شرکت آسان سیستم مبنا

www.easysoft.ir

EasySoft

برنامه نویسی ژنتیک

نگارش:

بهزاد شیرمحمدی

زمستان ۱۳۹۳

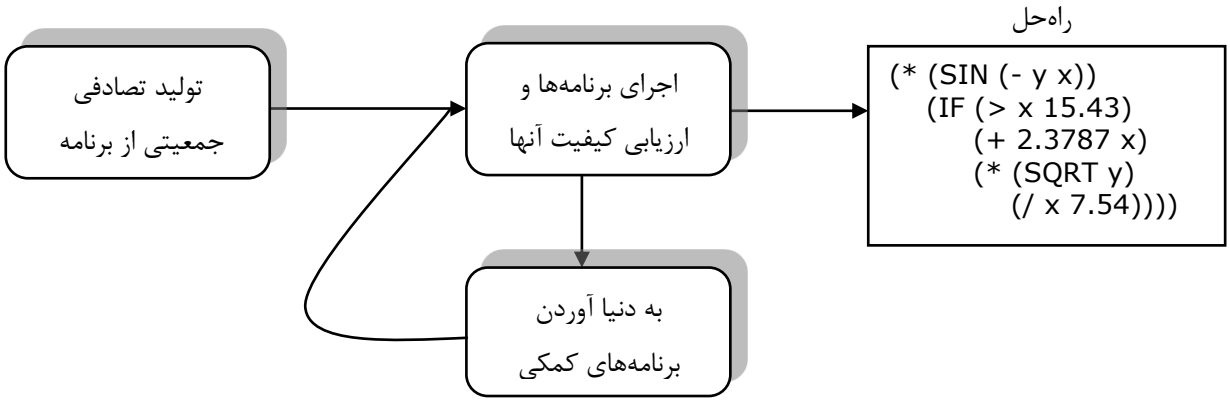
برنامه‌نویسی ژنتیک[1] شاخه‌ای از الگوریتم‌های تکاملی[2] می‌باشد که مسائل را بدون نیاز به کاربر (که شکل یا ساختاری را در پیشروی راه‌حل مساله تعیین می‌کند) حل می‌کند. در برنامه‌نویسی ژنتیک سعی می‌شود با استفاده از الگوریتم‌های ژنتیک[3] و مفاهیم درخت‌های تجزیه[4]، به جای اینکه کد برنامه لازم نوشته شود، به کامپیوتر این امکان داده شود که تنها با دانستن مفهوم کلی از کار، برنامه مورد نظر را آماده کند. در واقع یک دستور سطح بالا به کامپیوتر داده می‌شود و خود کامپیوتر برنامه لازم را برای اجرای برنامه مورد نظر آماده می‌کند، سپس برنامه را اجرا و خروجی مطلوب را ارائه می‌دهد.

الگوریتم‌های تکاملی از شاخه‌های زیر تشکیل شده است:

۱. الگوریتم‌های ژنتیک (GA)
۲. برنامه‌نویسی ژنتیک (GP)
۳. استراتژی‌های تکاملی[5] (ES)
۴. برنامه‌نویسی تکاملی[6] (EP)

تاریخچه برنامه‌نویسی ژنتیک به اوائل دهه ۹۰ میلادی باز می‌گردد و از افرادی که بیشترین تلاش‌ها را برای پیشرفت این شاخه انجام داده است می‌توان به جان کوزا[7] اشاره کرد.

در برنامه‌نویسی ژنتیک یک جمعیت از برنامه‌های کامپیوتری ایجاد می‌شود. که نسل به نسل، به صورت تصادفی تبدیل به جمعیتی از برنامه‌ای جدید و احتمالاً بهینه می‌شود. شکل ۲ روند کنترل برای برنامه‌نویسی ژنتیک را نشان می‌دهد. که در آن انتخاب طبیعی برای پیدا کردن راه‌حل‌ها استفاده شده است. برنامه‌نویسی ژنتیک، همانند طبیعت، یک فرآیند تصادفی است و هرگز نمی‌تواند نتایج را تضمین کند. هم‌چنین، GP موفقیت‌های زیادی در ارزیابی راه‌های نو و غیرمنتظره در حل مسائل دارد.



شکل ۲: روند کنترل برای برنامه‌نویسی ژنتیک

[1] genetic programming
[2] evolutionary algorithms
[3] genetic algorithms
[4] parse trees
[5] evolutionary strategies
[6] evolutionary programming
[7] John Koza

مراحل اساسی یک سامانه GP در الگوریتم ۱ نشان داده شده است. GP به وسیله اجرای این الگوریتم ایده‌آل‌ترین برنامه را پیدا می‌کند. ابتدا این الگوریتم یک جمعیت اولیه از برنامه‌ها به صورت تصادفی تولید می‌کند. سپس، با استفاده از اجرای هر یک از برنامه‌ها قابلیت یا برازندگی[8] آن‌ها را تعیین می‌کند (سطر ۳). یک یا دو برنامه از جمعیت با یک احتمال براساس قابلیت آن‌ها برای شرکت در عملیات ژنتیک انتخاب می‌شود (سطر۴). برنامه‌های جدید به وسیله به کار بردن عملیات ژنتیک برای نسل بعدی تولید می‌شود (سطر ۵). تا زمانی که یک راه‌حل قابل قبول پیدا نشود یا بعضی شرایط توقف محیا نشود این کار ادامه می‌یابد. در آخر بهترین برنامه تولید شده تا اینجا برگشت داده می‌شود.

عملیات ژنتیک اولیه برای تولید برنامه‌های جدید در زیر آمده است:

- **بازترکیبی (Crossover):** خلق یک برنامه فرزند جدید به وسیله ترکیب قسمت‌های انتخاب شده تصادفی از دو برنامه والد.
- **جهش (Mutation):** خلق یک برنامه فرزند جدید به وسیله تغییرات تصادفی یک قسمت انتخاب شده تصادفی از یک برنامه والد.

```
procedure: genetic_programming
1:    Randomly create an initial population of programs from the available primitives.
2:    repeat
3:        Execute each program and ascertain its fitness.
4:        Select one or two program(s) from the population with a probability based on fitness to
          participate in genetic operations.
5:        Create new individual program(s) by applying genetic operations with specified probabilities.
6:    until an acceptable solution is found or some other stopping condition is met (e.g., a maximum
      number of generations is reached).
7:    return the best-so-far individual.
```

الگوریتم ۱: برنامه‌نویسی ژنتیک

بنابراین، یک الگوریتم GP روی یک جمعیت از اعضا کار می‌کند که هر یک از آن‌ها نشان دهنده یک راه‌حل بالقوه از مساله است. مراحل GP در زیر آمده است:

۱- تولید یک جمعیت اولیه به صورت تصادفی با استفاده از ترکیب توابع و پایانه‌های مساله
۲- اجرای هریک از برنامه‌های جمعیت و نسبت دادن یک مقدار برازندگی به آن‌ها
۳- ساختن برنامه‌های کامپیوتری جدید، با نام فرزند[9]

الف) کپی کردن بهترین برنامه موجود، با نام تکثیر[10]

ب) ساختن یک برنامه جدید به وسیله جهش

ج) ساختن یک برنامه جدید به وسیله باز ترکیبی

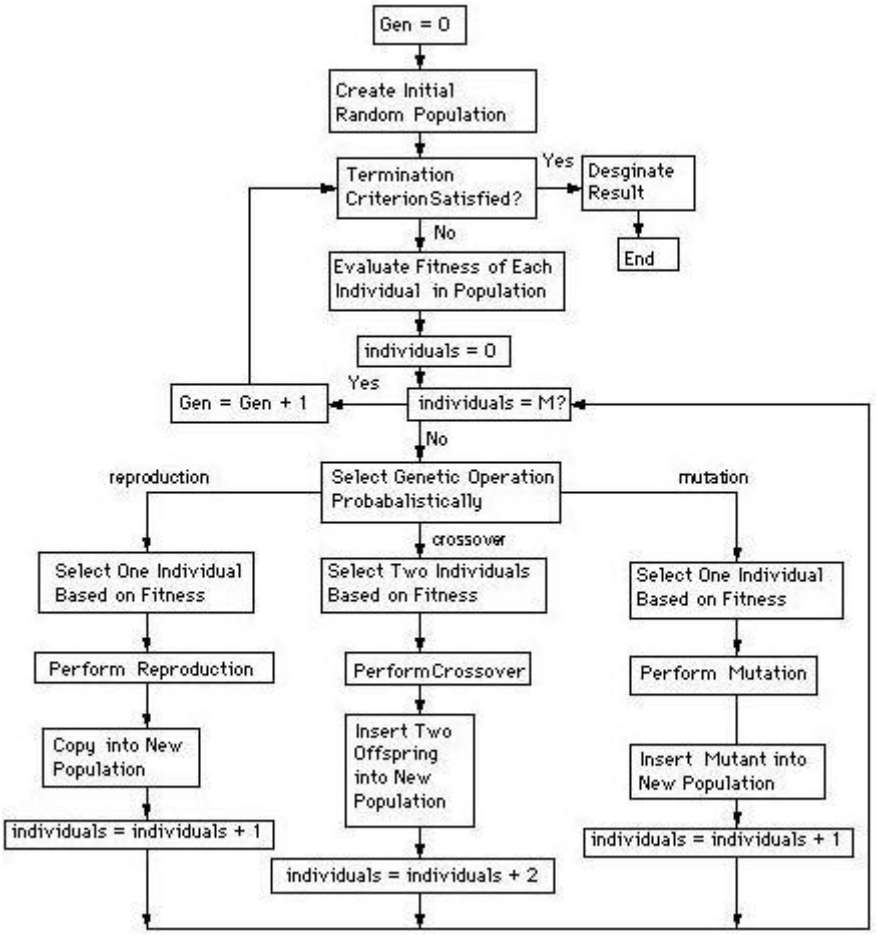
[8] fitness

[9] offspring

[10] reproduction

۴- بهترین برنامه کامپیوتری به دست آمده در هر نسل، بهترین تا اینجا، به عنوان نتیجه اجرای GP نشان داده می‌شود.

یک فلوچارت از مراحل GP در شکل ۳ آمده است.

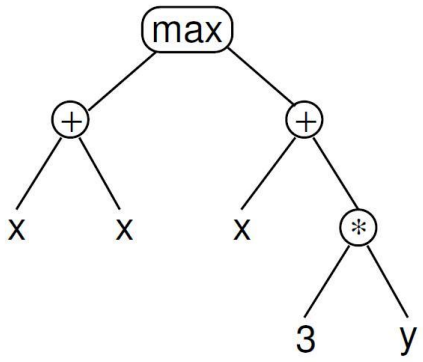


شکل ۳: فلوچارت GP [30]

در GP، برنامه‌ها معمولاً به وسیله نحو درخت‌ها بیان می‌شوند. برای مثال شکل ۴ درخت برنامه max(x+x,x+3*y) را نشان می‌دهد. متغیرها و ثابت‌ها در برنامه (x، y و 3) برگ‌های درخت هستند که در GP به آن‌ها پایانه[11] می‌گویند. عملگرهای محاسباتی (+، * و max) گره‌های داخلی هستند که به آن‌ها تابع[12] می‌گویند. مجموعه توابع و پایانه‌های مجاز باهم مجموعه اولیه از سامانه GP را تشکیل می‌دهند.

[11] terminal

[12] function



شکل ۴: نمایش نحو درخت GP: max(x+x,x+3*y)

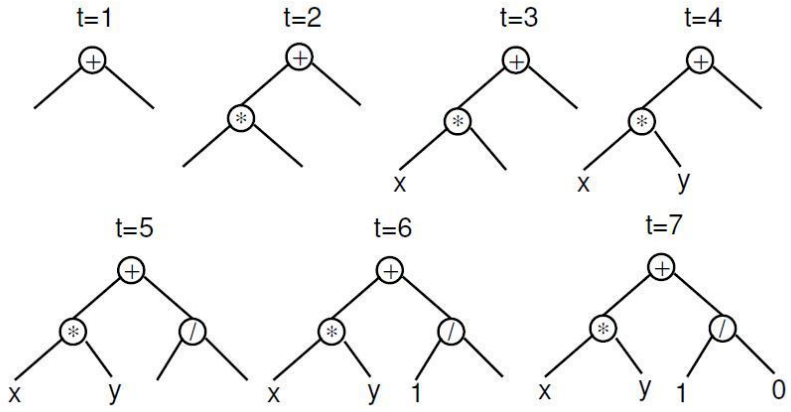
در ادبیات GP رایج است عبارات منطقی با یک علامت‌گذاری پیشوندی شبیه آنچه در لیسپ[۱۳] یا اسکیم[۱۴] استفاده شده است بیان شود. برای مثال max(x+x,x+3*y) برابر با (max(+xx)(+x(*3y))) است. این علامت‌گذاری ارتباط بین عبارت منطقی و درخت‌های متناظر را آسانتر می‌کند. با توجه به آنکه تمامی توابع به کار رفته در برنامه‌نویسی ژنتیک تعداد ثابتی از آرگومان‌ها را قبول می‌کند، پس پارانتزها در علامت‌گذاری پیشوندی عبارات منطقی زاید هستند و می‌توانند حذف شوند. بنابراین درخت‌ها همانند ترتیب‌های خطی ساده می‌توانند موثرتر نشان داده شوند. برای مثال عبارت منطقی (max(+x (x)(+x(*3y))) می‌تواند به این صورت نوشته شود: max+xx+x*3y.

پارامترهای برنامه‌نویسی ژنتیک

شبیه دیگر الگوریتم‌های تصادفی، در برنامه‌نویسی ژنتیک نیز اشخاص جمعیت اولیه به صورت تصادفی تولید می‌شوند. روش‌های متفاوتی برای تولید جمعیت اولیه به صورت تصادفی وجود دارد. از ساده‌ترین روش‌ها می‌توان به روش‌های کامل[۱۵]، رشد کردن[۱۶] و ترکیبی از این دو با نام نصف به نصف[۱۷] اشاره کرد. در هر دو روش کامل و رشد کردن جمعیت اولیه تولید شده از حداکثر عمقی که کاربر معین می‌کند تجاوز نمی‌کنند. عمق یک گره برابر است با تعداد یال‌هایی که گره نیاز دارد تا به گره ریشه برسد (گره ریشه با عمق صفر فرض شده است). عمق یک درخت برابر با عمق عمیق‌ترین برگ است (برای مثال، درخت شکل ۲-۱ دارای عمق ۳ است).

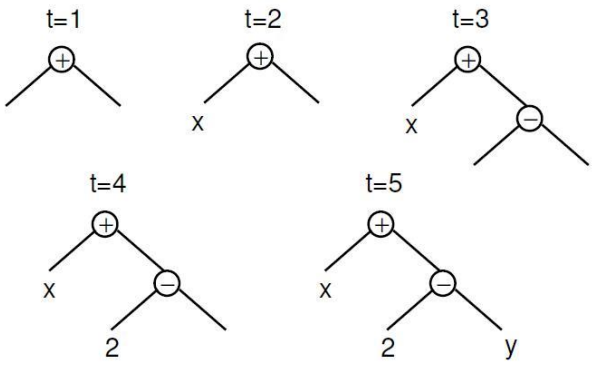
[13] lisp
[14] scheme
[15] full method
[16] grow method
[17] ramped half-and-half method

در روش کامل (به دلیل آنکه این روش درخت کامل، یعنی در آن همه برگ‌ها عمق مشابه دارند، تولید می‌کند به آن روش کامل می‌گویند) گره‌ها به صورت تصادفی تا زمانی که به عمق ماکزیمم برسد از مجموعه تابع انتخاب می‌شوند و در برگ‌ها فقط پایانه‌ها می‌توانند انتخاب شوند. شکل ۵ یک سری تصاویر لحظه‌ای از ساختمان یک درخت کامل به عمق ۲ را نشان می‌دهد. فرزندان گره‌های / و * باید برگ باشند در غیر این صورت درخت عمیق‌تر می‌شود. بنابراین، در مراحل t=3، t=4، t=6 و t=7 یک پایانه باید انتخاب شود.



شکل ۵: ایجاد یک درخت کامل با حداکثر عمق ۲ با استفاده از روش کامل

برخلاف روش کامل، روش رشد کردن همیشه درخت‌هایی با اندازه‌ها و شکل‌های گوناگون تولید می‌کند. در این روش گره‌ها به صورت تصادفی تا زمانی که به عمق ماکزیمم برسد از مجموعه اولیه (یعنی تابع و پایانه) انتخاب می‌شوند و در برگ‌ها فقط پایانه‌ها می‌توانند انتخاب شوند. شکل ۶ این فرآیند را برای ساختن یک درخت با عمق ۲ نشان می‌دهد.



شکل ۶: ایجاد یک درخت با ۵ گره با حداکثر عمق ۲ با استفاده از روش رشد کردن

شبه کد اجرای بازگشتی هر دو روش کامل و رشد کردن در الگوریتم ۲ آمده است. در این کد func_set مجموعه تابع و term_set مجموعه پایانه است و max_d نشان دهنده حداکثر عمق برای عبارات منطقی، method برابر کامل یا رشد کردن است. expr عبارت تولید شده در علامت‌گذاری پیشوندی و rand() یک تابع است که اعداد تصادفی یکنواخت توزیع شده بین ۰ و ۱ را برمی‌گرداند.

```
procedure: gen_rnd_expr(func_set,term_set,max_d,method)
1:      if max_d = 0 or (method = grow and rand() < |term_set|/(|term_set|+|func_set|)) then
2:              expr = choose_random_element( term_set )
3:      else
4:              func = choose_random_element( func_set )
5:              for i = 1 to arity(func) do
6:                      arg_i = gen_rnd_expr( func_set, term_set, max_d - 1, method );
7:              end for
8:              expr = (func, arg_1, arg_2, ...);
9:      end if
10:     return expr
```

الگوریتم ۲: شبه کد اجرای بازگشتی روش‌های کامل و رشد کردن

به دلیل آنکه هیچکدام از روش‌های کامل و رشد کردن یک ناحیه گسترده از اندازه‌ها و شکل‌ها روی خودشان تولید نمی‌کنند، ترکیبی از این دو روش با نام روش نصف به نصف فرض شده است که در آن نصف جمعیت اولیه به وسیله روش کامل و نصف دیگر با روش رشد کردن ساخته می‌شود. به این ترتیب با استفاده از این روش درختان با اندازه‌ها و شکل‌های متفاوت در یک ناحیه با عمق محدود تولید می‌شوند.

بعد از راه‌اندازی جمعیت، تعدادی از اشخاص جمعیت به صورت تصادفی انتخاب می‌شوند. آن‌ها با دیگری مقایسه می‌شوند تا بهترین آن‌ها از نظر برازندگی به عنوان والد برای انجام عملیات بازترکیبی و جهش انتخاب شود.

چندین روش انتخاب[18] وجود دارد که از آن جمله می‌توان به روش‌های زیر اشاره کرد:

- **روش چرخ رولت**[19]: الگوریتم روش چرخ رولت برنامه‌ها را بر طبق مقادیر برازندگی برای نسل جدید انتخاب می‌کند. یک برنامه با مقدار برازندگی زیاد، شانس زیادی برای عضو شدن در نسل جدید دارد و هر عضو می‌تواند چندین بار برای نسل جدید انتخاب شود. چرخ رولت به سکتورها تقسیم می‌شود که اندازه هر سکتور متناسب با مقدار برازندگی برنامه‌ها است. اندازه چرخ نیز برابر با مجموع تمام مقادیر برازندگی است. برای انتخاب یک عضو، یک عدد تصادفی انتخاب می‌شود، عدد انتخاب شده در هر سکتور که قرار داشت عضو نسبت داده شده به آن سکتور انتخاب می‌شود.
- **روش رتبه‌بندی**[20]: این الگوریتم مشابه الگوریتم چرخ رولت می‌باشد، اما یک تفاوت در روش محاسبه اندازه سکتورها دارد. اندازه سکتور بدترین برنامه برابر است با ۱، اندازه سکتور برنامه بعدی برابر با ۲ و به همین ترتیب.

[18] selection method

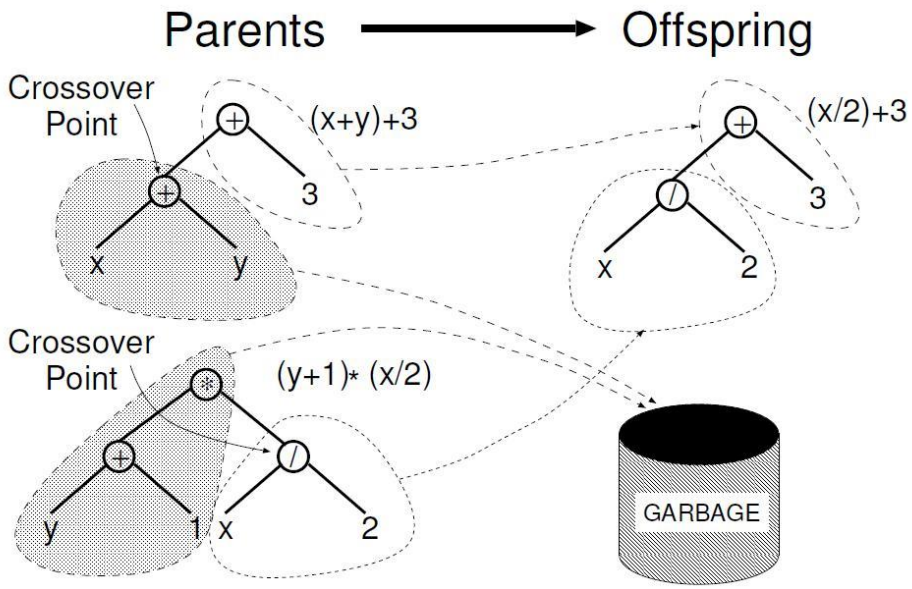
[19] roulette wheel method

[20] rank method

اندازه چرخ نیز همواره برابر است با size*(size+1)/2. که در این رابطه size همان اندازه جمعیت فعلی است. همانند روش چرخ رولت برای انتخاب یک عضو، یک عدد تصادفی انتخاب می‌شود، عدد انتخاب شده در هر سکتور که قرار داشت عضو نسبت داده شده به آن سکتور انتخاب می‌شود.

- **روش مسابقه**[۲۱]: در روش انتخاب مسابقه K عضو از جمعیت به صورت تصادفی برای مسابقه انتخاب می‌شوند. هر عضو می‌تواند چندین بار انتخاب شود. از بین K عضو انتخاب شده، بهترین عضو از نظر برازندگی به عنوان عضو انتخاب شده الگوریتم برگزیده می‌شود. این روش چندین بار انجام می‌شود تا جمعیت نسل میانی تولید شود. هم-چنین، مقدار K یک مقدار اختیاری می‌باشد و عواملی همانند اندازه جمعیت بستگی دارد.

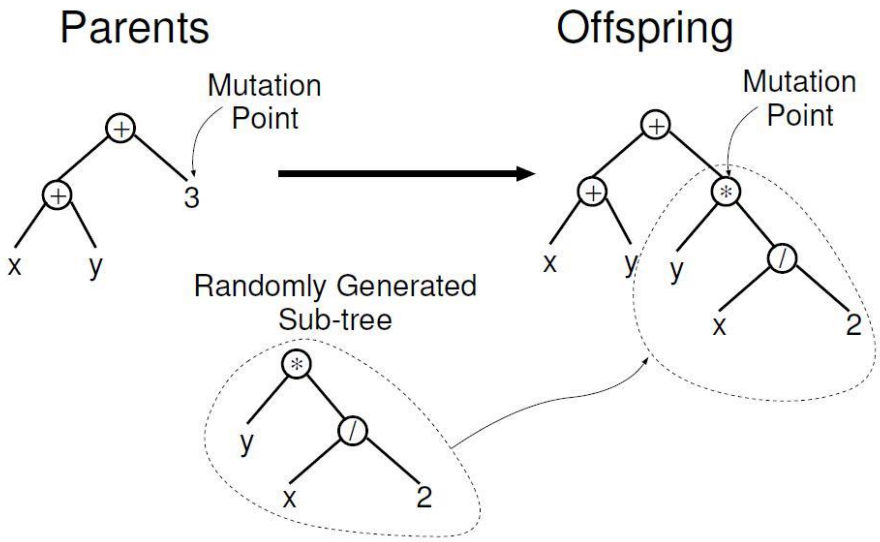
جهت انجام عملیات بازترکیبی، دو والد معین انتخاب می‌شوند، از هر درخت یک نقطه (یک گره) به عنوان نقطه بازترکیبی انتخاب می‌شود. خلق فرزند به وسیله جایگزینی ریشه زیردرخت نقطه بازترکیبی از والد اول با یک کپی از زیردرخت بازترکیبی والد دوم صورت می‌گیرد. مثالی از بازترکیبی در شکل ۷ نشان داده شده است.



شکل ۷: مثالی از عملیات بازترکیبی

جهت انجام عملیات جهش یک والد انتخاب می‌شود، یک نقطه از درخت به عنوان نقطه جهش انتخاب می‌شود. سپس این زیردرخت با زیردرختی که به صورت تصادفی تولید شده است جایگزین می‌شود. مثالی از جهش در شکل ۸ نشان داده شده است.

[21] tournament method



شکل ۸: مثالی از عملیات جهش

پس از مشخص شدن اعضای جمعیت نسل میانی، این نسل باید جایگزین نسل فعلی شود. به همین منظور چندین روش جایگزینی[۲۲] وجود دارد که در زیر آمد است:

- روش عمومیت دادن[۲۳]: در این روش تمامی برنامه‌های نسل میانی نظیر به نظیر جایگزین نسل فعلی می‌شود. این جایگزینی به دو صورت با نخبه‌کشی[۲۴] و بدون نخبه‌کشی انجام پذیر است. اگر از حالت نخبه‌کشی استفاده شود، بهترین برنامه نسل فعلی از نظر برازندگی با برنامه نظیر در نسل میانی مقایسه خواهد شد و اگر مقدار برازندگی بهتری داشت در جای خود باقی می‌ماند. اما در روش بدون نخبه‌کشی این مقایسه انجام نخواهد شد.
- روش حالت دائمی[۲۵]: در این روش جهت جایگزینی برنامه‌های نسل میانی، تمام برنامه‌های نسل میانی از نظر برازندگی با برنامه نظیر خود در نسل فعلی مقایسه می‌شود اگر مقدار برازندگی بهتری داشته باشد جایگزینی انجام خواهد شد.

بنابراین، در حالت کلی پارامترهای برنامه‌نویسی ژنتیک عبارتند از: اندازه جمعیت، تعداد نسل‌های برنامه‌نویسی ژنتیک، احتمال انجام دادن عملیات ژنتیک (عملیات بازترکیبی و جهش)، حداکثر اندازه برنامه‌ها (حداکثر عمق درخت‌ها)، تعیین روش تولید جمعیت اولیه (روش کامل، رشد کردن و نصف به نصف)، تعیین روش انتخاب (روش چرخ رولت، رتبه‌بندی و مسابقه) و تعیین روش جایگزینی (روش عمومیت دادن و حالت دائمی).

[22] replacement methods

[23] generalization method

[24] elitism

[25] steady-state method

اجرای برنامه‌نویسی ژنتیک

مراحل اجرای برنامه‌نویسی ژنتیک در زیر آمده است.

مرحله ۱: مجموعه پایانه

اولین مرحله تعیین مجموعه پایانه می‌باشد. مجموعه پایانه ممکن شامل موارد زیر باشد:

- *ثابت‌ها:* ثابت‌ها می‌توانند به صورت تصادفی در یک قسمت از فرآیند تولید درخت ایجاد شوند.
- *ورودی‌های خارجی برنامه‌ها:* ورودی‌های خارجی نوعاً شکلی از نام متغیرها را می‌گیرند (یعنی x و y)
- *توابع بدون آرگومان:* این توابع هر زمانی که فراخوانی می‌شوند مقادیر مختلفی را برمی‌گردانند. مانند تابع rand() که اعداد تصادفی را برمی‌گرداند یا تابع dist_to_wall() که فاصله یا مانع با ربات را برمی‌گرداند.

مرحله ۲: مجموعه تابع

مجموعه تابع استفاده شده در GP به طبیعت ناحیه مساله بستگی دارد. برای مثال، در یک مساله عددی ساده مجموعه تابع ممکن است فقط شامل توابع الگوریتمی باشد (+، -، * و /). بعضی اوقات مجموعه اولیه شامل پایانه‌ها و توابع برای حل مسائل در یک ناحیه ویژه از مساله طراحی شده‌اند. برای مثال، اگر هدف برنامه ربات تمیز کردن کف زمین باشد مجموعه تابع ممکن است شامل عمل‌های حرکت، چرخش و تمیز کردن باشد. جدول ۷ نشان دهنده یک مثال از برخی توابع و پایانه‌های دیده شده در ادبیات GP می‌باشد.

جدول ۷: مثالی از مجموعه تابع و پایانه در GP

مجموعه تابع	
نوع اولیه	**مثال**
الگوریتمی	+, *, /
ریاضی	sin, cos, exp
بولی	AND, OR, NOT
شرطی	IF-THEN-ELSE
حلقه	FOR, REPEAT
. . .	...
مجموعه پایانه	
نوع اولیه	**مثال**
مقادیر ثابت	3, 0.45
متغیر	x, y
توابع بدون آرگومان	rand, go_left

قبل از ادامه اجرای برنامه‌نویسی ژنتیک، لازم است که به دو مساله توجه شود.

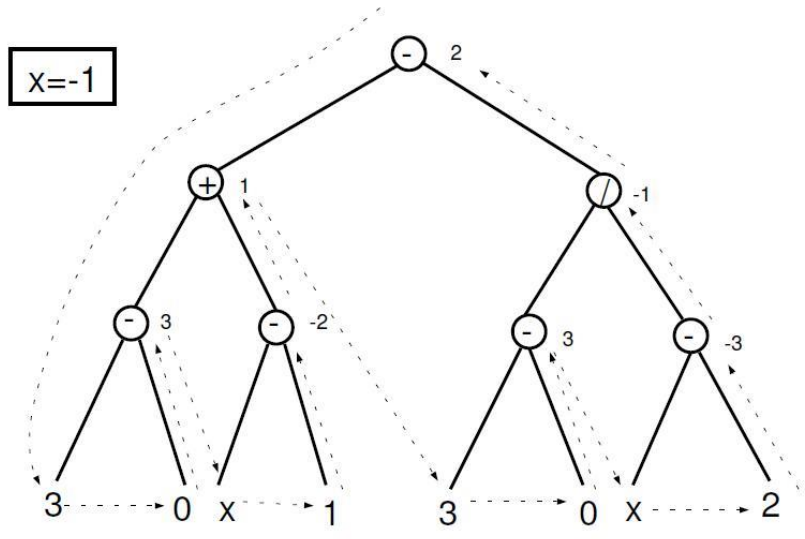
اول باید مساله بستار (Closure) رعایت شده باشد. یعنی اینکه هر تابعی که مقدار می‌گیرد، قادر باشد که تمام نتایج خروجی را مدیریت کند. به عنوان مثال فرض می‌شود برنامه‌ها فقط از عملگرهای ریاضی استاندارد یعنی {- ,+ ,/ ,*} به عنوان توابع مورد استفاده و از مقادیر {... ,2 ,1 ,0} به عنوان مقادیر پایانه استفاده می‌کنند. در این حالت قانون بستار رعایت نشده است. زیرا با این اعمال تمام حالت‌های خروجی معتبر نیستند، چرا که یک برنامه مانند (0 2 /) باعث بروز خطای تقسیم بر صفر می‌شود. برای رعایت قانون بستار بایستی که تابع تقسیم به گونه‌ای تعریف شود که سامانه را از کار نیاندازد و موجب خاتمه غیرعادی برنامه نشود. به عنوان مثال در عملگر تقسیم، اگر آرگومان دوم صفر باشد بعضی مقادیر پیش فرض مانند 1 را برمی‌گرداند (بدون توجه به مقدار آرگومان اول). به این نسخه از عمگر تقسیم نسخه حفاظت شده می‌گویند و اغلب با ٪ نمایش می‌دهند. در مورد مواردی مانند عملگرهای تک عملوندی مانند عملگر Not، می‌توان به این شکل عمل کرد که این عملگر دو عملوند به عنوان ورودی بپذیرد ولی عملوند دوم هیچ تاثیری نداشته باشد.

دومین مساله‌ای که بایستی مورد توجه قرار بگیرد این است که مساله‌ای که برای آن برنامه نوشته می‌شود باید با ترکیبی از پایانه‌ها و توابعی که در نظر گرفته شده است قابل حل باشد. به عنوان مثال اگر هدف یک برنامه محاسبه لگاریتم یک عدد را باشد، در این حالت با استفاده از توابع {- ,+} و پایانه‌های {... ,3 ,2 ,1} برنامه در بهترین حالت نیز قادر نیست که مقدار لگاریتم را با تقریب مناسب محاسبه کند. چرا که نتایج که به دست آمده، قطعاً فقط اعداد صحیح هستند.

مرحله ۳: تابع برازندگی

برازندگی می‌تواند به وسیله روش‌های مختلف اندازه‌گیری شود. برای مثال، برحسب: میزان خطای بین خروجی و خروجی مطلوب؛ میزان زمان لازم برای رساندن یک سامانه به حالت هدف مطلوب؛ دقت و صحت برنامه در الگوهای تعیین شده؛ بازدهی که یک برنامه ارائه می‌دهد.

به دلیل آنکه ساختارهای ارزیابی شده در GP برنامه‌های کامپیوتری هستند، ارزیابی برازندگی معمولاً نیازمند اجرای تمامی برنامه‌های جمعیت می‌باشد. با توجه به آنکه مقدار پردازش توسط یک کامپایلر قابل توجه است، معمولاً از یک مفسر برای ارزیابی برنامه‌ها استفاده می‌شود. تفسیر کردن یک درخت برنامه به این معنی است که گره‌ها در درخت به ترتیبی باید اجرا شوند که تضمین کند که قبل از آنکه مقدار آرگومان آن‌ها مشخص نشده است اجرا نمی‌شوند. به وسیله گردش درخت به صورت بازگشتی با شروع از گره ریشه، و به تعویق انداختن ارزیابی هر گره تا زمانی که مقادیر فرزندان آن شناخته شده باشد این کار شدنی است. فرآیند بازگشتی اول عمق در شکل ۹ نشان داده شده است. الگوریتم ۳ شبه کد پیاده‌سازی پروسه تفسیر را نشان می‌دهد. در این الگوریتم فرض شده است که برنامه‌ها با عبارات منطقی علامت‌گذاری پیشوندی نشان داده می‌شوند.



شکل ۹: مثالی از تفسیر نحوی درخت

```
procedure: eval( expr )
1:      if expr is a list then
2:              proc = expr(1) {Non-terminal: extract root}
3:              if proc is a function then
4:                      value = proc( eval(expr(2)), eval(expr(3)), ... ) {Function: evaluate arguments}
5:              else
6:                      value = proc( expr(2), expr(3), ...) {Macro: don't evaluate arguments}
7:              end if
8:      else
9:              if expr is a variable or expr is a constant then
10:                     value = expr {Terminal variable or constant: just read the value}
11:             else
12:                     value = expr() {Terminal 0-arity function: execute}
13:             end if
14:     end if
15:     return value
```

الگوریتم ۳: مفسر برنامه‌نویسی ژنتیک

مرحله ۴: پارامترهای GP

چهارمین مرحله تعیین کردن پارامترهای کنترل برای اجرا است. مهمترین پارامتر کنترل اندازه جمعیت است، پارامترهای کنترل دیگری همانند احتمال انجام دادن عملیات ژنتیک، حداکثر اندازه برای برنامه‌ها و دیگر جزئیات اجرا وجود دارد. معمولاً جمعیت اولیه به صورت تصادفی با استفاد از روش نصف به نصف با ناحیه عمق ۲ تا ۶ تولید می‌شود. اندازه درخت‌های اولیه به تعداد توابع، تعداد پایانه‌ها و تعداد آرگومان‌های توابع وابسته است. در بیشتر حالت‌ها محدودیت اصلی روی اندازه جمعیت به دلیل زمان لازم برای ارزیابی برازندگی‌ها می‌باشد و نه برای ذخیره اعضای جمعیت. معمولاً اندازه جمعیت باید حداقل شامل

۵۰۰ عضو باشد. زمان اجرای GP می‌تواند تخمین زده شود به وسیله: تعداد اجراها R، تعداد نسل G، اندازه جمعیت P، میانگین اندازه برنامه‌ها S و تعداد حالت برازندگی F.

مرحله ۵: خاتمه و طراحی راه‌حل

پنجمین مرحله شامل تعیین کردن ملاک خاتمه و روش نشان دادن نتایج اجرا است. ملاک خاتمه ممکن شامل حداکثر تعداد نسل‌ها برای اجرای یک برنامه مخصوص باشد و بهترین عضو تا اینجا محصول تولید شده می‌باشد.

مثالی از اجرای برنامه‌نویسی ژنتیک

این بخش یک مثال از اجرای GP را نشان می‌دهد، هدف تولید خودکار یک برنامه با رفتار ورودی/خروجی مشخص می‌باشد. مثال این بخش استنتاج کردن یک عبارت نظیر چند جمله‌ای درجه دوم x^2+x+1 در ناحیه [-1,+1] می‌باشد.

مراحل اجرا

به دلیل آنکه مساله پیدا کردن تابع ریاضی از روی یک متغیر مستقل، x، است، مجموعه پایانه‌ها شامل این متغیر می‌باشد. این مجموعه هم‌چنین شامل ثابت‌های تصادفی در یک ناحیه متعارف، مثلاً از 0.5- تا 0.5+ می‌باشد. بنابراین مجموعه پایانه، T، برابر است با:

$$T = \{x, R\}$$

یک انتخاب ساده برای مجموعه تابع همان چهار تابع الگوریتمی معمولی است: +، -، * و /. بیشتر مساله‌های رگرسیون عددی عملیات بزرگ با توابع اضافی همانند sin() و log() را لازم دارد. ولی در این مثال از مجموعه تابع ساده استفاده شده است:

$$F = \{+, -, *, \%\}$$

% همان تقسیم محاظت شده است. نکته که چند جمله‌ای مورد نظر دقیقاً می‌تواند با استفاده از مجموعه تابع و پایانه انتخاب شده بیان شود.

سومین مرحله شامل ساختن اندازه برازندگی است. هدف سطح بالای این مساله پیدا کردن مساله‌ای است که خروجی آن برابر با مقدار خروجی چند جمله‌ای درجه دوم x^2+x+1 باشد. بنابراین برازندگی عضو مخصوص جمعیت باید خروجی خیلی نزدیک به چند جمله‌ای هدف را منعکس کند. در این مساله برازندگی به صورت مجموع خطاهای مطلق اندازه‌گیری شده با مقادیر متفاوت از متغیر مستقل x در ناحیه [-1.0,+1.0] تعریف شده است. در این مساله، خطاها برای مقادیر زیر اندازه‌گیری می‌شوند:

$$x \in \{-1.0, -0.9, ..., 0.9, 1.0\}$$

بنابراین، هرچه مقدار برازندگی (خطا) کمتر باشد برنامه بهتر است و یک برازندگی (خطا) با مقدار صفر نشان دهنده یک تناسب کامل است.

در مرحله چهارم پارامترهای اجرا باید تنظیم شوند. اندازه جمعیت در این مثال کوچک فقط ۴ در نظر گرفته شده است. عملیات بازترکیبی استفاده شده تقریباً ۹۰ درصد از اعضای جمعیت را تولید می‌کند؛ عملیات تکثیر (که در آن یک عضو متناسب به صورت ساده کپی می‌شود) تقریباً ۸ درصد از اعضای جمعیت را تولید می‌کند؛ عملیات جهش ۱ درصد از جمعیت را تشکیل می‌دهد؛ و عملیات تغییر معماری تقریباً ۱ درصد از اعضای جمعیت را تشکیل می‌دهد. به دلیل انکه این مثال شامل یک جمعیت کوچک غیر عادی است، در این مثال عملیات بازترکیبی دو بار استفاده خواهد شد برابر با ۵۰ درصد از جمعیت؛ عملیات جهش و تکثیر یک بار استفاده خواهد شد برابر با ۲۵ درصد از جمعیت؛ و برای سادگی عملیات تغییر معماری در این مثال استفاده نشده است.

مرحله پنجم و نهایی یک شرط خاتمه تعیین می‌کند. یک شرط خاتمه منطقی برای این مساله این است که اجرا از نسل به نسل ادامه خواهد یافت تا زمانی که برازندگی (خطا) بعضی از اعضا کوچکتر از 0.1 شود.

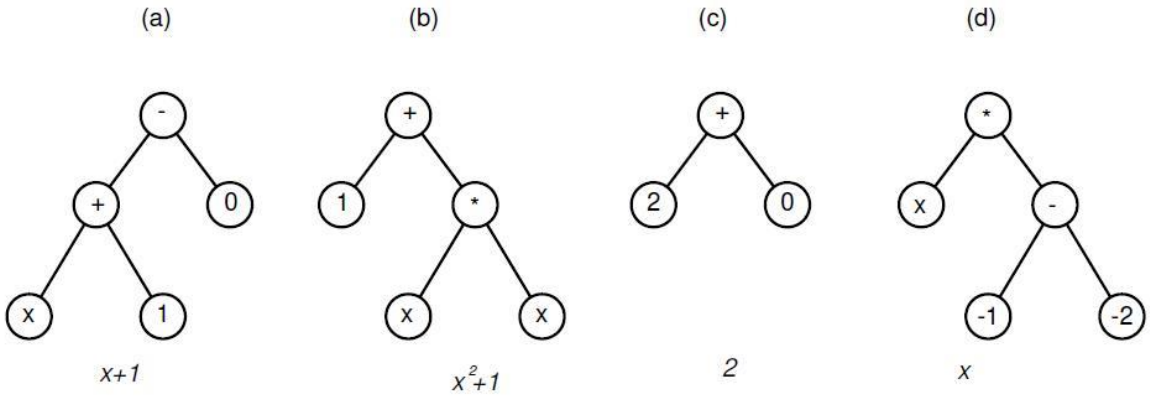
اجرای مرحله به مرحله

بعد از مشخص شدن مراحل اجرا حال می‌توان، GP را مرحله به مرحله اجرا کرد. خلاصه اجرای GP در جدول ۸ آمده است.

جدول ۸: پارامترهای مثال اجرای برنامه‌نویسی ژنتیک

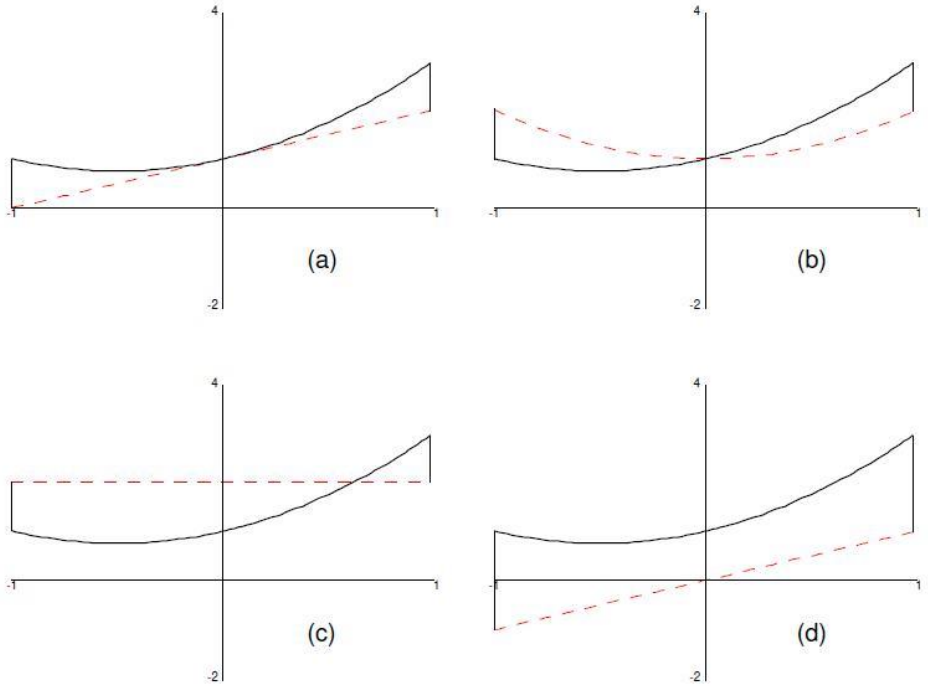
هدف	پیدا کردن برنامه‌ای که خروجی آن نظیر x^2+x+1 در ناحیه $-1 \leq x \leq +1$ باشد.
مجموعه تابع	+، -، *، % (تقسیم حفاظت شده)؛ همه عملیات اعشاری هستند.
مجموعه پایانه	x و ثابت‌های انتخاب شده به صورت تصادفی بین -5 و +5.
برازندگی	مجموع خطاهای مطلق برای $x \in \{-1.0, -0.9, ..., 0.9, 1.0\}$
انتخاب	برحسب برازندگی
جمعیت اولیه	روش نصف به نصف (عمق ۱ تا ۲، ۵۰ درصد از پایانه‌ها ثابت‌ها هستند)
پارامترها	اندازه جمعیت ۴، ۵۰ درصد زیردرخت بازترکیبی، ۲۵ درصد تکثیر، ۲۵ درصد زیردرخت جهش، اندازه درخت بدون محدودیت.
خاتمه	عضوی با برازندگی بهتر از 0.1 پیدا شود.

فرآیند مقداردهی اولیه با تولید جمعیت به صورت تصادفی از چهار عضو برنامه‌های کامپیوتری شروع می‌شود. چهار برنامه در شکل ۱۰ به صورت درخت نشان داده شده‌اند. اولین برنامه ایجاد شده به صورت تصادفی (شکل ۱۰ a) برابر است با عبارت x+1؛ برنامه دوم (شکل ۱۰ b) برابر با x^2+1؛ برنامه سوم (شکل ۱۰ c) برابر با مقدار ثابت 2 و برنامه چهارم برابر با عبارت x می‌باشد (شکل ۱۰ d).



شکل ۱۰: جمعیت اولیه تولید شده به صورت تصادفی با چهار عضو از نسل 0

برنامه‌های کامپیوتری تولید شده به صورت تصادفی معمولاً در حل مساله خیلی ضعیف هستند. چهار عضو تصادفی نسل 0 در شکل ۱۰ خروجی‌هایی تولید می‌کنند که به وسیله مقدارهای مختلف از تابع x^2+x+1 منحرف می‌شوند. برای ارزیابی برازندگی هر یک از برنامه‌ها شکل ۱۱ نمودارهای آن‌ها را با نمودار تابع x^2+x+1 مقایسه می‌کند. مجموع خطاهای مطلق برای خط مستقیم x+1 برابر با 7.7 می‌باشد (شکل ۱۱ a)؛ مجموع خطاهای مطلق برای سهمی x^2+1 برابر با 11.0 است (شکل ۱۱ b) و مجموع خطاهای مطلق برای دو عضو باقیمانده برابر با 17.98 (شکل ۱۱ c) و 28.7 (شکل ۱۱ d) است.



شکل ۱۱: نمودارهای توابع ارزیابی نسل 0

در شکل ۱۱ مشاهده می‌شود که خط مستقیم x+1 در ناحیه 1- تا 1+ نسبت به سه برنامه دیگر به سهمی x^2+x+1 نزدیکتر است. این خط مستقیم مسلماً برابر با سهمی مورد نظر نیست، آن حتی یک تابع درجه دوم نیز نیست، آن فقط بهترین برنامه از نسل 0 می‌باشد.

بعد از آنکه برازندگی هر عضو در جمعیت مشخص شد، برای تولید نسل بعدی GP برنامه‌هایی را از این نسل انتخاب می‌کند. عملیات ژنتیک در اعضای انتخاب شده به کار می‌رود تا برنامه‌های فرزند تولید شوند. نکته کلیدی آن است که فرآیند انتخاب حریصانه نیست و اعضای نامرغوب نیز شانسی برای انتخاب دارند، بهترین عضو تضمینی برای انتخاب نیست و بدترین عضو در جمعیت استثنا نیست.

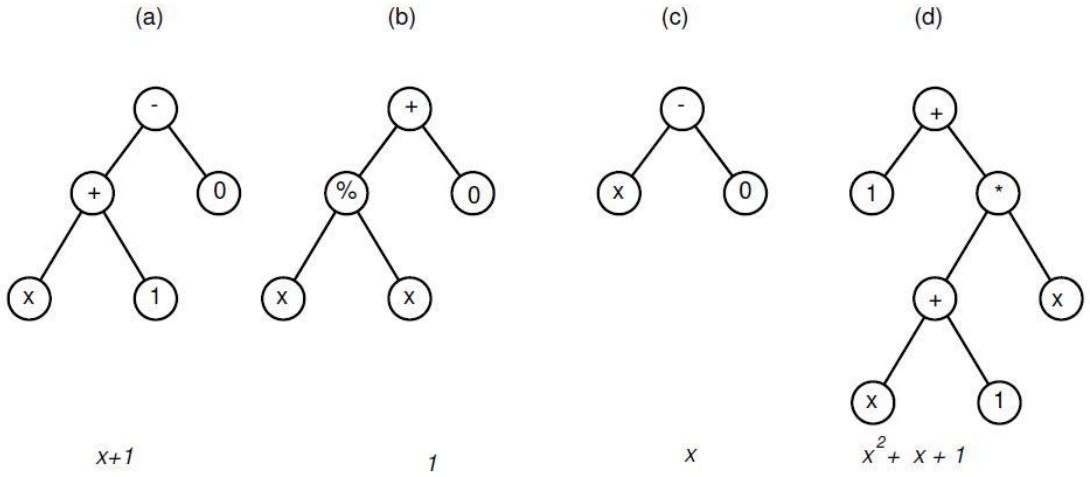
این مثال با عملیات تکثیر شروع می‌شود، به دلیل آنکه عضو اول (شکل ۱۰ a) بیشترین تناسب را در جمعیت دارد، این عضو شانس زیادی برای شرکت عملیات ژنتیک دارد. بنابراین فرض کنید این عضو برای تکثیر انتخاب می‌شود و آن همان کپی کردن در نسل بعدی (نسل 1) است که در شکل ۱۲ a نشان داده شده است.

عملیات بعدی عملیات جهش است. با توجه به آنکه انتخاب‌ها احتمالی است، عضو سوم جمعیت (شکل ۱۰ c) برای این عملیات انتخاب می‌شود. یکی از سه گره این عضو برای انجام عملیات جهش به صورت تصادفی باید انتخاب شود، پایانه 2 انتخاب می‌شود. این برنامه به صورت تصادفی جهش می‌یابد و آن برابر است با حذف زیردرخت کامل در نقطه برگزیده (پایانه 2) و درج یک زیردرخت که به صورت تصادفی ساخته شده است. زیردرخت ساخته شده برابر با x تقسیم بر x است با استفاده

از تقسیم محافظت شده که در شکل ۱۲ b نشان داده شده است. بنابراین این جهش مخصوص برابر است با تغییر ثابت 2 به ثابت 1، که برازندگی آن از 17.98 به 11.0 بهبود پیدا می‌کند.

در نهایت از عملیات بازترکیبی برای تولید دو عضو نهایی نسل جدید استفاده می‌شود. به دلیل آنکه اعضای اول و دوم در نسل 0 بیشترین تناسب را دارند، این دو برای شرکت در عملیات بازترکیبی محتمل‌تر هستند. ولی در این مثال برای اولین بازترکیبی از دو جفت درخت تناسب بالا (شکل ۱۰ a) و درخت تناسب پایین (شکل ۱۰ d) استفاده می‌شود. یک نقطه از والد اول به نام تابع + و یک نقطه از والد دوم به نام سمت چپ‌ترین x به عنوان نقطه بازترکیبی انتخاب می‌شود. عملیات بازترکیبی روی دو والد انجام می‌شود و نتیجه برابر است با عبارت x (شکل ۱۲ c).

برای عملیات بازترکیبی دوم دو عضوی که بیشترین تناسب را دارند به عنوان والد انتخاب می‌شوند. عضو شکل ۱۰ b والد اول و عضو ۱۰ a والد دوم. به صورت تصادفی سمت چپ‌ترین x در شکل ۱۰ b و تابع + در شکل ۱۰ a به عنوان نقاط بازترکیبی انتخاب می‌شوند. حال فرزند (شکل ۱۲ d) برابر است با عبارت x^2+x+1 که دارای برازندگی صفر است.



شکل ۱۲: جمعیت نسل 1 (بعد از عملیات‌های تکثیر، جهش و بازترکیبی)

به دلیل آنکه برازندگی عضو شکل ۱۲ d زیر 1.0 است، شرط خاتمه برای اجرا کافی است و اجرا به طور خودکار خاتمه می‌یابد. بنابراین، بهترین عضو تا به حال شکل ۱۲ d است و به عنوان نتیجه اجرا طراحی می‌شود.