصلح و برتر می‌باشد.

همچنین مجدداً یادآور می‌شویم که این الگوریتم برای بهینه سازی، جستجو و یادگیری ماشین مورد استفاده قرار می‌گیرد. اساس این الگوریتم قانونِ تکاملِ داروین (بقا بهترین) است که می‌گوید: موجودات ضعیف‌تر از بین می‌روند و موجودات قوی‌تر باقی می‌مانند. در واقع تکامل فرآیندی است که روی رشته‌ها صورت می‌گیرد، نه روی موجودات زنده‌ای که معرف موجودات رشته است. در واقع، قانون انتخاب طبیعی برای بقا می‌گوید که هر چه امکان تطبیق موجود بیشتر باشد بقای موجود امکان‌پذیرتر است و احتمال تولید مثل بیشتری، برایش وجود دارد. این قانون بر اساس پیوند بین رشته‌ها و عملکرد ساختمان‌های رمزگشایی شده آنها می‌باشد.

الگوریتم ژنتیک به دلیل تقلید نمودن از طبیعت دارای چند اختلاف اساسی با روش‌های جستجوی مرسوم می‌باشد که در زیر به تعدادی از آنها اشاره می‌کنیم.

- الگوریتم ژنتیک با رشته‌های بیتی کار می‌کند که هر کدام از این رشته‌ها کلّ مجموعه متغیرها را نشان می‌دهد حال آنکه بیشتر روش‌ها به طور مستقل با متغیرهای ویژه برخورد می‌کنند.
- الگوریتم ژنتیک برای راهنمایی جهت جستجو، انتخاب تصادفی انجام می‌دهد که به این ترتیب به اطلاعات مشتق نیاز ندارد.
- در الگوریتم ژنتیک روش‌های جستجو بر اساس مکانیزم انتخاب و ژنتیک طبیعی عمل می‌نمایند.

این الگوریتم‌ها مناسب‌ترین رشته‌ها را از میان اطلاعات تصادفی سازماندهی شده انتخاب می‌کنند. در هر نسل یک گروه جدید رشته‌ها با استفاده از بهترین قسمت‌های دنباله‌های قبلی و

---

[۱] Chromosome

[۲] Gene

بخش جدید اتفاقی برای رسیدن به یک جواب مناسب به وجود می‌آیند. با وجود اینکه الگوریتم‌ها تصادفی هستند ولی در زمره الگوریتم‌های تصادفی ساده نیستند. آنها به طور کارآمدی به اکتشاف اطلاعات گذشته در فضای جستجو می‌پردازند تا در یک نقطه جستجوی جدیدی با پاسخ‌های بهتر به سمت بهترین جواب پیش روند. هنگام پیش‌آمدسازی[1] الگوریتم‌های ژنتیک عمل پیش‌آمدسازی ساده را نمی‌پیمایند بلکه آنها داده‌های پیشین را با تفکّرِ انتخابِ جستجویِ جدید برای رسیدن پیشرفتِ مورد نظر توأم می‌کنند.

- الگوریتم ژنتیک در هر تکرار چند نقطه از فضای جستجو را در نظر می‌گیرد بنابراین شانس اینکه به یک ماکزیمم محلی همگرا شود کاهش می‌یابد.

در بیشتر روش‌های جستجوی مرسوم (روش گرادیان) قاعده تصمیم حاکم به این صورت عمل می‌کند که از این یک نقطه به نقطهٔ دیگر حرکت می‌کند. این روش‌ها می‌توانند در فضاهای جستجو دارای چند بیشینهٔ خطرناک باشند. زیرا ممکن است آنها به یک ماکزیمم محلی همگرا شوند. لیکن الگوریتم ژنتیک جمعیت‌های کاملی از رشته‌ها (نقاط) را تولید می‌کند سپس هر نقطه را به صورت انفرادی امتحان می‌کند و با ترکیب محتویات آنها یک جمعیت جدید را که شامل نقاط بهبود یافته است تشکیل می‌دهد. صرف نظر از انجام یک جستجو ملاحظهٔ هم‌زمانِ تعدادی نقطه در الگوریتم ژنتیک آنها را با ماشین‌های موازی تطبیق می‌سازد زیرا در اینجا تکامل هر نقطه یک فرآیند مستقل است. لذا الگوریتم ژنتیک فقط نیاز به اطلاعاتی در مورد کیفیت حل‌های ایجاد شده به وسیلهٔ هر مجموعه از متغیرها دارد، در صورتی که بعضی از روش‌های بهینه‌سازی نیاز به اطلاعات یا حتی نیاز به شناخت کامل از ساختمان مسأله و متغیرها دارند. چون الگوریتم ژنتیک نیاز به چنین اطلاعات مشخصی از مسأله ندارد بنابراین قابل انعطاف‌تر از بیشتر روش‌های جستجو است. همچنین الگوریتم ژنتیک از روش‌های جستجوی نوعی که برای راهنمایی جهت روش‌های جستجویشان از انتخاب تصادفی استفاده می‌کنند متفاوت است زیرا اگر چه برای تعریف روش‌های

[1] Randomization

تصمیم‌گیری از تصادف و شانس استفاده می‌کند ولی در فضای جستجو به صورت تصادفی قدم نمی‌زند.[13]

- الگوریتم ژنتیک از قوانین احتمالی پیروی می‌کند و نه از قوانین قطعی.[5]

۲-۳- مکانیزم الگوریتم ژنتیک

الگوریتم ژنتیک به عنوان یک الگوریتم محاسباتیِ بهینه‌سازی با در نظر گرفتن مجموعه‌ای از نقاط فضای جواب در هر تکرار محاسباتی به نحو مؤثری نواحی مختلف فضای جواب را جستجو می‌کند. در مکانیزم جستجو گرچه مقدار تابع هدف تمام فضای جواب محاسبه نمی‌شود ولی مقدار محاسبه شده تابع هدف برای هر نقطه، در متوسط‌گیری آماری تابع هدف برای هر نقطه، در متوسط‌گیری آماری تابع هدف در کلیه زیر فضاهایی که آن نقطه به آنها وابسته بوده دخالت داده می‌شود و این زیر فضاها به طور موازی از نظر تابع هدف متوسط‌گیری آماری می‌شوند. این مکانیزم را توازی ضمنی[1] می‌گویند. این روند باعث می‌شود که جستجوی فضا به نواحی از آن که متوسط آماری تابع هدف در آنها زیاد بوده و امکان وجود نقطه بهینه مطلق در آنها بیشتر است سوق پیدا کند. چون در این روش برخلاف روش‌های تک‌مسیری فضای جواب به طور همه جانبه جستجو می‌شود، امکان کمتری برای همگرایی به یک نقطه بهینه محلی وجود خواهد داشت.

امتیاز دیگر این الگوریتم آن است که هیچ محدودیتی برای تابع بهینه شونده، مثل مشتق‌پذیری یا پیوستگی لازم ندارد و در روند جستجو خود تنها به تعیین مقدار تابع هدف در نقاط مختلف نیاز دارد و هیچ اطلاعاتِ کمکی دیگری، مثل مشتق تابع را استفاده نمی‌کند. لذا می‌توان در مسائل مختلف اعم از خطی، پیوسته یا گسسته استفاده می‌شود و به سهولت با مسائل مختلف قابل تطبیق است.

در هر تکرار هر یک از رشته‌های موجود در جمعیت رشته‌ها، رمزگشایی شده و مقدار تابع هدف برای آن به دست می‌آید. بر اساس مقادیر به دست آمده تابع هدف در جمعیت رشته‌ها، به

[1] Implicit Parallelism

هر رشته یک عدد برازندگی نسبت داده می‌شود. این عدد برازندگی احتمال انتخاب را برای هر رشته تعیین خواهد کرد. بر اساس این احتمال انتخاب، مجموعه‌ای از رشته‌ها انتخاب شده و با اعمال عملکردهای ژنتیکی روی آنها رشته‌های جدید جایگزین رشته‌هایی از جمعیت اولیه می‌شوند تا تعداد جمعیت رشته‌ها در تکرارهای محاسباتی مختلف ثابت باشد.

مکانیزم‌های تصادفی که روی انتخاب و حذف رشته‌ها عمل می‌کنند به گونه‌ای هستند که رشته‌هایی که عدد برازندگی بیشتری دارند، احتمال بیشتری برای ترکیب و تولید رشته‌های جدید داشته و در مرحله جایگزینی نسبت به دیگر رشته‌ها مقاوم‌تر هستند. بدین لحاظ جمعیت دنباله‌ها در یک رقابت بر اساس تابع هدف در طیّ نسل‌های مختلف، کامل شده و متوسط مقدار تابع هدف در جمعیت رشته‌ها افزایش می‌یابد. بطور کلی در این الگوریتم ضمن آنکه در هر تکرار محاسباتی، توسط عملگرهای ژنتیکی نقاطی جدید از فضای جواب مورد جستجو قرار می‌گیرند توسط مکانیزم انتخاب، روند جستجوی نواحی از فضا را که متوسط آماری تابع هدف در آنها بیشتر است، کنکاش می‌کند. بر اساس سیکل اجرایی فوق، در هر تکرار محاسباتی، توسط عملگرهای ژنتیکی نقاط جدیدی از فضای جواب مورد جستجو قرار می‌گیرند توسط مکانیزم انتخاب، روند جستجو نواحی از فضا را که توسط آماری تابع هدف در آنها بیشتر است، کنکاش می‌کند. که بر این اساس، در هر تکرار محاسباتی، سه عملگر اصلی روی رشته‌ها عمل می‌کند؛ این سه عملگر عبارتند از: دو عملگر ژنتیکی و عملکرد انتخابی تصادفی.

«گلد برگ»[۱] الگوریتم ژنتیکی «جان هولند» را با عنوان الگوریتم ژنتیک ساده[۲] معرفی می‌کند؛ الگوریتم ژنتیک را از الگوریتم ژنتیک طبیعی اقتباس کردند.

در فصل یک گفتیم که: بدن همه موجودات زنده از سلول‌ها تشکیل شده است و در هر سلولی دسته کروموزوم‌های یکسانی وجود دارد. کروموزوم‌ها رشته‌هایی از DNA هستند که در واقع الگویی برای تمام بدن هستند. هر کروموزومی محتوی دسته‌هایی DNA است که ژن نامیده

[۱] Gold Berg

[۲] Simple Genetic Algorithm - SGA

می‌شوند و هر ژنی پروتئین خاصی را رمزگذاری می‌کند. اساساً می‌توان گفت که هر ژن، ویژگی خاصی (مثلا رنگ چشم) را رمزگذاری می‌کند. حالت‌های مختلف یک خصیصه (آبی، قهوه‌ای) آلل[۱] نامیده می‌شود. هر ژنی موقعیت خاص خود را بر روی کروموزوم دارد که این موقعیت لوکاس[۲] نامیده می‌شود. مجموعه کاملی از مواد ژنتیکی (همهٔ کروموزوم‌ها) ژنوم نامیده می‌شود. دستهٔ خاصی از ژن‌های موجود در ژنوم، ژنوتیپ نامیده می‌شود. ژنوتیپ به همراه تغییرات پس از تولّد، پایه و اساس فنوتیپ موجود زنده (ارگانیسم)، ویژگی‌های فیزیکی و ذهنی از قبیل رنگ چشم و هوش و غیره است.

در تولید مثل، ابتدا ترکیب(یا تغییر)[۳] اتفاق می‌افتد. ژن‌های والدین برای ایجاد کروموزوم‌های جدید ترکیب می‌شوند. سپس جنین تشکیل شده دچار تغییر می‌شود. جهش[۴] به این معناست که عناصر DNA کمی تغییر پیدا می‌کنند و این تغییرات اغلب نتیجه نسخه‌برداری غلط از ژن‌های والدین است. میزان شایستگی[۵] موجود زنده(جنین) به واسطه بقای آن اندازه گیری می‌شود.

در الگوریتم ژنتیک، مجموعه ای از متغیرهای طراحی را توسط رشته‌هایی با طول ثابت[۶] یا متغیر[۷] کدکذاری می‌کنند که در سیستم‌های بیولوژیکی آنها را کروموزوم یا فرد[۸] می‌نامند. هر رشته یا کروموزوم یک نقطهٔ پاسخ در فضای جستجو را نشان می‌دهد. به ساختمان رشته‌ها یعنی مجموعه‌ای از پارامترها که توسط یک کروموزوم خاص نمایش داده می‌شود ژنوتیپ[۹] و به مقدار

---

۱ Allele
۲ Locus
۳ Crossover
۴ Mutation
۵ Fitness
۶ Fixed Length
۷ variable
۸ Individual
۹ Genotype

رمزگشایی آن فنوتیپ[۱] می‌گویند. الگوریتم‌های وراثتی فرآیندهای تکراری هستند، که هر مرحلهٔ تکراری را نسل و مجموعه‌هایی از پاسخ‌ها در هر نسل را جمعیت نامیده‌اند.

الگوریتم‌های ژنتیک، جستجوی اصلی را در فضای پاسخ به اجرا می‌گذارند. این الگوریتم‌ها با تولید نسل[۲] آغاز می‌شوند که وظیفه ایجاد مجموعه نقاط جستجوی اولیه به نام «جمعیت اولیه»[۳] را بر عهده دارند و به طور انتخابی یا تصادفی تعیین می‌شوند. از آنجایی که الگوریتم‌های ژنتیک برای هدایت عملیات جستجو به طرف نقطه بهینه از روش‌های آماری استفاده می‌کنند، در فرآیندی که به انتخاب طبیعی وابسته است، جمعیت موجود به تناسب برازندگی افراد آن برای نسل بعد انتخاب می‌شود. سپس عملگرهای ژنتیکی شامل انتخاب[۴] ، پیوند(ترکیب)، جهش و دیگر عملگرهای احتمالی اِعمال شده و جمعیت جدید به وجود می‌آید. پس از آن جمعیت جدیدی جایگزین جمعیت پیشین می‌شود و این چرخه ادامه می‌یابد.

معمولاً جمعیت جدید برازندگی بیشتری دارد این بدان معناست که از نسلی به نسل دیگر جمعیت بهبود می‌آید. هنگامی جستجو نتیجه‌بخش خواهد بود که به حداکثر نسل ممکن رسیده باشیم یا همگرایی حاصل شده باشد و یا معیارهای توقف برآورده شده باشد.[13]

۲-۴- عملگرهای الگوریتم ژنتیک

به طور خلاصه الگوریتم ژنتیک از عملگرهای زیر تشکیل شده است:

۲-۴-۱- کدگذاری[۵]

این مرحله شاید مشکلترین مرحله حل مسأله به روش الگوریتم باشد. الگوریتم ژنتیک به جای اینکه بر روی پارامترها یا متغیرهای مسأله کار کند، با شکل کد شدهٔ آنها سروکار دارد. یکی

---

۱ Phenotype

۲ Seeding

۳ Initial Population

٤ Selection

٥ Encoding

از روشهای کد کردن، کد کردن دودویی می باشد که در آن هدف تبدیل جواب مسأله به رشته‌ای از اعداد باینری (در مبنای ۲) است.

۲-۴-۲- ارزیابی[۱]

تابع برازندگی را از اِعمال تبدیل مناسب بر روی تابع هدف یعنی تابعی که قرار است بهینه شود به دست می‌آورند. این تابع هر رشته را با یک مقدار عددی ارزیابی می‌کند که کیفیت آن را مشخص می‌نماید. هر چه کیفیت رشته جواب بالاتر باشد مقدار برازندگی جواب بیشتر است و احتمال مشارکت برای تولید نسل بعدی نیز افزایش خواهد یافت.

۲-۴-۳- ترکیب[۲]

مهمترین عملگر در الگوریتم ژنتیک، عملگر ترکیب است. ترکیب فرآیندی است که در آن نسل قدیمی کروموزوم‌ها با یکدیگر مخلوط و ترکیب می‌شوند تا نسل تازه‌ای از کروموزوم‌ها بوجود بیاید.

جفت‌هایی که در قسمت انتخاب به عنوان والد در نظر گرفته شدند در این قسمت ژن‌هایشان را با هم مبادله می‌کنند و اعضای جدید بوجود می‌آورند. ترکیب در الگوریتم ژنتیک باعث از بین رفتن پراکندگی یا تنوع ژنتیکی جمعیت می‌شود زیرا اجازه می‌دهد ژن‌های خوب یکدیگر را بیابند.

۲-۴-۴- جهش[۳]

---

[۱] Evaluation

[۲] Crossover

[۳] Mutation

جهش نیز عملگر دیگری هست که جواب‌های ممکن دیگری را متولد می‌کند. در الگوریتم ژنتیک بعد از اینکه یک عضو در جمعیت جدید بوجود آمد هر ژن آن با احتمال جهش، جهش می‌یابد. در جهش ممکن است ژنی از مجموعه ژن‌های جمعیت حذف شود یا ژنی که تا به حال در جمعیت وجود نداشته است به آن اضافه شود. جهش یک ژن به معنای تغییر آن ژن است و وابسته به نوع کدگذاری روش‌های متفاوت جهش استفاده می‌شود.

۲-۴-۵- رمزگشایی[۱]

رمزگشایی، عکسِ عمل رمزگذاری است. در این مرحله بعد از اینکه الگوریتم بهترین جواب را برای مسأله ارائه کرد لازم است عکس عمل رمزگذاری روی جواب‌ها یا همان عمل رمزگشایی اعمال شود تا بتوانیم نسخه واقعی جواب را به وضوح در دست داشته باشیم.

۲-۵- چارت الگوریتم به همراه شبه کد آن

در حالت کلی وقتی یک الگوریتم ژنتیکی اعمال می‌شود چرخه زیر را طی می‌کند:

ابتدا یک جمعیت اولیه از افراد به طور اتفاقی و بدون در نظرگرفتن معیار خاصی انتخاب می‌شود. برای تمامی کروموزوم‌های(افراد) نسل صفر مقدار برازش با توجه به تابع پردازش که ممکن است بسیار ساده یا پیچیده باشد تعیین می‌شود. سپس با مکانیزم‌های مختلف تعریف شده برای عملگر انتخاب زیرمجموعه‌ای از جمعیت اولیه انتخاب خواهد شد. سپس روی این افراد انتخاب شده عملیات برش و جهش در صورت لزوم با توجه به صورت مسأله اعمال خواهد شد.

حال باید این افراد که مکانیزم الگوریتم ژنتیک در موردشان اعمال شده است با افراد جمعیت اولیه (نسل صفر) از لحاظ مقدار برازش مقایسه شوند. (قطعاً توقّع داریم که افراد نسل اول با توجه به یکبار اعمال الگوریتم‌های ژنتیک روی آنان از شایستگی بیشتری برخوردار باشند، اما الزاماً چنین

[۱] Decoding

نخواهد بود.) به هرحال افرادی باقی خواهند ماند که بیشترین مقدار برازش را داشته باشند. چنین افرادی در مقام یک مجموعه به عنوان جمعیت اولیه برای مرحله بعدی الگوریتم عمل خواهد کرد.

هر مرحله تکرار الگوریتم یک نسل جدید را ایجاد می‌کند که با توجه به اصلاحاتی که در آن صورت پذیرفته است رو به سوی تکامل خواهد داشت. تذکر این نکته خالی از لطف نیست که هر چند الگوریتم‌های ژنتیک دارای پایه ریاضی متقن و مشخصی نیستند اما به عنوان یک مدل اجرایی و مطمئن که به خوبی نیز پیاده سازی می‌شود کارایی خود را نشان داده‌اند.

۱-۱ ۲-۵-۱- شبه کد و توضیح آن

در اینجا الگوریتم ژنتیک به صورت شبه کد بیان شده است.

```
PSEUDO CODE of GA
t:=0;                                   // start with an initial time
nit_population P(t);                    // initialize a usually random population of individuals
evaluate P(t);                          // evaluate fitness of all initial individuals of population
while (not done) do                     // test for termination criterion (time, fitness, etc.)
t:=t+1;                                 // increase the time counter
P':=select_parents P(t);                // select a sub-population for offspring production
recombine P' (t);                       // recombine the "genes" of selected parents
mutate P' (t);                          // perturb the mated population stochastically
evaluate P'(t);                         // evaluate it's new fitness
P:=survive P,P'(t);                     // select the survivors from actual fitness
```

end GA.

طرح کلی یک الگوریتم به شرح زیر می باشد:

۱-آغاز: جمعیت n کروموزومی به صورت تصادفی ایجاد کنید (راه‌حل‌های مناسب مسأله).

۲-ارزش گذاری: برازندگی f(x) هر کروموزوم X در جمعیت را ارزیابی کنید.

۳-جمعیت جدید: جمعیت جدیدی را تشکیل دهید. مراحل زیر را تکرار کنید تا جمعیت جدید کامل شود:

انتخاب: دو کروموزوم (والدین) را با توجه به برازندگی آنها از میان جمعیت انتخاب کنید(هر چه برازندگی بیشتر باشد شانس انتخاب بیبشتر است.)

ترکیب: با توجه به احتمال ترکیب شدن[۱] والدین را برای تشکیل فرزندان[۲] جدید با هم ترکیب کنید.

جهش: با توجه به احتمال جهش[۳] فرزندان را در هر لوکاس(موقعیت در کروموزوم)[۴] مورد جهش قرار دهید.

پذیرفتن: فرزندان جدید را در جمعیت جدید بگنجانید.

جایگزینی: جمعیت جدید ایجاد شده را برای روند الگوریتم بکار ببرید.

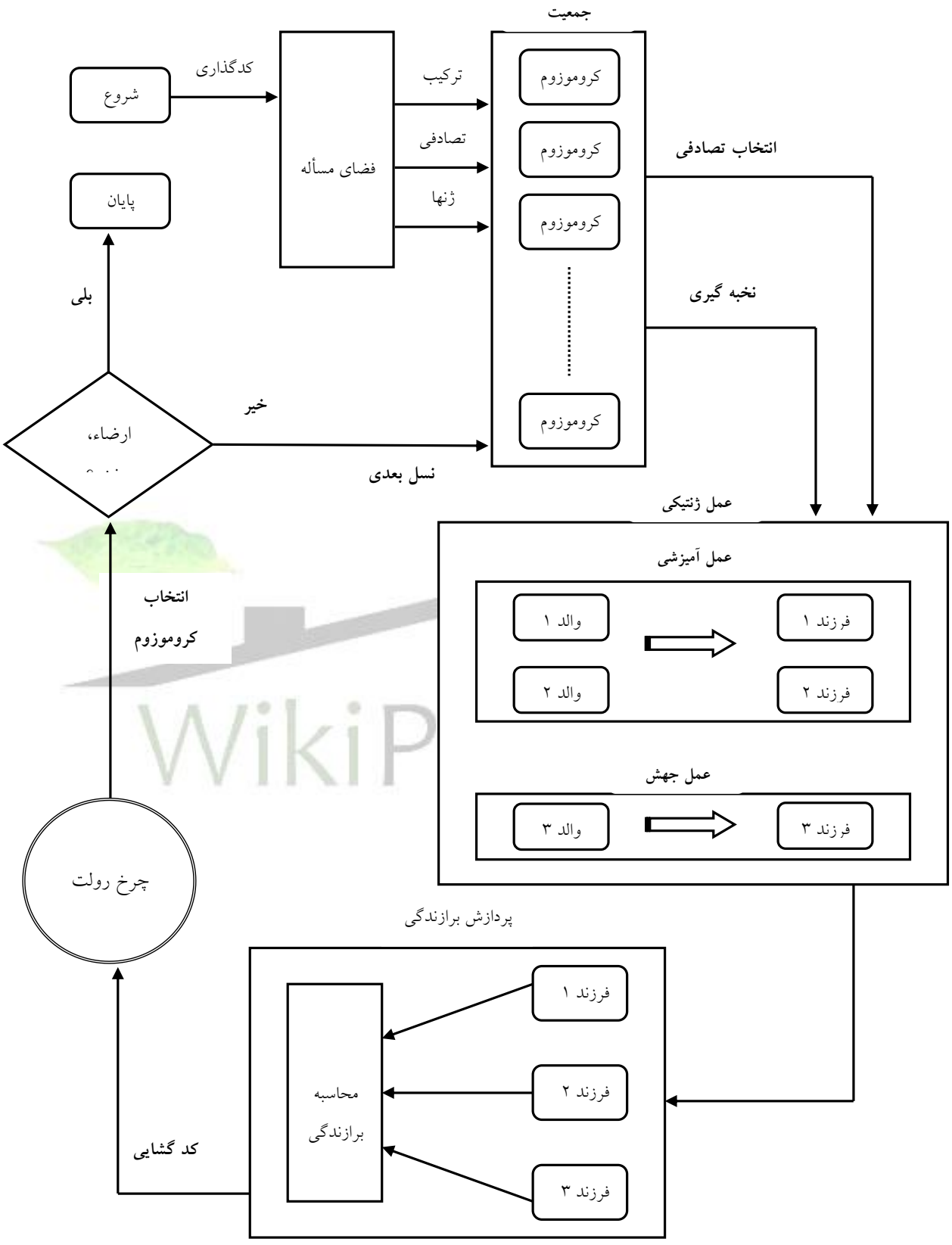
۲-۵-۲- چارت الگوریتم ژنتیک

[۱] Probability Crossover

[۲] Offspring

[۳] Probability Mutation

[۴] Lockus



شکل ۲-۱- چارت الگوریتم ژنتیک.

۲-۶- تابع هدف

تابع هدف، هدف و خواستهٔ ما از طرح مسأله است. یعنی، تابع هدف، شاخصی از نحوهٔ عملکرد افراد در فضای مسأله می‌باشد.[3]

۲-۷- روش‌های کد کردن

در این قسمت به بررسی کامل انواع کدینگ(کدگذاری) خواهیم پرداخت. هر چند همانطور که قبلاً هم گفته شد معمولاً از کد کردن دودویی استفاده می‌شود، اما در بسیاری از موارد کدینگ‌های دیگری به دلیل ماهیت مسأله مورد نیاز است.

انواع کدینگ(کدگذاری):

۱- کدینگ باینری[۱]

۲- کدینگ جایگشتی[۲]

۳- کدینگ مقدار[۳]

۴- کدینگ درختی[۴]

الگوریتم ژنتیک به جای این که بر روی پارامترها یا متغیرهای مسأله کار کند، با شکل کد شده آنها سروکار دارد. یکی از روش‌های کد کردن، کد کردن دودویی می‌باشد که در آن هدف تبدیل جواب مسأله به رشته‌ای از اعداد باینری (در مبنای ۲) است.

تعداد بیت‌هایی که برای کد گذاری متغییرها استفاده می‌شود، به دقت مورد نظر برای جواب‌ها، محدوده تغییر پارامترها و رابطه بین متغیرها وابسته است. رشته یا دنباله‌ای از بیت‌ها که به عنوان شکل کد شده یک جواب از مسأله مورد نظر می‌باشد، کروموزوم نامیده می‌شود. در

---

۱ Binary Coding

۲ Permutation Coding

۳ Value Coding

۴ Tree Coding

حقیقت بیت‌های یک کروموزوم نقش ژن‌ها در طبیعت را بازی می‌کنند. یکی از ویژگی‌های اصلی الگوریتم‌های ژنتیک آن است که به طور متناوب بر روی فضای کدینگ و فضای جواب کار می‌کنند. اعمال ژنتیکی بر روی فضای کدینگ یا کروموزوم‌ها اِعمال می‌شود، در حالی که انتخاب و ارزیابی بر روی فضای جواب عمل می‌نماید.

در طبیعت نیز به همین شکل است یعنی افراد (کروموزوم‌ها) در یک فضای حقیقی غیر کد شده در حالت فنوتیپ حضور دارند. در صورت کد شدن با هر مکانیزمی حالت ژنوتیپ خود را بروز می‌دهند.

ذکر این نکته ضروری است که هر زمان از کدینگ صحبت به میان می‌آید بطور پیش فرض منظور کد کردن از نوع باینری می‌باشد (رشته‌های دودویی). همچنین ساختمان داده مورد استفاده بطور پیش فرض، رشته می‌باشد.[13]

۲-۷-۱- کدینگ باینری

این تبدیل، تبدیل استاندارد در الگوریتم‌های ژنتیک می‌باشد. کد گذاری باینری ساده‌ترین کد گذاری و بهترین تبدیل برای عملگرهای ژنتیک است اما در مسائل پیچیده این نوع تبدیل چندان مناسب نیست چون معمولاً باعث می‌شود طول کروموزوم‌ها برای نگهداری اطلاعات پاسخ، بسیار بزرگ شود. در تبدیل باینری، اعضای جمعیت به رشته‌هایی از ۰ها و ۱ها تبدیل می‌شوند.

به عنوان مثال فرض کنید الگوریتم می‌خواهد ماکزیمم تابع F(x,y,z) را پیدا کند. در نظر بگیرید جستجو باید در اعداد صحیحِ مثبت و در محدوده ۰ تا ۲۵۵ انجام شود هر پاسخ ممکن شامل سه عدد X وY و Z می‌باشد.

طول هر عدد در محدوده مورد نظر مسأله در تبدیل باینری حداکثر ۸ بیت می‌باشد. اگر هر کروموزوم را به صورت XYZ در نظر بگیریم بنابراین برای پوشش دادن به تمام پاسخ‌های ممکن لازم است طول کروموزوم $3\times8=24$ بیت باشد. برای این مسأله کروموزوم C می‌تواند به شکل زیر باشد.

C=11010010 11100011 00110111

در همین مسأله اگر لازم باشد اعداد منفی نیز جستجو شود می‌توان یک بیت به ابتدای هر رشته اضافه کرد که مثلاً اگر ۰ باشد، عدد، مثبت و اگر ۱ باشد عدد، منفی در نظر گرفته شود.

000000001=1

100000001=-1

تبدیل اعداد اعشاری نیز می‌تواند با استفاده از چنین تمهیداتی انجام شود.

البته برای این نوع کدینگ دو روش در قسمت «نمایش رشته‌ها» به طور کامل بیان شده است.[13]

۲-۷-۲- کدینگ جایگشتی

در این روش، کروموزوم‌ها به صورت رشته‌ای از اعداد طبیعی نشان داده می‌شوند که هرکدام از این اعداد، مربوط به پارامتر ویژه‌ای در فضای حلّ مسأله است. ترتیب قرارگیری این اعداد مهم بوده و طول رشته دقیقاً با تعداد پارامترهای تعریف شده در مسأله برابر است. کاربرد این نوع کدگذاری در حل مسألهٔ فروشندهٔ دوره‌گرد است که تعریف آن در زیر آمده است.[3]

در بسیاری از مسایل مانند مسأله «فروشنده دوره گرد» با جایگشت‌های مختلفی از مجموعه راه‌حل‌ها رو به رو هستیم. در این مسأله تعدادی شهر داریم که فاصله میان آنها معلوم است و با شروع از یک شهر و ختم به همان شهر می‌بایست:

۱- از تمام شهرها فقط و فقط یکبار عبور نمائیم.

۲- کمترین مسافت ممکنه را طی نماییم.

نکته‌ای که در اینجا مهم است و باعث شده تا کدینگ باینری روش مناسبی برای این مسأله نباشد، این نکته است که حتما باید برش میان دو والد به نحوی صورت بگیرد که هیچ عنصری تکراری وجود نداشته باشد.

روش تک نقطه (در قسمت ترکیب که جلوتر بیان خواهد شد به طور کامل شرح داده خواهد شد) به این شکل اصلاح می‌شود که تمام قسمت قبل از نقطه برش در والد اول عیناً در فرزند کپی می‌گردد. بقیه ژن‌های والد اول که مطمئناً هنوز در فرزند تکرار نشده‌اند، مطابق با ترتیب قرار گرفتنشان در والد دوم در فرزند کپی می‌شوند.

(123456789) + (453689712) = (123456897)

شکل۲-۲- ترکیب تک نقطه.

البته در حالت جایگشتی می‌توان از PMX هم استفاده نمود.

After Crossover	Before Crossover
IHD | BCJ | AGEF	IHD | EFG | ACBJ
HCA | EFG | IBDJ	HGA | BCJ | IEDF

شکل۲-۳- ترکیب جایگشتی.

برای جهش نیز از مکانیزم زیر استفاده می‌شود:

دو موقعیت کاملاً دلخواه انتخاب شده و جای آنها با یکدیگر تعویض می‌شود.

(123456789) = (183456729)

شکل۲-۴- جهش: کدینگ جایگشتی.

[13]

۲-۷-۳- کد گذاری مقدار

در این نوع روش کدگذاری، کروموزوم‌ها می‌توانند هر نوع داده مرتبط با مسأله را در رشتهٔ خود اختیار نمایند. این داده‌ها می توانند از نوع اعداد حقیقی، عبارات منطقی، دستورات جهت‌یابی، داده‌های کد شده به صورت رشته‌های حرفی و... باشند.[3]

در این نوع کدگذاری تمامی مکانیزم‌های عملگر برش مانند حالت باینری، استفاده می‌شود. برای عملگر جهش نیز مانند زیر عمل می‌شود.

(1.29 5.68 **2.86 4.11** 5.55) = (1.29 5.68 **2.73 4.22** 5.55)

شکل ۲-۵- جهش: کدینگ مقدار.

یعنی به بعضی ژن‌ها بطور تصادفی عددی اضافه شده یا از آنها کم می‌گردد.[13]

۲-۷-۴- کدینگ درخت

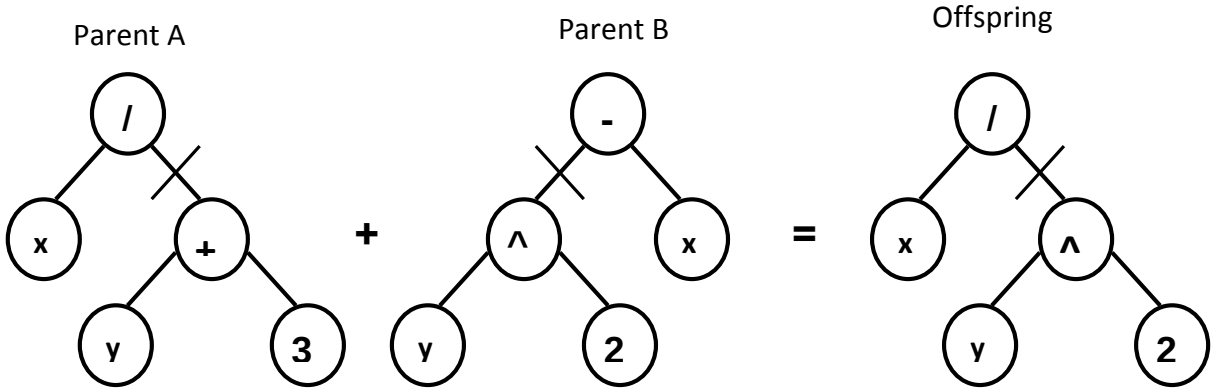
این روش که توسط «جان کوزا»[۱] توسعه یافت، بیشتر برای برنامه‌نویسی ژنتیک[۲] توسط نرم‌افزار «لیسپ»[۳] مورد استفاده قرار می‌گیرد،[5] که برنامه‌ها را به عنوان شاخه‌های داده در ساختار درخت نشان می‌دهد. در این روش تغییرات تصادفی می‌توانند با عوض کردن عملگرها یا تغییر دادن ارزش یک گره داده شده در درخت، یا عوض کردن یک زیردرخت با دیگری به وجود آیند.[10] یا به زبان ساده‌تر این طورمی‌توان گفت که: در این نوع کد گذاری یک درخت دودویی برای عبارت (کروموزوم) تشکیل می‌دهیم که معادل درخت پارس است و تمام اعمال مربوط به درخت پارس بر روی درخت قابل انجام است.[13]

[۱] John Koza

[۲] Genetic Programming

[۳] LISP

این روش بیشتر در نرم‌افزار کامپایلر فوق بکار برده می‌شود البته در نرم افزارهایی که تحت عنوان «کدهای الگوریتم ژنتیک» برای یک مسألهٔ خاص نوشته می‌شوند از این نوع روش رمزگذاری و آدرس‌دهی کروموزوم استفاده می‌شود.[5]



شکل ۲-۶- کدینگ درختی.

برای جهش هم یک نود دلخواه تغییر می‌کند.

۲-۸- نمایش رشته‌ها

نمایش مناسب رشته‌ها به ویژگی‌های فضای جستجو بستگی دارد ولی معمولاً به صورت رشته‌های دودوئی نمایش داده می‌شوند. در حل با الگوریتم ژنتیک متغیرها عموماً به صورت دودوئی و با طول رشته ثابت کد گذاری می‌شوند.

برای حلّ یک مسأله بهینه‌سازی به کمک الگوریتم ژنتیک ابتدا متغیرهای مسقل مسأله تشخیص داده می‌شود و سپس دامنهٔ تغییرات معین می‌شود.

بنابر اینکه هر متغیر پیوسته یا گسسته باشد یکی از روش‌های زیر را انتخاب می‌شود:

االف) متغیرهای پیوسته:

با فرض اینکه متغیر مورد نظر از $X_{\min}$ تا $X_{\max}$ در حال تغییر باشد و برای نشان دادن آن از یک رشته بیتی استفاده شده باشد.

ب) متغییرهای گسسته:

معمولاً در مسائلِ مربوط به طراحی استفاده از متغیرهای گسسته اجتناب‌ناپذیر است. برای مثال می‌توان از انتخاب جنس مواد و یا استفاده از جداول استاندارد نام برد. در هریک از موارد با فرض اینکه متغیر مورد نظر دارای n مقدار قابل قبول باشد. طول زیر رشته معادل آن از رابطه زیر به دست می‌آید.

برای مثال اگر متغیری دارای ۸ مقدار باشد، برای تمام حالات به یک زیر رشته با سه خانه نیاز است. حال اگر متغیر مورد نظر دارای ۱۰ مقدار متفاوت باشد به یک زیر رشته حداقل با ۴ خانه نیاز است. ولی زیر رشته‌ای با ۴ خانه توانایی ۲۴ مقدار متفاوت را دارد. برای حل این مشکل می‌توان به تعداد مورد نظر (در اینجا ۱۴ مورد) از مقادیری که در اختیار است به طور تصادفی انتخاب کرده و به صورت تکراری جایگزین کرد.

برای مثال فرض کنید در یک مسأله طراحی جنس قطعه مورد بررسی به عنوان متغیر بهینه‌سازی از میان جنس‌های آهن، چدن و برنج قابل انتخاب باشد برای بیان این متغیر $L_i = 2$ خواهد بود که به صورت زیر کد گذاری می‌شود:

آهن= ۰۰ برنج= ۰۱ چدن= ۱۰ آهن= ۱۱

همانگونه که نشان داده شده است برای فضای چهارم به صورت تصادفی یکی از سه جنس قابل قبول قرار داده شده است پس از اینکه تمام n متغیر طراحی بصورت زیر دسته‌هایی در نظر گرفته شدند با کنار هم قرار دادن این زیر رشته‌ها به یک رشته دودوئی از اعداد به طول L_c خواهیم رسید. این رشته از اعداد که معرّف یک طرح خواهد بود همان کروموزوم است. فرمول ریاضی که برای محاسبهٔ طول کروموزوم‌ها بکار می رود به شکل زیر می‌باشد:

$$L_c = \sum L$$

رابطه ۲-۱-محاسبه طول کروموزوم.

حال با داشتن مکان ابتدا و انتهای هر ژن می‌توان از روی کروموزوم مقدار هر ژن را محاسبه کرد. هر کروموزوم باید اطلاعاتی درباره راه‌حلی که نشان می‌دهد داشته باشد. یکی از روش‌های رمز گذاری که بیشتر مورد استفاده قرار می‌گیرد، رشته دوتایی[1] است. نمونه‌ای از کروموزوم‌ها در این حالت، در شکل زیر نشان داده شده است.

Chromosome 1	۱۱۰۱۱۰۰۱۰۰۱۱۰۱۱۰
Chromosome 2	۱۰۱۰۰۱۱۱۰۱۱۱۰۱۰۱

شکل ۲-۷- نمونه کروموزوم الگوریتم ژنتیکی.

هر کروموزوم به صورت رشته‌ای باینری نشان داده می‌شود. هر بخش موجود در رشته، ویژگی‌هایی از راه‌حل را نشان می‌دهد. احتمال دیگر آهن است که تمام رشته نمایان گر یک عدد باشد. البته راه‌های دیگر بسیاری برای رمز گذاری وجود دارد. رمز گذاری، عمدتاً به مسأله حل شده بستگی دارد. به طور مثال می‌توان مستقیماً اعداد صحیح و یا حقیقی را رمز گذاری کرد.

روش معمول برای رمز گشائی متغیرهای گسسته به این صورت است که جدولی از تمام حالات زیر رشته مربوط تشکیل می‌شود و هنگام رمز گشائی، معادل هر مقدار زیر رشته از جدول انتخاب می‌شود.[13]

۲-۹- انواع روش‌های تشکیل رشته

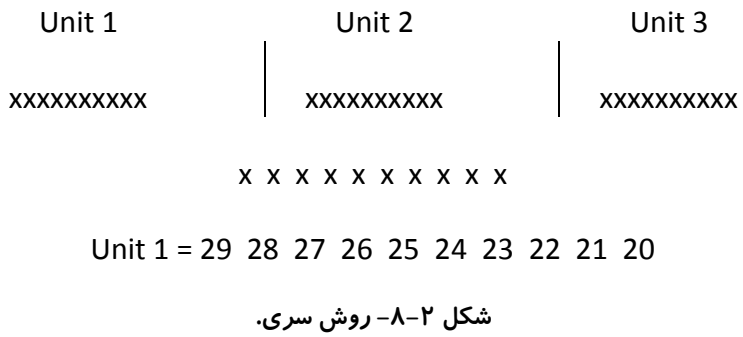
برای تشکیل رشته دو روش وجود دارد:

الف) روش سری

در روش سری بیت‌های عددی متناظر هر واحد مطابق شکل (۲-۸) کنار هم قرار می‌گیرند.

[1] Binary String

لازم به توضیح است که تا آخر برنامه هم محل هر واحد ثابت است و هیچ تغییری نمی‌کند، البته در ابتدای تشکیل رشته هیچ محدودیتی در انتخاب جایگاه وجود ندارد ولی بعد از تشکیل رشته تحت هیچ شرایطی نباید جابجا شوند.

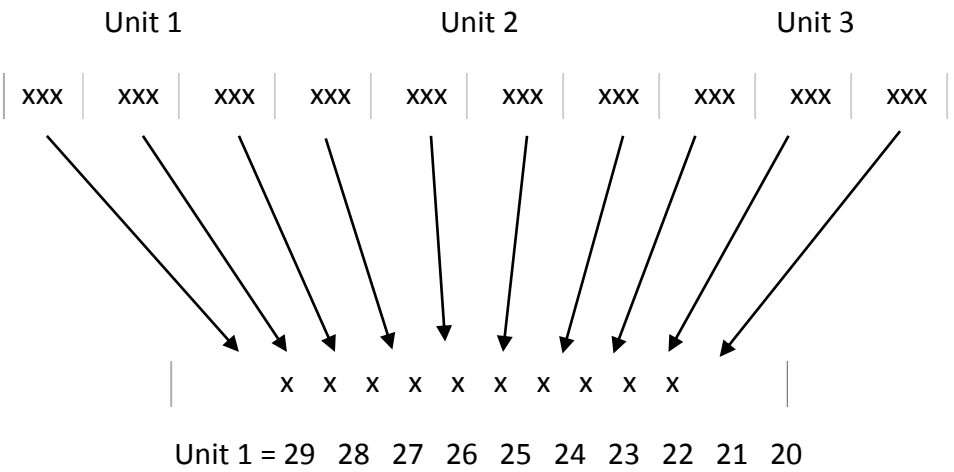


شکل ۲-۸- روش سری.

ب) روش محاطی

در روش محاطی بیت‌های عددی متناظر واحدها مطابق شکل (۲-۹) طوری کنار هم قرار می‌گیرند که در NP (تعداد واحدها) بیتی در کنار هم قرار می‌گیرند و هر بیت متعلق به یک واحد می‌باشد.

به بیان ساده‌تر اینکه، جایگاه‌های بیتی در هر NP بیت به طور متناوب متعلق به یک واحد خاص می‌باشد.



شکل ۲-۹- روش محاطی.

[13]

۲-۱۰- باز گرداندن رشته‌ها به مجموعه متغیرها

در روند اجرای الگوریتم ژنتیک لازم است رشته‌ها به متغییرها تبدیل شوند و به عبارت دیگر نمادهای مربوط به هر رشته (نماد ژنی) به دست آیند، تا از آنجا با قرار دادن متغیرها در تابع هدف، مقدار تابع هدف و در نتیجه ارزیابی آن رشته به دست آید. بنابراین یکی از قسمت‌های مهمّ الگوریتم ژنتیک قسمت رمز گشایی است که در مرحلهٔ ارزیابی انجام می‌شود.

برای باز گرداندن هر رشته به فضای جستجوی واقعی (فضای متغییرها) باید تعداد بیت(های) مربوط به تک‌تک متغیرها، نوع متغیرها (پیوسته یا گسسته) و محل هر متغیر در رشته مشخص باشند.

به هر حال با دانستن اطلاعات مربوط به هر متغیر، زیر رشته متناظر با این متغیر را استخراج و با توجه به محتویات آن مقدار واقعی متغیر را به دست می‌آوریم. در صورتی که متغیر X پیوسته باشد، این فرآیند به صورت زیر انجام می شود.

ابتدا زیررشته مربوط به این متغیر مشخص می‌شود. سپس مقدار واقعی آن در مبنای ۱۰ به دست می‌آید. اکنون برای به دست آوردن مقدار متغیر X با توجه داشته باشیم که صورتی که تعداد بیت مربوط به این متغیر برابر n باشد، در واقع ما متغیر X را در یک فاصله $[0,2^n-1]$ نگاشت کرده‌ایم که عدد به دست آمده در مبنای ۱۰ در واقع عدد مربوط به این فاصله می‌باشد. حال مقدار متغییر X با توجه به رابطه (۲-۲) محاسبه می‌شود.

$$X=\left[\frac{X_{\max}-X_{\min}}{2^N-1}\right]m+X_{\min}$$

رابطه ۲-۲- بازگرداندن رشته‌های متناظر به مجموعهٔ متغیرها.

در صورتی که متغیر مورد نظر ناپیوسته باشد، مقدار اعشاری حاصل از زیر رشته متناظر با متغیر ناپیوسته y در واقع نمایانگر اندیس دِرایه از یک بُردار است که مقدار آن درایه همان مقدار متغیر y می‌باشد. در واقع در حالت‌هایی که متغیرها ناپیوسته باشند، بجای رمز نمودن خود متغیرها اندیس‌های مربوط به آن متغیرها رمز می‌شوند و فرآیندهای ژنی نیز روی اندیس‌ها صورت می‌گیرد نه روی متغیرها.[13]

۲-۱۰-۱- تعداد بیت‌های متناظر با هر متغیر

تعداد بیت‌های متناظر با هر متغیر در صورتی که از رشته‌های بیتی استفاده شود، به صورت زیر به دست می‌آید.

$$2^n = \frac{X_{\max} - X_{\min}}{d}$$

رابطه ۲-۳- محاسبه تعداد بیت‌های متناظر با هر متغیر.

که در رابطه فوق $X_{\max}$ بیسترین مقدار مجاز متغیر و $X_{\min}$ کمترین مقدار مجاز متغیر X است و d نیز دقت مورد نظر برای این متغیر می‌باشد. اکنون با استفاده از رابطه (۲-۴) مقدار n محاسبه می‌کنیم. اگر در محاسبه مقدار n اعشاری شود، کوچکترین عدد صحیح بزرگتر یا مساوی آن را در نظر می‌گیریم.

$$n = \log_2\left[\frac{X_{\max} - X_{\min}}{d}\right]$$

رابطه ۲-۴- محاسبه مقدار n برای استفاده در رابطه ۲-۳.

همانطور که گفته شد، تعداد بیت مورد نیاز برای تک‌تک متغیرهای مسأله به دست می‌آید و در نهایت در صورتی که k متغیر داشته باشیم طول هر رشته برابر خواهد بود با:

$$N_s = N_1 + N_2 + ... + N_k$$

رابطه ۲-۵- محاسبه طول رشته برای k متغیر.

[13]

۲-۱۱- جمعیت

مفهوم جمعیت در الگوریتم ژنتیک شبیه به چیزی است که در زندگی طبیعی وجود دارد. برای مسأله گزاره‌هایی وجود دارند که می‌توانند به عنوان پاسخ، چه درست، چه غلط در نظر گرفته شوند. به این گزاره‌ها پاسخ‌های ممکن یا شدنی می‌گوییم. مثلاً اگر مسأله یافتن ماکزیمم یک تابع در مجموعه اعداد صحیح باشد، تمام اعداد صحیح می‌توانند به عنوان پاسخ شدنی مسأله در نظر گرفته شوند.

در الگوریتم ژنتیک به عنوان اولین مرحله لازم است مجموعه‌ای از جواب‌های شدنی به عنوان جمعیت اولیه ایجاد شود. اعضای این مجموعه معمولاً به صورت تصادفی انتخاب می‌شوند اما در الگوریتم‌های بهینه، از قیدهایی استفاده می‌شود تا جمعیت پراکندگی بیش از حد نداشته باشد. تعداد اعضای جمعیت به نوع مسأله بستگی دارد.

در واقع تعداد اعضا پارامتری است که با تغییر آن می‌توان دقت جواب‌ها و سرعت همگرایی جستجو را بهبود بخشید. در برخی مسائل یک جمعیت ۸ عضوی کاملاً مناسب است در حالی که در برخی یک جمعیت ۱۰۰ عضوی نیز کافی نیست. براساس تجربه بهتر است تعداد اعضای جمعیت عددی بین ۱۰ تا ۱۶۰ باشد.[13]

۲-۱۱-۱- ایجادجمعیت اولیه

پس از تعیین سیستم کدینگ و مشخص شدن روش تبدیل هر جواب به کروموزوم، باید جمعیت اولیه‌ای از کروموزوم‌ها تولید نمود. در اکثر موارد، جمعیت اولیه به صورت تصادفی تولید می‌شود. اما گاهی اوقات برای بالا بردن سرعت و کیفیت الگوریتم از روش‌های ابتکاری نیز برای

تولید جمعیت اولیه استفاده می‌گردد. در هر صورت عمومی‌ترین و راحت‌ترین روش، استفاده از یک رویکرد تصادفی می‌باشد.

اندازهٔ جمعیت اولیه معمولاً به سایز رشتهٔ کد شده وابسته است. به عنوان مثال اگر کروموزوم‌ها دریک مسأله ۳۲ بیتی هستند، قطعاً باید جمعیت انتخابی اولیه بیشتر از حالتی باشد که کروموزوم‌ها به عنوان مثال ۱۶ بیتی هستند.

معمولاً احتمال برش بین ۸۰ تا ۹۵ درصد، احتمال جهش بین نیم تا ۱ درصد و اندازه جمعیت بین ۲۰ تا ۳۰ در نظر گرفته می‌شود. آنگاه به کروموزوم‌های انتخاب شده با توجه به یک تابع برازش، مقداری حقیقی که نشان دهنده ارزش آنها است تخصیص داده می‌شود و مراحل الگوریتم‌های ژنتیک ادامه می‌یابد.[13]

۲-۱۱-۲- اندازه جمعیت

«گلدبرگ» برای محاسبه بهترین اندازه جمعیت برای کدهای دودوئی متغیرهای پیوسته تا طول حداکثر ۶۰ رشته مقدار زیر را پیشنهاد می‌کند.

$$N_{Pop} = 1.65 \times 2^{(0.21 \times Lc)}$$

رابطه ۲-۶- محاسبه اندازه جمعیت.

[13]

به طور مثال اگر طول هر کروموزوم برابر با ۲۵ باشد، آنگاه داریم: $N_{Pop} = 1.65 \times 2^{(0.21 \times 25)} = 62$.

و یا اگر طول هر کروموزوم برابر با ۳۵ باشد آنگاه خواهیم داشت: $N_{Pop} = 1.65 \times 2^{(0.21 \times 35)} = 270$ [۱].

اگر تعداد کروموزوم‌ها بسیار کم باشد، الگوریتم ژنتیک امکان انجام عمل ترکیب کمتری خواهد داشت و تنها بخشی کوچک از فضای جستجو کشف خواهد شد. از طرفی دیگر، اگر تعداد کروموزوم‌ها بسیار زیاد باشد، روند الگوریتم ژنتیک کند خواهد بود، بررسی نشان داده است که

[۱] طرح مثال از اینجانب(مهدی خلیلی‌نیا).

در نتیجه برخی محدودیت‌ها (که عمدتاً به کد گذاری و خود مسأله بستگی دارد) استفاده از جمعیت زیاد، ثمربخش نخواهد بود، زیرا این کار، مسأله را سریعتر از حالتی که جمعیتی متوسط استفاده می‌شود، حل نخواهد کرد.

در صورتی که تعداد اعضای جمعیت بسیار زیاد باشد، اگرچه وضعیت جستجو ممکن است به صورت بهتری نمایش داده شود زیرا با افزایش تعداد رشته‌ها، انتخاب رشتهٔ بهتر امکان‌پذیرتر می‌شود، ولی از طرفی نیازمندی‌های حافظه‌های کامپیوتر و زمان اجرای برنامه زیاد می‌شود. اگر تعداد اعضای جمعیت نیز کوچکتر از حدّ مشخصی باشد، جمعیت مورد نظر فقط قسمت کوچکی از فضای جستجو را نشان می‌دهد و ممکن است جستجو برای رسیدن به حلّ بهینه در این جمعیت موفقیت‌آمیز نباشد یا مستلزم صرف زمان زیادی باشد، در عمل تعداد اعضای جمعیت مقداری است که به صورت تجربی به دست آمده و نشان داده شده است. با این تعداد رشته در جمعیت، می توان به حل‌های مناسبی دست یافت این تعداد ۲ الی ۲/۵ برابر طول هر رشته می‌باشد.[13]

۲-۱۲- محاسبه برازندگی (تابع ارزش)

تابع برازندگی از اِعمال تبدیل مناسب بر روی تابع هدف یعنی تابعی که قرار است بهینه شود به دست می‌آید. این تابع هر رشته را با یک مقدار عددی ارزیابی می‌کند که کیفیت آن را مشخص می‌نماید. هر چه کیفیت رشته جواب بالاتر باشد مقدار برازندگی جواب بیشتر است و احتمال مشارکت برای تولید نسل بعدی نیز افزایش خواهد یافت. بسته به اینکه مسأله مورد نظر بیشینه‌سازی یا کمینه‌سازی باشد برازندگیِ بیشتر مترادف با بیشینه یا کمینه بودن تابع هدف خواهد بود، از آنجایی که الگوریتم ژنتیک طبیعتاً به دنبال بیشینه تابع است باید مسائل کمینه‌سازی به بیشینه‌سازی تبدیل شود.

چندین روش برای تبدیل تابع هدف برازندگی وجود دارد. ساده‌ترین حالت مساوی قرار دادن تابع برازندگی با تابع هدف است. این روش در مسائلی که تابع هدف بایستی بیشینه شود مناسب است.

برای تبدیل مسائل کمینه‌سازی به بیشینه‌سازی روش‌های مختلفی وجود دارد، با فرض اینکه مقدارِ ϕ_i تابع هدف معادل فرد i ام باشد، ساده‌ترین راه، کم کردن ϕ_i از یک مقدار ثابت C است بطوریکه به ازای تمام نسل‌ها $\phi_i < C$ باشد.

$$f_i = C - \phi_i$$

رابطه ۲-۷- تبدیل مسائل کمینه‌سازی به بیشینه‌سازی.

در صورتی که نتوانیم بزرگترین مقدار تابع هدف را حدس بزنیم می‌توانیم در هر نسل $\phi_{\max}$ و $\phi_{\min}$ یافته و برازندگی را به صورت زیر محاسبه کنیم.

$$f_i = (\phi_{\max} + \phi_{\min}) - \phi_i$$

رابطه ۲-۸- تبدیل مسائل کمینه‌سازی به بیشینه‌سازی.

روش دیگر، استفاده از تابع نمائی زیر است:

$$f_i = e^{-\phi_i}$$

رابطه ۲-۹- تبدیل مسائل کمینه‌سازی به بیشینه‌سازی.

معمولاً مسائل بهینه‌سازی دارای قیدهایی هستند که هنگام حل مسأله باید ارضاء شوند در صورتیکه قیدها کم یا ساده باشند و احتمال ارضاء شدن آنها به خودیِ‌خود زیاد باشد می‌توان در هر نسل افرادی را که قیدها را ارضاء نمی‌کنند با افراد جدید به طور تصادفی جایگزین کرد ولی در شرایط پیچیده قیدها به صورت تابع جریمه در تابع هدف منظور می‌شوند.

$$\phi_{inew} = P_i + \phi_i$$

رابطه ۲-۱۰- جایگزینی افراد جدید با افرادی که قیدهای مسأله را ارضاء نمی کنند.

علامت و مقدار P_i باید به گونه‌ای باشد که موجب شود افرادی از هر نسل که کمتر قیدها را ارضاء می‌کنند در نهایت از برازندگی کمتری بر خوردار باشند.[13]

۲-۱۳- انواع روش‌های انتخاب[۱]

در مرحله انتخاب، یک جفت از کروموزوم‌ها برگزیده می‌شوند تا با هم ترکیب شوند، عملگر انتخاب رابط بین دو نسل است و بعضی از اعضای نسل کنونی را به نسل آینده منتقل می‌کند، بعد از انتخاب، عملگرهای ژنتیک روی دو عضو برگزیده اعمال می‌شوند، معیار در انتخاب اعضاء ارزش تطابق آنها می‌باشد اما روند انتخاب حالتی تصادفی دارد.

شاید انتخاب مستقیم و ترتیبی به این شکل که بهترین اعضاء دوبه‌دو انتخاب شوند در نگاه اول روش مناسبی به نظر برسد اما باید به نکته‌ای توجه داشت؛ در الگوریتم ژنتیک ما با ژن‌ها روبه‌رو هستیم، یک عضو با تطابق پایین اگرچه در نسل خودش عضو مناسبی نمی‌باشد اما ممکن است شامل ژن‌های خوب باشد و اگر شانس انتخاب شدنش ۰ باشد، این ژن‌های خوب نمی‌توانند به نسل‌های بعد منتقل شوند. پس روش انتخاب باید به گونه‌ای باشد که به این عضو نیز شانس انتخاب شدن داده شود. راه‌حل مناسب، طراحی روش انتخاب به گونه‌ای است که احتمال انتخاب شدن اعضای با تطابق بالاتر بیشتر باشد. انتخاب باید به گونه‌ای صورت بگیرد که تا جایی که ممکن است هر نسل جدید نسبت به نسل قبلی‌اش تطابق میانگین بهتری داشته باشد.[13]

روش‌های متداول انتخاب عبارتند از:

- **انتخاب چرخ رولت**
- **انتخاب ترتیبی**

[۱] Select

- انتخاب بولتزمن
- انتخاب حالت پایدار
- نخبه سالاری
- انتخاب رقابتی
- انتخاب قطع سر
- انتخاب قطعی برینذل
- انتخاب جایگزینی نسلی اصلاح شده
- انتخاب مسابقه
- انتخاب مسابقه تصادفی

[13]

۲-۱۳-۱- انتخاب چرخ رولت

انتخاب چرخ رولت که اولین بار توسط «هولند» پیشنهاد شد یکی از مناسب‌ترین انتخاب‌های تصادفی بوده که ایدهٔ آن، احتمال انتخاب می‌باشد. احتمال انتخاب متناظر با هر کروموزوم، براساس برازندگیِ آن محاسبه شده که اگر f_k مقدار برازندگی کروموزوم k ام باشد، احتمال بقای متناظر با آن کروموزوم عبارت است از:

$$P_k = \frac{f_k}{\sum_{i=1}^{n} f_i}$$

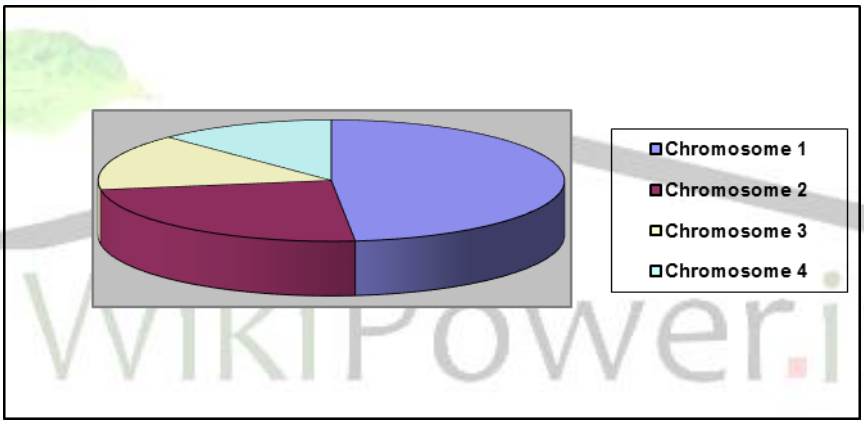
رابطه ۲-۱۱- احتمال انتخاب در روش چرخ رولت.

حال کروموزوم‌ها را براساس P_k مرتب کرده و q_k که همان مقادیر تجمعی P_k می باشد که به صورت زیر به دست می‌آید:

$$q_k = \sum_{i}^{k} P_i$$

رابطه ۲-۱۲- محاسبه مقادیر تجمعی در چرخ رولت.

چرخ رولت به این صورت عمل می‌کند که برای انتخاب هر کروموزوم یک عدد تصادفی بین یک و صفر تولید کرده و عدد مذکور در هر بازه‌ای که قرار گرفت، کروموزوم متناظر با آن انتخاب می‌شود. البته روش پیاده‌سازیِ چرخ رولت به این شکل است که ما یک دایره را در نظر گرفته و آن را به تعداد کروموزوم‌ها طوری تقسیم می‌کنیم که هر بخش متناظر با مقدار برازندگی کروموزوم مربوط باشد، حال چرخ را چرخانده و هر کجا که چرخ متوقف شد به شاخص چرخ نگاه کرده، کروموزوم مربوط به آن بخش انتخاب می‌گردد.



شکل ۲-۱۰- چرخ رولت.

انتخاب چرخ رولت، روشی است که نسبت مقدار تطابق، اعضاء را انتخاب می‌کند. این روش یک چرخ رولت را شبیه‌سازی می‌کند تا تعیین کند کدام اعضاء شانس باز تولید را دارند.

هر عضو به نسبت تطابقش، تعدادی از بخش‌های چرخ رولت را به خود اختصاص می‌دهد. سپس در هر مرحله انتخاب یک عضو و برگزیده می‌شود و روند آنقدر تکرار می‌شود تا به اندازه کافی، جفت برای تشکیل نسل بعد انتخاب گردد. این روش انتخاب را می‌توان به صورت زیر بیان کرد:

بردای مانند V در نظر بگیرید:

$$v=[1,...,M]$$

M تعداد عناصر بردار است اگر تعداد اعضای مجموعه N باشد، هر عضو i 1,...N دارای تطابقی مانند f_i می‌باشد. هر عضو i به نسبت f_i ، P_i بار در v تکرار می‌شود. هر چه f_i بیشتر باشد، عضو مکان‌های بیشتری را به خود اختصاص می‌دهد. عدد از تشکیل بردار v یک مقدار تصادفی $1\le r\le M$ انتخاب می‌شود. این مقدار به مکانی در بردار اشاره می‌کند که آن مکان خود معرّف عضوی از اعضای جمعیت است، به عنوان مثال اگر جمعیتی با N=4 داشته باشیم و تطابق اعضا عبارت باشد از:

$$f_1=10,\ f_2=10,\ f_3=15,\ f_4=25$$

مقدار مجموع تطابق‌ها عبارت است از:

$$\sum_{i=1}^{N} f_i=60$$

بردار v را برداری با ۶۰ عنصر در نظر می‌گیریم. این بردار به صورت زیر پر می‌شود. به عضو ۱، ۱۰ مکان، به عضو ۲، ۱۰ مکان، به عضو ۳، ۱۵مکان و به عضو ۴، ۲۵ مکان اختصاص می‌یابد.

$$v=\{1,1,..,1,2,2,...,2,3,3,...,3,4,4,...,4\}$$

حالا r بین ۱ تا ۶۰ به تصادف انتخاب می‌شود. فرض کنید r=32 نتیجه می‌شود:

$$v=[32]=32$$

پس عضو ۳ انتخاب می شود.[13]

۲-۱۳-۲- انتخاب حالت پایدار[۱]

در اکثر الگوریتم‌های ژنتیک که در مقالات ارائه شده‌اند، جمعیت جدید به طور کامل توسط فرزندان بوجود می‌آید و این فرزندان جایگزین والدین خود می‌شوند. در بعضی روش‌ها به برخی

[۱] Steady State Selection

از اعضای جمعیت قدیمی، اجازه حضور در جمعیت جدید، داده می‌شود. انتخاب حالت پایدار یکی از این روش‌هاست.

در این روش، فقط تعداد اندکی از اعضای جمعیت کنونی، با اعضای جدید جایگزین می‌شوند، به عبارت دیگر، بدترین اعضا با فرزندانی که از بهترین اعضا بوجود آمده‌اند تعویض می‌شوند اما بافت کلّی جمعیت، چندان تغییر نمی‌کند.[13]

۲-۱۳-۳- انتخاب نخبه گرایی[۱]

ایدهٔ نخبه‌سالاری یا گرایی، ویژگی تازه‌ای به پروسهٔ انتخاب اضافه می‌کند، در نخبه سالاری، بهترین عضو هر جمعیت، زنده می‌ماند و در جمعیت بعد حضور دارد، به عبارت دیگر عضوی که بالاترین تطابق را دارد به طور خودکار به جمعیت جدید منتقل می‌شود. این روش ابتدا در سال ۱۹۷۵ توسط «کندی جونز» معرفی شد. اعمال نخبه سالاری در الگوریتم ژنتیک، معمولاً باعث بهبود کارایی آن می‌شود.[13]

۲-۱۳-۴- انتخاب رقابتی[۲]

این روش تعدادی از اعضای جمعیت را به تصادف انتخاب می‌کند و سپس اگر شرطی خاص برقرار باشد، بهترین یا تعدادی از بهترین‌های آنها را به عنوان والد بر می‌گزیند، اگر شرط برقرار نشود، بدترین عضو یا تعدای از بدترین‌ها، در تشکیل جمعیت آینده به عنوان والد در نظر گرفته می‌شوند.

شکل استاندارد این روش، رقابت دوتایی یا باینری است و به شکل زیر می‌باشد:

۱- ۲عضو به تصادف انتخاب می‌شوند.

۲- مقدار r بین ۰ و ۱ به تصادف تعین می‌شود.

۳- پارامتر $0 \leq K \leq 1$ توسط کاربر تعیین می‌شود. (مثلاً K=۰,۷۵)

[۱] Elitist Selection

[۲] Rivalry Selection

۴- اگر $r < k$ عضو برتر و اگر $r \leq K$ عضو بدتر بین این دو عضو، به عنوان والد انتخاب می‌شود.

۵- دو عضو انتخاب شده برای رقابت، به جمعیت بر می‌گردند و می‌توانند دوباره در رقابت شرکت کنند.

روش انتخاب رقابتی می‌تواند به صورت رقابت n تایی نیز انجام شود.[13]

۲-۱۳-۵- انتخاب قطع سر

در این روش که توسط «گلدنبرگ» معرفی و ارائه شده، ابتدا یک عدد T که کوچکتر از صد می‌باشد، تعریف شده سپس کروموزوم‌ها را بر مبنای مقادیر برازندگی مرتب کرده و Tدرصدِ برتر را انتخاب می‌کنیم. حال از هر یک از آنها $\frac{100}{T}$ کپی به نسل بعد انتقال می‌دهیم مثلاً فرض کنید T=20 و تعداد کروموزوم‌ها نیز ۲۰ عدد باشد، بنابراین ۲۰٪ کروموزوم‌های اول یعنی $\frac{100}{5} = 4$ تای اول را از لیست مرتب شده انتخاب کرده و از هر کدام ۵ کپی در نظر می‌گیریم.[13]

۲-۱۳-۶- انتخاب قطعی برینذل

این روش که توسط «برینذل» معرفی و ارائه شده، به این صورت است که احتمال انتخاب برای هر کروموزوم طبق فرمول زیر محاسبه می‌شود:

$$(P_k = \frac{f_k}{\sum f_i})$$

رابطه ۲-۱۳- احتمال انتخاب در روش «قطعی برینذل».

و تعداد مورد انتظار برای هر کروموزوم نیز از رابطه به دست می‌آید.

$$e_k = P_k.(Pop\_Size)$$

رابطه ۲-۱۴- محاسبه مقادیر تجمعی در روش «قطعی بریندل».

حال هر کروموزوم بر طبق قسمت صحیح مقدار انتظار، به نمونه تخصیص داده می‌شود و سپس جمعیت بر حسب قسمت اعشار تعداد مورد انتظار، مرتب شده و به تعداد مورد نیاز جهت تکمیل جمعیت، به ترتیب از بالای لیست برداشته می‌شود.[13]

۲-۱۳-۷- انتخاب جایگزینی نسلی اصلاح شده

در این روش که توسط «وایتلی»[۱] ارائه شده، ابتدا کروموزوم‌ها براساس مقدار برازش منظم شده، سپس به تعداد نوزادان تولید شده از انتهای لیست کروموزوم‌ها حذف می‌گردند، آنگاه نوزادان جایگزین کروموزوم‌های حذف شده می‌شوند. مثلاً اگر ۱۰ کروموزوم وجود داشته باشد، ابتدا آنها را مرتب کرده و بعد اگر قرار باشد ۴ نوزاد نیز تولید شود پس از تولید نوزادان، آنها جایگزین ۴ کروموزوم آخر لیست می‌شوند.[13]

۲-۱۳-۸- انتخاب مسابقه[۲]

در این روش که توسط «گلدبرگ» ارائه شده، به تعداد Pop-Siz[۳]، مجموعه شامل چند عضو که از قبل مشخص شده، تولید کرده و در هر مجموعه بهترین عضو انتخاب می‌گردد. اندازه مجموعه فوق که به آن اندازه مسابقه گفته می‌شود معمولاً برابر ۲ فرض می‌گردد.[13]

۲-۱۳-۹- انتخاب مسابقه تصادفی

[۱] Whitley

[۲] Tournament Selection

[۳] تعداد کل جمعیت در هر نسل.

این روش که توسط «وزل» ارائه شده، مانند حالت قبل بوده، با این تفاوت که به جای این که مجموعه به صورت تصادفی انتخاب شود با کمک چرخ رولت انتخاب شده و بهترین آن به عنوان یک عضو از نسل جدید در نظر گرفته می‌شود.[13]

۲-۱۴- انواع روش‌های ترکیب

در طبیعت بقای نسل یکی از مهمترین فاکتورهاست و تنها عملگر ممکن برای این امر آمیزش است. در الگوریتم‌های ژنتیکی نیز آمیزش وجود دارد. آمیزش با تعویض ژن‌ها، بین دو کروموزوم انجام می‌گردد و هر کدام از کروموزوم‌ها خصوصیاتی از خود را به فرزندان انتقال می‌دهند. بدیهی است کروموزوم‌هایی که دارای برازندگی بیشتری هستند شانس بیشتری برای آمیزش دارند.

مهمترین عملگر در الگوریتم ژنتیک، عملگر ترکیب است. ترکیب، فرآیندی است که در آن نسل قدیمی کروموزوم‌ها با یکدیگر مخلوط و ترکیب می‌شوند تا نسل تازه‌ای از کروموزوم‌ها بوجود بیاید.

جفت‌هایی که در قسمت انتخاب، به عنوان والد در نظر گرفته شدند، در این قسمت ژن‌هایشان را با هم مبادله می‌کنند و اعضایی جدید بوجود می‌آورند.

براساس مباحث تئوری (که در ادامه بررسی خواهد شد) ترکیب اعضا با تطابق بالا باعث بوجود آمدن اعضایی می‌شود که از تطابق میانگین، تطابق بیشتری دارند. ترکیب در الگوریتم ژنتیک باعث از بین رفتن پراکندگی یا تنوع ژنتیکی جمعیت می‌شود، زیرا اجازه می‌دهد ژن‌های خوب یکدیگر را بیابند.[13]

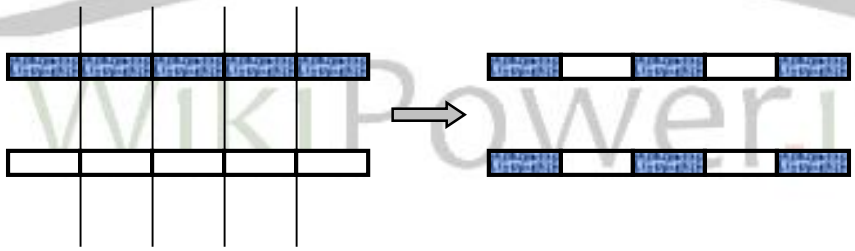
در ادامه متداولترین روشهای ترکیب شرح داده خواهد شد.

۲-۱۴-۱- جابه‌جایی دودوئی[1]

[1] Binary Crossover

روش‌های معمول جابجایی تک نقطه[۱]، دو نقطه[۲] و جابجایی یکنواخت[۳] می‌باشد. ساده‌ترین جابجا کردن، جابجایی تک نقطه‌ای است. در جابجایی تک نقطه‌ای، ابتدا جفت کروموزوم والد (رشته دودوئی) در نقطه مناسبی در طول رشته بریده شده و سپس قسمت‌هایی از نقطه برش، با هم عوض می‌شوند، بدین ترتیب دو کروموزوم جدید به دست می‌آید که هر نقطه از آن ژن‌هایی را از کروموزوم‌های الد به ارث می‌برند.

برای جابجایی چند نقطه‌ای[۴]، m موقعیت جابجا شدن، $k_i \in \{1,2,...,l-1\}$ که k_i نقطه جابجایی و l طول کروموزوم می‌باشد را به صورت تصادفی و بدون تکرار انتخاب می‌کنیم، سپس جهت ایجاد فرزندی جدید بیت‌های بین نقاط مشخص شده در والدین با هم عوض می‌شوند. این عملیات در شکل (۲-۱۱) نشان داده شده است. فلسفه انجام جابجایی این است که قسمت‌هایی از کروموزوم که بیان کننده سهم بسزایی در عملکرد بهتر یک عضو خاص هستند ممکن است در زیر رشته‌های همسایه یافت نشوند.



شکل ۲-۱۱- جابجایی چند نقطه.

به نظر می‌رسد نحوه عملکرد جابجایی در چند نقطه نسبت به روش همگرایی به مقادیر بالاتر برازندگی به پیشرفت و توسعه جستجو در فضای داده‌های مربوطه بیشتر کمک می‌کند، لذا جستجو در دامنه جواب قوی‌تر می‌شود.

---

۱ Single Point Crossover

۲ Two Point Crossover

۳ Monotonous Crossover

۴ Multi Crossover

«یانگ»[1] عملکرد جابجایی چند نقطه را مورد بررسی قرار داده و ثابت کرد که عملگرجابه جایی بیشتر، عملکرد الگوریتم ژنتیک را کاهش می‌دهد.

عملگرهای جابه‌جایی یک نقطه‌ای و چند نقطه‌ای در جایی اثر می‌کنند که کروموزوم دقیقاً در آن نقاط فقط می‌تواند جدا شود اما عملگر جابه‌جایی یکنواخت پتانسیل جابجا شوندگی را به تمام نقاط یک کروموزوم به صورت یکنواخت نسبت می‌دهد. به این معنی که احتمال جابجا شدن کروموزوم در هر نقطه برابر خواهد بود. یک الگوی بیان کننده عمل جابه‌جایی (به همان طولی که کروموزوم‌ها دارند) به صورت تصادفی ایجاد می‌شود و مقدار تعیین شده درهر بیت از این نمونه نشان می‌دهد که کدام یک از والدین به عنوان مرجع مقداردهی برای آن بین از فرزند خواهد بود. تعداد نقاط برش ثابت نیست ولی به طور متوسط برابر $\frac{l}{2}$ است.

$$P_1 = 1011000111$$
$$P_2 = 0001111000$$
$$Mask = 0011001100$$
$$O_1 = 0011110100$$
$$O_2 = 1001001011$$

دو کروموزوم اولیه (والدین) و الگوی عملگر جابه‌جایی و فرزندان حاصل را در نظر بگیرید. در اینجا مشاهده می‌شود که فرزند اول O_1 بدین صورت ایجاد شده است که اگر بیت مربوط در الگوی جابه‌جایی $(Mask)$ ۱ باشد آن بیت در O_1 برابر با مقدار بیت متناظر در P_1 و همچنین اگر بیت مربوط در الگوی جابه‌جایی ۰ باشد مقدار آن بیت در O_1 برابر با مقدار بیت متناظر در P_2 است. رشته O_2 با جابجا کردن P_1 و P_2 و در نظر گرفتن همین شیوه ایجاد شده است یعنی مقدار ۱ در رشته $Mask$ به معنی مقدار بیت متناظر در P_2 برای همان بیت در O_2 است.

مشخصاً عملگر جابه‌جایی یکنواخت همانند عملگر جابه‌جائی چند نقطه‌ای باعث کاهش خطای همگرایی ناشی از طول باینری استفاده شده و نوع کد کردن سری پارامترهای داده شده می‌شود. این مسأله کمک می‌کند که بر خطای همگرایی موجود در حالت جابه‌جایی تک نقطه‌ای در

[1] Jong

زیررشته‌های کوتاه غلبه کنیم بدون آنکه نیاز به دانستن مقادیر بیت‌های اعضا در کروموزوم‌های ارائه شده داشته باشیم.

«اسپرز»[۱] و «دیجانگ»[۲] نشان داده‌اند که چگونه جابه‌جایی یکنواخت به وسیله احتمال عوض شدن و جابجا شدن بیت‌ها پارامتری می‌شود. این پارامتر فوق‌العاده می‌تواند بدون توصیف یک همگرایی مربوط به طول رشته‌های استفاده شده در کنترل مقدار تغییر یافته در طول ترکیب‌بندی مجدد استفاده شود هنگامی که از عملگر جابه‌جایی یکنواخت در مقادیر حقیقی استفاده شود به آن «ترکیب‌بندی منفصل» گفته می‌شود.

در مقایسه‌هایی که بین عملگرهای دودوئی به هر دو صورت تئوری و تجربی انجام شده و نتایج به دست آمده نشان می‌دهد که هیچ یک از این عملگرها نمی‌تواند به طور مطلق بهترین بوده و اختلاف در سرعت این روش‌ها هم نمی‌تواند بیش از ۲٪ باشد.

عملگر جابه‌جایی دیگری که مطرح می‌شود به اسم «مخلوط» (شافل)[۳] است. یک نقطه قطع مجزا انتخاب می‌شود اما قبل از اینکه بیت‌ها تعویض شوند در هر دو والد بیت‌ها به صورت تصادفی جابجا می‌شوند. بعد از ترکیب‌بندی مجدد بیت‌ها در رشته فرزند جایگذاری می‌شوند. این عملگر نیز خطای همگرایی رشته‌ها را با جابه‌جایی تصادفی بیت‌ها در هر جایی که عملگر جابه‌جایی انجام می‌شود حذف می‌کند.

عملگر دیگری نیز عمل جابه‌جایی را مقیّد می‌کند که همیشه اعضای جدیدی ایجا کند در هر جایی که ممکن باشد. معمولاً این عملگر بدین صورت عمل می‌کند که مکان نقاط قطع را محدود می‌کند به گونه‌ای که نقاط قطع تنها جایی اتفاق می‌افتد که مقادیر ژن در دو کروموزوم متفاوت است.[13]

---

[۱] Spears

[۲] Dejong

[۳] Shuffle

۲-۱۴-۲- جابه‌جایی حقیقی[۱]

در کد گذاری حقیقی که کروموزوم‌ها به صورت برداری از اعداد حقیقی می‌باشند روش‌های زیادی برای عملگر جابه‌جایی حقیقی ارائه شده که اکثر آنها در دو دسته زیر خلاصه می‌شود:

۱- جابه‌جایی عمومی

۲- جابه‌جایی محاسباتی

عملگرهای جابه‌جایی عمومی با توسعه روش‌های جابه‌جایی دودوئی برای کد گذاری حقیقی تهیه می‌شود که مثال ساده از آن عملگر جابه‌جایی ساده می‌باشد که شامل جابه‌جایی تک نقطه، دو نقطه و چند نقطه است که مشابه همان حالت دودوئی می‌باشند با این تفاوت که در این جابه‌جایی یک بیت دودوئی $(1,0)$ یک عدد حقیقی در رشته است، با فرض اینکه $C=(C_1^1,...,C_n^1)$ و $C=(C_1^2,...,C_n^1)$ دو کروموزومی می‌باشند که تحت عمل جابه‌جایی قرار می‌گیرند و نقطه $i\in\{1,2,...,n-1\}$ نقطه جابجایی باشد دو کروموزوم جدید که از اعمال عملگر جابجایی ساده حاصل می‌شود و به صورت روابط زیر خواهد بود:

$$H1=(C_1^1,C_2^1,...,C_i^1,C_{i+1}^2,...,C_n^2)$$
$$H2=(C_1^2,C_2^2,...,C_i^2,C_{i+1}^1,...,C_n^1)$$

رابطه ۲-۱۵- جابجایی ساده.

جابجایی محاسباتی بر اساس مفهوم ترکیب خطی بردارها تهیه شده $Hk=(h_1^k,...,h_i^k,...,h_n^k)$ است. با این فرض که دو فرزند $k=\frac{1}{2}$ است بر اساس این عملگر تولید شده باشد، در این صورت تحت شرایط مختلف برای λ_1 و λ_2 ضرایب در روابط مذکور، سه نوع مختلف از این عملگر ایجاد می‌شود؛

[۱] Real Crossover

۱- جابجایی خطی[۱]؛ که در آن λ_1 و λ_2 هر دو حقیقی هستند.

۲- جابجایی سلبی[۲]؛ که در آن $\lambda_1+\lambda_2=1$ است.

۳- جابجایی برجسته(محدّب)[۳]؛ که در آن $\lambda_1+\lambda_2=1$ بوده و λ_1 و λ_2 هر دو حقیقی مثبت هستند.

اسامی «جابجایی خطی»، «جابجایی سلبی» و «جابجایی برجسته» از تئوری مجموعه‌های محدّب وام گرفته شده است.

عملگر «جابجایی برجسته» معمولاً بیشتر از بقیه کاربرد دارد. این عملگرها به طور شماتیکی در شکل --- نشان داده شده است.

$$h_i^1=\lambda_1 C_i^2+\lambda_2 C_i^1 \qquad h_1^1=\lambda_1 C_i^1+\lambda_2 C_i^2$$

رابطه ۲-۱۶- جابجایی محاسباتی.

[13]

۲-۱۴-۳- ترکیب تک‌نقطه‌ای

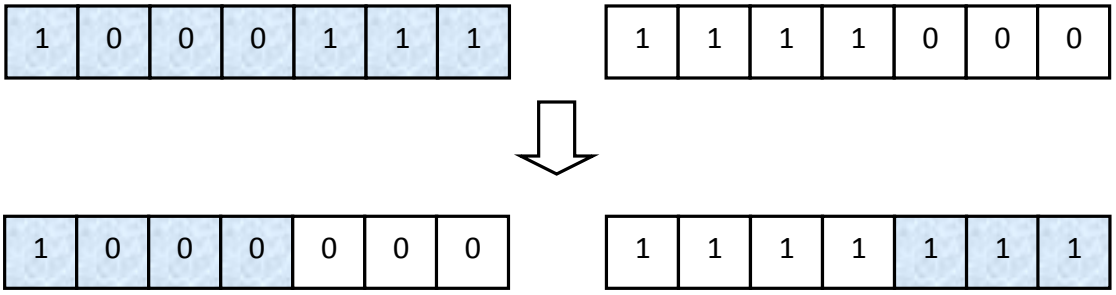
ترکیب تک نقطه‌ای دو کروموزوم را با انتخاب تصادف موقعیتی مانند P ، ترکیب می‌کند P مقداری کمتر یا مساوی طول کزوموزوم‌ها است. اگر تعداد (طول) ژن‌ها در کروموزوم‌ها N باشد از دو کروموزوم والد، دو فرزند به صورت زیر بوجود می‌آید.

یک فرزند با کپی کردن ژن‌های $1...(P_1-1)$ از کروموزوم والد اول و ژن‌های $P...N$ از کروموزوم دوم، ساخته می‌شود و فرزند دیگر به طور مشابه، این بار با کپی کردن ژن‌های $1...(P_1-1)$ از والد دوم و ژن‌های $P...N$ از والد اول، بوجود می‌آید. در این نوع ترکیب از دو والد، دو فرزند بوجود می‌آید. به عنوان مثال، این نوع ترکیب در شکل نشان داده شده است. در این مثال $P=4$ می‌باشد.

[۱] Linear Crossover

[۲] Affine Crossover

[۳] Convex Crossover



شکل ۲-۱۲- ترکیب تک نقطه‌ای.

لازم به ذکر است که اگر P برابر ۱ شود یا برابر طول کروموزوم‌ها، آنگاه دو والد بدون تغییر وارد جمعیت بعدی می‌شوند.[13]

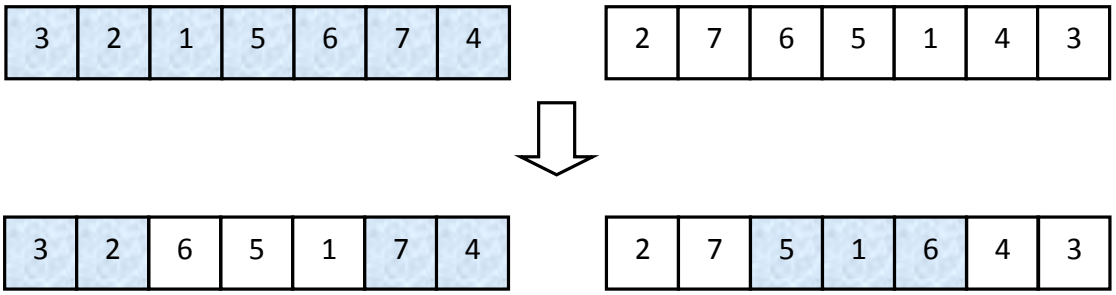
۲-۱۴-۴- ترکیب دو نقطه‌ای

در ترکیب دو نقطه‌ای، دو موقعیت P_1 و P_2 به عنوان موقعیت‌های ترکیب، به طور تصادفی بین ۱ و طول کروموزوم‌ها (N) انتخاب می‌شود. روش ایجاد فرزندان مانند ترکیب تک نقطه‌ای است.

فرزند اول، ژن‌های $1...(P_1-1)$ را از والد اول، ژن‌های $P_1...(P_2-1)$ را از والد دوم و ژن‌های $P_2...N$ را مجدداً از والد اول، به ارث می‌برد.

فرزند دوم، ژن‌های $1...(P_1-1)$ را از والد دوم، ژن‌های $P_1...(P_2-1)$ را از والد اول و ژن‌های $P_2...N$ را مجدداً از والد دوم، به دست می‌آورد.

در این روش ترکیب نیز، از یک جفت، دو فرزند بوجود می‌آید، در این روش احتمال اینکه والدها بدون تغییر به جمعیت بعد منتقل شوند، کمتر است. در شکل نمونه‌هایی از این ترکیب با موقعیت‌های ترکیب $P_1=2$ و $P_2=5$ نشان داده شده است.



شکل ۲-۱۳- ترکیب دو نقطه‌ای.

[13]

۲-۱۴-۵- ترکیب n نقطه‌ای

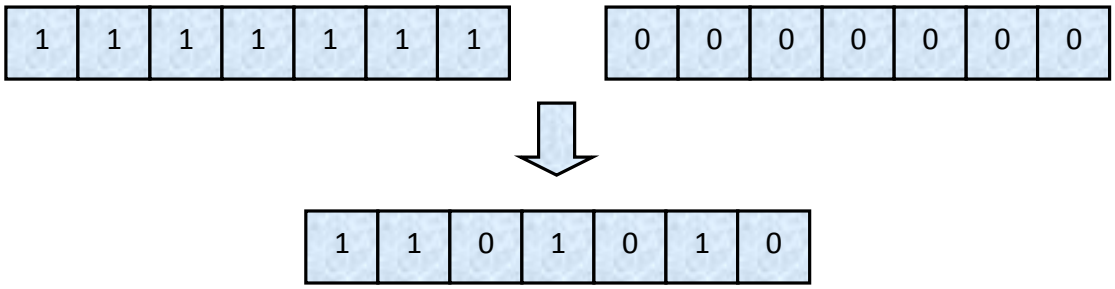
با انتخاب n موقعیت ترکیب و چیدن ژن‌ها مشابه آنچه در ترکیب تک نقطه‌ای و دو نقطه‌ای گفته شد، ترکیب n نقطه‌ای خواهیم داشت.[13]

۲-۱۴-۶- ترکیب یکنواخت

در ترکیب یکنواخت، هر ژن کروموزوم جدید به صورت جداگانه انتخاب می‌شود. هر ژن وابسته به موقعیتش به صورت تصادفی از یکی از دو والد انتخاب می‌شود، مثلاً ژن اول از والد اول، ژن دوم از والد دوم، ژن سوم از والد اول تا ژن آخر، برخلاف ترکیب‌هایی که قبلاً ذکر شد، این نوع ترکیب، یک فرزند بوجود می‌آورد، در واقع در این حالت از یک ماسک استفاده می‌شود.

جمعیت جدیدی که با ترکیب یکنواخت بوجود می‌آید، دارای تنوع ژنتیکی بیشتری نسبت به ترکیب‌های تک نقطه‌ای و دو نقطه‌ای می‌باشد به همین دلیل این نوع ترکیب در جمعیت‌هایی که اعضای کمی دارند اثر بهتری دارد تا جمعیت‌هایی که تعداد اعضای زیادی دارند.

در جمعیت‌های کوچک، ممکن است به تنوع ژنتیکی نیاز باشد تا روش، سریعتر همگرا شود، اما در جمعیت‌های بزرگ، معمولاً تنوع ژنتیکی لازم فراهم است. در شکل (۲-۱۴) نمونه‌ای از ترکیب یکنواخت مشاهده می‌شود.



شکل ۲-۱۴- ترکیب یکنواخت

[13]

۲-۱۴-۷- ترکیب حسابی

ترکیب حسابی به صورت زیر بیان می‌شود.

اگر A و B دو عضو از جمعیت فعلی باشند که به عنوان والد انتخاب شده‌اند از آنها دو فرزند a و b به صورت زیر بوجود می‌آید:

$$a = \delta A + (1-\delta)B$$

$$b = \delta B + (1-\delta)A$$

رابطه ۲-۱۷- ترکیب حسابی.

پارامتر δ مقداری در بازه $[0,1]$ می‌باشد که در هر ترکیب می‌تواند مقدار مختلفی داشته باشد.[13]

۲-۱۴-۸- ترتیب

این روش توسط «دیویس» معرفی گردیده و به روش OX معروف می‌باشد در این روش دو عدد را به صورت تصادفی به عنوان نقاط به دست آورده سپس قسمت مابین را در دو طرف کروموزوم ثابت نگه داشته ولی قسمت‌های دو طرف به این صورت به دست می‌آید که برای نوزاد

اول در والد دوم از ابتدای کروموزوم شروع کرده و آنهایی را که در قسمت ما بین نوزاد وجود ندارد در جای خالی قرار می‌گیرند.

با مثال زیر بحث روشن می‌شود:

والد اول: ۴۷۶/۳۵۹۸/۱۲

والد دوم: ۸۳۵/۶۴۱/۲۹

برای نوزاد اول قسمت ما بین والد یک، بدون تغییر جابجا می‌شود.

نوزاد اول: ---/۳۵۹۸/--

حال قسمت خالی از روی والد دوم پر می‌گردد.

نوزاد اول. ۷۴۶/۳۵۹۷/۱۲

برای نوزاد دوم نیز به همان صورت عمل می شود:

نوزاد دوم: ۳۵۹/۷۶۴۱/۸۲

[13]

۲-۱۴-۹- چرخه

این روش توسط «الیور اسمیت»[1] و «هولند» معرفی شده و به نام عملگر CX معروف می‌باشد و به اینگونه عمل می‌شود که ابتدا اولین ژن را عیناً از والد اول به نوزاد اول کپی کرده، سپس یک چرخه بین ژن‌هایی که در دو کروموزوم والد اول و دوم وجود دارد ایجاد می‌گردد، نقطه شروع همان ژن فوق می‌باشد. برای ایجاد چرخه باید ابتدا همان ژن را در کروموزوم دوم یافته و مکان آن را در نظر بگیریم. سپس ژن موجود در همان مکان از کروموزوم اول تثبیت کنیم. این عمل تا جایی که چرخه کامل شود ادامه می‌یابد (تا جایی که به نقطه شروع برسد). در این لحظه برای تکمیل نوزاد اول باید ژنهای باقیمانده از کروموزوم دوم را به همان ترتیب نوزاد اول جایگذاری نمود.

[1] Oliver Smeat

که مثال زیر بحث را روشن می کند.

والد اول: ۳۴۶۵۷۸۲۹۱

والد دوم: ۷۹۴۳۵۶۲۸۱

برای تولید نوزاد اول، ابتدا ژن از کروموزوم والد اول را منتقل کرده و عمل ادامه می‌یابد:

نوزاد اول: ----۵۷--۳

حال که چرخه کامل شد عددهای به دست آمده را در والد دوم حذف کرده باقیمانده به ترتیب عبارت است از ۹۴۶۲۸۱ که به همین حالت در جاهای خالی نوزاد اول قرار می‌گیرد:

نوزاد اول: ۳۹۴۵۷۶۲۸۱

برای نوزاد دوم نیز به همان صورت عمل می‌شود.

نوزاد دوم: ----۳۵--۷

که با روند فوق رشته زیر حاصل می‌شود.

نوزاد دوم: ۷۴۶۳۵۸۲۹۱

[13]

۲-۱۴-۱۰- محدّب

در این عملگر اگر والد اول P_1 و والد دوم P_2 باشد نوزاد اول و دوم به صورت زیر حاصل می‌شود:

نوزاد اول: $c_1 = \lambda_1 P_1 + \lambda_2 P_2$

نوزاد دوم: $c_2 = \lambda_1 P_2 + \lambda_2 P_2$

اگر $\lambda_1 = \lambda_2 = 0.5$ باشد به آن عملگر «تقاطعی متوسط» می‌گویند. اگر $\lambda_1 = 1.5$ و $\lambda_2 = -0.5$ باشد به آن «نسبت سلبی» گفته و در صورتی که λ_1 و λ_2 به صورت تصادفی از بازه $[-d, 1=d]$ انتخاب شود به آن «تقاطعی میانه توسعه یافته» می‌گویند.

«تقاطعی متوسط» توسط «دیویس» ، «تقاطعی میانه توسعه یافته» توسط «مولن بین» و «نسبت سلبی» توسط «رایت» ارائه شده‌اند، البته «چنگ» و «جن» حالتی را که λ_1 و λ_2 دو عدد تصادفی بوده و دارای شرط $\lambda_1 > 1$ و $\lambda_2 > 0$ و $\lambda_1 + \lambda_2 \leq 2$ باشد را تحت عنوان «عملگر خطی» ارائه نمودند.[13]

۲-۱۴-۱۱- بخش_نگاشته

این روش که توسط «گلدبرگ» و «لینگل» معرفی شده به روش PMX معروف بوده و در حقیقت همان عملگر دو نقطه برش می‌باشد که برای حالت خاصی ارائه شده است.

در این روش دو عدد به صورت تصادفی به عنوان نقاط برش به دست آورده، سپس قسمت مابین دو نقطه برش را در دو کروموزوم تعویض کرد و آنگاه قسمت‌های دو طرف طوری مقدار گذاری می‌شوند که در هیچکدام از دو کروموزوم، تکراری صورت نگیرد.

روش کار با یک مثال روشن شده است.

والد اول: ۴۳/۵۶۲۸/۷۹۱

والد دوم: ۶۵/۸۳۴۹/۲۱۷

حال برای تولید نوزاد به این صورت عمل می‌شود که قسمت مابین را عوض کرده بعد در والد اول از ابتدای کروموزوم شروع نموده هر عددی را که در قسمت مابین کروموزوم جدید نباشد عیناً نوشته و برای تکراری‌ها جای خالی قرار داده می‌شود سپس در والد دوم از ابتدای کروموزوم شروع کرده و هر عددی که در نوزاد جدید نباشد به جای محل خالی، گذاشته می‌شود تا کلیهٔ جاهای خالی پر گردد. برای نوزاد دوم نیز به همین صورت عمل می‌شود. بنابراین قسمت‌های میانی را جابجا کرده دو کروموزوم زیر حاصل می‌شود.

نوزاد اول: --/۸۳۴۹/---

نوزاد دوم: --/۵۶۲۸/---

سپس جاهای خالی نوزاد در صورت تکراری نبودن ژن مورد نظر پر می شود.

نوزاد اول: --/۸۳۴۹/۷-۱

حال جاهای خالی باقیمانده نوزاد اول پر می شود.

نوزاد اول: ۶۵/۸۳۴۹/۷۲۱

باتوجه به مطالب گفته شده، نوزاد دوم به دست زیر به دست می آید.

نوزاد اول: --/۵۶۲۸/-۱۷

نوزاد دوم: ۴۳/۵۶۲۸/۹۱۷

[13]

۲-۱۵- احتمال ترکیب

ترکیب لازم نیست در هر نسل اتفاق بیفتد، در واقع ممکن است نسل‌هایی بدون عملگر ترکیب به نسل‌های جدید تبدیل شوند. برای تعیین رخ دادن یا ندادن ترکیب از پارامتری به نام احتمال ترکیب P_c استفاده می شود که این پارامتر مقدار بین ۰ و ۱ است. از آنجاکه ترکیب نقشی اساسی در رشد مقدار میانگین تطابق جمعیت دارد، مقدار P_c بین ۰,۵ تا۰,۸ و بیشتر بین ۰,۷ تا ۰,۸ در نظر گرفته می‌شود.

اگر ترکیبی صورت نگیرد، فرزندان دقیقاً همانند والدین خواهند بود. اگر ترکیب صورت بگیرد، فرزندان از بخش‌هایی از کروموزوم‌های والدین به وجود می‌آیند. اگر احتمال ترکیب ۱۰۰٪ باشد، در این صورت همه فرزندان در نتیجه ترکیب به وجود آمده‌اند. اگر این احتمال ۰٪ باشد، کل نسل جدید در اثر نسخه برداری عینی کروموزوم های نسل قدیم بوجود آمده است (این به این معنی نیست که نسل جدید همانند نسل قدیم است). ترکیب با این امید صورت می‌گیرد که کروموزوم‌های جدید، حاوی بخش‌های مناسب کروموزوم‌های قدیمی است و در نتیجه

کروموزوم‌های جدید بهتر خواهد بود. با این حال خوب است که برخی از قسمت‌های نسل قدیم برای نسل بعدی باقی بمانند.[13]

۲-۱۶- تحلیل مکانیزم جابجایی

به دلیل اهمیّت مرحله ترکیب، بسیاری از تحقیقات دربارهٔ الگوریتم‌های ژنتیکی به روش‌های ترکیب و تحلیل آنها اختصاص یافته است. در این قسمت تعدادی از مهمترین این روش‌ها مورد بحث قرار می‌گیرد.

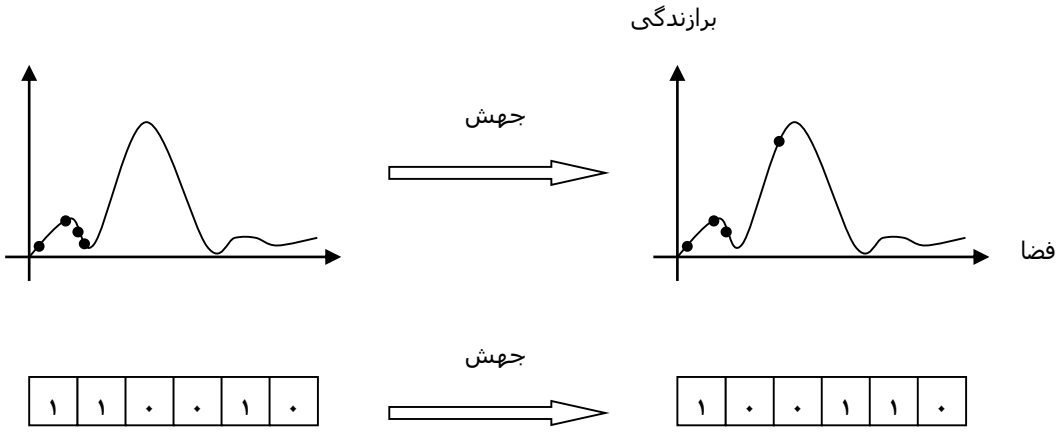
معمولاً کاربرهای الگوریتم‌های ژنتیکی تعداد نقاط شکست کروموزوم در عمل ترکیب را یک یا دو نقطه انتخاب می‌کنند. اگر روش ترکیب دو نقطه‌ای را گسترش دهیم، ترکیب چند نقطه‌ای خواهیم داشت که هر رشته به صورت حلقه‌ای از بیت‌ها در نظر گرفته می‌شود که به K قسمت تعریف شده است؛ K تعداد نقاط شکست می‌باشد. در ترکیب چند نقطه‌ای، قسمت‌های دو کروموزوم یکی در میان باهم تعویض می‌شوند.

بین اندازه جمعیت و نوع ترکیب ارتباط مستقیم وجود دارد؛ تجربیات نشان می‌دهد که ترکیب تک‌نقطه‌ای در جمعیت‌های کوچک بسیار مناسب است، اما برای جمعیت‌های بزرگتر ترکیب دو نقطه‌ای مناسبتر است. ترکیب‌های با تعداد نقاط شکست ثابت در جمعیت‌های کوچک امکان جستجوی بیشتری به ما می‌دهد. این واگرائی ذاتی در جمعیت‌های بزرگ لزوم جستجوی بیشتر را کاهش می‌دهد و بنابراین ترکیب دو نقطه‌ای جواب مناسب را به دست می‌آورد.[13]

۲-۱۷- جهش

در طبیعت برخی عوامل مانند تابش اشعهٔ ماوراءِ بنفش باعث به وجود آمدن تغییرات غیرقابل پیش‌بینی در کروموزوم‌ها می‌شوند. از آنجایی که الگوریتم‌های ژنتیکی از قانون تکامل پیروی می‌کنند در این الگوریتم‌ها نیز عملگر جهش با احتمال کم اعمال می‌شود. جهش باعث جستجو در فضاهای دست نخورده مسأله می‌شود می‌توان استنباط کرد که مهمترین وظیفه جهش

اجتناب از همگرایی به بهینه محلّی است. در اَشکال زیر نحوه جهش و کارکرد آن نمایش داده شده است:



شکل ۲-۱۵- شبیه سازی جهش به کمک نمودار.

به تعبیری دیگر می‌توان جهش را مشابه شروع مجدد تصادفی الگوریتم «تپّه نوردی»[۱] هنگام گیر افتادن در فلات دانست.

در الگوریتم ژنتیک نیز بعد از اینکه یک عضو در جمعیت جدید به وجود آمد، هر ژن آن با احتمال جهش[۲]، جهش می‌یابد. در جهش ممکن است ژنی از مجموعه ژن‌های جمعیت حذف شود یا ژنی که تا حال در جمعیت وجود نداشته است به آن اضافه شود. جهش یک ژن به معنای تغییر آن ژن است و وابسته به نوع کدگذاری، روش‌های متفاوت جهش استفاده می‌شود.

اگر مرحله جهش صورت نگیرد، فرزندان بلافاصله بعد از ترکیب و بدون هیچ تغییری بوجود می‌آیند (یا مستقیماً نسخه برداری می‌شوند که در نتیجه عمل ترکیب هم صورت نگرفته است). اگر تغییر صورت بگیرد، یک یا بیش از یک قسمت از کروموزوم تغییر می‌کند. اگر احتمال تغییر ۱۰۰٪ باشد، یعنی همه کروموزوم‌های تغییر کرده‌اند و اگر ۰٪ باشد، هیچ تغییر نکرده است.

---

۱ Hill Climbing

۲ Presumption Mutation - P_m

به طور کلی جهش از قرار گرفتن GA[۱] در اکسترمم‌های محلی جلوگیری می‌کند. جهش نباید زیاد صورت بگیرد زیرا در این صورت الگوریتم ژنتیک به جستوی کاملا تصادفی تبدیل خواهد شد.

همانطور که گفته شد، هر عضو، وابسته به احتمال جهش، جهش می‌یابد. احتمال جهش(P_m) مقداری است که توسط کاربر تعیین می‌شود. در الگوریتم استاندارد ژنتیک بنا به دلایلی مقدار این پارامتر، بسیار کوچک، مثل $P_m = 0.01$ یا حتی $P_m = 0.001$ در نظر گرفته می‌شود. اما از آنجا که کمتر از شکل استاندارد این الگوریتم استفاده می‌شود، به عنوان یک پیشنهاد می‌توان P_m را به صورت زیر تخمین زد:

$$P_m = \frac{1}{N}$$

رابطه ۲-۱۸- احتمال جهش.

که N تعداد ژن‌های کروموزوم است.

در هر حال P_m بین ۰ و ۱ است و معمولاً عددی کوچک انتخاب می‌شود در فرزندی که بوجود آمده است (توسط ترکیب)، به ترتیب مقداری تصادفی بین ۰ و ۱ به هر ژن اختصاص می‌یابد. اگر این مقدار اختصاص داده شده از P_m کمتر باشد، ژن جهش می‌یابد و اگر بیشتر باشد ژن تغییر نمی‌کند. نرخ بالای جهش، باعث تنوع و پراکندگی ژنتیکی در جمعیت می‌شود که این پراکنندگی ممکن است همگرایی را به تأخیر بیندازد. به همین دلیل پیشنهاد می‌شود برای جمعیت‌های بزرگ یا در نسل‌های آخر از P_m های کوچکتر و برای جمعیت‌های کوچک یا در نسل‌های ابتدایی از P_m های بزرگتر استفاده شود.

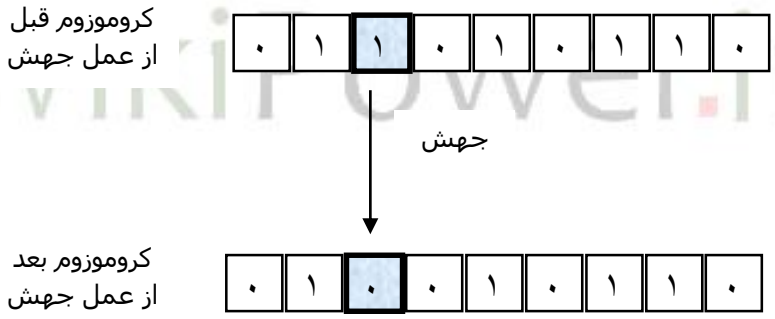
[۱] GA=الگوریتم ژنتیک.

GA پارامترهای دیگری نیز دارد که از قرار گرفتن GA در بهینهٔ محلّی جلوگیری می‌کند، از پارامترهای مهم دیگر: تعداد جمعیت[1] و احتمال ترکیب می‌باشد.[13]

در ادامه چند روش جهت پیاده‌سازی جهش بیان می‌شود.

۲-۱۷-۱- جهش باینری[2]

در الگوریتم ژنتیک با کد گذاری باینری، این عملگر اغلب با تولید تصادفی یکی از اعداد ۰ و ۱ جایگزینی آن به جای بیت مورد نظر صورت می‌گیرد، اما در برخی کاربردهای ژنتیک عمل جهش دودوئی در یک بیت با متمم ساختن آن بیت انجام می‌شود. به این صورت که اگر بیت مورد نظر ۰ بوده به بیت ۱ و بر عکس تبدیل خواهد شد، که آزمایش‌ها نشان داده شده است که روش دوم مناسبتر است.



شکل ۲-۱۶- جهش باینری.

[13]

۲-۱۷-۲- جهش حقیقی[3]

[1] Pop_Size

[2] Binary Mutation

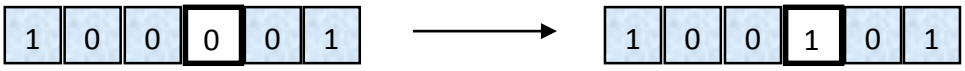
[3] Real Mutation

در کدگذاری حقیقی، عملگر جهش باعث تولید تصادفی یک مقدار جدید در یک موقعیت خاص در کروموزوم می‌شود، در نتیجهٔ این تغییراتِ تصادفی در جمعیت کروموزوم‌ها، نواحی بیشتری از فضای کاوش بررسی شده و از همگرایی بی‌موقع (ناگهانی محلّی) الگوریتم جلوگیری می‌شود.

یک مثال از عملگر جهش حقیقی، جهش تصادفی یا یکنواخت می‌باشد. با این فرض که $C=(c_1,...,c_i,...,c_n)$ یک کروموزوم و C_i یک ژنی باشد که تحت عمل جهش قرار می‌گیرد آنگاه C_i یک مقدار انتخابی تصادفی جدید از محدوده C_i می باشد که بجای ژن C_i در کروموزوم جدید جایگذاری خواهد شد. مثال دیگری برای این روش عملگر جهش مرزی که در آن یکی از ژن‌های کروموزوم به طور تصادفی با حد پایین یا بالای محدوده آن ژن جایگزین می‌شود $C_i=b_i$ یا $C_i=a_i$ [13].

۲-۱۷-۳- وارونه سازی بیت

از این نوع جهش هنگامی استفاده می‌شود که کد گذاری، کد گذاری باینری باشد. در اینجا بیتی که شرایط جهش را دارد اگر ۰ باشد به ۱ و اگر ۱ باشد به ۰ تغییر می‌دهد. به عنوان نمونه اگر در شکل ژن چهارم شرایط جهش را داشته باشد به صورت نشان داده شده، جهش می‌یابد.



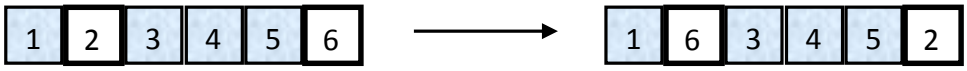
شکل ۲-۱۷- جهش: وارونه سازی بیت.

[13]

۲-۱۷-۴- تغییر ترتیب قرارگیری

از این نوع جهش مخصوصاً در الگوریتم‌هایی استفاده می‌شود که کد گذاری بر اساس مقدار باشد البته در دیگر کد گذاری‌ها مثل کد گذاری باینری هم می‌توان این جهش را بکار برد. در این

جهش، محل قرارگیری دو ژنی که می‌خواهد جهش بیابند در کروموزوم تعویض می‌شود. در شکل نمونه‌ای از این جهش نشان داده شده است.



شکل ۲-۱۸- جهش: تغییر ترتیب قرارگیری.

[13]

۲-۱۷-۵- وارون سازی

این عمل در طبیعت بسیار رخ می‌دهد ولی در الگوریتم‌های ژنتیکی به ندرت استفاده می‌شود و دلیل آن ایجاد تخریب زیاد است. این عملگر کروموزوم را معکوس می‌کند.

شکل ۲-۱۹- جهش: وارون سازی.

[13]

۲-۱۷-۶- تغییر مقدار

این نوع جهش را نمی‌توان برای کدگذاری باینری یا کدگذاری‌های مشابه که امکان تغییر ژن‌ها وجود ندارد، به کار برد. در این جهش به ژنی که شرایط جهش را دارد مقداری اضافه یا کم می‌شود. اضافه شدن و کم شدن می‌تواند به تصادف انتخاب شود یا الگوریتم مقید به استفاده از یکی از این دو عمل باشد.

مقداری که به ژن افزوده یا از آن کاسته می‌شود، وابسته به محدوده مقدار ژن است و باز می‌تواند به تصادف انتخاب شود یا برای الگوریتم تعریف شود. بدیهی است مقدارهای بزرگ پراکندگی ژنتیکی را افزایش می‌دهند. در شکل --- نمونه‌ای از این جهش نشان داده شده است.

این جهش خصوصاً برای کدگذاری‌هایی که در آنها، ژن‌ها به صورت اعداد حقیقی هستند مناسب است.

(1.29, 5.68, **2.86**, **4.11**, 5.55) ➡ (1.29, 5.68, **2.73**, **4.22**, 5.55)

شکل ۲-۲۰- جهش: تغییر مقدار.

همانطور که در شکل دیده می‌شود، ژن‌های دوم و سوم کروموزوم جهش یافته‌اند. از شکل پیداست که مقدار انتخابی برای افزودن یا کاستن می‌تواند از ژنی به ژن دیگر متفاوت باشد.[13]

۲-۱۸- محک اختتام اجرای الگوریتم ژنتیک

برای اینکه تشخیص دهیم چه موقع الگوریتم از اجرا متوقف شود، از شیوه‌های مختلفی می‌توان استفاده کرد. به عنوان نمونه می‌توان همگرا شدن کل جمعیت را در نظر گرفت و یا اینکه فاصله ارزیابی (برازندگی) بهترین فرد جمعیت از متوسط ارزیابی‌ها (برازندگی‌ها) را در نظر گرفت که در این حالت باید از حد مشخصی کوچکتر باشد، یا مقدار تابع هدف از حد مشخصی بیشتر باشد یا می‌توان تعداد نسل‌های مشخصی را به عنوان محک اختتام در نظر گرفت:

۱- به دست آوردن جواب نهایی مورد نظر بعد از چند تکرار کم و قابل قبول بودن جواب به ازای خطای خاص.

۲- اگر با پیشروی الگوریتم هیچ نوع بهبودی مشاهده نشد خواه الگوریتم جواب دلخواه را پیدا کرده باشد و یا اینکه در مینیمم محلی گیر کرده باشد.

۳- اگر مقدار میانگین تابع هدف به ازای تعدادی تکرر به مقدار خاصی رسیده باشد.

۴- الگوریتم به تعداد ثابتی از نسل‌ها رسیده باشد.

۵- بیشترین درجه برازش فرزندان حاصل شود یا دیگر نتایج بهتری حاصل نشود.

۶- بازرسی دستی.

۷- ترکیب‌های بالا.

[13]

۲-۱۹- انواع الگوریتم‌های ژنتیکی

الگوریتم‌های ژنتیک که نمونه اولیه آن توسط «هولند» در سال ۱۹۷۵ ارائه شد، تکامل طبیعی را در سطح ژن و کروموزوم شبیه‌سازی می‌کنند. عملکرد غالب در تولید نسل جدید، پیوند کروموزوم هاست، گرچه جهش در ژن‌ها نیز به عنوان یک عملکرد ثانوی به کار می‌رود.[13]

انواع بسیاری برای GA شناخته شده است که در اینجا به تعدادی از آنها اشاره می‌کنیم:

۱- الگوریتم ژنتیک سری[۱]

۲- الگوریتم ژنتیک موازی[۲]

۳- الگوریتم ژنتیک آشفته[۳]

۴- الگوریتم ژنتیک هیبرید[۴]

۵- الگوریتم ژنتیک خودسازمان[۵]

۶- الگوریتم ژنتیک زایشی[۶]

۷- الگوریتم ژنتیک حالت دائمی[۷]

[3]

[۱] Sequential GA
[۲] Parallel GA
[۳] Messy GA
[۴] Hybrid GA
[۵] Adaptive GA
[۶] Generational GA
[۷] Steady State GA

۲-۱۹-۱- الگوریتم ژنتیکی سری

الگوریتم ژنتیک سری همان الگوریتم ژنتیک معمولی است که در مقابل نوع موازی سری نام گرفته است.

تکامل یک پروسه بهینه‌سازی مبتنی بر تغییرات تصادفی تدریجی نمونه‌های مختلف در یک جمعیت و انتخاب اَحسن آنهاست. با مدل سازی این پروسه می‌توان یک تکنیک بهینه‌سازی آماری را به دست آورد که امروزه در مسائل پیچیده مختلف و بخصوص مسائل طراحی، کارائی خود را نشان داده است. در الگوریتم ژنتیک به عنوان یکی از الگوریتم‌های تکاملی، اثر کدهای ژنتیکی در ترکیب و انتقال اطلاعات و همچنین فرآیند انتخاب طبیعی بر اساس سازگاری موجود با شرایط زیست‌محیطی مدل‌سازی است. در این الگوریتم نمونه‌هایی که در پروسه تکاملی قرار می‌گیرند، جواب‌های مختلف در فضاهای جواب هستند. متناظر هر جواب (نقطه در فضای جواب)، یک نمونه ژنتیکی (Genotype) به صورت یک رشته از کاراکترها (ژن‌ها)، نسبت داده می‌شود.

الگوریتم ژنتیک در هر تکرار محاسباتی (نسل) روی جمعیتی از رشته‌ها عمل می‌کند.

تغییرات تصادفی روی مجموعه نمونه‌ها، از طریق اعمال مدل‌های ایده‌آل فرآیندهای ژنتیکی روی رشته‌ها انجام می‌شود، اما انتخاب طبیعی براساس نمود رفتاری (Phenotype) هر رشته انجام می‌شود.

بدین مفهوم که رشته‌ها رمزگشایی می‌شوند و جواب‌های مختلف از نظر عملکرد بر اساس تابع هدف ارزیابی شده و انتخاب، بر مبنای این ارزیابی و تصادف انجام می‌شود.[13]

۲-۱۹-۲- الگوریتم ژنتیکی موازی

تا کنون دو مدل اصلی در الگوریتم ژنتیک موازی مطرح گشته است:

- **مدل جزیره‌ای**[۱]
- **مدل همسایگی**[۲]

[۱] Island Model

[۲] Find Grained Model

در مدل جزیره‌ای چندین زیرجمعیت مجزّا مطابق با الگوریتم ژنتیک معمولی تکامل می‌یابد و هر از چند گاهی زیر جمعیت‌های همسایه، بهترین کروموزوم یکدیگر را معاوضه می‌کنند.

در مدل همسایگی یک مدل منفرد تکامل می‌یابد. هر کروزوزوم این جمعیت در یک سلول از یک شبکه مشبک قرار دارد و الگوریتم ژنتیک سری، به صورت مجزا به هر سلول و همسایگانش که بر حسب شعاع همسایگی مشخص می‌شوند، اعمال می‌گردد. شبکه به صورت تروید[۱] در نظر گرفته می‌شود تا از اثرات مرزی اجتناب گرد.

مقایسه‌ای بین رفتار این الگوریتم با الگوریتم‌های معمولی نشان می‌دهد که مدل همسایگی به خاطر مکانیزیم انتخاب محلی که از فشار انتخاب می‌کاهد، کاوش دقیقتری را در فضای جستجو فراهم می‌سازد.

از این جهت در مسائل ساده‌تر بدون بهبودی در عملکرد روش، تنها بار محاسباتی اضافی‌تر تحمل می‌گردد. ولی مسائل مشکلتر از این طریقه جستجو سود خواهد برد.

شعاع همسایگی مناسب نیز به مسأله مورد حل بستگی دارد و حتی همسایگی‌های کوچک به شعاع یک یا دو، انتخابی مقاوم و اطمینان از رفتاری خوب را فراهم می‌سازند.[13]

مطالعهٔ دیگر انواع GA به خوانندهٔ محترم واگذار می‌شود که با مراجعه به منبع [11] می‌تواند اطلاعات کافی و مناسبی دریافت کند.

۲-۲۰- مقایسه الگوریتم ژنتیک با سیستم‌های طبیعی

سیستم‌های طبیعی	الگوریتم ژنتیک
کروموزوم بسته‌های ژنی هستند که اطلاعات وراثتی را از نسلی به نسل دیگر عیناً انتقال می‌یابند.	پاسخ‌های ممکن مسأله به صورت رشته‌های عددی رمزگذاری شده است.

[۱] Triode

محیط شرایط محیطی را که جمعیت در آن قرار دارد.	تابع برازش مساله به صورت یک رابطه ریاضی در آمده کا تابع برازش می‌نامند.
اصل انتخاب طبیعی معیار بقای موجود زنده و تکثیر آن، سازش با محیط است.	تکثیر هر رشته جمعیت را به عنوان متغیر تابع برازش در نظر گرفته و مقدار تابع برازش هر رشته محاسبه می‌شود، متناسب با مقدار تابع برازش، رشته‌های جمعیت جدید انتخاب می‌شود.
تقاطع در نتیجه تقاطع یا تبادل قسمتی از کروموزوم‌ها مبادله ژن‌های پیوسته صورت می‌گیرد.	تقاطع(crossover) رشته های جمعیت به صورت دو به دو مزدوج می‌شوند. زوج رشته‌ها از یک نقطه قطع می‌شوند. نیم بخش‌های بین دو رشته تعویض می‌شوند.
جهش جانشین شدن ژنی به جای ژن دیگر در طول زنجیره DNA با تغییرات ایجاد شده در ژن	جهش(mutation) یک بیت از رشته عددی به صورت تصادفی انتخاب می‌شود و دچار تغییر می‌گردد.
ایجاد نسل های جدید و تکامل موجودات	تکرار مراحل فوق بعد از هر مرحله تکثیر

[13]

۲-۲۱- نقاط قوّت الگوریتم‌های ژنتیک

اولین و مهمترین نقطه قوّت این الگوریتم‌ها این است که الگوریتم‌های ژنتیک ذاتاً موازی‌اند. اکثر الگوریتم‌های دیگر موازی نیستند و فقط می‌توانند فضای مسأله مورد نظر را در یک جهت در یک لحظه جستجو کنندو اگر راه‌حل پیدا شده یک جواب بهینه محلی باشد و یا زیر مجموعه‌ای از جواب اصلی باشد باید تمام کارهایی که تا به حال انجام شده را کنار گذاشت و دوباره از اول شروع کرد. از آنجایی که GAچندین نقطه شروع دارد، در یک لحظه می‌تواند فضای مسأله را از چند جهت مختلف جستجو کند. اگر یکی به نتیجه نرسید سایر راه‌ها ادامه می‌یابند و منابع بیشتری را در اختیارشان قرار می‌گیرد. در نظر بگیرید: همه ۸ عدد رشته باینری یک فضای جستجو را تشکیل می‌دهند، که می‌تواند به صورت ******** نشان داده شود.رشته ۰۱۱۰۱۰۱۰ یکی از اعضای این فضاست. همچنین عضوی از فضاهای ۰******* و ******۱۰ و ۰******* و ۰*۱*۱*۱*۰ و ۰**۱*۰۱*۰ و الی آخر باشد.

به دلیل موازی بودن و این که چندین رشـــته در یک لحظه مورد ارزیابی قرار می‌گیرند الگوریتمهای ژنتیک برای مسائلی که فضای راه‌حل بزرگی دارند بسیار مفیدند. اکثر مسائلی که این گونه‌اند به عنوان غیرخطی شناخته شده‌اند. در یک مسأله خطی، Fitness هر عنصر مستقل است، پس هر تغییری در یک قسمت بر تغییر و پیشرفت کل سیستم تأثیر مستقیم دارد. می‌دانیم که تعداد کمی از مسائل دنیای واقعی به صورت خطی‌اند. در مسائل غیرخطی تغییر در یک قسمت ممکن است تاثیری ناهماهنگ بر کل سیستم و یا تغییر در چند عنصر تاثیر فراوانی بر سیستم بگذارد. خوشـبختانه موازی بودن GA باعث حل این مسـأله می‌شـود و در مدت کمی مشـکل حل می شود. مثلاً برای حل یک مسأله خطی ۱۰۰۰ رقمی ۲۰۰۰ امکان حل وجود دارد ولی برای یک غیرخطی ۱۰۰۰ رقمی 2^{1000} امکان.

یکی از نقاط قوت الگوریتم‌های ژنتیک که در ابتدا یک کمبود به نظر می‌رسد این است که: GA ها هیچ چیزی در مورد مسائلی که حل می‌کنند نمی‌دانند و اصطلاحاً به آنها «ساعت‌ساز نابینا»[۱]

[۱] Blind Watchmakers

می‌گوییم. آنها تغییرات تصادفی را در راه‌حل‌های کاندیدشان می‌دهند و سپس از تابع برازش برای سنجش این که آیا آن تغییرات پیشرفتی ایجاد کرده‌اند یا نه، استفاده می‌کنند. مزیت این تکنیک این است که به GA اجازه می‌دهد تا با ذهنی باز شرو به حل مسائل کند. از آنجایی که تصمیمات آن اساساً تصادفی است، بر اساس تئوری همه راه‌حل‌های ممکن به روی مسأله باز است، ولی مسائلی که محدود و به اطلاعات هستند باید از راه قیاس تصمیم بگیرند و در این صورت بسیاری از راه‌حل‌های نو و جدید را از دست می‌دهند.

یکی دیگر از مزایای الگوریتم این است که آنها می‌توانند چندین پارامتر را همزمان تغییر دهند. بسیاری از مسائل واقعی نمی‌توانند محدود به یک ویژگی شوند تا آن ویژگی ماکسیمم شود و باید چند جانبه در نظر گرفته شوند. GAها در حل این گونه مسائل بسیار مفیدند، و در حقیقت قابلیت موازی کار کردن آنها این خاصیت را به آنها می‌بخشد. و ممکن است برای یک مسأله ۲ یا چند راه‌حل پیدا شود، که هر کدام با در نظرگرفتن یک پارامتر خاص به جواب رسیده‌اند.

به طور خلاصه مزایای الگوریتم ژنتیک را می‌توان در موارد زیر برشمرد:

۱- با متغیرهای پیوسته و هم گسسته می‌تواند عمل بهینه‌سازی را انجام دهد.

۲- نیازی به محاسبه مشتق توابع ندارد.

۳- بطور همزمان می‌تواند تمامی ناحیه جستجو شونده وسیع تابع هزینه را جستجو کند.

۴- قادر به بهینه سازی مسائل با تعداد متغیرهای زیاد می‌باشد.

۵- قابل اجرا از طریق کامپیوترهای موازی است.

۶- توابع هزینه‌ای که بسیار پیچیده با شند نیز از این طریق قابل بهینه سازی می‌باشند و الگوریتم در اکسترمم محلی به دام نمی‌افتد.

۷- قادر ا ست تا چند جواب بهینه را بطور همزمان به د ست آورد نه فقط یک جواب.

۸- الگوریتم‌های ژنتیک بر روی مجموعه‌ای از راه‌حل‌ها اعمال می‌شوند و نه بر روی یک راه‌حل خاص.

۹- قادر است تا متغیرها را کد بندی نموده و بهینه‌سازی را با متغیرهای کدبندی شده انجام دهد. کد بندی سرعت همگرایی الگوریتم را افزایش می‌دهد.

۱۰- الگوریتم توانایی کار کردن با داده‌های عددی تولید شده و داده‌های تجربی را علاوه بر توابع تحلیلی دارد.

۱۱- فرآیند ارائه شده توسط الگوریتم‌های ژنتیک بر روی فضایی از مجموعه نمایندگان یا همان فضای کروموزوم‌ها اعمال می‌گردد و نه بر روی خود فضای راه‌حل‌ها.

۱۲- الگوریتم‌های ژنتیک از قوانین انتقالی احتمالی بجای قوانین انتقالی قطعی استفاده می‌کنند، بدین معنا که حرکت آن در هر نقطه از الگوریتم کاملا احتمالی بوده و بر اساس قطعیت صورت نمی‌پذیرد. این امر از مزایای مهم این روش بوده و از افتادن سیستم در کمینه محلی جلوگیری می‌نماید.

البته میزان احتمال به گونه‌ای است که احتمال حرکت به سمت مسأله بیشتر از احتمال حرکت آن به سمت مخالف جواب می‌باشد.

۱۳- تنها ملاک ارزشیابی و سنجش میزان شایستگی هر راه‌حل توسط الگوریتم‌های ژنتیک، مقدار تابع شایستگی آن در فضای کروموزوم‌ها می‌باشد و نه معیارهای مورد نظر در سطح فضای راه‌حل‌ها.

۱۴- برای حل برخی از مسائلی از رده NP-Hard نیز استفاده می‌شود.

۱۵- این الگوریتم بیشتر در مسائل بهینه سازی و امثالهم بکار می‌رود.

[13]

۲-۲۲- محدودیت‌های GAها

یک مشکل چگونگی نوشتن عملگر ارزیاب است که منجر به بهترین راه‌حل برای مسأله شود. اگر این کارکرد برازش به خوبی و قوی انتخاب نشود ممکن است باعث شود که راه‌حلی برای مسأله پیدا نکنیم یا مسأله‌ای دیگر را به اشتباه حل کنیم. به علاوه برای انتخاب تابع مناسب برای ارزیاب، پارامترهای دیگری مثل اندازه جمعیت، نرخ ترکیب، قدرت و نوع انتخاب هم باید مورد توجه قرار گیرند.

مشکل دیگر، که آن را «نارس» می‌نامیم این است که اگر یک ژنوم که فاصله‌اش با سایز ژنوم‌های نسل‌اش زیاد باشد. (خیلی بهتر از بقیه باشد) خیلی زود دیده می شود (ایجاد می شود) ممکن است محدودیت ایجاد کند و راه‌حل را به سوی جواب بهینه محلی سوق دهد. این اتفاق معمولاً در جمعیت‌های کم اتفاق می‌افتد. روش‌های Rank Scaling , Tournament Selection **بر این مشکل غلبه می‌کنند.**[13]

۲-۲۳- استراتژی برخورد با محدودیت‌ها

بحث دیگری که در اجرای الگوریتم ژنتیک وجود دارد چگونگی برخورد با محدودیت‌های مسأله می‌باشد، زیرا عملگرهای ژنتیک مورد استفاده در الگوریتم باعث تولید کروموزوم‌های غیرموجّه می‌شود. «میکالویچ» چند تکنیک معمول جهت مواجهه با محدودیت‌ها تقسیم‌بندی نموده است که در ادامه به چند تا از آنها اشاره می‌شود.[13]

۲-۲۳-۱- استراتژی اصلاح عملگرهای ژنتیک

یک روش برای جلوگیری از تولید کروموزوم غیرموجّه این است که عملگر ژنتیکی طوری تعریف گردد که پس از عمل بر روی کروموزم‌ها، کروموزوم تولید شده نیز موجّه باشد. در این حالت یکسری مشکلات وجود دارد. مثلاً پیدا کردن عملگری که دارای شرط فوق باشد بسیار دشوار بوده و از مسأله دیگر متفاوت می‌باشد.[13]

۲-۲۳-۲- استراتژی رَدّی

در این روش پس از تولید هر کروموزوم آنرا از نظر موجّه بودن تست کرده و در صورت غیرموجّه بودن حذف می‌گردد. این روش بسیار ساده و کارا می‌باشد.[13]

۲-۲۳-۳- استراتژی اصلاحی

در این روش به جای اینکه کروموزوم غیرموجّه حذف گردد تبدیل به یک کروموزوم موجّه می شود. این روش نیز مانند روش اول به مسأله وابسته بوده و یافتن فرآیند اصلاح گاهی بسیار پیچیده می‌باشد.[13]

۲-۲۳-۴- استراتژی جریمه‌ای

در این روش بر خلاف سه روش قبل که از ورود جواب‌های غیرموجّه جلوگیری می‌کردند، جواب غیرموجّه با احتمال کم امکان حضور می‌یابند. سه روش فوق دارای این عیب بودند که به هیچ نقطه‌ای بیرون از فضای موجّه توجّه نمی‌کردند، اما در بعضی مسائل بهینه سازی، جواب‌های

غیرموجّه درصد زیادی از جمعیت را اشغال می‌کنند. در چنین شرایطی اگر جستجو فقط در ناحیه موجّه انجام گیرد شاید یافتن جواب موجّه خیلی وقت‌گیر و مشکل باشد.

استراتژی جریمه‌ای از متداولترین تکنیک‌های مورد استفاده برای سر و کار داشتن با جواب‌های غیرموجّه می‌باشد که در آن ابتدا محدودیت‌های مسأله در نظر گرفته نمی‌شوند پس برای هر تخلّف از محدودیت‌ها یک جریمه اختصاص داده می‌شود که این جریمه در تابع هدف قرار می‌گیرد.

مسأله اصلی چگونگی انتخاب یک مقدار مناسب برای مقدار جریمه می‌باشد تا در حل مسائل به ما کمک نماید.

نکته‌ای که در روش جریمه وجود دارد این است که یک جواب غیرموجّه به سادگی حذف نمی شود زیرا ممکن ا ست در ژنهای آن اطلاعات مفیدی وجود دا شته با شد که با اندکی تغییر به جواب بهینه تبدیل شود.[13]

۲-۲۴- بهبود الگوریتم ژنتیک

برای بهبود دادن GA می‌توانیم تغییرات زیر را اعمال کنیم:

۱- استفاده از بهینه‌گر محلی.

۲- تغییر پارامتر هائی از قبیل تغییر جمعیت اولیه، آهنگ جهش و کسر ادغام.

۳- تغییر GA باینری به پیوسته و بالعکس.

[13]

۲-۲۵- چند نمونه از کاربردهای الگوریتم‌های ژنتیک

نرم افزار شنا سایی چهره (شنا سایی چهره با ا ستفاده از ت صویر ثبت شده. در این روش، شنا سایی چهره بر ا ساس فا صله اجزای چهره و ویژگی‌های محلی و هند سی صورت می گیرد که تغیرات ناشـــی از گریم، تغییرات نور و افزایش ســـن کمترین تأثیر را خواهد داشـــت. همچنین گراف‌ها برای چهره‌های جدید با ا ستفاده از الگوریتم‌های ژنتیک ساخته شده و با ا ستفاده از یک تابع تشابه، قابل مقایسه با یکدیگر هستند که این امر تأثیر به سزایی در افزایش سرعت شناسایی خواهد داشت).

توپولوژی‌های شبکه‌هایی کامپیوتری توزیع شده.

بهینه سازی ساختار مولکولی شیمیایی(شیمی).

مهندسی برق برای ساخت آنتن‌های خمیده[۱].

مهندسی نرم‌افزار.

بازی‌های کامپیوتری.

مهندسی مواد.

مهندسی سیستم.

[۱] Crooked-Wire Genetic Antenna

رباتیک[۱] .

تشخیص الگو و استخراج داده[۲] .

حل مسأله فروشنده دوره گرد.

آموزش شبکه‌های عصبی مصنوعی.

یاددهی رفتار به رباتها با GA.

یادگیری قوانین فازی با استفاده از الگوریتم‌های ژنتیک.

یک مثال ساده.

مثال: فرض کنید تابع $f(x)=-x^2+6-3$ را داریم می‌خواهیم برای $0\le x\le 15$ و $x\in N$ ماکزیمم تابع را بیابیم.

مراحل حل مسأله به روش الگوریتم ژنتیک به شرح زیر است:

۱- کد کردن (Encoding)

برای اینکه ما جواب‌های ممکن یعنی $0\le x\le 15$ را کدگذاری کنیم یکی از روش‌هایی که در پیش داریم تبدیل هر عدد به یک عدد باینری متناظر است. چون بیشترین جواب ممکن ما ۱۵ است و متناظر باینری آن ۱۱۱۱ چهار بیتی است لذا تمام جواب‌های ما (کروموزوم ها) دارای طولی برابر $l=4$ می‌باشد.

[۱] Robotics
[۲] Data Mining

۲- تابع ارزش(Evaluation)

تابعی که مقداری برای Fitness حساب می‌کند مستقیماً از خود تابع $f(x)$ به دست می‌آید.

یعنی اگر به ازای یک $f(b) > f(a) : b,a$ باشد آنگاه b دارای Fitness بیشتری است. یا اینکه b از a بهتر است.

۳- انتخاب (Selection)

فرض کنیم که در آن جمعیت اولیه m فرد که m<n دو عدد ۳ و ۸ دارای Fitness بیشتری نسبت به جمعیت خود بودند و احتمال انتخاب بیشتری داشتند. بعد از انتخاب این دو عدد به عنوان Parent الگوریتم وارد مراحل تولید نسل جدید می‌شود.

$3 \rightarrow 0011$ $\quad f(3) = 6$

$8 \rightarrow 1000$ $\quad f(8) = -19$

۴- ترکیب (Crossover)

فرض کنیم که ترکیب عمل زیر را انجام دهد.

۰۰۱۰ Point=3 ۰۰۱۱

۱۰۰۱ → ۱۰۰۰

۵- جهش (Mutation)

و نیز عملگر جهش:

0110=6 ۰۰۱۰

Point=2

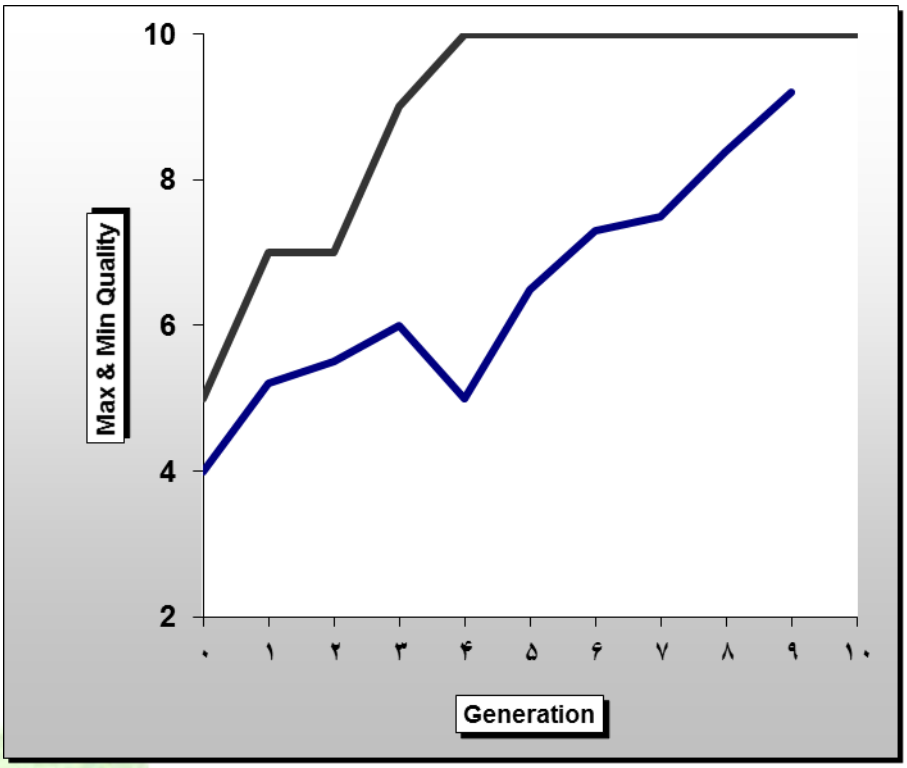
۱۰۰۱ 1101=13

→

حال کروموزوم‌های متولد شده جزو نسل جدید به حساب می‌آیند و به جمعیت اولیه افزوده می‌شـــوند و جواب‌های با Fitness پایین حذف می‌شـــوند و الگوریتم دوباره با n فرد به کار خود ادامه می‌دهد. یک نکته لازم به تذکر اســـت که ممکن اســـت بهترین جواب (برای مثال x=6) طی مراحل بعد به نحوی از بین برود. برای جلوگیری از این امر بعد از هر بار انجام الگوریتم بهترین جواب را در جایی کنار می‌گذاریم (far) تا همیشه بهترین جواب حاصل از هر بار اجرای الگوریتم را داشته باشیم.

معیارهای مختلفی را می‌توان برای توقف الگوریتم در نظر گرفت و معمولاً چند معیار برای توقف استفاده می‌شود تا احتمال‌های مختلف وقوع پیشامدها در طی اجرای الگوریتم حساب شوند. یک معیار می‌تواند این باشـد که بهترین جواب را بعد از اجرای تعداد مشـخصـی بار از الگوریتم تغییر ندهد. یا معیار دیگر اینکه میانگین برازندگی جواب‌های موجود در جمعیت جاری همان برازندگی بهترین جواب یا بسیار نزدیک به آن باشد. یا اینکه می‌توانیم از پیش قرارداد کنیم که الگوریتم به تعداد مشـخصـی اجرا شـود. معمولاً روتین‌های توقّف مختلف اسـت و بسـتگی به پیچیدگی و چگونگی مسـأله دارد. در اینجا ما برای تعریف تابع ارزشـیابی از خود تابع اسـتفاده کردیم و لزومی برای نسبت دهی یک مقدار به عنوان کیفیت[۱] نبود.

[۱] Quality



شکل ۲-۲۱- نمودار بررسی رابطه های جمعیت، کیفیت جواب و معیار توقف با یکدیگر.

اگر ما این مراحل را به گونه‌ای دنبال کنیم که بعد از هر بار اجرا جواب های بد حذف و جواب های نسل جدید که احتمالا Fitness بالایی دارند جایگزین آنها شوند و به مرور فاصله Fitness بین Min و Max کم می شود و میانگین جواب‌ها به بهترین جواب میل می‌کند و این نشان می‌دهد که به طور کلی به بهترین جواب نزدیک می‌شویم.

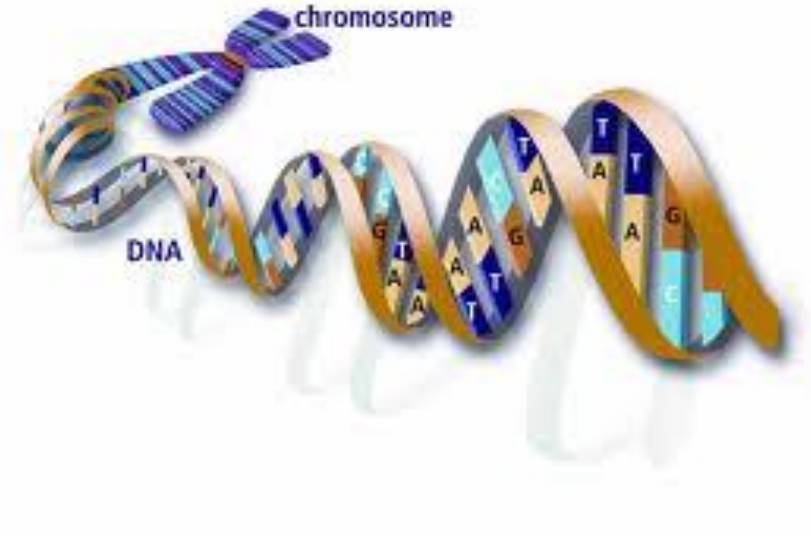
سؤالی که در اینجا مطرح می شود اینست که اگر در مثال قبل به جای فضای گسسته فضای پیوسته $[0,15]$ مطرح می‌شد الگوریتم به چه شکلی انجام می‌گرفت؟

در این حالت ما یک ε تعریف می‌کنیم تا هر بیت جواب در مراحلی مثل Mutation به اندازه ε تغییر کند. بدیهی اســـت هر چقدر ما این ε را کوچکتر فرض کنیم میزان محســـابات و تکرار الگوریتم بالا می‌رود و به همین دلیل زمان اتمام الگوریتم بالا می‌رود. عامل دیگری که در زمان مؤثر اســـت n اولیه در جمعیت اولیه اســـت که با زیاد فرض کردن n زمان اتمام بالا می‌رود.

[13]

فصل سوم:

مسائل کلاسیک



۳-۱- مقدمه

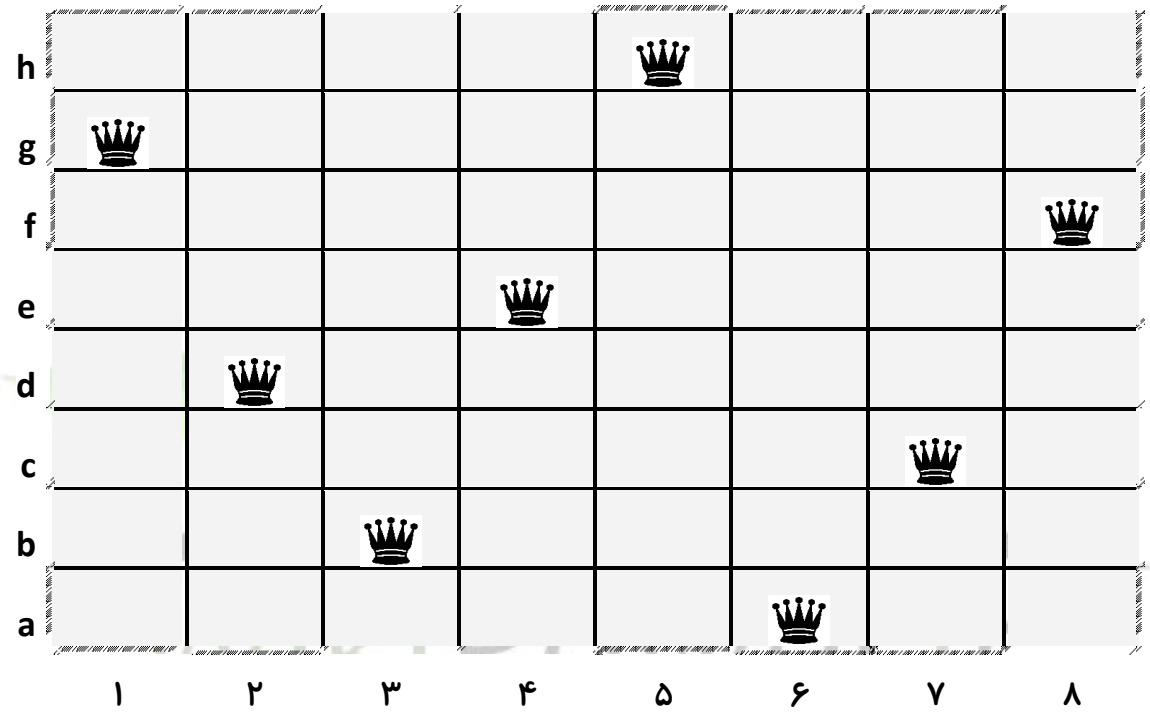
امید می‌رود که خوانندهٔ گرامی بهره و حظ وافی را از فصل دوم برده باشد و با تمام مفاهیمِ مطرح در الگوریتم ژنتیک و برنامه نویسی ژنتیک آشنایی کامل پیدا کرده باشد.

پژوهشگرانی که بر روی روش‌ها و تکنیک‌های جستجو کار می‌کنند عمدتاً برای ارزیابی الگوریتم‌های جستجو از یک سِری مسائلِ کلاسیک کمک می‌گیرند. ما نیز در فصلی که پیش رو دارید روی برنامه نویسی ژنتیک متمرکز شده، تعدادی از مسائل کلاسیک را انتخاب و به کمک روند ژنتیکی پیاده‌سازی کرده‌ایم؛ یک مسألهٔ مرتب‌سازی (که در اینجا فقط جنبهٔ آشنایی با برنامه نویسی ژنتیک دارد و تنها شِمای بسیار کوچکی از اَبَرمسائل مربوط به مرتب سازی را نمایش می دهد) به همراه سه مسألهٔ کلاسیکِ معمّای هشت وزیر، فروشندهٔ دوره‌گرد و حلّ معمای سودوکو که به کمک نرم‌افزار دلفی[۱] پیاده‌سازی شده‌اند به حضور شما خوانندهٔ عزیز تقدیم می‌گردد.

[۱] Delphi

۳-۲- حلّ معمای هشت وزیر[۱]

معمای هشت وزیر یکی از مسائل کلاسیک و متداول برای سنجش توانایی الگوریتم‌های جستجو است. در این معمای قدیمی باید هشت مهره وزیر شطرنج به گونه‌ای بر صفحه چیده شوند، که هیچ‌یک در تیررس قرار نگیرند. شکل زیر یکی از این حالت‌ها را نشان می‌دهد.

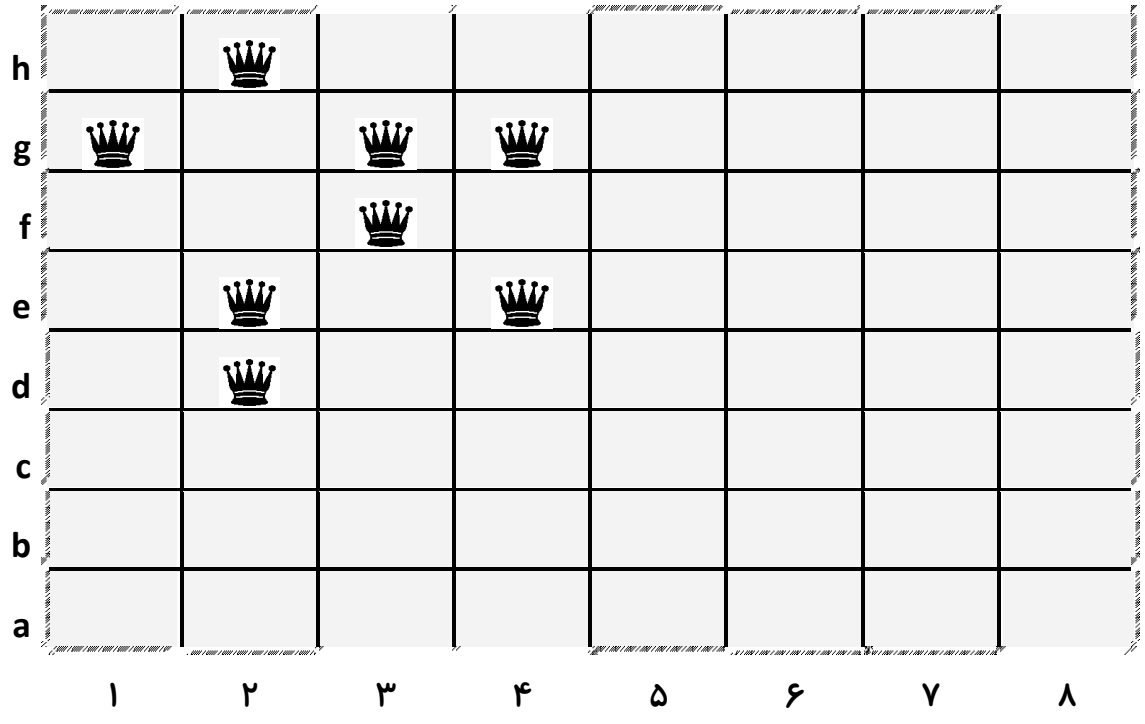


شکل ۳-۱- چینش هشت مهره وزیر در صفحه شطرنج بدون تهدید یکدیگر.

این مسأله پاسخ‌های بسیار دارد. ابتدایی‌ترین راه‌حل برای حلّ چنین مسأله‌ای استفاده از هشت حلقهٔ تودرتو ۶۴تایی (برای هشت مهره که هرکدام امکان جایگیری برای هر یک از ۶۴خانه صفحه شطرنج را دارند) برای تولید همه حالت‌های ممکن جانشانی مهره‌ها بر صفحه و سپس آزمون هر یک از آن حالت‌ها برای یافتن پاسخ‌های احتمالی است. اما روشن است که چنین

[۱] 8-Queen

روشـــی، انبوهی از حالت‌ها را پدید می‌آورد، $64^8 = 2.8\times10^{14}$ حالت که که بســیاری از آنها مانند شکل زیر به طور بدیهی پاسخ نیستند.



شکل ۳-۲- چینش هشت مهره وزیر در صفحه شطرنج(در تیررس یکدیگر).

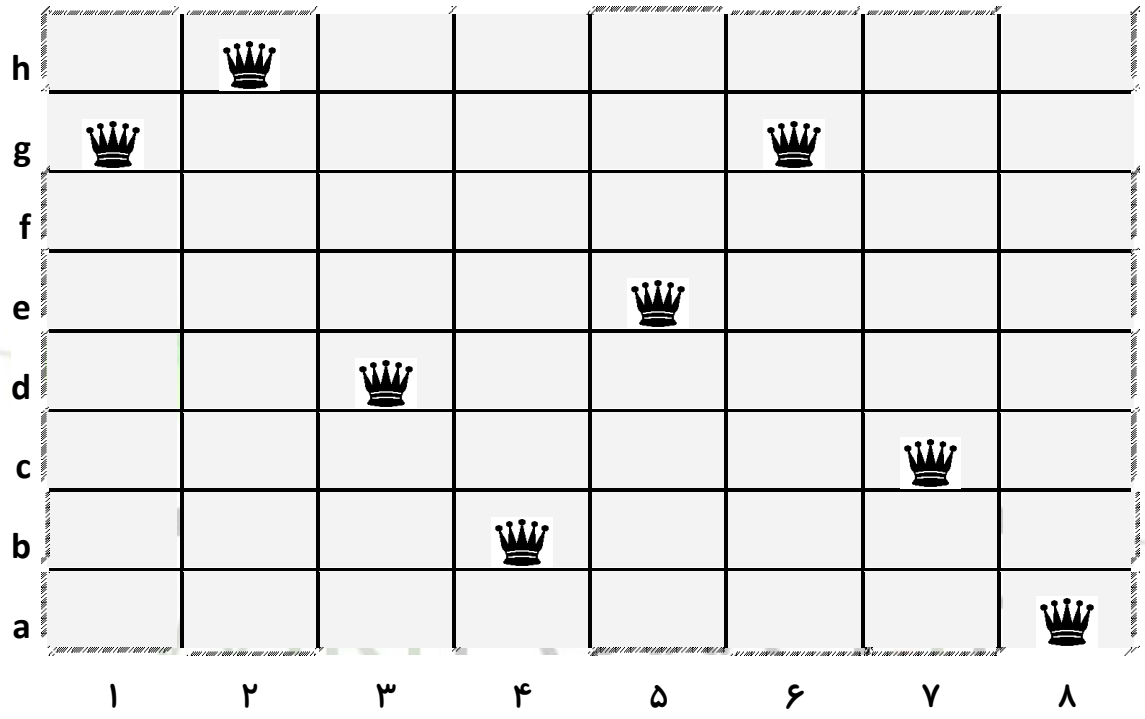
چنین مسـأله‌ای، با وجود آنکه پاسـخ‌های متناهی دارد، در زمرهٔ مسـائل بی‌پایان می‌گنجد و اگر فرض کنیم که کامپیوتر برای تولید و مقایســه هر حالت، تنها به یک هزارم ثانیه زمان نیاز داشته باشد، باز هم به بیش از ۹۰۰۰سال زمان برای آزمون کل حالت‌ها نیاز خواهد داشت!

افزودن یک آگاهی کوچک و ساده به برنامه، پاسخ سریعتری را در دسترس قرار می‌دهد: این که در هر ستون تنها یک مهره قرار گیرد؛ بنابراین تعداد حالت‌ها میلیون‌ها بار کاهش می‌یابد و مسأله را قابل حل می‌سازد.

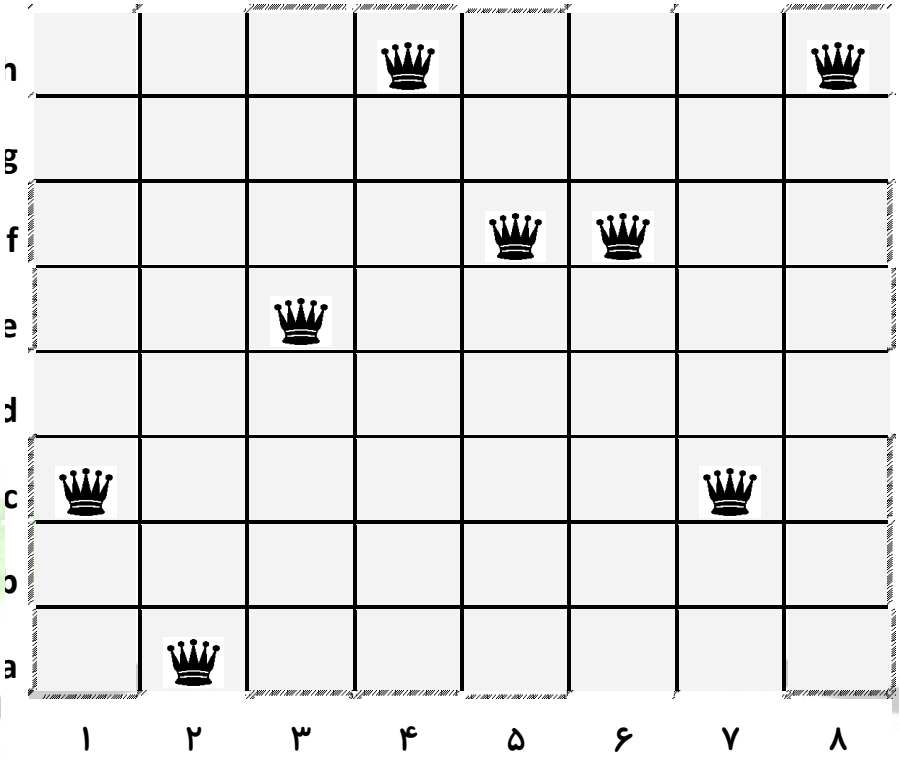
برای حلّ چنین مســأله‌ای، روش‌ها و تکنیک‌های بســیاری پیشــنهاد می‌شــود. اما یکی از جالب‌ترین و سریعترین روش‌ها برای دستیابی به یکی از پاسخ‌ها، همان الگوریتم ژنتیک است که در ادامه مسأله را با الگوریتم ژنتیک مدل سازی می‌کنیم.

۳-۲-۱- جمعیت آغازین

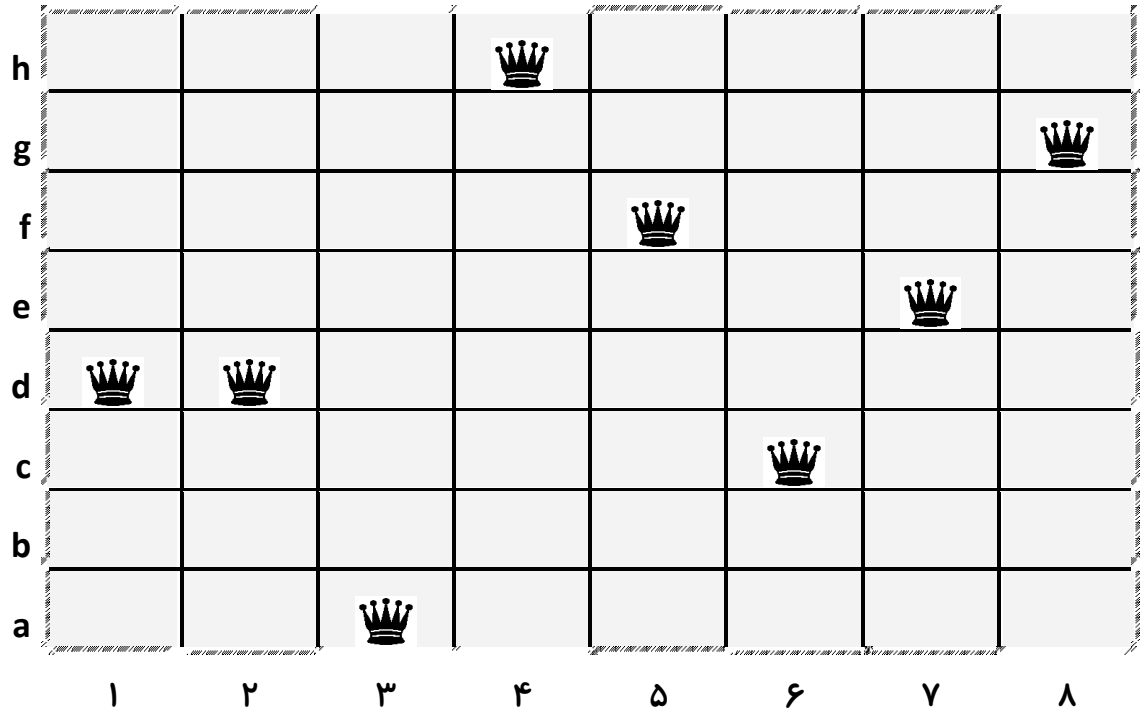
فرض می‌کنیم که چهار حالت اتفاقی از قرار گرفتن مهره‌ها بر صفحه (م شروط بر اینکه در هر ستون فقط یک مهره قرار گیرد) مانند شکل‌های زیر تولید کرده‌ایم.



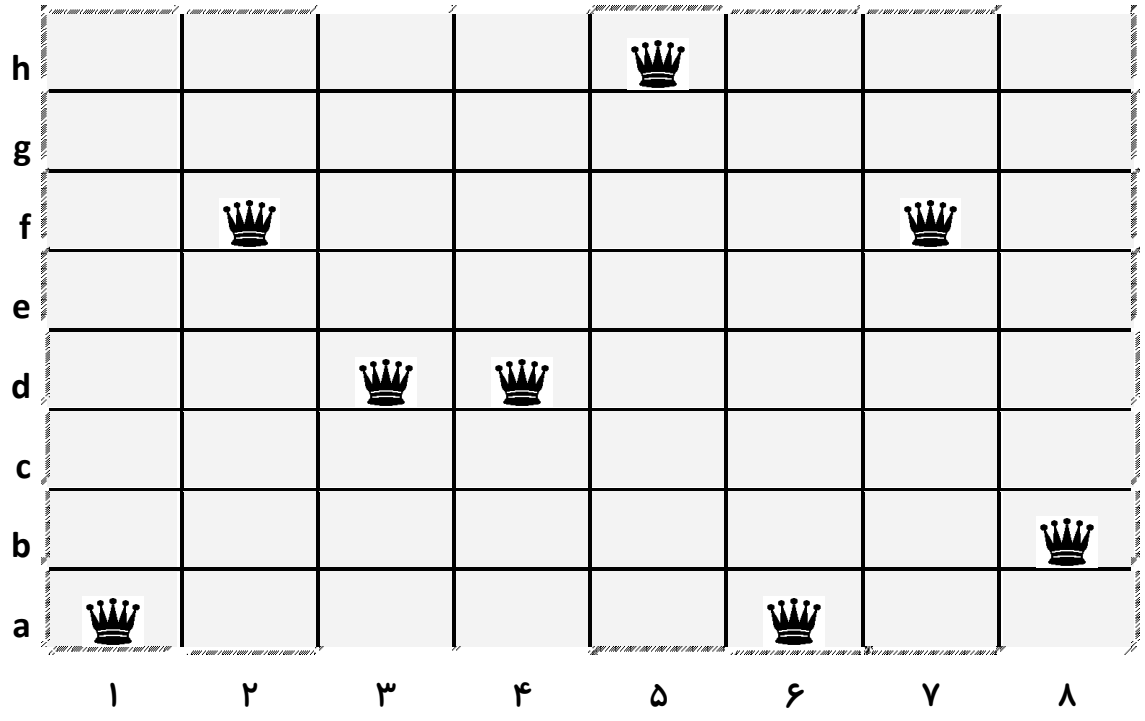
شکل ۳-۳- چینش هشت مهره وزیر در صفحه شطرنج(در تیررس یکدیگر).



شکل ۳-۴- چینش هشت مهره وزیر در صفحه شطرنج(در تیررس یکدیگر).



شکل ۳-۵- چینش هشت مهره وزیر در صفحه شطرنج(در تیررس یکدیگر).



شکل ۳-۶- چینش هشت مهره وزیر در صفحه شطرنج(در تیررس یکدیگر).

برای ســادگی در نمایش می‌توان هریک از حالت‌ها را به صــورت زیر نمایش داد و آن را ساختار ژنتیکی هر یک از اعضای گروه در نظر گرفت:

[1] ghdbegca

[2] caehffch

[3] ddahfceg

[4] afddhafb

اینها اعضای اولیهٔ جامعهٔ ما هستند که پی از آمیزش با یکدیگر، نسل‌های بعدی را به وجود می‌آورند؛ تا جایی که یکی از اعضاء پاسخ دلخواه مسأله باشد.

۳-۲-۲- تابع برازندگی

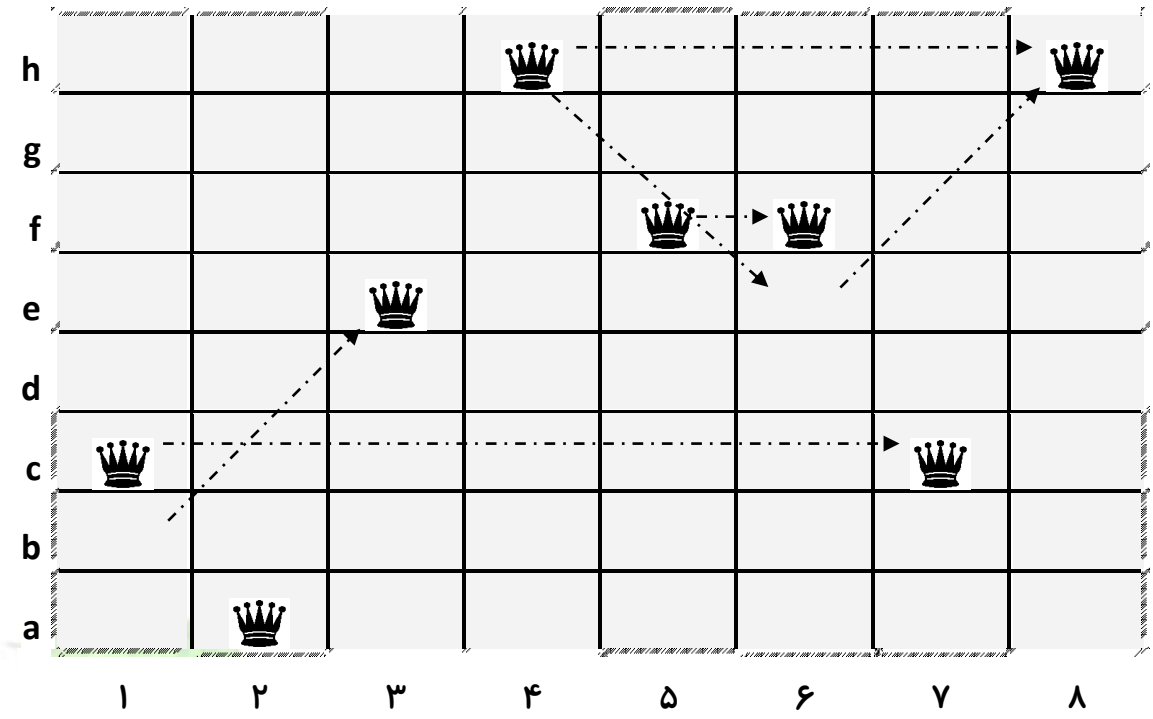
تدوین تابع برازندگی اغلب سخت‌ترین کار در حل مسائل به کمک GA می‌باشد. طراحان، بسته به سلیقه یا توانایی فکری خود، ممکن است راه حل‌های گوناگونی را برای طراحی تابع برازندگی چنین مسأله‌ای ارائه دهند. در اینجا ما یک روش ساده را به کار می‌گیریم؛ به گونه‌ای که هر عضو به ازای هر یک از حالت‌های برخورد، یک نمره منفی دریافت کند. در زیر نمره منفی هر یک از کروموزوم‌ها به همراه نقشهٔ یکی از آنها درج شده است.

[1] ghdbegca (5)

[2] caehffch (3)

[3] ddahfceg (6)

[4] afddhafb (7)



شکل ۳-۷- چگونگی برازش هشت مهره وزیر در صفحه شطرنج.

به یاد داشته باشید که بدترین حالت (همه مهره‌ها بر یک خط قرار گیرند) ۲۸ نمره منفی خواهد داشت؛ و هر یک از پاسخ‌ها صفر نمره منفی. با این احتساب روشن است که ساختار چهارم بهترین و ساختار سوم بدترین کروموزوم‌ها برای آمیزش هستند و می‌توان احتمال مشارکت در تولید نسل آینده برای هریک از آنها (بر پایه تابع برازندگی یاد شده در بالا) چنین نوشت:

$$(\frac{5}{21})\times 100 = \%24$$ **[1] ghdbegca**

$$(\frac{3}{21})\times 100 = \%14$$ **[2] caehffch**

$$(\frac{6}{21})\times 100 = \%29$$ **[3] ddahfceg**

$(\frac{7}{21})\times 100 = \%33$ **[4] afddhafb**

۳-۲-۳- آمیزش

بر پایه همین احتمالات است که برخی از اعضا برای تولید نسل آینده برگزیده می‌شوند. این کار نیاز به استفاده از توابع تولید اعداد تصادفی دارد که در اینجا (روی کاغذ) امکان پذیر نیست. پس اجازه دهید فرض کنیم که بر پایه تابع برازش ما، ژن دوم(با کمترین میزان برخورد) یک بار با ژن یکم و یک بار هم با ژن سوم ترکیب می شود و ژن چهارم در هیچ آمیزشی شرکت نمی کند(که چندان هم دور از انتظار نیست). و باز هم فرض می کنیم که محل گسست و ترکیب تصادفی کروموزوم ها به ترتیب ژن سوم و چهارم بوده است و از هر آمیزش، دو فرزند با ترکیب ژن‌های چپ و راست محل گسست دو عضو پدید می‌آید(دوقلوهای کاملاً ناهمسان). بنابراین خواهیم داشت:

[2] cae/hffch + [1] ghd/begca = [5] cae/begca , [6] ghd/hffch

[2] caeh/ffch + [3] ddah/fceg = [7] caeh/fceg , [8] ddah/ffch

[1] ghdbegca (5)

[2] caehffch (3)

[3] ddahfceg (6)

[4] afddhafb (7)

۳-۲-۴- جهش ژنتیکی

اکنون فرض کنید که به طور تصادفی بر روی ژن هفتم از کروموزوم پنجم و ژن ششم از کروموزوم هفتم، یک جهش ژنتیکی نیزبه صورت زیر رخ می‌دهد:

[5] cae/begca ➡ [5] cae/begha

[7] caeh/fceg ➡ [7] caeh/faeg

حال چهار عضو نوپدید را به جمعیت اولیه می‌افزائیم و پس از محاسبه نمرات منفی(تعداد برخوردها)، تابع برازندگی را مشخص می‌کنیم:

[1] ghdbegca (5)

[2] caehffch (3)

[3] ddahfceg (6)

[4] afddhafb (7)

[6] ghd/hffch (6)

[5] cae/begha (4)

[7] caeh/faeg (5)

[8] ddah/ffch (5)

این کار تا آنجا ادامه می‌یابد که یکی از تعداد برخورد مهره‌های یکی از ساختارها به صفر برسد. اما شگفت اینجاست که GA این مسأله را سریعتر از روش‌های دیگر به پاسخ می‌رساند! [2]

۳-۳- الگوریتم ژنتیک و حلّ مسألهٔ فروشندهٔ دوره‌گرد[۱]

[۱] Traveling Salesman Problem - TSP

مسأله فروشندهٔ دوره‌گرد (TSP) به این شکل است که می‌خواهیم در یک تعداد شهر n دوری پیدا کنیم که از هر شهر دقیقاً یک بار عبور کند و در پایان به شهر آغازین بازگردد به طوری که طول دور مینیمم باشد.

هر چند منشأ TSP دقیقاً مشخص نیست ولی قطعاً در حدود ۱۹۳۱ بوده است. اولین نمونه شبیه به این مسأله توسط «اویلر»[۱] در سال ۱۷۵۹ مطرح شد و به این صورت بود که یک مهره اسب می‌بایست روی برد شطرنج حرکت کند و از هر خانه دقیقاً یک بار عبور کند.

این مسأله را می‌توان به صورت ریاضی هم شبیه‌سازی کرد. به این ترتیب که ما در یک گراف وزن دار $G(v,e)$ دوری فراگیر (اویلری) با مینیمم مجموع وزن های یال های گذر نده می‌خواهیم بیابیم. به روش ریاضی مسأله با یافتن تعداد جایگشت‌های n شئ متمایز و سپس ارزیابی هر حالت بررسی می‌شود. تعداد جایگشتها !n است. برای یافتن مینیمم دورها نیز به حداکثر !n محاسبه احتیاج داریم. ولی اگر n را زیاد فرض کنیم تعداد محاسبات بسیار بالا خواهد بود به همین دلیل گفته می‌شود که الگوریتم حل مسأله در زمان «چندجمله‌ای» نیست[۲].

حال می‌خواهیم چند الگوریتم ارائه شده برای حل مسأله TSP را مطرح کنیم. تا به امروز الگوریتمی به دست نیامده است که این مسأله را در زمان چندجمله‌ای (Polynomial) حل کند. به همین دلیل ما بهینگی را فدای زمان می‌کنیم تا بتوانیم در یک زمان معقول به یک جواب خوب برسیم.

یکی از الگوریتم‌های ارائه شده الگوریتم حریصانه[۳] است؛ به این شکل که الگوریتم یک لیست از همه یال‌ها در گراف ایجاد می‌کند و سپس آنها را از کمترین هزینه به بیشترین هزینه

[۱] Euler

[۲] None Polynomial

[۳] Greedy

مرتب می‌کند. سپس ابتدا یالی با کمترین هزینه را انتخاب می‌کند و چک می‌کند که این یال سبب نشود که دوری در گراف ایجاد شود که از همه رئوس عبور نکند.

روش دیگری نیز هست که شبیه به الگوریتم حریصانه است که به نام نزدیکترین‌همسایه[۱] مطرح شده است.

به این شکل که یک نقطه آغازین به صورت تصادفی انتخاب می‌کنیم و بعد به نزدیکترین رأس بعد می‌رویم و از آن رأس نیز همین کار را ادامه می‌دهیم. البته باز هم شرط می‌کنیم که در هر مرحله دوری که اویلری نباشد ایجاد نشود.

این روش‌ها راه‌حلّ خوبی ارائه نمی‌دهند چون اغلب آخرین یال اضافه شده وزن نسبتا زیادی دارد.

الگوریتم بعدی «درخت پوشای مینیمم»[۲] می‌باشد. ابتدا ما یک درخت پوشای مینیمم با n-1 یال می‌یابیم. سپس می‌توانیم یک دور به وسیله رفتار و چگونگی یال‌ها در درخت پوشایمان ایجاد کنیم (مثل یال‌های دو جهتی) بعد از یک شهر که فقط به یک شهر دیگر متصل شده است شروع می‌کنیم و با دنبال کردن یال‌های طی نشده برای شهرهای جدید ادامه می‌دهیم. اگر یال طی نشده‌ای وجود نداشت به یا قبلی بر می‌گردیم و این کار را ادامه می‌دهیم تا به شهر ابتدایی باز گردیم. با این کار ما یک کران بالا برای TSP خواهیم داشت.

قابل توجه است که ما بعضی از شهرها را بیش از یک بار بازدید کردیم. وقتی که نیاز داریم به عقب حرکت می‌کنیم ولی در عوض به شهر بازدید نشده بعدی می‌رویم. بعد از اینکه همه شهرها بازدید شدند به شهر شروع بر می‌گردیم.

۳-۳-۱- حل مسأله TSP به وسیله الگوریتم ژنتیک

---

[۱] Nearest Neighbor

[۲] Minimum Spanning Tree

مراحل الگوریتم ژنتیک برای مسأله به صورت زیر است

۱- Encoding

برای Encoding می‌توانیم یک ماتریس مجاورت گراف ایجاد کنیم که شامل ۱ در مکان i و j است اگر یک یال از رأس i به رأس j وجود داشته باشد و در غیر این صورت 0 است.

حال می‌توانیم از این ماتریس همانگونه که هست استفاده کنیم یا به این صورت که سطرهای ماتریس را به هم به این صورت اولین متد Encoding ساخته می‌شود.

برای مثال ماتریس مقابل:

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

دوری را نشان می‌دهد که از شهر ۱ به شهر ۳ و از شهر ۳ به شهر ۲ و از شهر ۲ به شهر ۱ می‌رود.

متد بعدی که مورد بررسی قرار می گیرد متدی است که یک رشته از اعداد که معرف شماره شهر است را می‌سازد.

اولین شیوه ساخت این متد به این صورت است که برای مثال رشته $v=1234$ این مفهوم را می‌رساند که ابتدا از شهر ۱ شروع کرده و سپس به شهر ۲ می‌رویم و از آنجا به شهر ۳ و بعد به شهر ۴ حرکت می‌کنیم. در پایان هم به شهر اول بر میگردیم. توجه کنید که دو رشته ۱۲۳۴ و ۲۳۴۱ معادلند. شیوه دوم این است که اگر رشته $v=3214$ را داشته باشیم به این معنی است که دور از شهر ۱ به شهر ۳ و از شهر ۳ به ۲ و از شهر ۲ به ۴ و در پایان از شهر ۴ به شهر ۲ می‌رویم.

قابل توجه اســت که هر رشــته ممکن در اینجا یک دور مجاز را نشــان نمی‌دهد. مثلاً رشــته ۳۴۱۲ نشان می‌دهد که ما از شهر ۱ به شه ۳ می‌رویم و به شهر ۱ باز میگردیم و از شهر ۲ به شهر ۴ می‌رویم و به شهر ۲ باز می‌گردیم که یک رشته بی ربط است.

۲- Crossover

متد اول Partially Matched Crossover (PMX) می‌باشد.

اگر دو رشته مقابل را داشته باشیم:

$$1234|567|8$$

$$8521|567|7$$

و یک Crossover دو نقطه‌ای انجام دهیم خواهیم داشت:

$$v_1 = 1234|364|8$$

$$v_2 = 8521|567|7$$

که به طور بدیهی غیرمجاز هستند چون v_1 شهر ۵ یا ۷ را بازدید نمی‌کند و شهرهای ۳ و ۴ را دو بار بازدید می‌کند. به طور مشابه v_2 شهرهای ۳ و ۴ را نمی‌بیند و شهرهای ۵ و ۷ را دو بار می‌بیند.

PMX بدون اســتفاده از ابزاری این مشــکل را به این گونه حل می‌کند که تعویض‌های $3 \leftrightarrow 5$ و $6 \leftrightarrow 6$ و $4 \leftrightarrow 7$ را انجام می‌دهد ســپس این تعویض‌ها را عیناً روی ژن‌های خارج از نقاط Crossover تکرار می‌کند.

و به این ترتیب رشته‌های زیر ساخته می‌شود:

۱ ۲ ۵ ۷ ۳ ۶ ۴ ۸

۸ ۳ ۲ ۱ ۵ ۶ ۷ ۴

ولی ما در این روش لزوماً یک دور مجاز تولید نمی‌کنیم به همین دلیل نیاز داریم تا با یک روتین بهتر برای Crossover راه‌حل بهتری پیدا کنیم و دورهایی که مجازند را تولید کنیم.

متد دوم Cycle Crossover (CX) می‌باشد.

لازم به ذکر است که این متد Crossover روی اولین شیوه Encoding عمل می‌کند یعنی رشته ۱۲۳۴ به این مفهوم است که به ترتیب از ۱ به ۲ و به ۳ و به ۴ می‌رویم و در آخر به ۱ بر می‌گردیم.

فرض کنید دو رشته زیر را داریم:

۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸

۸ ۵ ۲ ۱ ۳ ۶ ۴ ۷

ما برای تولید یک رشته جدید v_1 به این شکل عمل می‌کنیم:

اولین ژن را از یکی از Parent ها انتخاب می‌کنیم:

$$v_1 = 1 - - - - - - -$$

با ید هر عنصر را از یکی از Parent ها برداریم و آنرا در موقعیتی که قبلاً بوده قرار می‌دهیم.

از آنجایی که اولین موقعیت توسط ۱ اشغال شده است عدد ۸ از رشته دوم نمی‌تواند به آن محل برود پس مجبوریم ۸ را هم از رشته اول برداریم:

$$v_1 = 1------8$$

و به همین شکل ۷ و ۴ را نیز از رشته اول برداشته و در محل خود قرار می‌دهیم:

$$v_1 = 1--4--78$$

حال با پر کردن بقیه محل‌ها با عناصر آن محل‌ها از رشته دوم v_1 را می‌سازیم.

این متد همیشه یک کروموزوم مجاز می‌سازد.

البته ممکن است فرزند درست شده همان Parent باشد ولی این مشکل نیست چون نشانگر اینست که Fitness و Parent بالایی دارد و باز هم می‌تواند یک انتخاب باشد.

متد سوم Order Crossover می‌باشد که خیلی به PMX شبیه است.

همان دو نقطه را در نظر می‌گیرد ولی به جای اصلاح کروموزوم‌ها با تعویض تکرارها به طور ساده‌تری بقیه ژن‌ها را مرتب می‌کند تا یک دور مجاز بدهد.

مثلا اگر دو رشته زیر را داشته باشیم:

$$135|762|48$$

$$563|821|47$$

با تعویض کردن ژنهای بین دو نقطه به دست می‌آید:

$$v_1 = ---|821|--$$

$$v_2 = - - - |762| - -$$

سپس با شروع از دومین نقطه Crossover ژن‌ها را از کروموزوم Parent ثبت می‌کنیم.

۴ ۸ ۱ ۳ ۵ ۷ ۶ ۲

۴ ۷ ۵ ۶ ۳ ۸ ۲ ۱

سپس ژن‌هایی را که بین نقاط Crossover بوده‌اند را حذف می‌کنیم. یعنی ۸ و ۲ و ۱ را از لیست v_1 و ۷ و ۶ و ۲ را از لیست v_2 حذف می‌کنیم تا رشته‌های زیر به دست آیند:

۴ ۳ ۵ ۷ ۶

۴ ۵ ۳ ۸ ۱

و با جایگزینی از دومین نقطه Crossover با کروموزوم‌های فرزند رشته‌های نهایی زیر به دست می‌آیند.

$$v_1 = 57682143$$

$$v_2 = 38176245$$

متد چهارم Crossover Matrix می‌باشد همان Crossover یک یا دو نقطه‌ای می‌باشد.

اگر ماتریس‌های زیر را داشته باشیم:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$B=\begin{bmatrix}0 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}$$

نقاط Crossover را بعد از اولین ستون و بعد از دومین ستون انتخاب می‌کنیم. Crossover کردن ستون‌ها نتیجه می‌دهد:

$$A'=\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 1\\ 1 & 1 & 0\end{bmatrix}$$

$$B'=\begin{bmatrix}0 & 1 & 1\\ 1 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}$$

حالا ما چندین ۱ در بعضی سطرها داریم و بعضی از سطرها اصلا ۱ ندارند. این را بو سیله یکی از ۱ ها از سطری که چندین ۱ دارد به سطری که هیچ یکی ندارد درست می‌کنیم.

این انتخاب بین سطرهای شامل چند ۱ تصادفی است.

$$A''=\begin{bmatrix}1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{bmatrix}$$

$$B''=\begin{bmatrix}0 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}$$

حال با ملاحظه ماتریس اول می‌بینیم که $a\leftrightarrow a$ و $b\rightarrow c\rightarrow d$. پس ما دو دور مختلف ایجاد کرده‌ایم ولی این را بوسیله برش و متصل کردن درست می‌کنیم.

متد پنجم Modified Order Crossover (MOX) شبیه Crossover یک نقطه‌ای است.

یک نقطه Crossover تصـادفی در Parent انتخاب می‌کنیم و به طور معمول ژن‌های قبل از نقطه را همانطور که هسـتند رها می‌کنیم. سـپس ژن‌های بعد از نقطه Crossover را به ترتیبی که در دومین کروموزوم Parent ظاهر شده‌اند دوباره مرتب می‌کنیم.

اگر دو کروموزوم زیر را داشته باشیم:

123|456

364|215

به دست خواهیم آورد:

$v_1 = 123|645$

$v_2 = 364|125$

Crossover تا حالا تمرکز روی موقعیت شـهر در دور را جسـتجو کرده در حالیکه در واقع یال‌ها مهمترین قسـمت‌های سـفر فروشـندهٔ دوره‌گرد هسـتند زیرا آنها وزن‌ها را معین می‌کنند بنابراین چیزی که ما واقعاً می‌خواهیم این ا ست که با یال‌ها بیشتر از موقعیت هر شهر سر و کار داشته باشیم.

Grefenstette (1981) یک روتین Crossover اختراع کرده که هر رأس را از یکی از آنهایی که برای رأس جاری در یکی از Parentها لازم است بر می‌دارد. این را بوسیله ایجاد یک لیست یال برای هر رأس انجام می‌دهیم.

کروموزوم‌های:

$v_1 = 123456$

$$v_2 = 123645$$

ابتدا یکی از نخســتین رأس ها را از یکی از Parent ها یعنی ۱ یا ۳ در این مثال انتخاب می‌کنیم آن را که کمترین عدد از رأس‌های لازم را دارد انتخاب می‌کنیم و یا اگر آنها عدد یکسانی دارند تصادفی یکی را انتخاب می‌کنیم.

سپس مشاهده می‌کنیم که رأس‌هایی با رأس ۱ تلاقی می‌کنند. چون این همان رأ سی ا ست که اول انتخاب کردیم، دوباره رأســی با کمترین عدد از راســهای لازم که قبلا انتخاب نشــده‌اند انتخاب می‌کنیم ســپس رأس ۲ را انتخاب می‌کنیم این فرآیند انتخاب رأس‌های فرضــی را ادامه می‌دهیم اگر به حالتی برخورد کنیم که نتوانیم رأ سی را که قبلاً انتخاب نشده را انتخاب کنیم یک گره که قبلاً انتخاب نشده است را تصادفی انتخاب می‌کنیم. این به آن معنی است که ما می‌خواهیم رأ سی را به د ست آوریم که در رأس جاری ما در یکی از Parent ها می‌توانند فرزند تولید کنند. توجه داشــته باشــید که ما در تمام مدت در توانایی انتخاب رأس‌هایی که در یکی از Parent ها لازم بودند موفق بودیم. ما فقط یک فرزند از این Crossover به دســت می‌آوریم تا بســیاری از Crossover ها را دو بار انجام دهیم تا یک نســل جد ید ایجاد کنیم. همچنین عملکرد های Crossover ای داریم که از اطلاعات مکاشفه‌ای استفاده می‌کند.

Heuristic Crossover یک رأس تصادفی برای شروع انتخاب می‌کند و سپس به دو یالی که یک دور را نشان نمی‌دهد بر می‌دارد. اگر هر دو یال یک دور را نشان می‌دهند بطور تصادفی یالی را که این کار را انجام ندهد انتخاب می‌کنیم.

۳- Mutation:

ابتدا عملگر «دو گزینه‌ای» (2-opt)[1] را نشان می‌دهیم. دو یال (a,b) و (c,d) را از دورمان انتخاب می‌کنیم و چک می‌کنیم که آیا می‌توانیم این ۴ رأس را با یک روش متفاوت به هم وصل

[1] 2-Operation

کنیم تا کمترین وزن را به ما بدهد یا خیر. برای انجام این کار چک می‌کنیم که اگر $cab+ccd>cac+cdb$ باشد یال‌های (c,d) و (a,b) را با یال‌های (d,b) و (a,c) عوض می‌کنیم. توجه کنید که فرض کرده‌ایم که a و b و c و d با ترتیب مشخصی در دور ظاهر شده‌اند حتّی اگر b و c متصل نباشند.

همچنین یک عملگر «سه گزینه‌ای» (3-opt)[1] داریم که بجای دو یال سه یال تصادفی را نشان می‌دهد. اگر یال‌های $cab+ccd+cef>cac+cbe+cdf$ باشد یال‌های (a , b) و (c , d) و (e , f) را با یال‌های (a , c) و (b , e) و (d , f) عوض کنیم.

عملگر (or – opt) شبیه (2- opt) است. یک مجموعه از رأس‌های متصل را تصادفی انتخاب می‌کنیم و چک می‌کنیم که آیا این رشته می‌تواند بین دو رأس دیگر اضافه شود تا وزن تقلیل یابد یا خیر. ما می‌توانیم این را بوسیله پیدا کردن مجموع وزن‌های یالهای اضافه شده و مجموع وزن‌های یال‌های حذف شده محاسبه می‌کنیم. اگر وزن یال‌های حذف شده بیشتر بود تعویض انجام گیرد.

سه عملگر Mutation دیگر نیز وجود دارند که یک شهر انتخابی را به یک مکان انتخاب شده تصادفی اضافه می‌کند. همچنین ما هنگامی که دو شهر تصادفی را انتخاب می‌کنیم و آنها را تعویض می‌کنیم یک Mutation دو جانبه داریم.

۳-۳-۲- مقایسه روشهای مختلف الگوریتم و ژنتیک برای TSP

تا به حال به فرم های مختلفی رمز گذاری ها Encoding و عملگر های Crossover و Mutation را در حل مسـأله TSP به روش الگوریتم ژنتیک دیدیم. این ترکیب‌ها با هم ترکیب شوند و منجر به ر سیدن به راه‌حل‌های مختلفی برای TSP به روش الگوریتم ژنتیک شوند. ولی از آنجایی که متدهای Crossover روی Encoding های خاصی عمل می‌کنند در نتیجه الگوریتم‌های

[1] 3-Operation

ژنتیک خیلی متفاوتی برای جستجو نداریم حال به بررسی الگوریتم‌های ژنتیک خیلی متفاوت برای جستجو نداریم.

حال به بررسی الگوریتم‌های ژنتیک محض یعنی بدون استفاده از Heuristic Information می‌پردازیم.

فرض کنید که PMX Crossover نتیجه‌ای برابر ۵۱۷ می‌دهد. از آنجایی که می‌دانیم در این مسأله خاص (۳۰ شهر) بهترین جواب طولی برابر ۴۲۰ دارد به نظر می‌رسد Order Crossoverجوابی بهتر از بقیه بر می‌گرداند.

حال به بررسی Matrix Crossover می‌پردازیم. اگر ا یک Crossover دو نقطه‌ای استفاده کنیم مشاهده می‌کنیم که برای ۳۰ و ۵۰ و ۷۵ و ۱۰۰ و ۳۱۸ دورهایی با طول ۴۲۰ و ۴۲۶ و ۵۲۵ و ۶۲۹ و ۴۲۱۵۴ را ارائه می‌کند. که همه این جواب‌ها کمتر از ۲ درصد بیشتر از جواب بهینه هستند. پس احتمالاً ا ستفاده از یال‌ها بسیار امیدوار کننده‌تر از ا ستفاده از رأس‌ها به عنوان متغیر است. توجه کنید که به هر حال نمایش ماتریسی فضای بیشتری را برای ذخیره کردن نسبت به نمایش به صورت عدد صحیح و Crossover ساده می‌خواهد و در ضمن محاسبات و Mutation در ماتریس پیچیده‌تر و زمانبرتر است.

همچنین روش دیگری که تست شده اینست که ما از (2- opt) برای Mutation ا ستفاده کنیم و از Crossover استفاده نکنیم. این روش نیز جواب خوبی ارائه می‌دهد ولی جواب قبلی بهتر از این روش است. در ضمن برای وقتی که n را زیاد می‌کنیم این روش جوابی مناسب ارائه نمی‌دهد.

Heuristic algorithm نیز به جواب خوبی می‌رسد. Heuristic algorithm وقتی که با Mutation (2 – opt) ترکیب می‌شود بهترین جواب را در مقایسه با متدهایی که تا به حال

گفتیم بر می‌گرداند. به طوری که این جواب بسیار نزدیک به مقدار بهترین جواب است. البته این روش فضای زیادی را اشغال می‌کند و نیز وزن هر یال باید در جایی ذخیره شود.

در نتیجه می‌بینیم که الگوریتم ژنتیک وقتی که از نمایش ماتریس برای Encoding و از Matrix Crossover یا Heuristic Crossover استفاده می‌کند بهترین جواب را بر می‌گرداند و بهتر از دیگر روش ها کار می‌کند. در هر دو روش Crossover بالا استفاده از (2- opt) Mutation کیفیت الگوریتم را افزایش می‌دهد.

۳-۳-۳- نتیجه گیری

الگوریتم‌های ژننتیک به نظر می‌رسد که یک جواب خوب برای TSP پیدا می‌کند. در حالیکه کارایی الگوریتم به میزان زیادی به نحوه Encode کردن و نیزه Crossover و Mutation بستگی دارد. به نظر می‌رسد که استفاده از نمایش Matrix و Heuristic Information بهتر از بقیه روش‌ها کار می‌کند و جواب قابل قبولی را بر می‌گرداند که به جواب واقعی بسیار نزدیک است.

احتمالاً الگوریتم ژنتیک روش بهتری نسبت به دیگر روش‌ها برای TSP است ولی هنوز جواب بهتری نسبت به دیگر روشهای موجود پیدا نکرده است. ولی این را نیز می‌دانیم که بهترین الگوریتم‌های غیرژنتیکی ارائه شده برای TSP در حالت‌های خاصی از الگوریتمهای ژنتیک ارائه شده است. پس ما امیدواریم که با ارائه روتین‌های بهتری برای Encoding و Crossover و Mutation راه‌حل‌های مناسبتری برای مسأله فروشندهٔ دوره‌گرد ارائه شود.[13]

۳-۴- حلّ مسأله معمای سودوکو[1]

[1] Sodoku

احتمالاً همه شما با جداول سودوکو آشنایید، جداول بازی با اعدادی که اکنون در اکثر روزنامه‌ها و مجله‌ها در قسمت سرگرمی فضایی را به خود اختصاص داده‌اند یا شاید در اطرافتان افرادی باشند که به صورت جدی به حل مداوم اینگونه جداول می‌پردازند. جداول سودوکو در حقیقت یک جدول ۹×۹ است که بعضی از خانه‌های آن با اعدادی بین ۱ تا ۹ پر شده است. شما می‌توانید در هر خانه یکی از اعداد ۱ تا ۹ را قرار دهید اما باید این کار را طوری انجام دهید که سه شرط زیر برقرار باشد.

۱- در هر سطر هیچ عدد تکراری وجود نداشته باشد.

۲- در هر ستون هیچ عدد تکراری نباید وجود داشته باشد.

۳- در هر جدول ۳×۳ طبق شکل نیز باید هیچ عدد تکراری وجود داشته باشد.

اگر به حل یک نمونه جدول سودوکو تمایل دارید جدول زیر را حل کنید.

8			4		6			7
						4		
	1					6	5	
5		9		3		7	8	
				7				
	4	8		2		1		3
	5	2					9	
		1						
3			9		2			5

شکل ۳-۸- جدول سودوکو.

مسأله مورد نظر در اینجا بدین صورت است که فرض کنید یک جدول سودوکوی خالی دارید و از شما می‌خواهند مثلاً پنجاه راه‌حل معرفی کنید. یعنی طراحی و رسم پنجاه جدول که خاصیت سودوکو را داشته باشد. توجه کنید که در مسأله ما همه خانه‌ها خالی هستند. برای این که زمان

اجرای این برنامه کمتر شود و امکان رسیدن به جواب در خانه نیز مهیّا باشد، در صورت مسأله مورد نظر تغییر کوچکی داده شده است، به این صورت که فرض می‌شود در هر خانه غیر از اعداد ۱ تا ۹ عدد صفر را نیز می‌توان قرار داد.

اگر بخواهیم بیان پیشرفته‌تری از صورت مسأله داشته باشیم، در حقیقت این مسأله، پیدا کردن حالت‌های مناسبی است در بین همه حالات ممکن برای جدول مزبور ما. در هر خانه جدول ده عدد مختلف می‌تواند قرار بگیرد. از طرفی دارای ۸۱ خانه هستیم.

پس مقدار حالات ممکن برای مسأله ما برابر $(81)^{10}$ است و از میان همه این حالات ما می‌خواهیم به دنبال حالت‌هایی بگردیم که شرایط مورد نظر ما را داشته باشد. بدیهی است پیمایش همه این حالات، حتی برای کامپیوترهای امروزی که ما از آنها استفاده می‌کنیم و در مدت زمان طبیعی، امری محال است.

۳-۴-۱- حل مسأله

با توجه به این تعداد حالات مسأله بسیار زیاد است، پیمایش کل فضای درخت حالت مسأله غیرممکن است. ازطرفی ما نیازمند پنجاه جواب درست هستیم و مطمئناً جواب‌های مسأله ما در فضای درخت حالت مسأله پراکنده هستند و در نهایت این شرایط این ایده را به وجود می‌آورد که برای حل مسأله از الگوریتم‌های ژنتیک استفاده کنیم.

۳-۴-۲- تعیین کروموزم

قدم اول در حل مسأله تعیین ساختار کروموزم است. همانطور که می‌دانیم هر کروموزم درالگوریتم ژنتیک، معادل یک وضعیت از حالات ممکن برای فضای حالت مسأله است.

ما جدول سودوکو را در قالب یک آرایه یک بعدی از رکوردها[۱] تصور کردیم یعنی برای پیاده‌سازی ساختار این برنامه از ساختمان‌داده رکورد استفاده کردیم؛ بدین صورت که بخش اول

[۱] Record

رکورد مجموعه ایست از متغیرهای ناهمگون؛ برخلاف آرایه که عناصرآن تماماً از یک نوعند.

رکورد آرایه‌ای است دو بعدی به ابعاد ۹×۹ که برای نگهداری کروموزوم‌ها استفاده می‌شود و بخش دوم متغیری است که برای ذخیره مقدار تابع برازش مورد استفاده قرار می‌گیرد.

(کد)

```
Type
Tchromo=Record
Bits:Array [0..8,0..8] of Byte;
Fitness:Byte;
End;
Var  G_Population : Array [0..49] of  Tchromo;
G_Temp : Array [0..999] of  Tchromo;
```

۳-۴-۳- ساختن جمعیت آغازین یا نسل اول

همانطور که می‌دانیم، جمعیت آغازین را می‌توانیم به صورت تصادفی ایجاد کنیم. یعنی در هر خانه از یک نمونه از جدول مورد نظر یک عدد تصادفی بین ۰ تا ۹ قرار دهیم. در اینجا ذکر چند نکته لازم است: اول این‌که، در کد نوشته شده تعداد کروموزم‌های جمعیت آغازین را برابر پنجاه فرض می کنیم و از این به بعد از هر نسل پنجاه عنصر انتخاب می‌شوند و به عنوان عناصر سازنده نسل بعدی مورد استفاده قرار می‌گیرند، از طرف دیگر، اگر بتوانیم جمعیت آغازین خود را به گونه‌ای انتخاب کنیم که در عین تصادفی بودن قسمت‌هایی از شروط را نیز در خود داشته باشد، مطمئناً بسیار سریع‌تر به جواب می‌رسیم؛ چراکه هر چه جمعیت آغازین بهتر باشد، احتمال دسترسی سریع به جواب بیشتر است.

با توجه به این ایده می‌توانیم جمعیت آغازین را به‌گونه‌ای طراحی کنیم که با این‌که در هر خانه یک عدد تصادفی قرار داشته باشد، در هر سطر هیچ عدد تکراری نداشته باشیم.

۳-۴-۴- ساختن تابع از ارزش

در ادامه باید تابعی ایجاد کنیم که بتوانیم توسط آن به هر نمونه عددی متناسب با میزان خوب بودن آن را اختصاص دهیم. این تابع را که AssignFit می‌نامیم، بدین‌صورت تعریف می‌کنیم: به ازای هر سطر یا ستون یا هر مربع ۳×۳ که شرط جدول سودوکو را داشته باشد، یک واحد به جواب تابع AssignFit اضافه می‌کنیم، بدین‌صورت تابع AssignFit تابعی خواهد بود که دارای جواب بین صفر تا ۲۷ باشد. البته بدیهی است که اگر نمونه‌ای دارای AssignFit برابر ۲۷ باشد، بدین معنی است که معادل یکی از المان‌های جواب نهایی سودوکو مسأله ما است. همانطور که ذکر شد جمعیت اولیه ما برابر ۵۰ خواهد بود(N=50) یعنی یک آرایه ۵۰عنصری از نوع رکورد Tchromo. همچنین باید توجه داشت که خروجی تابع AssignFit در متغیرFitness از رکورد Tchromo ذخیره خواهد شد.

(کد)

```
Function AssignFit(Popu:TChoromo):Byte;
var
Row,Col,TT,Rank:Byte;
FlgA:Boolean;
begin
Rank:=0;

//Test Row...
for Row:=0 to 8 do
begin
FlgA:=True;
```

```
for Col:=0 to 8 do
for TT:=0 to 8 do
if (Popu.bits[Row,Col]=Popu.bits[Row,TT]) and (Col<>TT) then
FlgA:=False;

if FlgA=True then
Rank:=Rank+1;

end;

//Test Column...
...
//Test Squar3*3
...
...
AssignFit:=Rank;

end;
```


۳-۴-۵- ترکیب نمونه‌ها و ساختن جواب جدید

در ادامه باید هر بار دو نمونه را انتخاب کنیم و از ترکیب هر دو نمونه دو جواب جدید به دست آوردیم. باید توجه داشت که نباید یک نمونه با خودش ترکیب شود؛ چراکه دو جواب عیناً مثل خودش تولید می‌کند. در نهایت اگر بخواهیم به طور خلاصه بیان کنیم، با انتخاب دو عدد

تصادفی دو نمونه را انتخاب می‌کنیم و البته باید این اعداد تصادفی به گونه‌ای باشند که نمونه‌هایی که دارای مقدار تابع ارزش بیشتری هستند، به همان نسبت نیز دارای شانس انتخاب بیشتری نسبت به دیگر اعضای جامعه داشته باشند. ما در اینجا از تکنیک چرخ رولت استفاده می‌کنیم. می‌توان گفت روشی که ما در اینجا برای پیاده سازی چرخ رولت استفاده می‌کنیم نوعی روش ابداعی توسط خود اینجانب[1] است، که نمونه‌ای از آن را جایی مشاهده نکردم!

(کد)

```
Function Roulette (TotalFitness:Byte):TChoromo;

var j,m:Integer;
SliceRoulette:Single;

begin
//Randomize;
SliceRoulette:=(Random(TotalFitness));

for j:=0 to Pop_Size-1 do
begin
SliceRoulette:=SliceRoulette-G_Population[j].Fitness;

if (SliceRoulette<=0) and (j<>NoRep1) and (j<>NoRep2) then
begin
```

[1] «مهدی خلیلی‌نیا»

```
for m:=0 to 8 do
Result:=G_Population[j];

if (SWT=true) then
NoRep1:=j
else
NoRep2:=j;

SWT:=not(SWT);
Break;
end;

end;
end;
```


متغیرهای NoRep1 و NoRep2 از انتخاب دو مرتبه یک عنصر برای ترکیب(با خودش) جلوگیری می‌کنند. پس از این انتخاب دو نمونه، با تولید یک عدد تصادفی دیگر میزان احتمال ترکیب این دو عنصر با یکدیگر را محاسبه و سپس با انتخاب دو عدد تصادفی بین صفر تا هشت شماره سطر و ستون نقطه شکست را به دست می‌آوریم و عمل ترکیب را انجام می‌دهیم و دو جواب جدید به دست می‌آوریم، اما صرف عمل ترکیب ما را به جواب نهایی رهنمون نخواهد کرد، بلکه باید از عمل جهش نیز استفاده کنیم.

بنابراین به ازای هر جواب به دست آمده است این اعمال را انجام دهیم. ابتدا یک عدد تصادفی بین یک تا صفر انتخاب می‌کنیم. اگر عدد تولید شده کوچکتر از نرخ جهش(این عدد را معمولاً برابر ۰,۰۰۱ می گیرند) بود، عمل جهش را انجام نمی‌دهیم، اما اگر بیشتر بود، دو عدد

تصادفی دیگر بین صفر تا هشت به نشانه شماره سطر و ستون خانه‌ای که باید در آن جهش انجام شود و دو عدد تصادفی بین صفر تا نُه به عنوان خانه هایی که باید مقادیر آنها جابجا شوند به دست می‌آوریم و عمل جهش را در آنها انجام می‌دهیم. بدین‌ترتیب هر جواب ممکن است با احتمال یک هزارم در دوتا از ژن‌های خود دچار جهش بشود.

(کد)

```
Procedure Mutate(var Table:TChoromo);
var m,n,P1,P2,Tmp:Byte;
begin
for i:=0 to 8 do
if (Random<Mutaion_Rate) then
begin
while (P1=P2) do
begin
P1:=Random(9);
P2:=Random(9);
end;

Tmp:=Table.bits[i,P1];
Table.bits[i,P1]:=Table.bits[i,P2];
Table.bits[i,P2]:=Tmp;
end

end;
```

```
//*****************************************

if (Random<Crossover_Rate) then

begin

while (Cr1>=Cr2) do

begin

Cr1:=Random(9);

Cr2:=Random(9);

end;


for RR:=0 to Cr1 do

for CC:=0 to 8 do

begin

G_Temp[CPop].bits[RR,CC]:=Bit1.Bits[RR,CC];

G_Temp[CPop+1].bits[RR,CC]:=Bit2.Bits[RR,CC];

end;


for RR:=Cr1 to Cr2-1 do

for CC:=0 to 8 do

begin

G_Temp[CPop].bits[RR+1,CC]:=Bit2.Bits[RR+1,CC];

G_Temp[CPop+1].bits[RR+1,CC]:=Bit1.Bits[RR+1,CC];

end;
```

```
for CC:=0 to Cr2 do
begin
G_Temp[CPop].bits[Cr1,CC]:=Bit1.Bits[Cr1,CC];
G_Temp[CPop+1].bits[Cr1,CC]:=Bit2.Bits[Cr1,CC];
end;

for CC:=Cr2+1 to 8 do
begin
G_Temp[CPop].bits[Cr1,CC]:=Bit2.Bits[Cr1,CC];
G_Temp[CPop+1].bits[Cr1,CC]:=Bit1.Bits[Cr1,CC];
end;
...
//Now Mutation...
Mutate(G_Temp[CPop]);
Mutate(G_Temp[CPop+1]);
```


۳-۴-۶- ارزشیابی مجموعه جواب

اکنون دارای یک آرایه با هزار عنصر از نوع رکورد Tchromo به عنوان جواب هستیم که مطمئناً بعضی از این جواب‌ها نسبت به بقیه بهتر هستند و همانطور که می‌دانیم، برای ساختن نسل بعد، باید از جواب‌هایی استفاده کنیم که از نظر ارزش، دارای رتبه بهتری باشند. بدین‌ترتیب باید

در ادامه جواب‌های خود را ارزشیابی کنیم. این عمل دقیقاً همان تابع AssignFit است، اما این بار مجموعه هزارتایی را ارزشیابی می‌نماید

۳-۴-۷- ساختن نسل بعد

در اینجا باید از میان هزار جواب به دست آمده، جواب‌های برتر را به عنوان والدهای نسل بعدی انتخاب کنیم و بقیه را دور بریزیم، که در آن از همان تکنیک چرخ رولت استفاده شده است.

بقیه مراحل مشخص است که دوباره تکرار مراحل قبل می‌باشد، یعنی ارزشیابی مجموعه والد، انتخاب دو والد تصادفی متناسب با تابع ارزششان و ساختن جواب جدید و تکرار این مرحله تا به دست آوردن نسل بعدی و در نهایت انتخاب والدهای بعدی از میان نسلی به دست آمده و تکرار دوباره... [13]

۳-۵- مرتب سازی به کمک GA

یکی از مهمترین مسائل کلاسیک و آموزشی که همه دانشجویان رشته کامپیوتر-نرم‌افزار در طول دورهٔ تحصیلی فرا می‌گیرند و با آن سروکار دارند حلّ مسائل مربوط به مرتب‌سازی (اعم از حروف، آرایه‌ها و دیگر ساختمان داده‌های مشابه) به صورت نزولی یا صعودی به کمک بعضی از الگوریتم‌ها می‌باشد.

برای مرتب‌سازی آرایه‌ای با طول نَه چندان بلند، شاید GA گزینهٔ مناسبی نباشد و بتوان از دیگر الگوریتم‌ها مانند: مرتب سازی حبابی، مرتب سازی درجی، مرتب سازی تعویضی یا جابجایی، مرتب سازی هرمی یا درخت نیمه مرتب و ... استفاده کرد که پیاده‌سازی و کد نویسی آنها نیز بسیار راحت تر از GA می‌باشد، ولی بدون شک در آرایه ها و یا دیگر ساختمان داده های بزرگتر و با طول بیشتر الگوریتم‌های ذکر شده از عملکرد زمانی خوبی بهره‌مند نمی‌باشند، حال آنکه به کمک GA می‌توان این زمان را به طرز شگفت‌انگیزی کاهش داد!

در ادامه فقط برای درک مطلب و آشنایی شما خواننده عزیز با این نوع مرتب‌سازی به کمک GA ، ما به عنوان یک مثال ساده و قابل فهم یک آرایه را با طول کوتاه مورد امتحان و بررسی قرار می‌دهیم.

۳-۵-۱- صورت مسأله

اعداد یک تا نُه در یک آرایه یک‌بعدی با طول نُه سلول قرار دارند، می‌خواهیم هریک از عناصر در جای خود (در خانه متناظر با مقدار خود) قرار گیرند؛ به عبارت دیگر در نظر داریم این آرایه را به صورت صعودی مرتب کنیم.

۳-۵-۲- جمعیت آغازین

ابتدا سعی می‌کنیم ساختار اصلی جمعیت را طراحی کنیم؛ طبق اصول GA، هر آرایه معادل یک کروموزوم و هر سول از این آرایه را یک ژن در نظر می‌گیریم. همچنین نیاز به یک متغیر برای نگهداری میزان برازندگی (تناسب) این کروموزوم داریم. برای راحتی کار از ساختمان داده رکورد استفاده می‌کنیم، بدین صورت که هر کروموزوم را به همراه متغیر برازندگی آن در یک رکورد قرار می‌دهیم، سپس مجدداً یک آرایه N عنصری به عنوان جامعه اولیه ایجاد می کنیم. (N=20)

(کد)

```
Type
Tchromo=Record
Bits:Array [0..8] of Byte;
Fitness:Byte;
End;
Var  G_Population : Array [0..N] of  Tchromo;
G_Temp : Array [0..T] of  Tchromo;
```

ما در اینجا(روی کاغذ) جمعیت اولیه را با شش عضو تشکیل می‌دهیم، توجه کنید که ما به کمک تابع تولید اعداد تصادفی، اعداد را طوری در آرایه می‌چینیم که هر عدد منحصر به فرد باشد؛ یعنی عدد تکراری در آرایه نداشته باشم.

(کد)

```
//************************************
Procedure GetRandomBit(var TABLE:TChromo);
var
chrm:TChromo;
Row,TT,RM:Byte;
FlgA,FlgB:Boolean;
str:String;
begin
TABLE:=chrm;
for Row:=0 to 8 do
begin
FlgB:=True;
while (FlgB=True) do
begin
RM:=Random(9)+1;
FlgA:=True;
for TT:=0 to 8 do
begin
if (TABLE.bits[TT]=RM) then
```

```
FlgA:=False;
end;

if (FlgA) then
begin
TABLE.bits[Row]:=RM;
FlgB:=False;
str:=str+IntToStr(TABLE.Bits[Row]);
end;

end;  //End While.
end;
str:=str+'  : '+IntToStr(Form1.AssFit(str));
Form1.Memo1.Lines.Add(str);
str:='';
TABLE.Fitness:=0;
end;
//************************************
```

جمعیت اولیه زیر را در نظر بگیرید.

1)	5	1	2	4	9	6	7	3	8
2)	1	2	3	4	5	7	6	8	9
3)	9	3	6	4	5	1	7	8	2
4)	6	5	4	9	2	3	1	8	7

5)	6	2	4	8	9	1	7	3	5
6)	4	9	8	2	1	3	7	5	6

شکل ۳-۱- جمعیت اولیه‌ای از آرایه یک‌بعدی با نُه سلول.

۳-۵-۳- تابع برازندگی

همانطور که می‌دانید مقدار تابع برازش (تناسب) میزان مشارکت هر کروموزوم در آمیزش و تشکیل جمعیت جدید را مشخص می‌نماید، و همینطور طراحی این تابع وابستگی مستقیم با میزان توانایی فکری و سلیقه شخص طراح دارد. تکنیکی که ما در اینجا به کار می بریم ساده‌ترین و صد البته راحت‌ترین راه ممکن برای پیاده‌سازی تابع برازش می‌باشد؛ بدین‌صورت که مقدار هر عنصر از آرایه را با موقعیت مکانیش در آرایه مقایسه می‌کنیم، اگر با یکدیگر برابر بودند یک واحد به تابع برازش اضافه می‌کنیم. پر واضح است که آرایه یا کروموزومی که دارای تناسبی با مقدار نُه باشد پاسخ مسأله خواهد بود. در زیر مقدار برازندگی هر کروموزوم به همراه میزان احتمال مشارکت هر یک از اعضای جمعیت ابتدایی نشان داده شده است. (به کمک رابطه $(P_k = \frac{f_k}{\sum_{i=1}^{n} f_i}$

$$f_1 = 3 \Rightarrow \%17$$
$$f_2 = 7 \Rightarrow \%39$$
$$f_3 = 4 \Rightarrow \%22$$
$$f_4 = 1 \Rightarrow \%5$$
$$f_5 = 2 \Rightarrow \%11$$
$$f_6 = 1 \Rightarrow \%5$$

۳-۵-۴- انتخاب

پس از تعیین میزان تناسب همه اعضاء نوبت به انتخاب عضوهای مناسب می‌باشد. در این مرحله تکنیک چرخ رولت به کمک ما می‌آید. تابع چرخ رولت جفت کروموزوم‌هایی را برای مشارکت در تولید نسل بعدی انتخاب می کند؛ عناصر ۱ و ۲ با هم، ۲ و ۳ باهم، ۱ و ۳ با هم، ۲ و

۵ نیز با یکدیگر ترکیب می شوند. کاملاً مشخص است که عناصر ۲ و ۳ به دلیل تناسب بالاتر از دیگر اعضاء، بیشتر در ترکیب‌ها شرکت می‌کنند و در مقابل ۶ و ۴ شرکت داده نمی‌شوند.

۳-۵-۵- ترکیب

در ترکیب کروموزوم‌ها از روش «ترتیب» استفاده می‌کنیم که که در مبحث مربوط به انواع ترکیب‌ها در فصول قبل کاملاً شرح داده شد. مزیت استفاده از این روش در این است که از به وجود آمدن عناصر تکراری در آرایه جلوگیری می‌کند.

دو عدد به صورت تصادفی انتخاب می‌کنیم:

$P_1 = 5$ و $P_2 = 8$.

1) 51249/673/8 + 2) 12345/768/9

⬇

New1) 12458/643/9 , New2) 51249/768/3

$P_1 = 2$ و $P_2 = 7$.

2) 12/34576/89 + 3) 93/64517/82

⬇

New3) 91/34576/82 , New4) 23/64517/89

$P_1 = 1$ و $P_2 = 6$.

1) 5/12496/738 + 3) 9/36451/782

⬇

New5) 3/12496/578 , New6) 2/36451/978

$P_1 = 3$ و $P_2 = 7$.

2) 123/4576/89 + 5) 624/8917/35

⬇

New7) 289/4576/13 , New8) 234/8917/56

۳-۵-۶- جهش

فزض می‌کنیم که در کروموزوم‌های شمارهٔ سه و شش و هفت جهش صورت می‌گیرد. مجدداً از تابع تولید اعداد تصادفی کمک می‌گیریم و دو عدد تصادفی برای هریک از کروموزوم‌ها به دست می‌آوریم، سپس با توجه به اعداد تولید شده جای مقادیر مورد نظر را با یکدیگر جابجا می‌کنیم.

New2) 513**4**9/768/**2**

New3) **2**1/34576/8**9**

New6) 2/36451/9**87**

New7) 289/45**67**/13

حال مجدداً تابع برازندگی را فراخوانی می‌کنیم تا جمعیت جدید را برازش کند.

1) 512496738 → 3

2) 123457689 → 7

3) 936451782 → 4

5) 624891735 → 1

New1) 124586439 → 4

New2) 513497682 → 3

New3) 213457689 → 5

New6) 236451987 → 3

New7) 289456713 → 4

New8) 234891756 → 1

روند انتخاب، ترکیب و جهش را برای جمعیت جدید ادامه می‌دهیم تا به جواب نهایی برسیم...

کدهای این بخش به صورت کامل در قسمت پیوست آمده است.

فهرست منابع و مراجع

[۱]- شاهمیری ا ، "جستاری بر الگوریتم‌ژنتیک"، روزنامه جام‌جم- ضمیمه کلیک- شماره ۲۰۹، تهران، ۱۳۸۷.

[۲]- شاهمیری ا ، "الگوریتم‌ژنتیک"، روزنامه جام‌جم- ضمیمه کلیک- شماره ۲۱۰، تهران، ۱۳۸۷.

[۳]- باوری ا ، صالحی م ، "الگوریتم‌های‌ژنتیک و بهینه‌سازی سازه‌های مرکب"، انتشارت عابد، تهران، ۱۳۸۷.

[۴]- کیا م ، "الگوریتم‌ژنتیک در MATLAB"، خدمات نشر کیان رایانه سبز، تهران، ۱۳۸۸.

[۵]- رضائی ع ، رنجبران س، "آموزش کاربردی الگوریتم‌ژنتیک در نرم‌افزار MATLAB"، انتشارات آذر، تهران، ۱۳۸۶.

[6]- www.en.wikipededia.org/wiki/Gergor_Mandel.htm

[7]-www.en.wikipededia.org/wiki/Genetic_Algorithm.htm

[8]-
www.aftabir.com/articles/view/applied_sciences/management/هیوریستیک.htm

[9]-www.fa.wikipededia.org/wiki/الگوریتم_جستجو.htm

[10]-www.fa.wikipededia.org/wiki/الگوریتم_ژنتیک.htm

[11]-www.fa.wikipededia.org/wiki/الگوریتم_ژنتیکی.htm

[12]-www.fa.wikipededia.org/wiki/ ژنتیک.htm

[13]-www.prozhe.com/مقاله_الگوریتم_ژنتیک.htm

/www.visat.ir/article/227417-[14]الگوریتم_های_هیوریستیک_چیستند؟.htm

[15]- www.mshams.ir/heuristic_and_metaheuristic.html

[16]-www.daneshju.ir/forum/f339.html

/www.fa.wikipededia.org/wiki- [17] ان‌پی_سخت.htm

پیوست

```
unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls;


type

TForm1 = class(TForm)

Memo1: TMemo;

Button1: TButton;

Button2: TButton;

procedure Button2Click(Sender: TObject);

procedure Button1Click(Sender: TObject);

private

{ Private declarations }

public

Function AssFit(st:String):Byte;

{ Public declarations }
```

```
end;

const
pop_size_P=40;
//Pop_Size_S=60;
Crossover_Rate=1;
Mutation_Rate=0.007;
Max_Allowable_Generation=30;
Chromo_Length=8;

var
Form1: TForm1;

Type
arr=array [0..8] of Byte;
TChromo=Record
Bits:array [0..8]of Byte;
Fitness:Byte;
end;

implementation

//**************
```

```
var
G_Population:array[0..pop_size_P-1] of TChromo;
G_temp:array[0..pop_size_P-1] of TChromo;
Bits1,Bits2,T1,T2,Tmp:arr;
bFound,SWT:Boolean;
NoRep1,NoRep2,GRTFS,Cross1,cross2,CPop,i,TotalFitness:Integer;

{$R *.dfm}

//*************************************
Procedure GetRandomBit(var TABLE:TChromo);
var
chrm:TChromo;
Row,TT,RM:Byte;
FlgA,FlgB:Boolean;
str:String;
begin
TABLE:=chrm;
for Row:=0 to 8 do
begin
FlgB:=True;
while (FlgB=True) do
begin
```

```
RM:=Random(9)+1;
FlgA:=True;
for TT:=0 to 8 do
begin
if (TABLE.bits[TT]=RM) then
FlgA:=False;
end;

if (FlgA) then
begin
TABLE.bits[Row]:=RM;
FlgB:=False;
str:=str+IntToStr(TABLE.Bits[Row]);
end;

end;  //End While.
end;
str:=str+'  : '+IntToStr(Form1.AssFit(str));
Form1.Memo1.Lines.Add(str);
str:='';
TABLE.Fitness:=0;
end;
//**********************************
```

```
Function TForm1.AssFit(st:String):Byte;
var F,i:Byte;
begin
F:=0;
for i:=0 to 8 do
if (st[i]=IntToStr(i+1)) then
F:=F+1;

AssFit:=F;
end;
//************************************
Procedure AssignFitness(var G_Pop:TChromo);
var m,fit:Integer;
begin
fit:=0;
for m:=0 to 8 do
if G_Pop.Bits[m]=m+1 then
fit:=fit+1;

G_Pop.Fitness:=fit;
end;
```

```
//*****************************************

procedure SortG_Pop();

var m,n:Integer;

tmp:TChromo;

begin

for m:=pop_size_P-1 downto 0 do

for n:=0 to m-1 do

if (G_Population[n].Fitness>G_Population[n+1].Fitness) then

begin

tmp:=G_Population[n];

G_Population[n]:=G_Population[n+1];

G_population[n+1]:=tmp;

end;

Beep;

end;

//*****************************************

{procedure SortG_Temp();

var m,n:Integer;

tmp:TChromo;

begin

for m:=pop_size_P-1 downto 0 do

for n:=0 to m-1 do

if (G_temp[n].Fitness>G_temp[n+1].Fitness) then
```

```
begin
tmp:=G_Temp[n];
G_temp[n]:=G_Temp[n+1];
G_temp[n+1]:=tmp;
end;
Beep;
end;}
//*****************************************
Function Roulette (TotalFitness:Byte):arr;

var j,m:Integer;
SliceRoulette:Single;

begin

Randomize;
SliceRoulette:=(Random(TotalFitness));

for j:=0 to pop_size_P-1 do
begin
SliceRoulette:=SliceRoulette-G_Population[j].Fitness;

if (SliceRoulette<=0) and (j<>NoRep1) and (j<>NoRep2) then
```

```
begin
for m:=0 to 8 do
Result[m]:=G_Population[j].bits[m];

if (SWT=true) then
NoRep1:=j
else
NoRep2:=j;

SWT:=not(SWT);
Break;
end;
end;
end;
//*****************************************
Function Elitism():arr;
var j,m:Integer;
begin
for j:=pop_size_P-1 downto 0 do
begin
if (j<>NoRep1) then
begin
```

```
for m:=0 to 8 do
Result[m]:=G_Population[j].bits[m];

NoRep1:=j;
Break;
end;
beep;
end;
end;

//*****************************************
Procedure Mutate(var MBits:arr);
var TM,Rnd1,Rnd2:Byte;
begin
Rnd1:=255;
Rnd2:=255;
if (random<Mutation_Rate) then
begin
Beep;
while (Rnd1=Rnd2) do
begin
Rnd1:=random(9);
Rnd2:=random(9);
```

```
end;
TM:=MBits[Rnd1];
MBits[Rnd1]:=MBits[Rnd2];
MBits[Rnd2]:=TM;
end;
end;
//*****************************************
procedure TForm1.Button1Click(Sender: TObject);
var j,m:Integer;
str1,str2:String;
FF:Boolean;
begin
m:=0;
for j:=0 to pop_size_P-1 do
begin

Memo1.Lines.Add('--- Chromosome : '+IntToStr(j+1)+' ---');
(GetRandomBit(G_Population[j]));
Memo1.Lines.Add('');
end;

GRTFS:=0;
```

```
bFound:=False;
while (not Bfound) do
begin
TotalFitness:=0;

for j:=0 to pop_size_P-1 do
begin
AssignFitness(G_Population[j]);
TotalFitness:=TotalFitness+g_Population[j].Fitness;
end;

for i:=0 to Pop_Size_P-1 do
if (G_Population[i].Fitness=9) then
begin
ShowMessage('Found : ' +IntToStr(i));
bFound:=True;
end;

CPop:=0;

NoRep1:=5000;
NoRep2:=5000;
```

```
while (CPop<pop_size_P) do
begin
SortG_Pop();
Bits1:=Roulette(TotalFitness);
Bits2:=Roulette(TotalFitness);

Randomize;

{       if (Random<Crossover_Rate) then
begin
Cross1:=Random(9);
Memo1.Lines.Add(IntToStr(Cross1));
T1:=Tmp;
T2:=Tmp;

For i:=0 to cross1 do
begin
T1[i]:=bits1[i];
T2[i]:=bits2[i];
end;
For i:=(Cross1)+1 to 8 do
begin
T1[i]:=bits2[i];
```

```
T2[i]:=bits1[i];
end;

end;
}
//if (Random<Crossover_Rate) then
begin
while(Cross1>=cross2) do
begin
Cross1:=Random(9);
Cross2:=Random(9);
end;

T1:=Tmp;
T2:=Tmp;

For i:=Cross1 to cross2 do
T1[i]:=bits1[i];
For i:=Cross1 to cross2 do
T2[i]:=bits2[i];
For i:=0 to 8 do
begin
FF:=True;
```

```
For j:=Cross1 to cross2 do
if T1[j]=Bits2[i] then
FF:=False;

if (FF) then
begin
while(T1[m]<>0)and (m<10)do
m:=m+1;
T1[m]:=bits2[i];
end;
end;
m:=0;
For i:=0 to 8 do
begin
FF:=True;
For j:=Cross1 to cross2 do
if T2[j]=Bits1[i] then
FF:=False;

if (FF) then
begin
while(T2[m]<>0)and (m<10)do
m:=m+1;
```

```
T2[m]:=bits1[i];
end;
end;

m:=0;


for i:=0 to 8 do
begin
str1:=str1+IntToStr(T1[i]);
str2:=str2+IntToStr(T2[i]);
end;
Memo1.Lines.Add(str1);
Memo1.Lines.Add(str2);

str1:='';
str2:='';
Assert(sizeof(T1)=9);
Assert(sizeof(T2)=9);

Bits1:=T1;
Bits2:=T2;
Mutate(Bits1);
```

```
Mutate(Bits2);

for i:=0 to 8 do
G_temp[CPop].Bits[i]:=bits1[i];

CPop:=CPop+1;

for i:=0 to 8 do
G_temp[CPop].Bits[i]:=bits2[i];

CPop:=CPop+1;

end;//End If(Crossover)
```



```
end;//End Loop(CPop<pop_size_S)...
Memo1.Lines.Add('---------');

for i:=0 to (pop_size_P)-1 do
AssignFitness(G_Temp[i]);

//SortG_Temp();
for i:=(0) to pop_size_P-1 do
G_Population[i]:=G_Temp[i];
```

```
GRTFS:=GRTFS+1;

if (GRTFS>Max_Allowable_Generation) then
begin
bFound:=True;
ShowMessage('No SoluTions Found !');
end;
end;
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
Application.Terminate;
end;
end.
```

واژه‌نامه

- Adaptation

اقتباس، تطبیق.

در زیست‌شناسی به تغییرات ژنتیکی برای رسیدن به یک منظور خاص گفته می‌شود، به نحوی که مستلزم اضافه شدن رشته‌هایی به ژن‌های اولیه بشود. در صورت عدم اضافه شدن چیزی، به جای فرآیند تطبیق، فرآیند کپی صورت گرفته است.

مثال: تطبیق چشم در تاریکی و روشنایی برای تنظیم میزان نور وارده به آن.

- Allele

حالت‌های مختلف یک ژن برای قرار گرفتن در یک لوکاس(موقعیت خاص).

هریک از مقادیر متفاوتی که یک ژن از یک کروموزوم در یک موقعیت خاص می‌تواند دارا باشد.

- Answer

جواب.

راه‌حل بهینه، یک یا چند راه‌حل(جواب) مسأله که دارای بالاترین برازش باشند.

- Chromosome

کروموزوم.

هر یک از راه‌حل‌های کاندیدا به عنوان جواب بهینه.

مجموعه‌ای از ژن‌ها با فیلدهای مختلف که به عنوان افراد در استخر ژنی اولیه حضور دارند.

- Crossover

برش، تقاطع، همبری.

یکی از عملگرهای مهم الگوریتم‌های ژنتیک.

در الگوریتم‌های ژنتیک به معنی Sexual Recombination(جفتگیری جنسی) نیز می‌باشد.

- Elitism

نخبه گرایی، نخبه سالاری، بهینه گزینی.

چون ممکن است بدلیل ماهیت تصادفی الگوریتم‌های ژنتیک، بهترین افراد یک نسل ازبین رفته و در نسل بعد حضور نداشته باشند، برای جلوگیری از این مسأله بهترین افراد را بدون هیچگونه تغییری در نسل بعدی کپی می‌نمائیم تا کارآیی را افزایش دهیم.

- Epitasis

خصوصیات و ویژگی‌های یک ژن که در یک نژاد مخفی می‌شود.

- Evolution

فرآیند تغییرات در طول زمان که با تغییر در آرایش ژنتیکی نوع خاصی همراه باشد.

- Feature

خصیصه، ویژگی خاصی که قابل کپی برداری یا انتقال به دیگر ژن‌ها یا کروموزوم‌ها باشد.

- Fitness

شایستگی، برازش.

قابلیت یک کروموزوم خاص از یک ژنوتیپ برای تولید مثل.

- Fitness – Function

تابع برازش، تعیین میزان شایستگی یک راه‌حل.

بررسی مقدار بهینگی یک راه‌حل، Objective function**(تابع هدف).**

- Gene

بلوکی از DNA **که دارای ساختار پروتئینی خاصی است و شکل کلی ارگانیسم را می‌سازد.**

هریک ازبیت‌های یک رشته کروموزومی، مثلاً ژن رنگ چشم یک جانور دارای این مشخصات است: دارای شماره 10 **است (**Locus **آن در** Position **شماره ۱۰ است.)،** Allele Value **آن آبی است (یا مشکی، سبز،....).**

- Genoe

راه‌حل‌های مختلف یک مسأله شامل داده و دستورالعمل که معمولاً به شکل رشته بیان می‌شود.

دنباله کامل DNA **از یک مجموعه از کروموزوم ها.**

یک مجموعه از پارامترها که راه‌حل مسأله را تعریف می‌نماید.

- Genotype

فضای جواب‌های کد شده شامل تمامی رشته‌ها.

نوع و معرّف نماینده یک جنس (از موجودات دارای صفات مشابه ارثی) که با مطالعه DNA **ژنوتیپ به طور کامل مشخص می‌شود.**

ژنوتیپ= فنوتیپ کد شده.

- Heredity

انتقال کاراکترها وصفات از والدین به فرزندان با کپی ژن‌های مختلف حین آمیزش.

- Implicit-Parallelism

توازی مجازی.

مسائلی با ماهیت‌های متفاوت در شاخه‌های مختلف علوم که بعد از کد شدن بسیار به یکدیگر شبیه شده و ممکن است دارای جواب‌های یکسانی نیز باشند.

- Individual

شخص، فرد، منحصر به فرد.

- Locus

موقعیت مکانی یک ژن روی یک کروموزوم که می‌تواند به وسیله هر کدام از آلل‌های موجود اِشغال شود، مثلاً در رشته ۱۱۱۰۱ عدد ۰ در موقعیت مکانی چهارم از سمت چپ قرار دارد.

- Mutation

جهش.

یکی از عملگرهای الگوریتم‌های ژنتیک.

با استفاده از این عملگر تضمین می‌شود که هیچ جوابی از دید الگوریتم مخفی نمی‌ماند.

- Natural – Selection

مکانیزم اصلی تکامل و انتخاب در طبیعت که به وسیله «چارلز داروین» توضیح داده شده است.

- Offspring

زاد و ولد، فرزند، اولاد، مبدأ، منشأ.

- Phenotype

فضای اولیه جواب‌ها با همه محیط آن که نشان دهنده رفتار فیزیکی واقعی است.

با مطالعه رفتار یک ارگانیسم، فنوتیپ بطور کامل مشخص می‌شود.

فنوتیپ = ژنوتیپ دیکد شده.

- Premature Convergence

همگرایی زودرس، همگرایی سریع به جواب بدون تضمین.

بهینگی بدین معنی که به احتمال قوی به یک نقطه بهینه محلی رسیده‌ایم. فلسفه استفاده از عملگر جهش برای اجتناب از این حالت است.

- Recombination

زیست شناسی: پدیده لقاح میان تخمک و اسپرم.

شیمی: ترکیب گازهای هیدروژن و اکسیژن در یک باتری در طول عمل شارژ شدن آن که باعث تولید آب می‌شود.

- Selection

زیست شناسی: آلل‌های خاصی از یک نوع که برای اعمال بعدی مانند: آمیزش، جهش و برش انتخاب می‌شوند. معمول‌ترین انواع آن در طبیعت شامل حالت‌های زیر است:

- Sexual

- Ecological
- Stabilizing
- Disruptive
- Directional

- Reproduction

کپی برداری.

تولید یک چیز از خودش به نحویکه به چیز اولیه اضافه یا از آن کم نشده باشد، این عمل به دو شیوه جنسی(اشتراک مواد جنسی از دو والد مختلف) و غیرجنسی (تولید کپی از والد) صورت می‌پذیرد.

- Simulated Annealing (SA)

یک روش اکتشافی بر اساس احتمالات در مسائل بهینه‌سازی در فضای جستجویِ بسیار بزرگ که توسط Glatt, Kirkpatrick و Vecchi در سال ۱۹۸۳ مطرح شد. ایدهٔ اصلی آن از فرآیند در مهندسی مواد (متالوژی) حاصل شده است که به معنی گداختن و سرد کردن کنترل شدهٔ مواد، برای افزایش اندازه کریستال‌های تشکیل دهنده و کاهش شکنندگی آنان است.

در استفاده از SA، هر نقطه دلخواه از فضای جستجو با یک حالت فیزیکی سیستم، معادل گرفته می‌شود.

می‌خواهیم تابع E(s) که انرژی درونی سیستم را بیان می‌کند مینیمم بشود. هدف ما شروع از یک وضعیت کاملاً دلخواه اولیه و رسیدن به حالت انرژی درونی است.

- Solution

راه‌حل، یک جواب محتمل مسأله.

بهترین راه‌حل که دارای بالاترین برازش می‌باشد، جواب مسأله نامیده می‌شود.

- Species

نوع، گونه، نوعی از ارگانیسم، انواع.

این کلمه هم به معنای مفرد بکار می‌رود و هم نشانه جمع است.

در شیمی: مولکول، یون، اتم، رادیکال آزاد.

- Stochastic- process

فرآیند تصادفی، اتفاقی، غیرقطعی، Random.

گرفته شده از واژه یونانی (stochos).

فرآیندی که حالت بعدی آن، بطور کامل از روی محیط و حالت قبلی، قابل تعیین نیست. به عنوان مثال می‌توان از سری‌های زمانی، سیگنال‌های صوتی و تصویری، موارد بسیاری در علم پزشکی مانند نوار قلب، نوار مغز، فشار خون، حرارت بدن و... نام برد.

- Trajectory

خط سیر، مسیر گلوله، گذرگاه.

مسیر مناسب و با بیشترین قابلیت احتمال عبور.