دانشکده مهندسی کامپیوتر

گزارش پایانی درس: مبانی هوش محاسباتی

عنوان پژوهش: تمرین اول (الگوریتم ژنتیک)

ارائه دهنده:

نرگس سادات موسوی جد

شادی شاهی محمدی

استاد درس:

دکتر کارشناس

زمستان ۱۴۰۳

فهرست:

۱– بخش اول: مبانی و مفاهیم الگوریتم ژنتیک (GA) و ویژگی‌های آن

۱–۱–سوال اول:

۱–۱–۱–الگوریتم تکاملی:

الگوریتم تکاملی یکی از تکنیک‌های بهینه‌سازی می‌باشد که از فرایند تکامل طبیعی الهام گرفته است. این الگوریتم برای یافتن راه حل در مسائل بهینه‌سازی مانند پیدا کردن بیشینه یا کمینه‌ی یک تابع استفاده می‌شود.[۱]

الگوریتم تکاملی از روش‌ها و فرایندهای تکامل زیستی مانند تولید مثل، جهش، ترکیب و انتخاب الهام گرفته است.[۲] در این الگوریتم مجموعه‌ای از جواب‌ها که در فضای جست‌وجو قرار دارند، به طور تصادفی انتخاب می‌شوند که به آن‌ها، افراد[۱] یا کروموزوم‌ها[۲] گفته می‌شود. همچنین تابعی برای بررسی میزان تطابق با محیط (برازندگی) در نظر گرفته شده است که به آن تابع برازندگی[۳] می‌گویند. راه‌حل‌ها، با کمک تابع برازندگی، اصلاح و ارزیابی می‌شوند؛ به طوری که احتمال بقا و تولید مثل برای افراد با عملکرد بهتر، (بر اساس تابع ارزیابی) بیشتر است. در حالی که احتمال حذف شدن افراد با عملکرد ضعیف بالاتر خواهد بود. این فرآیند انتخاب و تولید مثل، می‌تواند در طول زمان منجر به تکامل افراد با تناسب بیشتر شود که برای حل مسئله بهینه‌سازی مورد نظر کارایی بالاتری دارند.[۳] برای انجام این فرایند مراحلی طی می‌شوند که به شرح زیر هستند:

مرحله اول: در این مرحله فضای اصلی مسئله که در آن تکامل اتفاق می‌افتد، تعریف می‌شود که به آن نمایش مسئله می‌گویند. هدف از این نمایش در واقع مدل‌سازی مسئله از دنیای واقعی به دنیای الگوریتم‌های تکاملی می‌باشد.

مرحله دوم: در این مرحله تابع برازندگی تعریف می‌شود. این تابع به عنوان مبنایی برای انتخاب افراد در هر نسل و تعیین کیفیت عملکرد آن‌ها می‌باشد. بهبود راه حل‌ها با کمک این تابع ممکن می‌شود.

---

[۱] Individuals

[۲] Chromosomes

[۳] Fitness function

مرحله سوم: پس از تعریف مسئله و تابع برازنگی، مراحل اصلی الگوریتم انجام می‌شوند. در این مرحله، جمعیت اولیه، شامل افرادی که معمولا به صورت تصادفی یا به کمک یک تابع اکتشافی انتخاب می‌شوند، تشکیل می‌شود.

مرحله چهارم: در این مرحله برازندگی افراد جمعیت محاسبه می‌شود و افراد برازنده‌تر (با احتمال بالاتری) به عنوان والد انتخاب می‌شوند. وقتی فردی به عنوان والد انتخاب شود، تغییراتی روی ویژگی‌های (ژن‌ها) آن انجام می‌شود و فرزندان جدید بوجود می‌آیند.

مرحله پنجم: در این مرحله، عملگرهایی روی والدهای انتخاب شده در مرحله‌ی قبل، اعمال می‌شوند و فرزندانی بوجود می‌آیند. دو دسته عملگر اصلی، شامل ترکیب[۴] و جهش[۵] وجود دارد. عملگر جهش، یک عملگر یگانی است. یعنی فقط روی یک والد اعمال می‌شود و فرزندی ایجاد می‌کند. در حالی که عملگر ترکیب دوتایی است و روی دو یا چند والد اعمال و یک یا چند فرزند ایجاد می‌کند.

مرحله ششم: این مرحله شامل انتخاب بازماندگان برای نسل بعدی می‌باشد که مشابه مرحله‌ی چهارم بر اساس برازندگی افراد انجام می‌شود.

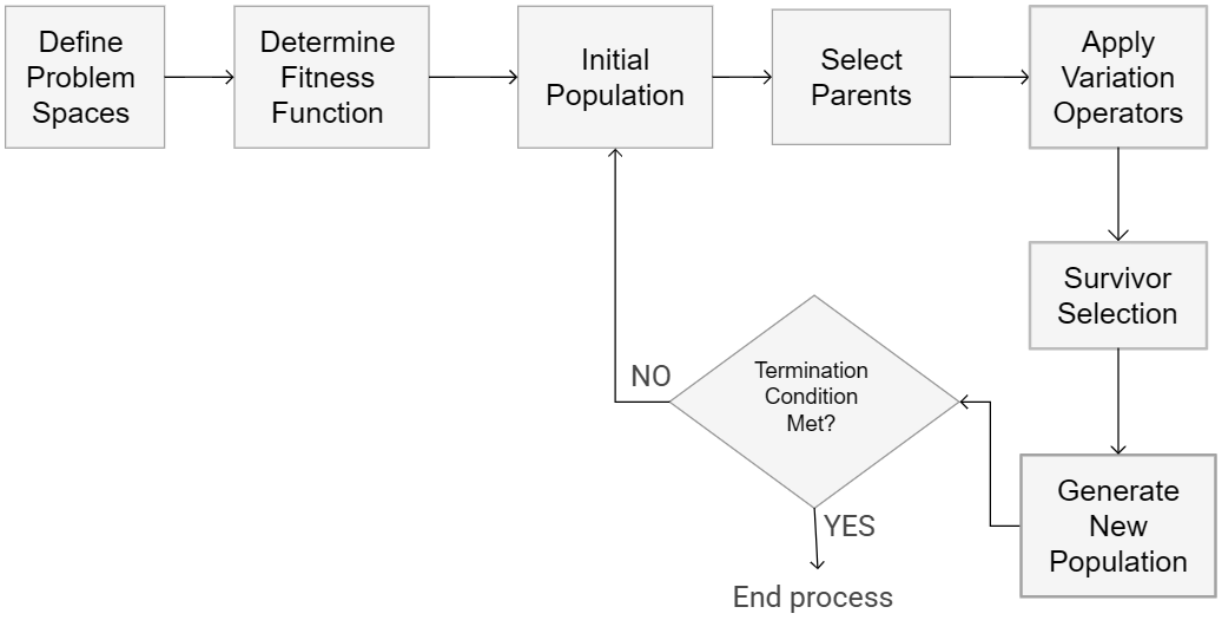
مرحله هفتم: از آنجایی که الگوریتم‌های تکاملی، ماهیت تصادفی دارند، لزوما رسیدن به جواب بهینه را تضمین نمی‌کنند. بنابراین ممکن است الگوریتم برای رسیدن به جواب بهینه تا زمان بی‌نهایتی اجرا شود. بنابراین نیاز به شرط‌هایی مانند حداکثر زمان CPU، تعداد ارزیابی‌ها و... داریم تا الگوریتم در زمان مناسبی متوقف شود. در این مرحله شرط‌های پایان بررسی شده و اگر هنوز محقق نشده‌اند الگوریتم از مرحله‌ی چهارم دوباره اجرا می‌شود.

---

[۴] crossover

[۵] mutation

فلوچارت مراحل الگوریتم تکاملی در شکل زیر نمایش داده شده است.



۱–۱–۲–دلیل برتری الگوریتم تکاملی از یادگیری تقویتی:

در مقاله‌ی پیشنهاد شده، هدف مقایسه‌ی عملکرد بین یکی از انواع الگوریتم‌های تکاملی به نام استراتژی تکاملی با روش‌های رایج یادگیری تقویتی مانند Q_learning در مسائلی مانند شبیه‌سازی حرکات ورزشی انسان در شبیه‌ساز MuJoCo و بازی آتاری است. اجرای یک نوع از ES به نام NES و موازی‌سازی آن باعث می‌شود که این الگوریتم بتواند یک شبیه‌سازی انسان‌گونه را در کمتر از ۱۰ دقیقه انجام دهد. همچنین این عامل، در بازی آتاری، بعد از یک ساعت تقریبا به همان اندازه خوب عمل می‌کند که الگوریتم A3C بعد از یک روز یادگیری، به آن سطح می‌رسد.[۴] دلایل این برتری به شرح زیر می‌باشد:

استفاده نکردن از پس‌انتشار[۶]: اکثر الگوریتم‌های یادگیری تقویتی مانند Deep Q-Learning، از شبکه‌های عصبی و در نتیجه، عمل پس‌انتشار برای یافتن سیاست بهینه استفاده می‌کنند؛ ولی الگوریتم‌های تکاملی، نیازی به پس‌انتشار ندارند. این موضوع باعث می‌شود که آن‌ها نسبت به RL دو تا سه برابر سریع‌تر باشند و چون نیازی به ذخیره‌ی اطلاعات اضافی نظیر مسیرها یا تجربیات، ندارند؛ در مصرف حافظه نیز بهینه‌تر عملی می‌کنند.
RLها در مسائلی که در آن عامل پس از طی یک توالی پیچیده، پاداشی دریافت می‌کند؛ به‌دلیل پس‌انتشار، به-

[۶] backpropagation

خوبی عمل نمی‌کنند. در حالی که چون EAها مستقیما از پاداش کل برای بهینه‌سازی پارامترهای سیاست استفاده می‌کنند، عملکرد قوی‌تری در این محیط‌ها دارند. همچنین در محیط‌هایی که کاوش دشوار است، الگوریتم‌های تکاملی بهتر عمل می‌کنند در حالی که RLها ممکن است در بهینه‌های محلی متوقف شوند یا در سیاست‌های اکتشافی مانند ε-حریصانه، به کاوش‌های قبلی متکی باشند.[۴], [۵]

قابلیت موازی‌سازی: الگوریتم‌های تکاملی (از جمله ES) قابلیت موازی‌سازی زیادی دارند. به‌طوری که برازندگی هر کروموزوم به صورت جداگانه توسط یک کارگر (نخ) محاسبه شده سپس این مقادیر برازندگی بین کارگران به‌اشتراک گذاشته می‌شود و در نهایت هر کارگر به‌صورت جداگانه عمل جهش را انجام می‌دهد. بنابراین اگر کارگران در ابتدای الگوریتم، seed مربوط به تصادفی‌سازی در تابع ()random را با هم هماهنگ کنند؛ اعداد تصادفی تولید شده از یک دیگر را می‌دانند و تنها جیزی که نیاز است به‌اشتراک بگذارند، مقادیر برازندگی می‌باشد. در حالی که الگوریتم‌های یادگیری تقویتی ذاتاً سری هستند؛ زیرا برای بروزرسانی سیاست برای حرکت بعدی، به نتایج انجام حرکت قبلی و تخمین ارزش آن حرکت نیاز است. بنابراین با داشتن مزیت موازی‌ساز، ES بسیار سریع‌تر می‌تواند به جواب برسد؛ چون امکان استفاده‌ی همزمان از چندین CPU برای محاسبات فراهم می‌شود.[۴], [۵]

استحکام بیشتر: الگوریتم‌های یادگیری تقویتی، ابرپارامترهای زیادی دارند که مقدار آن‌ها در موفقیت یا شکست عامل بسیار تعیین‌کننده است و از طرفی تعیین مقدار مناسب برای آن کار دشواری می‌باشد. مثلا در بازی آتاری، ابرپارامتر فریم-پرش[۷] بسیار مهم می‌باشد. این ابرپارامتر در واقع نشان می‌دهد که بین کنش عامل‌ها چند فریم رد می‌شود. اگر تنظیمات مختلف فریم-پرش برای RL می‌تواند باعث شود که مدل به خوبی یاد بگیرد یا حتی گاهی اوقات به‌طور کامل شکست بخورد. بنابراین RL، بسیار به این ابرپارامتر وابسته است؛ ES وابستگی چندانی به آن نداشته است. [۴], [۵]

اکتشاف ساختارمند: در الگوریتم‌های یادگیری تقویتی معمولا سیاست اولیه، به صورت تصادفی تعیین می‌شود. اثرات این فرض اولیه ممکن است تا مدت‌ها بر روند الگوریتم تاثیرگذار باشد. مثلا در یک بازی ماز ممکن است الگوریتم تا مدت‌ها کنش بی‌ارزشی به سمت دیوار انجام دهد. البته در برخی از RLها مانند Q_learning، که از سیاست‌های اکتشافی ε-حریصانه استفاده می‌کنند؛ این مشکل تقریبا حل شده است. در ε-حریصانه، عامل با احتمالی، یک حرکت کاملا تصادفی انجام می‌دهد و با یک احتمال دیگر، بر اساس تجربیاتش

[۷] frame_skip

حرکتی را انجام می‌دهد. با این سیاست می‌توان کنترل کرد که در شروع یادگیری، عامل حرکت‌های تصادفی بیشتری انجام دهد ولی با کسب تجربه، کم‌کم از دانش خود برای انجام کنش‌ها استفاده کند. از طرفی در EAها، اصلا سیاست تصادفی و مشکلات آن وجود ندارد. در این الگوریتم‌ها، هر سیاست، قطعی است و تعدادی پارامتر دارد که این پارامترها به صورت تصادفی و با جست‌وجو در فضای مسئله تعیین می‌شوند. در واقع در EA، پارامترها تصادفی هستند نه کنش‌ها. هنگامی که پارامترها انتخاب می‌شوند؛ سیاست قطعی مربوط به آن‌ها در محیط اجرا می‌شود و پاداش بدست آمده برای ارزیابی پارامترها، مورد استفاده قرار می‌گیرد و این روند به سمت سیاست‌هایی با پاداش بیشتر سوق پیدا می‌کند. [۴], [۵]

در نهایت می‌توان گفت که در بسیاری از مسائل، ES یک انتخاب جذاب است، مخصوصاً زمانی که تعداد گام‌های زمانی در یک اپیزود طولانی است، جایی که کنش‌ها دارای اثرات طولانی‌مدت هستند، یا اگر تخمین‌های تابع ارزش خوبی در دسترس نباشد.[۵]

۱–۲–سوال دوم:

۱–۲–۱–تعریف الگوریتم ژنتیک GA:

الگوریتم ژنتیک یکی از زیربخش‌های الگوریتم تکاملی می‌باشد. این الگوریتم بر پایه‌ی انتخاب طبیعی و ژن‌ها کار می‌کند. درواقع در این الگوریتم فرایند انتخاب طبیعی شبیه‌سازی شده است به این صورت که در هر نسل گونه‌هایی که بهتر می‌توانند با شرایط محیطی سازگار شوند، زنده می‌مانند و می‌توانند تولید مثل کنند و به نسل بعدی بروند. هر نسل از جمعیتی از افراد تشکلیل شده است و هر فرد نمایانگر یک نقطه در فضای جست-وجو است. هر فرد در واقع رشته‌ای از کاراکتر، عدد صحیح، عدد اعشاری یا بیت می‌باشد که به آن کرموزوم می-گویند.[6]

انتخاب والدین، ترکیب، جهش، تابع برازندگی از مهم‌ترین بخش‌های الگوریتم ژنتیک هستند. مراحل اجرای الگوریتم ژنتیک کلاسیک به اینصورت است که ابتدا یک جمعیت به نام Y متشکل از n کرموزوم به صورت تصادفی ایجاد می‌شود. سپس برازندگی هر کروموزم در Y محاسبه می‌شود. سپس چند کرموزوم از بین جمعیت به عنوان والدین، انتخاب می‌شوند؛ (و دو به دو با هم جفت می‌شوند.) در ادامه با احتمال C_p عمل ترکیب تک نقطه‌ای[۸] روی هر دو کرموزوم انجام می‌شود و فرزند جدیدی ایجاد می‌گردد. سپس عمل جهش

[۸] single-point crossover

یکنواخت[۹] روی این فرزند با احتمال Mp (به ازای هر ژن) انجام می‌شود و فرزند ایجاد شده جایگزین والد، در جمعیت می‌شود. عمل ترکیب و جهش روی بقیه والدین و فرزندان نیز انجام می‌شود تا جمعیت جدیدی با همان اندازه تکمیل و جایگزین قبلی شود. سپس دوباره فرایند انتخاب از سرگرفته می‌شود. شبه کد این الگوریتم به صورت زیر است: [۷]

Input:
Population Size, n
Maximum number of iterations, MAX
Output:
Global best solution, Y_{bt}

begin
Generate initial population of n chromosomes $Y_i \ (i = 1, 2, ..., n)$
Set iteration counter $t = 0$
Compute the fitness value of each chromosomes
while $(t < MAX)$
Select a pair of chromosomes from initial population based on fitness
Apply crossover operation on selected pair with crossover probability
Apply mutation on the offspring with mutation probability
Replace old population with newly generated population
Increment the current iteration t by 1.
end while
return the best solution, Y_{bt}
end

۱–۲–۲– الگوریتم استراتژی تکاملی ES(1+1):

در این الگوریتم هر نسل در واقع شامل یک فرد است. روش کار آن به این صورت است که ابتدا فرد x به صورت تصادفی، از فضای جست‌وجوی مسئله انتخاب می‌شود. سپس از x فقط یک فرزند، به نام y ایجاد می‌گردد. در واقع روی x، ترکیب انجام نمی‌شود (چون فقط یک فرد است.) و y فقط از طریق عمل جهش بوجود آمده است. عمل جهش در این الگوریتم به این صورت است که به هر ژن x یک مقدار تصادفی σ اضافه می‌شود. احتمال اضافه شدن یا نشدن این مقدار دارای توزیع نرمال استاندارد است (میانگین آن برابر ۱ و انحراف معیار آن صفر می‌باشد.). پس با احتمال زیاد مقدار یک ژن در x تغییر نمی‌کند چون عدد صفر در آن ضرب شده است.

[۹] uniform mutation

$$y_i = x_i + \sigma . N_i(0,1)$$

مقدار σ نیز از قانون یک پنجم جهش موفق پیروی می‌کند به این معنی که اگر تعداد جهش‌های موفق نسبت به تعداد کل جهش‌ها بیشتر از یک پنجم شود، مقدار σ افزایش می‌یابد و اگر این نسبت کمتر از یک پنجم شود، σ کاهش می‌یابد. در صورتی که این نسبت دقیقا برابر یک پنجم شود σ ثابت می‌ماند. در واقع در این قانون فرض بر این است که بهترین نتایج زمانی مشاهده می‌شوند که این نسبت برابر یک پنجم شود. در نهایت پس از ایجاد y، مقدار برازندگی x و y محاسبه می‌شود و هرکدام برازنده‌تر بود (مقدار برازندگی بیش‌تر) به عنوان نسل بعدی در نظر گرفته می‌شود.[۸]

۱-۲-۳-الگوریتم استراتژی تکاملی ES(μ+λ):

این الگوریتم تعمیم یافته‌ی ES(1+1) برای جمعیتی با بیش از یک عضو است. در این الگوریتم، هر فرد یک کرموزوم اضافی به نام σ هم دارد که در واقع میزان انحراف معیار هر ژن را مشخص می‌کند. شبه کد الگوریتم به صورت زیر است:[۸]

Algorithm 3 ES(μ+λ) Evolution Strategy

1: determine objective function (OF)
2: assign number of generation to 0 (t=0)
3: randomly create μ individuals in the initial population P(t)
4: evaluate individuals in population P(t) using OF
5: **while** termination criterion is not satisfied **do**
6: t=t+1
7: create the population T(t) by reproduction λ individuals from population P(t-1)
8: create the population M(t) by crossover and mutation of individuals from population T(t)
9: evaluate individuals in population M(t)
10: select μ the best individuals to population P(t) from the populations P(t-1) and M(t)
11: **end while**
12: return the best individual in population P(t)

۱-۲-۴-الگوریتم برنامه‌نویسی تکاملی EP:

این الگوریتم ابتدا برای یادگیری گرامر یک زبان ناشناخته، طراحی شد ولی بعد از آن به عنوان یک الگوریتم بهینه سازی عددی معروف شد. در برنامه‌نویسی تکاملی برخلاف استراتژی تکاملی و ژنتیک، عمل

انتخاب والدین انجام نمی‌شود بلکه برای ایجاد نسل بعدی عمل جهش روی کل نسل قبلی انجام شده و به تعداد جمعیت قبلی، فرزند ایجاد می‌شود. عمل جهش در این الگوریتم بر اساس تغییر تصادفی مقدار برخی ژن‌ها در فرد است. در ادامه بین نسل قبلی و نسل جدید، افراد برازنده تر، جمعیت بعدی را تشکیل می‌دهند. شبه کد الگوریتم به‌صورت زیر است:

Algorithm 4 Evolutionary Programming

```
1: determine objective function (OF)
2: assign number of generation to 0 (t=0)
3: randomly create individuals in initial population P(t)
4: evaluate individuals in population P(t) using OF
5: while termination criterion is not satisfied do
6:     t=t+1
7:     create population M(t) by the mutation of every individual from the population P(t-1)
8:     evaluate individuals in population M(t) using OF
9:     select the individuals to population P(t) from the sum of individuals in P(t-1) and M(t) using ranking selection
10: end while
11: return the best individual in population P(t)
```

۱–۲–۵–تفاوت الگوریتم ژنتیک با ES و EP:

- کروموزوم‌ها در الگوریتم ژنتیک کلاسیک معمولا به‌صورت رشته‌های بیتی یا با مقادیر گسسته نمایش داده می‌شوند در حالی که ES و EP به‌صورت اعداد حقیقی به نمایش در می‌آیند. بنابراین الگوریتم ژنتیک معمولا برای مسائل گسسته و ES و EP برای مسائل پیوسته مناسب-ترند.[۸]
- الگوریتم ژنتیک برای ایجاد فرزندان از دو عملگر ترکیب و جهش استفاده می‌کند؛ درحالی که ES و EP، معمولا فقط بر عملگر جهش تمرکز دارند. البته در ES$(\mu+\lambda)$ می‌توان از ترکیب هم استفاده کرد. نکته‌ی جالب توجه در ESها این است که میزان جهش در آن‌ها می‌تواند به‌صورت خودکار تعیین شود. درحالی که این امکان در ژنتیک و EP چندان مورد توجه نیست.
- در ژنتیک و ES، برای ایجاد فرزندان، از بین جمعیت والدینی انتخاب شده و سپس به کمک عملگرهای مختلف، فرزندان تولید می‌شوند؛ درحالی که در EP، والدینی انتخاب نمی‌شوند بلکه فرزندان جدید از کل جمعیت تولید می‌گردند.[۸]

- انتخاب افراد در الگوریتم ژنتیک و EP (هنگام انتخاب بازماندگان) معمولا به صورت تصادفی است؛ به‌طوری که افراد با برازندگی بیشتر، شانس بالاتری داده می‌شود ولی شانس افراد با برازندگی کمتر نیز صفر نیست. روش‌هایی مانند گردونه‌ی شانس و تورنمنت برای این کار بسیار رایج هستند. این در حالی است که در ES از روشی قطعی برای انتخاب باز ماندگان استفاده می‌شود. به عنوان مثال در $ES(\mu+\lambda)$ از بین فرزندان و والدین، μ تا از بهترین‌ها انتخاب شده و به عنوان جمعیت بعدی در نظر گرفته می‌شوند.

۱–۳–سوال سوم:

در الگوریتم‌های ژنتیک، عملگرهای ترکیب و جهش نقش اساسی در تولید نسل‌های جدید و جستجوی فضای پاسخ دارند. نحوه اعمال این عملگرها بسته به نوع نمایش کروموزوم‌ها متفاوت است. در ادامه، به تفصیل نحوه اعمال این عملگرها را بر روی رشته‌های بیتی[۱۰] و جایگشت‌ها[۱۱] بررسی می‌کنیم.

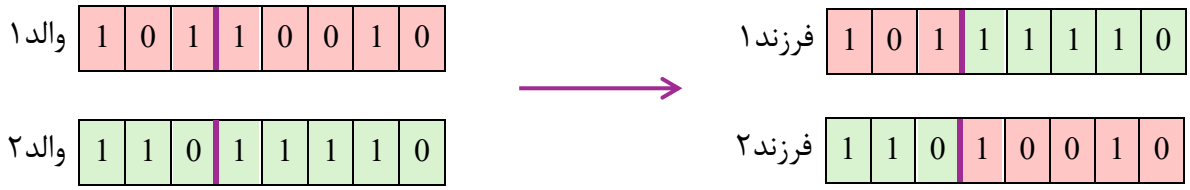
۱–۳–۱–اعمال عملگرها بر روی رشته‌های بیتی:

در نمایش بیتی، هر کروموزوم به صورت رشته‌ای از بیت‌ها (۰ و ۱) نمایش داده می‌شود

الف) ترکیب: انواع عملگرهای ترکیب برای این نمایش به شرح زیر است:

ترکیب تک نقطه‌ای: یک نقطه‌ی تصادفی در طول رشته انتخاب می‌شود و تمامی ژن‌هایی که در سمت راست تقطه قرار دارند، با یکدیگر مبادله خواهند شد و از این طریق، دو کروموزوم یا رشته جدید تولید می‌شود.[۹]

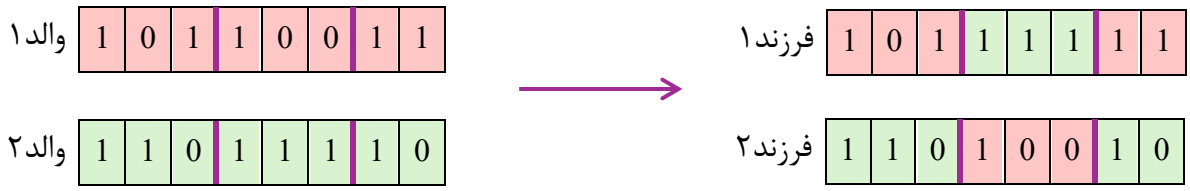
مثال:



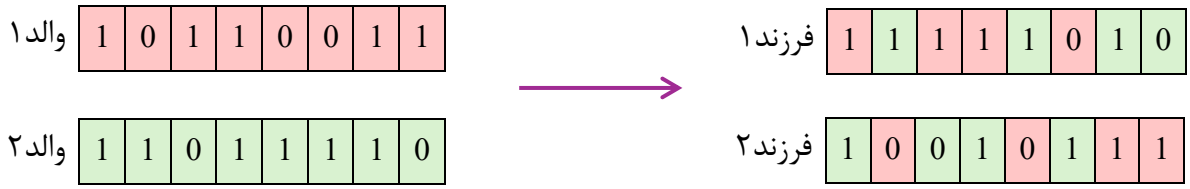
[۱۰] Bitstring

[۱۱] Permutations

ترکیب دو نقطه‌ای[۱۲]: حالت خاصی از ترکیب Nنقطه‌ای است. در این روش دو نقطه تصادفی انتخاب می‌شود و بخش میانی بین دو نقطه، بین والدین تبادل می‌شود.[۱۰] این عملگر برای زمانی که طول رشته‌ی کروموزوم طولانی می‌باشد، مناسب‌تر از ترکیب تک نقطه‌ای است.[۹] مثال:



ترکیب یکنواخت[۱۳]: هر ژن به صورت مستقل با احتمال مشخصی(معمولا یکنواخت) از یکی از والدین انتخاب می‌شود و ژن والد دیگر به فرزند دیگر داده می‌شود.[۱۰] مثال:



ب) جهش: در نمایش بیتی، عملگر جهش معمولاً با معکوس کردن مقدار یک یا چند ژن که به صورت تصادفی انتخاب شده‌اند، انجام می‌شود. به طوری که ژن‌های صفر، به یک و ژن‌های یک به صفر تبدیل می‌شوند.[۱۱] مثال:

۱-۳-۲-اعمال عملگرها بر روی جایگشت:

در مسائلی که نیاز به ترتیب‌دهی یا مسیریابی دارند (مانند مسئله فروشنده دوره‌گرد)، از نمایش جایگشتی استفاده می‌شود.

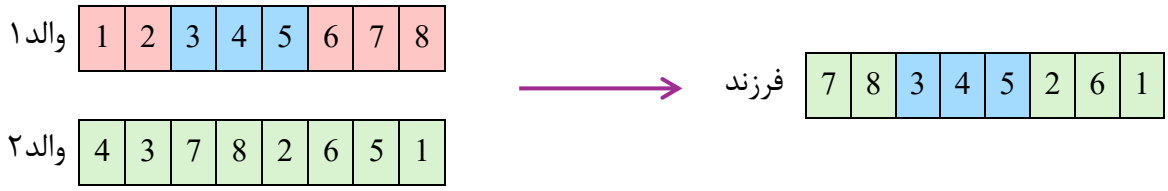
الف)ترکیب: در این نمایش، عملگرهای ترکیب باید به گونه‌ای طراحی شوند که جایگشت‌های معتبری تولید کنند.

---

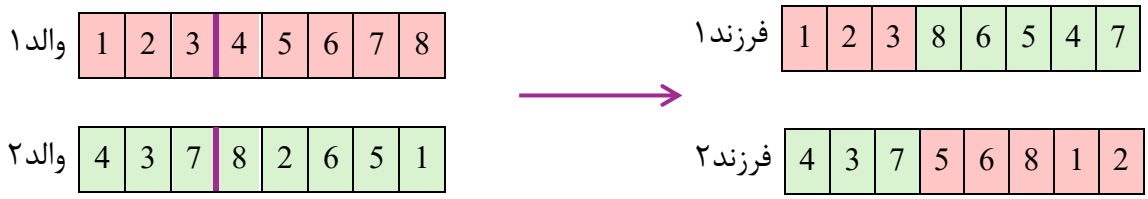
[۱۲] Two-Point Crossover

[۱۳] Uniform Crossover

ترکیب ترتیب‌محور[۱۴]: یک بخش تصادفی از والد اول انتخاب و در فرزند قرار می‌گیرد. سپس، به ترتیب عناصر والد دوم که در فرزند وجود ندارند، در مکان‌های خالی قرار می‌گیرند. مثال:



ترکیت قطع و تکمیل[۱۵]: مشابه ترکیب تک‌نقطه‌ای، یک نقطه‌ی تصادفی روی رشته انتخاب می‌شود سپس تکه‌ی دوم کروموزوم‌ها با هم جابه‌جا می‌شوند؛ البته به اینصورت که کروموزوم به عنوان یک لیست حلقوی پیمایش می‌شود و فقط ژن‌های غیر تکراری در بخش دوم نوشته می‌شوند. مثال:



الف)جهش: در نمایش جایگشتی، جهش باید به گونه‌ای باشد که جایگشت معتبر باقی بماند.[۱۱]

تعویض دوگانه[16] : دو موقعیت تصادفی انتخاب و مقادیر آن‌ها جابجا می‌شوند. مثال:

ب)معکوس‌سازی: یک بخش تصادفی انتخاب و ترتیب آن معکوس می‌شود. مثال:

۱-۴-سوال چهارم:

در الگوریتم ژنتیک، Properties Invariance به این معناست که برخی از ویژگی‌های کلیدی الگوریتم یا مسئله، تحت تغییرات خاصی (مانند جهش، تقاطع، یا تغییر در نمایش کروموزوم‌ها) بدون تغییر باقی بمانند. این

---

[۱۴] Order Crossover - OX

[۱۵] Cut and crossfill

[16] Swap Mutation

تغییرناپذیری‌ها به الگوریتم اجازه می‌دهند تا به‌طور سازگار و کارآمد عمل کند، حتی اگر تغییراتی در پارامترها یا ساختار مسئله ایجاد شود.

تغییر ناپذیری خواص (Properties Invariance)در الگوریتم ژنتیک به موارد زیر تقسیم می شود

۱- تغییرناپذیری انتقالی

در این تغییر ناپذیری اگر به همه مقادیر تابع برازندگی مقدار ثابتی اضافه شود بر روی عملکرد الگوریتم نباید تاثیری داشته باشد. برای مثال در بهینه سازی توابع ریاضی اگر به تابع برازندگی مقدار ثابتی اضافه کنیم ترتیب انتخاب نباید تغییر کند چون تفاوت نسبی برازندگی‌ها ثابت مانده است. اگر این خاصیت رعایت نشود ممکن است مقادیر مطلق برازندگی به اشتباه مقایسه شده و انتخاب‌ها در الگوریتم دچار خطا گردد.

مثلا مسئله حد آستانه نباید این خاصیت را داشته باشد و در مسائلی که مقدار مطلق تابع برازندگی مهم است نیز اضافه کردن مقدار ثابت احتمالا باعث می‌شود که برخی از بهترین جواب‌ها کنار بروند. اگر به اشتباه فرض کنیم این خاصیت لازم است، ممکن است برخی جواب‌های مناسب را نادیده بگیریم.

۲- تغییر ناپذیری مقیاس

در این تغییر ناپذیری اگر به همه مقادیر تابع برازندگی در ثابتی ضرب شود بر روی عملکرد الگوریتم نباید تاثیری داشته باشد برای مثال مسئله بشینه سازی تابع برازندگی باید این خاصیت را داشته باشد . اگر تابع برازندگی دو برابر یا نصف شود انتخاب نباید تغییر کند در برخی روش‌های انتخاب مقدار برازندگی قبل از محاسبه احتمال انتخاب بصورت نرمال شده مورد پردازش قرار می‌گیرند اگر در اینجا این خاصیت رعایت شود ممکن است تاثیر برخی مقادیر کم شود. اگر این تغییر ناپذیری به اشتباه لحاظ شود ممکن است برخی افراد نادرست از جمعیت انتخاب شوند و تکامل دچار مشکل شود.

۳-تغییر ناپذیری جایگشتی (Permutation Invariance)

در این تغییرناپذیری ترتیب ژن‌ها نباید روی عملکرد الگوریتم تاثیر داشته باشند در واقع جایگشت ژن‌ها باید منجر به کروموزوم‌های یکسان شود. به عموان مثال در مسئله‌ی کوله پشتی اگر همه‌ی اجناس، قیمت و وزن یکسانی داشته باشند، فقط تعداد اجناس اهمیت دارد بنابراین اگر هر ژن در رشته‌ی بیتی نمایانگر وجود یا عدم وجود یک جنس باشد؛ در اینجا جایگشت اهمیتی ندارد و فقط تعداد صفر و یک‌ها مهم می‌باشد. اگر در این حالت، الگوریتم ژنتیک، بین این جایگشت‌ها تفاوت قائل شود، باعث می‌شود که حالت‌های تکراری زیادی مورد

بررسی قرار گیرند و کارایی الگوریتم را کاهش می‌دهد. البته در مسئله‌ی کوله پشتی در حالت کلی جایگشت اهمیت دارد.

۴-تغییر ناپذیری نمایش (Representation Invariance)

در این تغییر ناپذیری طریقه نمایش کروموزوم نباید روی عملکرد الگوریتم تاثیر داشته باشد. مثلا برای رشته‌های بیتی اگر نمایش رشته به صورت باینری یا کد گری باشد، عملکر الگوریتم ژنتیک نسبت به آن باید یکسان باشد.

۵-تغییرناپذیری چرخش:

در این حالت اگر رشته‌ی بیتی یک چرخش انجام دهد، عملکرد الگوریتم تغییر نمی‌کند. مثلا عملکرد در قبال دو رشته‌ی ۱۰۱۰ و ۰۱۰۱ باید یکسان باشد. این تغییرناپذیری برای الگوریتم‌های مربوط به تقارن دایره‌ای مورد نیاز است.

۶-تغییرناپذیری طول کروموزوم:

در بسیاری از مسائل، طول کروموزوم در طول الگوریتم باید ثابت باشد. مثلا در مسئله‌ی فروشنده‌ی دورگرد اگر فضای جست‌وجو همه‌ی دورهای همیلتونی ممکن باشد، این طول باید ثابت باشد ولی اگر فضای جست‌وجو همه‌ی دورهای ممکن بود، طول می‌تواند متفاوت باشد. در رشته‌های بیتی نیز معمولا در مسائلی که صفر یا یک نشاندهنده‌ی وجود یا عدم وجود است این طول ثابت است ولی در مسائلی که رشته بیتی در واقع یک مقدار باینری است ممکن است متفاوت باشد. به طور کلی ثابت بودن طول رشته باعث سادگی بیشتر در پیاده‌سازی الگوریتم می‌شود.

۱–۵– سوال پنجم:

۱–۵–۱–جستجوی تصادفی یکنواخت[۱۷]:

در جستجوی تصادفی یکنواخت،رشته‌های بیتی به طور تصادفی از فضای جستجو را انتخاب می‌شوند و این روند تا رسیدن به جواب بهینه ادامه می‌یابد. بنابراین اگر طول رشته‌های بیتی برابر با n باشد، کل فضای جستجو شامل 2^n رشته‌ی ممکن است. پس احتمال یافتن جواب بهینه در یک انتخاب تصادفی برابر است با:

[۱۷] Uniform Random Search

$$\frac{1}{2^n}$$

بنابراین، تعداد مراحل مورد انتظار برای پیدا کردن جواب بهینه در جستجوی تصادفی یکنواخت از مرتبه‌ی $\mathbf{O(2^n)}$ برابر است. از آنجایی که $\mathbf{O(n^3)}$ مرتبه‌ی چندجمله‌ای است و $\mathbf{O(2^n)}$ یک مرتبه‌ی نمایی، الگوریتم ژنتیک بسیار کارآمدتر از جستجوی تصادفی می‌باشد.

۱-۵-۲-عوامل موثر بر عملکرد الگوریتم:

نمایش کروموزوم (رمزگذاری): نحوه نمایش راه‌حل‌ها در کروموزوم‌ها تأثیر مستقیمی بر کارایی GA دارد. انتخاب روش مناسب رمزگذاری می‌تواند به بهبود سرعت همگرایی و کیفیت نتایج کمک کند.

تابع برازندگی: کیفیت و دقت تابع برازندگی که برای ارزیابی راه‌حل‌ها استفاده می‌شود، تأثیر زیادی بر انتخاب والدین و در نتیجه بر کیفیت نسل‌های بعدی دارد.

عملیات انتخاب، ترکیب و جهش: پارامترهای مربوط به انتخاب (مانند انتخاب والدین)، ترکیب (مانند نوع و نرخ کراس‌اور) و جهش (مانند نرخ جهش) می‌توانند بر تنوع ژنتیکی جمعیت و سرعت همگرایی تأثیرگذار باشند.

اندازه جمعیت: اندازه جمعیت اولیه و همچنین اندازه جمعیت در طول نسل‌ها می‌تواند بر توانایی الگوریتم در جستجوی فضای حل و یافتن راه‌حل‌های بهینه تأثیر بگذارد.

تعداد نسل‌ها: تعیین تعداد مناسب نسل‌ها برای اجرای الگوریتم می‌تواند بر کیفیت و زمان اجرای الگوریتم تأثیرگذار باشد.

استراتژی‌های متنوع‌سازی: استفاده از تکنیک‌هایی برای حفظ تنوع ژنتیکی در جمعیت می‌تواند از همگرایی زودهنگام جلوگیری کرده و به کشف راه‌حل‌های بهینه‌تر کمک کند.

پارامترهای الگوریتمی: تنظیم پارامترهایی مانند نرخ تقاطع، نرخ جهش و اندازه جمعیت می‌تواند تأثیر زیادی بر عملکرد GA داشته باشد.

۲–بخش دوم: درک و حل مسائل با الگوریتم ژنتیک

۲–۱–سوال اول:

الف) در الگوریتم ژنتیک به‌کار رفته، هر کروموزوم یک دور همیلتونی در این گراف در نظر گرفته می‌شود و هدف، کمینه‌سازی طول این دورها است. بنابراین اگرn شهر داشته باشیم، هر کروموزوم دقیقاً شامل n ژن است، زیرا یک دور همیلتونی، باید از تمام راس‌ها یک بار عبور کرده و به راس مبدا برگردد؛ بنابراین تعداد یال‌هایی که باید از آن عبور کند برابر n می‌شود. در نتیجه اگر ۱۰ شهر داشته باشیم هر کروموزوم ۱۰ ژن دارد.

ب) الفبای الگوریتم به مجموعه‌ی مقادیر یکتای ممکن برای هر ژن اشاره دارد. در این مثال هر ژن به یک یال اشاره می‌کند. بنابراین الفبای الگوریتم، برابر مجموعه‌ی یال‌های گراف می‌شود و تعداد مقادیر ممکن یک ژن، همان تعداد یال‌ها می‌باشد. اگر گراف را کامل فرض کنیم، بین هر دو راسی یک یال وجود دارد بنابراین الفبای الگوریتم برابر است با:

$$\frac{n\times(n-1)}{2} = \frac{10\times9}{2} = 45$$

۲–۲–سوال دوم:

۲–۲–۱–محاسبه برازندگی:

با توجه به تابع برازندگی، برازندگی کروموزوم‌ها به صورت زیر محاسبه می‌شود:

$$f(x) = (a+b) - (c+d) + (e+f) - (g+h)$$

$$f(x_1) = (6+5) - (4+1) + (3+5) - (3+2) = 11 - 5 + 8 - 5 = 9$$

$$f(x_2) = (8+7) - (1+2) + (6+6) - (0+1) = 15 - 3 + 12 - 1 = 23$$

$$f(x_3) = (2+3) - (9+2) + (1+2) - (8+5) = 5 - 11 + 3 - 13 = -16$$

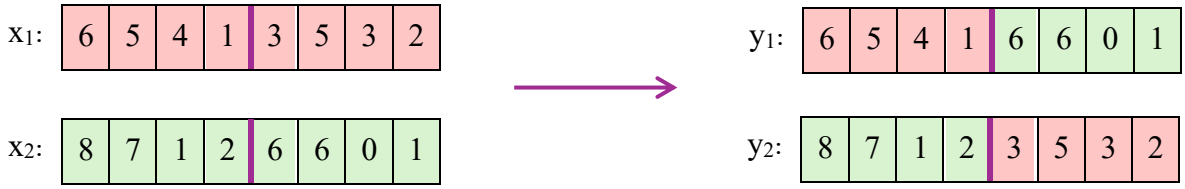
$$f(x_4) = (4+1) - (8+5) + (2+0) - (9+4) = 5 - 13 + 2 - 13 = -19$$

کروموزوم‌ها به ترتیب برازندگی به صورت زیر مرتب می‌شوند:

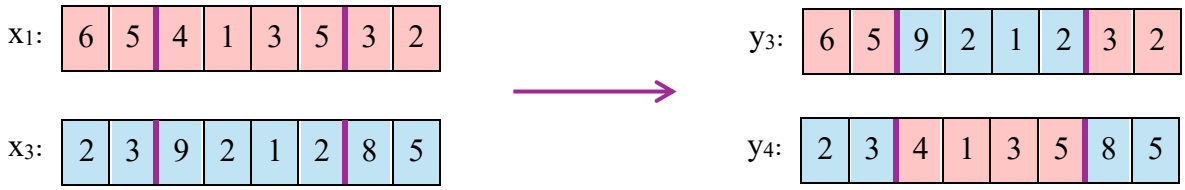
$$x_4 < x_3 < x_1 < x_2$$

۲-۲-۲-ترکیب روی کروموزوم ها:

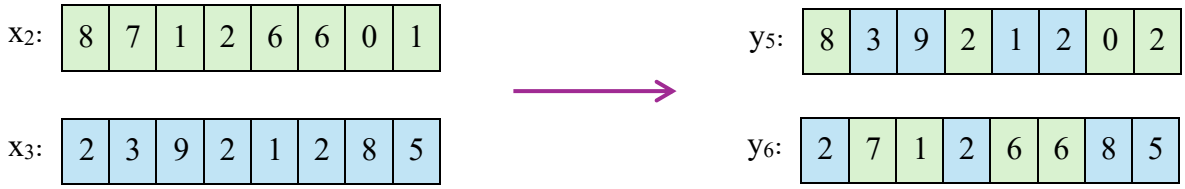
ترکیب تک نقطه‌ای روی کروموزوم x_2 و x_1:



ترکیب دو نقطه‌ای روی کروموزوم x_3 و x_1:



ترکیب یکنواخت روی کروموزوم x_3 و x_2:



۲-۲-۳- برازندگی فرزندان:

مقدار برازش شش فرزند جدید به صورت زیر محاسبه می‌شود:

$$f(y) = (a+b) - (c+d) + (e+f) - (g+h)$$

$$f(y_1) = (6+5) - (4+1) + (6+6) - (0+1) = 11 - 5 + 12 - 1 = 17$$

$$f(y_2) = (8+7) - (1+2) + (3+5) - (3+2) = 15 - 3 + 8 - 5 = 15$$

$$f(y_3) = (6+5) - (9+2) + (1+2) - (3+2) = 11 - 11 + 3 - 5 = -2$$

$$f(y_4) = (2+3) - (4+1) + (3+5) - (8+5) = 5 - 5 + 8 - 13 = -5$$

$$f(y_5) = (8+3) - (9+2) + (1+2) - (0+2) = 11 - 11 + 3 - 2 = 1$$

$$f(y_6) = (2+7) - (1+2) + (6+6) - (8+5) = 9 - 3 + 12 - 13 = 5$$

در اینجا به طور کلی مقدار برازش بهبود یافته است. زیرا مقادیر کمینه قبلا ۱۹- و ۱۶- بود ولی اکنون کمترین مقدار ۵- می‌باشد که یعنی از کمترین مقدار جمعیت قبلی ۱۴ واحد بیشتر است. البته برازندگی بهترین فرد در این جمعیت نسبت به به قبلی ۶ واحد افت کرده ولی در مقایسه با این افزایش ناچیز است.

۲-۲-۴-مقدار بهینه:

چون ژن‌های a, b, e, f تاثیر مثبتی در مقدار برازندگی دارند، مقدار آن‌ها باید بیشینه حالت ممکن یعنی ۹ باشد و چون ژن‌های c, d, g, h تاثیر منفی در برازندگی دارند، مقدار آن‌ها باید کمترین حالت ممکن یعنی صفر باشد بنابراین جواب بهینه مقدار ۹۹۰۰۹۹۰۰ خواهد بود که مقدار برازندگی آن به صورت زیر محاسبه می‌شود:

$$f(x) = (9 + 9) - (0 + 0) + (9 + 9) - (0 + 0) = 18 - 0 + 18 - 0 = 36$$

۲-۲-۵-بررسی نیاز به جهش برای رسیدن به مقدار بهینه:

با توجه ب جمعیت اولیه این الگوریتم حتما نیاز به عملگر جهش دارد زیرا مقادیر 9 و 0 در بسیاری از ژن‌ها دیده نمی‌شوند. مثلا در ژن a، هیچ یک از والدین، مقدار 9 ندارند، بنابراین فقط با ترکیب والدین نمی‌توان ژن a=9 را وارد جمعیت کرد.

۲-۳-سوال سوم:

۲-۳-۱-محاسبه برازندگی:

مقادیر برازندگی برای xها به صورت زیر محاسبه می‌شود:

$$F(x) = F_{raw}(x) = x^3 - 4x^2 + 7$$
$$F(1) = 1^3 - 4(1^2) + 7 = 4$$
$$F(2) = 2^3 - 4(2^2) + 7 = -1$$
$$F(3) = 3^3 - 4(3^2) + 7 = -2$$
$$F(4) = 4^3 - 4(4^2) + 7 = 7$$

۲-۳-۲-نامنفی کردن مقادیر برازندگی:

چون به ازای x=3 و x=2 مقدار برازندگی منفی می‌شود، مقدار ثابت ۲ به همه‌ی مقادیر برازندگی اضافه شده است تا همه‌ی آن‌ها نامنفی شوند. مقادیر جدید برازندگی:

$F(1) = 6$

$F(2) = 1$

$F(3) = 0$

$F(4) = 9$

۲-۳-۳-برازندگی کل جمعیت:

با توجه به تعداد نمونه‌ها از هر x، مقدار برازندگی کل جمعیت به صورت زیر محاسبه می‌شود:

$$\sum_{i=1}^{n}(n_i \times F_{raw}(x_i)) = F_{total} = (2 \times 6) + (3 \times 1) + (3 \times 0) + (2 \times 9) = 33$$

۲-۳-۴-احتمال انتخاب هر فرد در روش گردونه‌ی شانس:

از آن جایی که از هر مقدار x چند فرد وجود دارد و این افراد، یکسان هستند، بنابراین احتمال انتخاب یک فرد از یک مقدار x با این رابطه محاسبه می‌شود:

$$p(x_i) = \frac{n_i \times F_{raw}(x_i)}{F_{total}}$$

بنابراین احتمالات انتخاب هر فرد به صورت زیر محاسبه می‌گردد:

$$p(x = 1) = \frac{2 \times 6}{33} = \frac{12}{33} \sim 0.36$$

$$p(x = 2) = \frac{3 \times 1}{33} = \frac{3}{33} \sim 0.1$$

$$p(x = 3) = \frac{3 \times 0}{33} = \frac{0}{33} = 0$$

$$p(x = 4) = \frac{2 \times 9}{33} = \frac{18}{33} \sim 0.54$$

۲-۳-۵-احتمال انتخاب هر فرد با تابع برازندگی جدید:

تابع برازندگی جدید، در واقع به توان ۲ رسیده‌ی تابع قبلی می‌باشد. مزیت این تابع نسبت به قبلی این است که در اینجا مقدار برازندگی به ازای همه‌ی xها نامنفی است. مزیت دیگر آن نسبت به حالتی که با جمع

یک مقدار ثابت مقادیر را نامنفی کنیم، این است که احتمال انتخاب هیچ فردی صفر نمی‌شود. مقدار برازندگی جدید هر فرد برابر است با:

$$g(1) = 16$$

$$g(2) = 1$$

$$g(3) = 4$$

$$g(4) = 49$$

مقدار کل برازندگی نیز برابر است با:

$$\sum_{i=1}^{n}(n_i \times F_{raw}(x_i)) = F_{total} = (2 \times 16) + (3 \times 1) + (3 \times 4) + (2 \times 49) = 145$$

در نهایت احتمال انتخاب هر فرد به این صورت محاسبه شده است:

$$p(x = 1) = \frac{2 \times 16}{145} = \frac{32}{145} \sim 0.22$$

$$p(x = 2) = \frac{3 \times 1}{145} = \frac{3}{145} \sim 0.02$$

$$p(x = 3) = \frac{3 \times 4}{145} = \frac{12}{145} \sim 0.08$$

$$p(x = 4) = \frac{2 \times 49}{145} = \frac{98}{145} \sim 0.68$$

۲-۳-۶-تاثیر تابع برازندگی جدید بر فشار انتخاب[۱۸] و همگرایی و تنوع جمعیت:

وقتی مقدار برازندگی به توان دو می‌شود، تغییرات قابل توجهی در نحوه انتخاب کروموزوم‌ها در الگوریتم ژنتیک ایجاد خواهد شد. این تغییرات می‌توانند فشار انتخاب را افزایش دهند و تاثیر زیادی بر همگرایی و تنوع جمعیت داشته باشند.

فشار انتخاب به معنای تأثیر قدرت و میزان انتخاب افراد با برازندگی بالا است. در واقع هرچقدر احتمال انتخاب افرادی با برازندگی پایین، کمتر باشد؛ فشار انتخاب بر آن‌ها بیشتر است. همان طور که مشاهده می‌شود، زمانی که برازندگی به توان دو می‌رسد، فشار انتخاب افزایش می‌یابد زیرا مقدار برازندگی کروموزوم‌های برازنده‌تر

[۱۸] Selection Pressure

افزایش می‌یابد و فاصله‌ی آن با کروموزوم‌هایی با برازندگی پایین، زیاد می‌شود بنابراین شانس انتخاب کروموزوم‌های برازنده‌تر به شدت افزایش می‌یابد.

در اینجا قبل از به توان رسیدن احتمال انتخاب x=4، 0.54 بوده ولی بعد از آن به 0.68 افزایش می‌یابد. در حالی که احتمال انتخاب x=2 و x=1 نسبت به قبل کمتر شده است. البته در این مثال بدلیل اینکه احتمال انتخاب x=3، قبل از به توان دو رسیدن، صفر بوده، می‌توان افزایشی در احتمال انتخاب آن مشاهده کرد ولی دلیل این امر این است که پس از به توان رساندن دیگر نیازی به نامنفی کردن برازندگی‌ها نیست بنابراین برازندگی x=3، صفر نمی‌شود. بنابراین در حالت کلی می‌توان گفت فشار انتخاب افزایش یافته است.

همگرایی به این معنی است که جمعیت به یک راه‌حل واحد یا محدوده‌ای از راه‌حل‌ها که بهینه است، نزدیک می‌شود. افزایش فشار انتخاب باعث می‌شود کروموزوم‌های با برازندگی بالا به طور پیوسته انتخاب شوند. بنابراین جمعیت، خیلی سریع به سمت پاسخ‌های بهینه حرکت می‌کند. این موجب می‌شود که الگوریتم سریع‌تر همگرا شود. ولی ممکن است در نهایت به همگرایی زودهنگام منجر شود که در آن تنوع جغرافیایی (تنوع ژنتیکی) کاهش می‌یابد و جمعیت به یک راه‌حل محلی می‌رسد و در آن متوقف می‌شود.

تنوع جمعیت به میزان اختلاف بین کروموزوم‌های مختلف در جمعیت گفته می‌شود. وقتی برازندگی به توان دو می‌رسد، افراد با برازندگی بالا به طور مکرر انتخاب می‌شوند و تنوع جمعیت به شدت کاهش می‌یابد. این کار باعث می‌شود که جمعیت به سمت کروموزوم‌های مشابه حرکت کند و کروموزوم‌های با برازندگی پایین شانس کمی برای انتخاب شدن داشته باشند. در نتیجه، بعد از مدتی جمعیت می‌تواند به یک گروه از کروموزوم‌های مشابه برسد که در صورت وجود یک راه‌حل بهینه محلی، ممکن است از یافتن بهینه جهانی باز بماند.

بنابراین تاثیرات به توان دو رساندن تابع برازندگی به طور خلاصه به شرح زیر است:

افزایش فشار انتخاب: با به توان دو رساندن مقدار برازندگی، کروموزوم‌های با برازندگی بالا نسبت به دیگر کروموزوم‌ها انتخاب بیشتری خواهند داشت. این موضوع فشار انتخاب را افزایش داده و باعث می‌شود که الگوریتم خیلی سریع‌تر به سمت یک یا چند راه‌حل بهینه حرکت کند.

همگرایی زودهنگام: اگر برازندگی به توان دو برسد، احتمال همگرایی زودهنگام به راه‌حل بهینه محلی وجود دارد. به این معنی که جمعیت ممکن است به سرعت به یک جواب ثابت برسد، حتی اگر آن جواب بهینه جهانی نباشد.

کاهش تنوع جمعیت: افزایش فشار انتخاب باعث کاهش تنوع در جمعیت می‌شود. این باعث می‌شود که دیگر کروموزوم‌ها شانس کمی برای انتخاب شدن داشته باشند و تنوع ژنتیکی کاهش یابد.

۳–بخش سوم: پیاده‌سازی الگوریتم ژنتیک

۳–۱–پیش پردازش داده‌ها:

ابتدا داده‌های آموزش وارد برنامه شده، ستون ID از آن حذف شده و عملیات مدیریت داده‌های از دست رفته، حذف داده‌های پرت و رمزگذاری ویژگی‌های دسته‌ای روی آن انجام شده است. در ادامه نیز داده‌های آزمایش وارد برنامه شده‌اند و فقط عملیات کدگذاری بر روی آن انجام شده است.

```
def import_dataset():
    # read train data
    train_dataset = pd.read_csv("Train.csv")
    train_dataset.drop('ID', inplace=True, axis=1)
    # control missing data
    train_dataset.dropna(
        subset=['Gender', 'Age', 'Profession', 'Family_Size', 'Var_1',
'Spending_Score', 'Graduated', 'Ever_Married'],
        inplace=True)
    train_dataset['Work_Experience'] = train_dataset.apply(lambda row:
missing_value_control(row, train_dataset),
                                                           axis=1)
    # encoding data
    encoding_categories(train_dataset)
    # remove outliers
    remove_Outliers(train_dataset)

    test_dataset = pd.read_csv("Test.csv")
    test_dataset.drop('ID', inplace=True, axis=1)
    encoding_categories(test_dataset)
    return train_dataset, test_dataset
```

۳–۱–۱–مدیریت داده‌های از دست رفته:

سیاست کلی برای مدیریت داده‌های از دست رفته در اکثر ستون‌های مجموعه داده‌ی آموزش، حذف سطری است که حداقل یک ویژگی آن مقداردهی نشده است. با این وجود اگر امکان تخمین آن داده‌ی گمشده وجود داشته باشد؛ بسیار بهتر است زیرا هر سطر حاوی اطلاعات مفیدی می‌باشد و با حذف آن این اطلاعات از دست می‌رود. بنابراین برای ستون سابقه‌ی کاری، میانگین سابقه‌ی کاری افرادی که در یک محدوده‌ی سنی قرار دارند به عنوان مقدار تخمینی در نظرگرفته می‌شود.

```
def missing_value_control(row, dataset, age_range=6):
    if np.isnan(row['Work_Experience']):
        lower_bound = row['Age'] - (age_range // 2)
        upper_bound = row['Age'] + (age_range // 2)

        similar_ages = dataset[(dataset['Age'] >= lower_bound) &
(dataset['Age'] <= upper_bound) & dataset[
            'Work_Experience'].notna()]

        if not similar_ages.empty:
            return similar_ages['Work_Experience'].mean()
        else:
            return dataset['Work_Experience'].mean()
    return row['Work_Experience']
```

۳-۱-۲-حذف داده پرت:

داده‌هایی که فاصله‌ی آن‌ها تا چارک اول و سوم، بیش از ۱.۵ برابر فاصله‌ی بین این دو چارک می‌باشند، به عنوان داده‌ی پرت شناخته شده و حذف می‌شوند. به این روش IQR گفته می‌شود.[۱۲]

```
def remove_Outliers(dataset):
    Q1 = dataset.quantile(0.25)
    Q3 = dataset.quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    dataset[(dataset >= lower_bound) & (dataset <=
upper_bound)].dropna(inplace=True)
```

۳-۱-۳-کدگذاری داده‌ها:

داده‌های غیرعددی به روش Label Encoding کدگذاری می‌شوند.

```
def encoding_categories(dataset):
    label_encoder = preprocessing.LabelEncoder()
    dataset['Gender'] = label_encoder.fit_transform(dataset['Gender'])
    dataset['Ever_Married'] =
label_encoder.fit_transform(dataset['Ever_Married'])
    dataset['Graduated'] = label_encoder.fit_transform(dataset['Graduated'])
    dataset['Profession'] =
label_encoder.fit_transform(dataset['Profession'])
    dataset['Spending_Score'] =
label_encoder.fit_transform(dataset['Spending_Score'])
    dataset['Var_1'] = label_encoder.fit_transform(dataset['Var_1'])
    dataset['Segmentation'] =
label_encoder.fit_transform(dataset['Segmentation'])
```

۳-۲-پیاده‌سازی الگوریتم ژنتیک:

۳-۲-۱-تعریف کروموزوم:

در این الگوریتم کروموزوم‌ها رشته‌های بیتی به طول ویژگی‌های ورودی فروشگاه هستند. یک یا صفر بودن یک ژن نشان‌دهنده‌ی در نظر گرفتن یا نگرفتن ویژگی مربوطه در درخت تصمیم است. همچنین تعداد ویژگی‌هایی که در مدل استفاده می‌شوند از قبل مشخص است؛ بنابراین تعداد یک ها در کروموزوم‌ها باید ثابت باشد.

۳-۲-۲-پارامترهای الگوریتم:

train_set: مجموعه داده آموزشی (که به عنوان ورودی به الگوریتم داده می‌شود).

num_of_features: تعداد ویژگی‌هایی که باید انتخاب شوند.

population_size: تعداد کروموزوم‌ها در جمعیت.

genome_length: طول کروموزوم (که در اینجا همیشه ۹ است، زیرا مجموعه داده‌ای که استفاده می‌شود ۹ ویژگی دارد).

mutation_rate: نرخ جهش که تعیین می‌کند هر ژن در کروموزوم با چه احتمالی تغییر می‌کند.

crossover_rate: نرخ ترکیب که تعیین می‌کند چه زمانی والدها برای تولید نسل جدید با یکدیگر ترکیب می‌شوند.

generations: تعداد نسلی که الگوریتم اجرا می‌شود.

patience: پارامتری که برای جلوگیری از اجرای بی‌پایان الگوریتم استفاده می‌شود، یعنی اگر دقت به مدت خاصی تغییر نکرد، الگوریتم متوقف می‌شود.

۳-۲-۳-متد random_genome(self)

در این متد یک کروموزوم تصادفی ساخته می‌شود. کروموزوم‌ها به صورت آرایه‌ای از ۰ و ۱ هستند که نشان‌دهنده انتخاب یا عدم انتخاب یک ویژگی هستند. به طور خاص، ۱ نشان‌دهنده انتخاب ویژگی و ۰

نشان‌دهنده عدم انتخاب آن است. در اینجا، ویژگی‌ها به تعداد num_of_features به صورت تصادفی به ۱ تغییر می‌کنند و بقیه‌ی ویژگی‌ها ۰ باقی می‌مانند.

```
def random_genome(self):
    chromosome = np.zeros(9)
    features = sample(range(0, 9), self.num_of_features)
    for feature in features:
        chromosome[feature] = 1
    return chromosome
```

۳-۲-۴-متد init_population(self)

در این متد، جمعیت اولیه کروموزوم‌ها (هر کروموزوم یک آرایه از ۰ و ۱ است) ایجاد می‌شود. تعداد کروموزوم‌ها با استفاده از پارامتر population_size مشخص می‌شود.

```
def init_population(self):
    return [self.random_genome() for _ in range(self.population_size)]
```

۳-۲-۵- متد prepare_data_for_fitness(self, chromosome)

این متد برای آماده‌سازی داده‌ها به منظور ارزیابی برازندگی کروموزوم استفاده می‌شود. به این صورت که ابتدا ویژگی‌های انتخاب‌شده از کروموزوم استخراج می‌شود و سپس داده‌های ورودی (x) و خروجی (y) برای مدل آماده می‌شوند. این داده‌ها برای ارزیابی عملکرد مدل پیش‌بینی (درخت تصمیم) استفاده می‌شوند.

```
def prepare_data_for_fitness(self, chromosome):

    selected_features = [i for i, gene in enumerate(chromosome) if gene == 1]
    if not selected_features:
        return 0

    x = self.train_set.values[:, selected_features]
    y = self.train_set.values[:, 9]
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=100)
    return x_train, x_test, y_train, y_test
```

۳-۲-۶-متد fitness(self, chromosome)

این متد، برازندگی یا تناسب هر کروموزوم را محاسبه می‌کند. برای انجام این کار ابتدا ویژگی‌های انتخاب‌شده از کروموزوم استخراج می‌شود. مدل درخت تصمیم (DecisionTreeClassifier) برای پیش‌بینی بر اساس داده‌های آموزشی ساخته و آموزش داده می‌شود. مدل بر اساس داده‌های آزمایشی (x_test و y_test)

ارزیابی شده و دقت آن محاسبه می‌شود. دقت مدل معیاری است که در الگوریتم ژنتیک برای ارزیابی کیفیت کروموزوم‌ها استفاده می‌شود.

```
def fitness(self, chromosome):
    x_train, x_test, y_train, y_test =
self.prepare_data_for_fitness(chromosome)
    model = DecisionTreeClassifier(random_state=42, max_depth=10)
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    return accuracy_score(y_test, y_pred)
```

۳-۲-۷-متدهای انتخاب (Selection Methods)

این متدها مسئول انتخاب والدین برای تولید نسل بعدی هستند. سه روش مختلف انتخاب در اینجا پیاده‌سازی شده است:

tournament_selection(self, fitness_scores, k=3): در این روش، گروهی از کروموزوم‌ها به صورت تصادفی انتخاب می‌شوند و بهترین کروموزوم از بین آن‌ها انتخاب می‌شود.

```
def tournament_selection(self, fitness_scores, k=3):
    selected = random.sample(list(enumerate(fitness_scores)), k)
    winner = max(selected, key=lambda x: x[1])
    return self.population[winner[0]]
```

roulette_wheel_selection(self, fitness_scores): در این روش، انتخاب بر اساس احتمال است. کروموزوم‌هایی که دقت بیشتری دارند، احتمال انتخاب بیشتری خواهند داشت. این انتخاب به نوعی شبیه به چرخ رولت است.

```
def roulette_wheel_selection(self, fitness_scores):
    total_fitness = sum(fitness_scores)
    probabilities = [f / total_fitness for f in fitness_scores]
    selected_index = np.random.choice(len(self.population), p=probabilities)
    return self.population[selected_index]
```

rank_based_selection(self, fitness_scores): در این روش، کروموزوم‌ها بر اساس رتبه‌شان انتخاب می‌شوند. کروموزوم‌هایی که در رتبه‌های بالاتر قرار دارند، شانس بیشتری برای انتخاب دارند.

```
def rank_based_selection(self, fitness_scores):
    sorted_indices = np.argsort(fitness_scores)
    ranks = np.arange(1, len(self.population) + 1)
    probabilities = ranks / sum(ranks)
    selected_index = np.random.choice(sorted_indices, p=probabilities)
    return self.population[selected_index]
```

۳-۲-۸-متدهای ترکیب (Crossover Methods)

این متدها برای ترکیب دو والد و تولید نسل جدید استفاده می‌شوند. این متدها باید به گونه‌ای باشند که تعداد یک‌ها ثابت بماند بنابراین از دو روش دیگر که برای این منظور مناسب بوده‌اند استفاده شده است.:[۱۳]

Counter_based_crossover(self, parent1, parent2): این روش تلاش می‌کند که تعداد ویژگی‌های انتخاب‌شده در هر فرزند ثابت بماند به این منظور از دو شمارنده برای کنترل تعداد صفرها و یک‌ها استفاده می‌کند.

```
def Counter_based_crossover(self, parent1, parent2):
    child = [np.zeros(9), np.zeros(9)]
    for j in range(2):
        counter_one_child1 = 0
        counter_zero_child1 = 0
        for i in range(self.genome_length):
            if counter_one_child1 == self.num_of_features:
                child[j][i] = 0
            elif counter_zero_child1 == self.genome_length -
self.num_of_features:
                child[j][i] = 1
            else:
                child[j][i] = random.choice((parent1[i], parent2[i]))
                if child[j][i] == 0:
                    counter_zero_child1 += 1
                else:
                    counter_one_child1 += 1
    child1, child2 = child
    return child1, child2
```

Map_of_ones_crossover(self, parent1, parent2): در این روش، فرزندان محل قرارگیری یک‌ها را از والدین خود به ارث می‌برند.

```
def Map_of_ones_crossover(self, parent1, parent2):
    child = [np.zeros(9), np.zeros(9)]
    for i in range(2):
        pos1_p1 = [i for i, gene in enumerate(parent1) if gene == 1]
        pos1_p2 = [i for i, gene in enumerate(parent1) if gene == 1]
        for j in range(self.num_of_features):
            random_parent = random.choice((parent1, parent2))
            if (random_parent == parent1).all():
                random_pos = random.choice(pos1_p1)
            else:
                random_pos = random.choice(pos1_p2)
            child[i][random_pos] = 1
            try:
                pos1_p1.remove(random_pos)
                pos1_p2.remove(random_pos)
            except Exception:
                print("not found")
```

```
    child1, child2 = child
    return child1, child2
```

۳–۲–۹–متد جهش (Mutation Methods)

جهش‌ها برای تنوع در نسل‌ها و جلوگیری از همگرا شدن سریع الگوریتم استفاده می‌شوند. تابع زیر برای جهش در نظر گرفته شده است:

swap_mutation(self, chromosome): در این روش دو ژن به صورت تصادفی جابجا می‌شوند.

```
def swap_mutation(self, chromosome):
    if random.random() < self.mutation_rate:
        pos1, pos2 = random.sample(range(len(chromosome)), 2)
        chromosome[pos1], chromosome[pos2] = chromosome[pos2],
chromosome[pos1]
    return chromosome
```

۳–۲–۱۰–متد stopping_criteria(self, generation, best_fitness, last_best_fitness)

این متد بررسی می‌کند که آیا الگوریتم متوقف شود یا خیر. سه شرط برای توقف وجود دارد:

تعداد نسل‌ها به حداکثر تعداد مشخص‌شده (generations) رسیده باشد.

مدت خاصی (با استفاده از patience) هیچ بهبودی در دقت مشاهده نشود.

۳–۲–۱۱–متد run(self, selection_method, crossover_method)

این متد الگوریتم ژنتیک را برای یک ترکیب خاص از روش‌های انتخاب و ترکیب اجرا می‌کند. مراحل اصلی این متد عبارتند از:

انتخاب والدین با استفاده از روش‌های انتخاب.

تولید نسل جدید با استفاده از ترکیب و جهش.

ارزیابی دقت مدل در هر نسل.

بررسی معیارهای توقف.

در نهایت، بهترین کروموزوم و دقت آن برگشت داده می‌شود.

۳-۲-نتایج:

۳-۲-۱-بهترین ویژگی‌ها:

مطابق جدول زیر برای ۳ ویژگی، ۵ ویژگی و ۸ ویژگی الگوریتم اجرا شده و با حالتی که همه‌ی ویژگی‌ها محاسبه شوند، مقایسه شده است:

تعداد	ویژگی‌ها	دقت در داده تست
۳	Age, Profession, Spending_Score	۰.۳۴۰۷۶
۵	Age, Profession, Var_1 Ever_Married, Graduated	۰.۳۲۲۱۹
۸	همگی به جز Family_Size	۰.۳۱۳۸۳
۹	همه‌ی داده‌ها	۰.۳۰۸۲۶

همانطور که مشاهده می‌شود دقت مدل در داده‌های کمتر، بالاتر رفته است. بنابراین استفاده از الگوریتم ژنتیک باعث بهبود مدل نسبت به حالتی شده است که از همه‌ی ویژگی‌ها استفاده کرده‌ایم.

۳-۲-۲-مزایا و معایب استفاده از ویژگی‌های کمتر:

مزایا:

کاهش پیچیدگی مدل: کاهش تعداد ویژگی‌ها، باعث ساده‌تر شدن مدل می‌شود که این می‌تواند سرعت اجرای مدل را افزایش دهد.

بهبود دقت: با انتخاب ویژگی‌های مهم و کاهش ویژگی‌های بی‌اهمیت یا اضافی، مدل می‌تواند تمرکز بیشتری بر روی ویژگی‌های مهم‌تر داشته باشد که می‌تواند باعث بهبود دقت پیش‌بینی‌ها شود. ویژگی‌های بی‌اهمیت ممکن است نویز ایجاد کنند و از توانایی مدل برای یادگیری الگوهای صحیح جلوگیری کنند.

افزایش سرعت آموزش مدل: کاهش تعداد ویژگی‌ها باعث کاهش تعداد محاسبات مورد نیاز برای آموزش مدل می‌شود و بنابراین زمان آموزش کاهش می‌یابد. به‌ویژه در داده‌های با ابعاد بسیار بالا (مثل داده‌های تصویر یا متنی)، کاهش ابعاد می‌تواند تأثیر زیادی در زمان پردازش داشته باشد.

کاهش احتمال Overfitting: توجه مدل به ویژگی‌های غیرضروری و نادرست، ممکن است منجر به Overfitting و کاهش توانایی مدل در پیش‌بینی داده‌های جدید شود. استفاده از تعداد ویژگی‌های کمتر معمولاً به جلوگیری از این مشکل کمک می‌کند.

معایب:

از دست دادن اطلاعات مهم: انتخاب ویژگی‌های کمتر می‌تواند منجر به از دست دادن اطلاعات مفید در مورد مسئله شود. برخی از ویژگی‌های به ظاهر بی‌اهمیت ممکن است در واقع در مدل‌های پیچیده‌تر یا در ترکیب با ویژگی‌های دیگر مهم باشند. در صورتی که ویژگی‌های انتخاب‌شده به‌درستی شناسایی نشوند، ممکن است دقت مدل کاهش یابد.

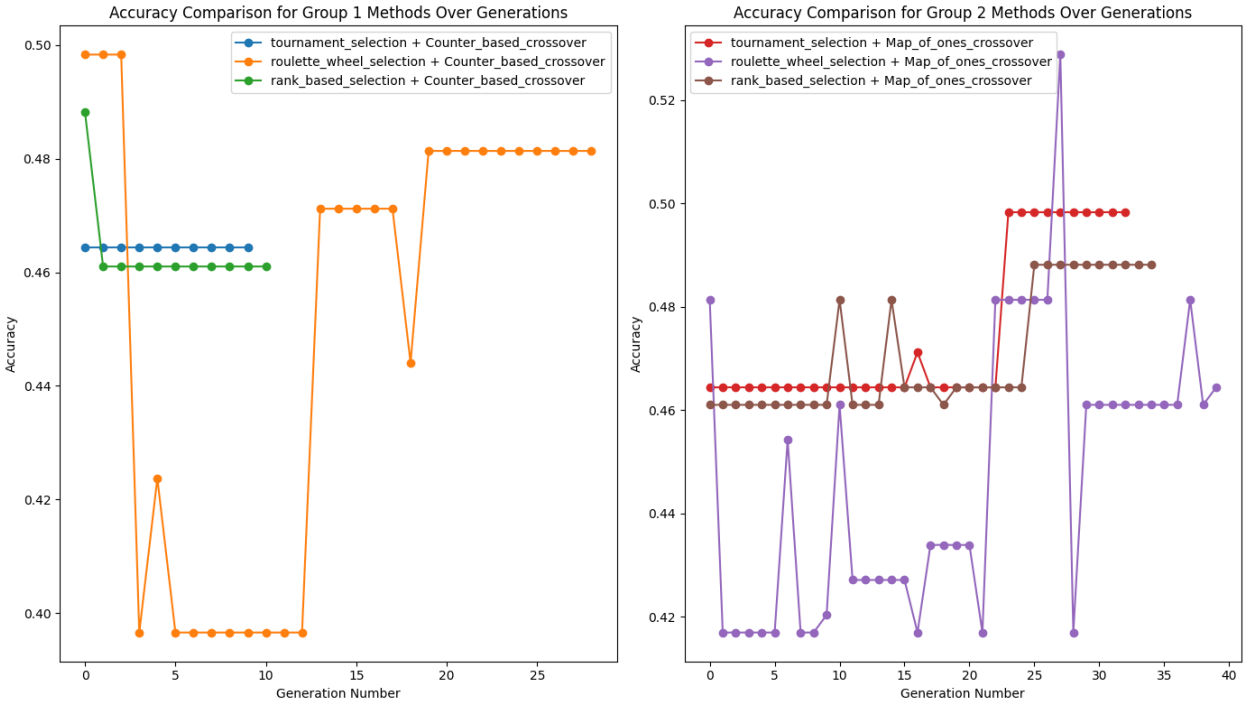
نیاز به انتخاب ویژگی دقیق: انتخاب ویژگی‌ها به‌طور دستی یا خودکار نیازمند یک فرآیند انتخاب دقیق است. این کار می‌تواند زمان‌بر باشد و نیاز به تحلیل و آزمون‌های مختلف داشته باشد.انتخاب نادرست ویژگی‌ها ممکن است منجر به نتایج ضعیف شود.

امکان افزایش خطای مدل در داده‌های غیر خطی یا پیچیده: در داده‌هایی که روابط پیچیده و غیر خطی دارند، کاهش تعداد ویژگی‌ها می‌تواند بر عملکرد مدل تأثیر منفی بگذارد. برخی ویژگی‌ها به‌ظاهر بی‌اهمیت، ممکن است در یک ترکیب خاص، اطلاعات مفیدی فراهم کنند که با حذف آن‌ها مدل دقت خود را از دست می‌دهد.

تعداد انتخاب‌های بالا در انتخاب ویژگی: اگر ویژگی‌ها به‌طور تصادفی انتخاب شوند، احتمالاً چندین ترکیب از ویژگی‌های مهم در داده‌های مختلف وجود دارد که انتخاب صحیح آن‌ها در فرآیند انتخاب ویژگی می‌تواند چالش‌برانگیز باشد.

۳-۲-۳-مقایسه‌ی روش‌های انتخاب و ترکیب:

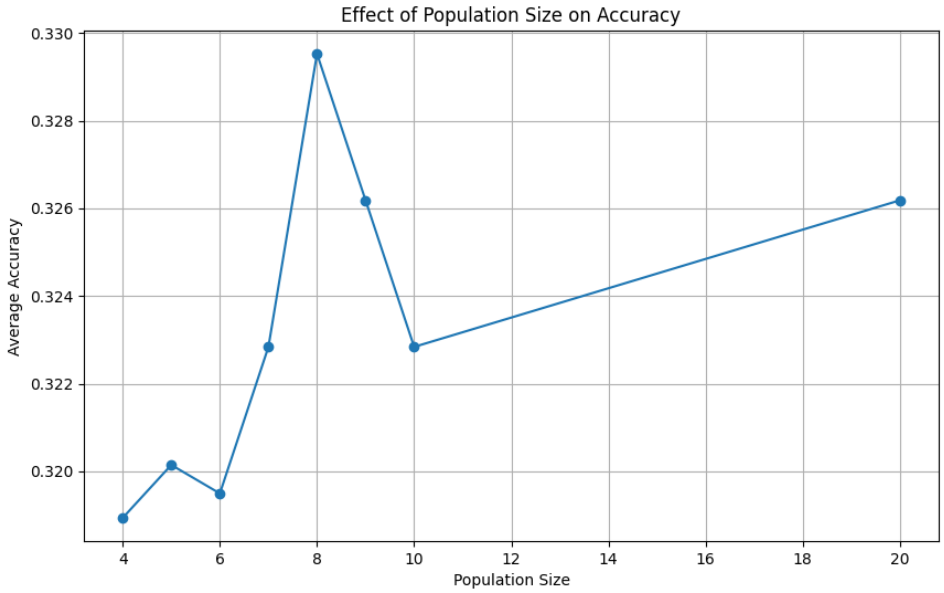
الگوریتم ژنتیک پیاده‌شده با ۳ روش انتخاب و ۲ روش ترکیب، عملا با ۶ روش می‌تواند اجرا شود در نمودار زیر دقت الگوریتم در طی نسل‌های مختلف برای هر روش محاسبه شده است.



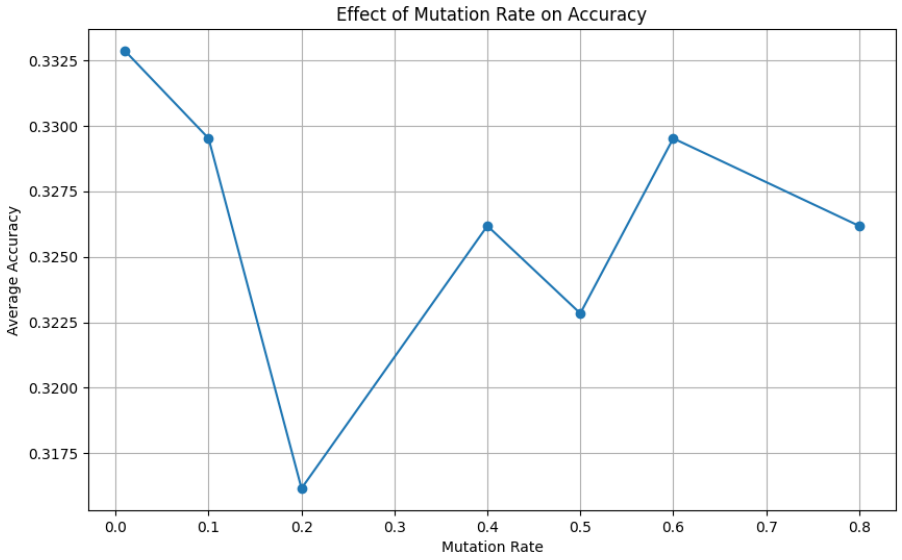
همان طور که مشاهده می‌شود، این نمودار مربوط به اجرای ۶ روش ذکر شده در الگوریتم ژنتیک با ۳ ویژگی می‌باشد. برخی ترکیب‌ها به سرعت همگرا می‌شوند مانند انتخاب تونمنت به همراه ترکیب counter_base و همچنین انتخاب رتبه‌بندی شده به همراه ترکیب counter_base. ولی برخی دیگر از روش‌ها بیشتر در فضای جست‌وجو حرکت می‌کنند و ماهیت تصادفی‌تری دارند مشابه روش چرخ رولت به همراه counter_base و یا روش چرخ رولت به همراه Map_1. البته این لزوما به معنی رسیدن به مقدار بهینه نیست چون در برخی اجراها ممکن است الگوریتم دیر همگرا شود و قبل از همگرا شدن در مکانی نامناسب متوقف شود. بنابراین روشی میانه که در زمان مناسبی همگرا شود مناسب‌تر است. مطابق این نمودار روش تورنمت به همراه Map_1 بهترین مقدار دقت را داشته و همگرایی آن پیش از پایان حداکثر تعداد نسل‌ها رخ داده است.

۳-۲-۴-بررسی تاثیر تعداد جمعیت و نرخ جهش:

با توجه به انتخاب بالا، تاثیر پارامترهای تعداد جمعیت و نرخ جهش، در روش تورنمت_ Map_1، محاسبه شده‌اند. طبق نمودار زیر، هرچه اندازه جمعیت بیشتر باشد نتایج بهتر است تا اینکه در جمعیتی با اندازه‌ی ۸ بهترین نتیجه را داریم. بعد از آن نتایج ضعیف‌تر می‌شوند.



طبق نمودار زیر نیز مشاهده می‌شود که رابطه‌ی چندانی بین نرخ جهش و دقت مدل وجود ندارد ولی در مقادیر کوچک نرخ جهش، دقت مدل بهتر بوده است.



منابع:

[1] "What are Evolutionary Algorithms | Deepchecks." Accessed: Mar. 18, 2025. [Online]. Available: https://www.deepchecks.com/glossary/evolutionary-algorithms/

[2] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," *Proceedings - International Conference on Global Trends in Signal Processing, Information Computing and Communication, ICGTSPICC 2016*, pp. 261–265, Jun. 2017, doi: 10.1109/ICGTSPICC.2016.7955308.

[3] "الگوریتم های تکاملی چیست؟ – به زبان ساده – فرادرس - مجله." Accessed: Mar. 19, 2025. [Online]. Available: https://blog.faradars.org/%D8%A7%D9%84%DA%AF%D9%88%D8%B1%DB%8C%D8%AA%D9%85-%D9%87%D8%A7%DB%8C-%D8%AA%DA%A9%D8%A7%D9%85%D9%84%DB%8C/

[4] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution Strategies as a Scalable Alternative to Reinforcement Learning," Mar. 2017, [Online]. Available: http://arxiv.org/abs/1703.03864

[5] "Evolution strategies as a scalable alternative to reinforcement learning | OpenAI." Accessed: Mar. 19, 2025. [Online]. Available: https://openai.com/index/evolution-strategies/

[6] "Genetic Algorithms - GeeksforGeeks." Accessed: Mar. 14, 2025. [Online]. Available: https://www.geeksforgeeks.org/genetic-algorithms/

[7] S. Katoch, S. S. Chauhan, and V. Kumar, “A review on genetic algorithm: past, present, and future,” *Multimed Tools Appl*, vol. 80, no. 5, pp. 8091–8126, Feb. 2021, doi: 10.1007/s11042-020-10139-6.

[8] A. Slowik and H. Kwasnicka, “Evolutionary algorithms and their applications to engineering problems,” Aug. 01, 2020, *Springer*. doi: 10.1007/s00521-020-04832-8.

[9] “مجله - فرادرس – از صفر تا صد الگوریتم ژنتیک.” Accessed: Mar. 16, 2025. [Online]. Available: https://blog.faradars.org/genetic-algorithm/

[10] “Crossover in Genetic Algorithm - GeeksforGeeks.” Accessed: Mar. 16, 2025. [Online]. Available: https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/

[11] “Mutation Algorithms for String Manipulation (GA) - GeeksforGeeks.” Accessed: Mar. 16, 2025. [Online]. Available: https://www.geeksforgeeks.org/mutation-algorithms-for-string-manipulation-ga/

[12] “Detect and Remove the Outliers using Python - GeeksforGeeks.” Accessed: Mar. 20, 2025. [Online]. Available: https://www.geeksforgeeks.org/detect-and-remove-the-outliers-using-python/

[13] L. Manzoni, L. Mariot, and E. Tuba, “Balanced crossover operators in Genetic Algorithms,” *Swarm Evol Comput*, vol. 54, p. 100646, May 2020, doi: 10.1016/J.SWEVO.2020.100646.

لینک کد:

https://colab.research.google.com/drive/1TM031NXvsxYKilolUiNmhqoYGZq95fjZ#scrollTo=KKaJXndwPvgV