نوع مقاله: پژوهشی

دریافت مقاله: ۱۴۰۰/۰۵/۱۵
پذیرش مقاله: ۱۴۰۰/۱۲/۱۰

ساخت توابع ریاضی و منطقی بهینه با استفاده از الگوریتم ژنتیک و کاربرد آن در تقریب نتایج

علیرضا رمضانی
کارشناس مهندسی کامپیوتر، دانشکده فنی فومن – دانشکدگان فنی دانشگاه تهران – فومن – ایران
پست الکترونیکی: alireza1376@gmail.com

عاطفه حسن‌زاده*
استادیار، دانشکده فنی فومن – دانشکدگان فنی دانشگاه تهران – فومن – ایران
پست الکترونیکی: hasanzadeh.a@ut.ac.ir

علی نقاش اسدی
مدرس، دانشکده فنی فومن – دانشکدگان فنی دانشگاه تهران – فومن – ایران
پست الکترونیکی: naghashasadi@guest.ac.ir

چکیده

در این مقاله، با استفاده از الگوریتم ژنتیک، به‌عنوان یک ابزار اصلی در بهینه‌سازی، روشی ارائه شده است که می‌تواند توابع ریاضی و منطقی پیچیده را بهینه کند. در اکثر روش‌های موجود، از الگوریتم‌های ژنتیک برای بهینه‌سازی مدارهای منطقی استفاده شده است ولی در روش ارائه‌شده در این مقاله، امکان تقریب نتایج توابع ریاضی گسسته و پیوسته نیز فراهم شده است. با استفاده از ساختار داده درخت، تقریباً امکان نمایش همه توابع به‌صورت سلسله مراتبی وجود خواهد داشت. از آنجایی که توابع ریاضی و منطقی می‌توانند پیچیده باشند و درخت آن‌ها بسیار بزرگ شود، بهینه‌سازی آن‌ها می‌تواند مزایای زیادی به همراه داشته باشد. از جمله مزایای بهینه‌سازی توابع می‌توان به کاهش زمان محاسباتی، کاهش هزینه‌های پیاده‌سازی و غیره اشاره کرد. با این حال، بهینه‌سازی یا ساده‌سازی توابع با کاهش تعداد عملگرها و متغیرها باعث می‌شود که نتایج به‌دست آمده از آن‌ها، دقت اولیه را نداشته باشند. اگر نتایج به‌دست آمده از توابع بهینه‌شده با توابع اصلی، تفاوت زیادی نداشته باشد، می‌توان به‌جای توابع اصلی، از توابع بهینه‌شده استفاده کرد. در روش ارائه‌شده در این مقاله، که با ارائه مثال‌هایی مورد ارزیابی قرار گرفته است، توابع بهینه با استفاده از الگوریتم ژنتیک به‌دست می‌آیند که نتایج حاصل از آن‌ها، تفاوت چندانی با نتایج توابع اصلی ندارند.

واژه‌های کلیدی: تقریب توابع ریاضی و منطقی، الگوریتم ژنتیک، ساختار داده درخت، بهینه‌سازی.

۱- مقدمه

تعداد عملگرها و متغیرها در توابع ریاضی و منطقی، پیچیدگی آن‌ها را مشخص می‌کند. به‌عبارت دیگر، هر چه تعداد عملگرها و متغیرها در توابع ریاضی و منطقی بیشتر باشد، پیچیدگی آن‌ها بیشتر شده و برای حل آن‌ها هزینه‌های بیشتری از نظر زمان و حافظه باید مصرف شود. بنابراین کاهش عملگرها و متغیرهای توابع ریاضی و منطقی، به‌عنوان یک هدف در نظر گرفته می‌شود. روش‌های بسیاری برای

* نویسندهٔ مسئول

این منظور ارائه شـــده است که اساس آن‌ها، کم کردن تعداد عملگرها اســـت. علاوه بر کاهش تعـــداد عملگرها به‌صورت مســـتقیم، افزایش تعداد ســـطوح توابع ریاضی نیز می‌تواند منجر به کاهش تعداد عملگرها شود. البته در بعضی موارد، در استفاده از برخی عملگرها محدودیت‌هایی وضع می‌شود و باید توابع ریاضی معادلی بدون اســـتفاده از آن عملگرها پیشـــنهاد کرد. برای ایـــن منظور می‌تـــوان از الگوریتم‌های تکاملـــی[1] به‌ویژه الگوریتم‌های ژنتیک[2] بـــرای طراحی توابع ریاضی اســـتفاده کرد تا هم پیچیدگـــی توابع ریاضی تا حد امکان کاهش یافته و محدودیت‌های موجود اعمال شوند و هم نتایج به‌دست آمده با نتایج مورد انتظار مطابقت داشته باشند.

در این مقاله با اســـتفاده از الگوریتم ژنتیک روشی ارائه شـــده است که به وسیله آن می‌توان توابع ریاضی و منطقی را با کاهش تعداد عملگرها و متغیرها بهینه‌سازی کرد. برای این منظور، ابتدا جمعیت اولیه‌ای از توابع ساده ایجاد می‌شود. این توابع با ساختار سلسله مراتبی درخت نگهداری می‌شوند. ســـپس با مشـــخص کردن سطح تناسب و شـــرایط توقف الگوریتم ژنتیک، هر یک از روابط ایجادشـــده مورد بررسی قرار گرفته و متناســـب با سیاست‌های تعریف‌شده، بعضی از آن‌ها انتخاب و با یکدیگر ترکیب شـــده و جهش می‌یابند. روابطی که مجدد ایجاد می‌شوند از نظر سطح تناسب با تابع اصلی مورد بررســـی قرار می‌گیرند. این چرخه تا زمانی که شرایط توقف الگوریتم ژنتیک فراهم شود، ادامه می‌یابد.

در ایـــن مقاله ابتدا بـــه معرفی مفاهیـــم اصلی همچون الگوریتم ژنتیک، نمایش توابع ریاضی و دروازه[3]های منطقی پرداخته می‌شـــود. سپس روش پیشنهادی با بیان فرضیات اولیه، ارزیابی سطح تناسب و بررسی شرایط توقف الگوریتم، روش‌های انتخـــاب، تقاطع (ترکیب) و جهـــش مورد بیان و بررســـی قرار می‌گیرد. لازم به ذکر است که در این مقاله از زبان برنامه‌نویســـی پایتون ۳/۶ و کتابخانه Graphviz برای پیاده‌سازی روش پیشـــنهادی و نمایش دادن ساختار داده درخت استفاده شده است [۱].

1- Evolutionary Algorithm
2- Genetic Algorithm
3- Gate

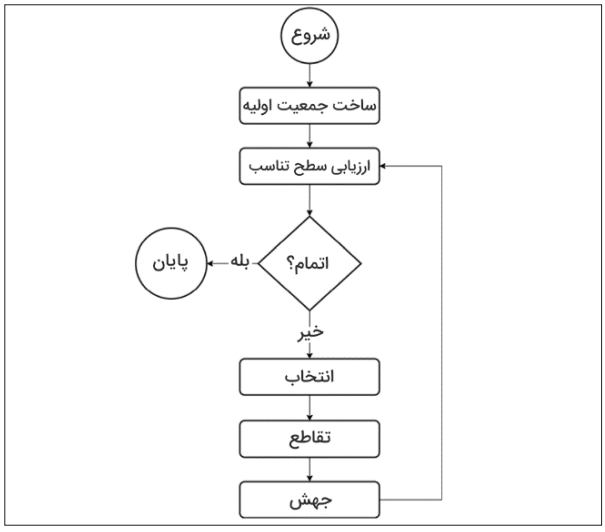
۲- مفاهیم اصلی

۱،۲. الگوریتم ژنتیک

الگوریتم ژنتیک یک روش بهینه‌سازی با الهام از طبیعت جانداران بر مبنای نظریه داروین است. این الگوریتم با اقتباس از علم ژنتیک و مبتنی بر تکرار اســـت. براســـاس الگوریتم ژنتیک از خصوصیت‌های نسل حاضر برای تولید نسل جدید استفاده می‌شـــود. این الگوریتم جستجو را در محیط پاسخ انجـــام می‌دهد [۲]. مطابق با شـــکل ۱، گام اول در الگوریتم ژنتیک، ساخت جمعیت اولیه به‌صورت تصادفی یا انتخابی است. در ادامه برای هر نمونه از جمعیت، مقدار تابع هدف به منظور ارزیابی سطح تناسب محاسبه شده و براساس مقدار تابع هدف، برازندگی[4] هر نمونه ارزیابی می‌شود. در ادامه با توجه به پارامتر برازندگی، نمونه‌هایی از جمعیت انتخاب[5] شده (در علم ژنتیک به آن‌ها والدین می‌گویند) و سپس عملگرهای ژنتیکی شامل تقاطع یا ترکیب[6] و جهش[7] بر روی نمونه‌های انتخاب‌شده، اعمال می‌شوند و جمعیت جدیدی تولید می‌گردد. پس از آن، نمونه‌های جدید جایگزین نمونه‌های پیشین شده و این چرخه ادامه پیدا می‌کند. به‌طور معمول با تولید نمونه‌های جدید، پارامتر برازندگی در جمعیت جدید بیشـــتر شده و در نتیجه جمعیت نسل به نسل بهبود می‌یابد. این چرخه تا زمانی که مقدار تابع هدف به ســـطح تناسب مورد نظر نرسد، ادامه می‌یابد. مولفه‌های اصلی در الگوریتم ژنتیک عبارتند از:

- **برازندگی**: یک نمونه با استفاده از مقدار تابع هدف (تابعی که باید بهینه شـــود) محاسبه می‌شـــود. پارامتر برازندگی کیفیت نمونه را مشخص نموده و احتمال مشارکت در تولید نمونه‌های جدید را افزایش می‌دهد.
- **تقاطع یا ترکیب**: مهم‌ترین عملگر الگوریتم ژنتیک است. در این بخش با اســـتفاده از نمونه‌هـــای موجود در جمعیت حاضر (والدین در علم ژنتیک)، نمونه‌های جدید (فرزندان در علم ژنتیک) تولید می‌شوند. اعمال ترکیب در الگوریتم ژنتیک موجب کاهش پراکندگی در نمونه‌ها می‌شود.

4- Fit
5- Selection
6- Crossover
7- Mutation



شکل ۱: فرآیند الگوریتم ژنتیک

● **جهش**: در نمونه‌های جدید تولیدشده همواره امکان ایجاد تغییراتی وجود دارد (در علم ژنتیک، تغییر در کروموزوم‌ها در نتیجه جهش ژنتیکی است). این تغییرات منجر به تغییر یک سری از نمونه‌ها در جمعیت جدید می‌شود، یعنی بخش کوچکی از جمعیت جدید با استفاده از جهش تغییر می‌یابند.

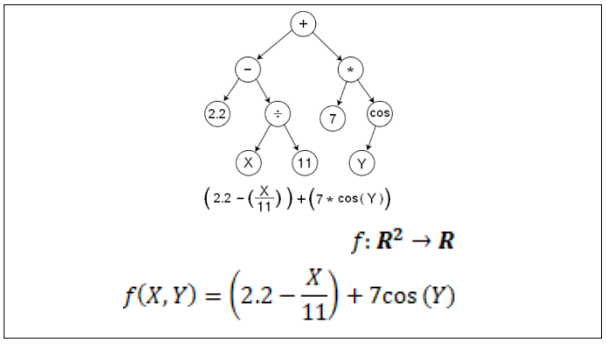
۲،۲. نمایش توابع ریاضی و منطقی

بسیاری از پدیده‌های درون طبیعت را می‌توان با توابع ریاضی یا منطقی نشان داد. اگرچه نمایش توابع ریاضی و منطقی به‌صورت معمول بسیار کمک‌کننده هستند، ولی این نوع نمایش در توضیح جنبه‌های ساختاری و سلسله مراتبی توابع از غنای کافی برخوردار نیستند. مطابق با شکل ۲، نمایش توابع با ساختار داده‌ای درخت می‌تواند این محدودیت‌ها را برطرف کند. در این ساختار داده، گره‌های انتهایی (برگ‌ها) متغیرهای روابط را نشان می‌دهند و گره‌های میانی، عملگرهای اعمال شده بر روی متغیرها هستند.

۳،۲. دروازه‌های منطقی

دروازه‌های منطقی یک دستگاه محاسباتی برای بیان توابع بولی[8] هستند. سه دروازهٔ منطقی اصلی عبارت‌اند از: OR، AND و NOT. این سه دروازه می‌توانند مجموعه کاملی از تابع را ایجاد کنند و برای تولید سایر دروازه‌های منطقی با ترکیب و ترتیب مناسب کافی هستند. هر دروازهٔ منطقی

8- Boolean



شکل ۲: نمایش سلسله مراتبی یک تابع با استفاده از ساختار داده درخت

با یک جدول درستی همراه است، که شامل تمام نگاشت‌ها (ورودی/خروجی) است [۳]. جدول درستی تعدادی از دروازه‌های منطقی در جدول ۱ نشان داده شده است.

۴،۲. تقریب توابع ریاضی و منطقی

توابع ریاضی و منطقی می‌توانند پیچیده باشند و درخت آن‌ها بسیار بزرگ شود. تعداد عملگرها و متغیرها در توابع ریاضی و منطقی، پیچیدگی آن‌ها را مشخص می‌کند. به‌عبارت دیگر، هر چه تعداد عملگرها و متغیرها در توابع ریاضی و منطقی بیشتر باشد، پیچیدگی آن‌ها بیشتر شده و برای حل آن‌ها هزینه‌های بیشتری از نظر زمان و حافظه باید مصرف شود. بهینه‌سازی توابع می‌تواند مزایای زیادی به همراه داشته باشد. از جمله مزایای بهینه‌سازی توابع می‌توان به کاهش زمان محاسباتی، کاهش هزینه‌های پیاده‌سازی و غیره اشاره کرد. با این حال، بهینه‌سازی یا ساده‌سازی توابع با کاهش تعداد عملگرها و متغیرهای باعث می‌شود که نتایج به‌دست آمده از آن‌ها، دقت اولیه را نداشته باشند. مطابق با جدول ۲، اگر نتایج به‌دست آمده از توابع بهینه‌شده با توابع اصلی، تفاوت زیادی نداشته باشد، می‌توان به‌جای توابع اصلی، از توابع بهینه‌شده استفاده کرد.

۳- کارهای مرتبط

اکثر روش‌های موجود از الگوریتم‌های ژنتیک برای بهینه‌سازی مدارهای منطقی استفاده می‌کنند و برای بهینه‌سازی توابع ریاضی کاربردی ندارند. در ادامه به بعضی از این مقالات اشاره شده است. در مقاله [٤]، یک

جدول ۱: جدول درستی دروازه‌های AND، OR، NOT، NAND و XOR

X	Y	AND	OR	NOT(Y)	NAND	XOR
0	0	0	0	1	1	0
0	1	0	1	0	0	1
1	0	0	1		0	1
1	1	1	1		0	0

روش جدید برای طراحی تکاملی مدارهای منطقی ترکیبی با استفاده از الگوریتم ژنتیک ارائه شده است. در این مقاله فقط از همتافتگر[۹] دو در یک به‌عنوان عملگر استفاده شده است؛ زیرا این مقاله بیان می‌کند که استفاده از یک نوع عملگر می‌تواند هزینه تولید مدارهای منطقی را کاهش دهد. در مقاله [۵] نیز، با استفاده از همتافتگر دو در یک و الگوریتم ژنتیک، پیچیدگی مدارهای منطقی و تاخیر ناشی از آن‌ها کاهش یافته است. در مقاله [٦] با استفاده از الگوریتم‌های تکاملی، روشی برای طراحی و بهینه‌سازی مدارهای منطقی ترتیبی ارائه شده است. در این مقاله، با محدود کردن فضای جستجو، متوسط تعداد نسل‌های بعدی را کاهش می‌دهد. در مقاله [۷]، دو روش جستجوی مبتنی بر جمعیت برای سنترکردن مدارهای برگشت‌پذیر ارائه شده است. نتایج این مقاله نشان می‌دهد که روش هزینه و زمان سنتز کاهش یافته است. برخلاف مقالات معرفی‌شده، در مقاله ما روش بهینه‌سازی با استفاده از الگوریتم ژنتیک ارائه می‌شود که علاوه بر توابع منطقی، می‌توان از آن برای بهینه‌سازی توابع ریاضی گسسته و پیوسته نیز استفاده کرد.

۴- روش پیشنهادی

روش پیشنهادی در این مقاله، مشابه با سایر روش‌های مرتبط با الگوریتم‌های ژنتیک، از بخش‌های مشخصی تشکیل شده است. مطابق با شکل ۱، این بخش‌ها شامل ساخت جمعیت اولیه، ارزیابی سطح تناسب، شرط اتمام الگوریتم، الگوریتم انتخاب نمونه‌ها از جمعیت، الگوریتم تقاطع (ترکیب) نمونه‌های انتخاب‌شده از جمعیت، و الگوریتم جهش نمونه جدید است. در ادامه هر یک از این بخش‌ها معرفی می‌شوند.

9- multiplexer

جدول ۲: مثالی از بهینه‌سازی توابع ریاضی و منطقی

نوع تابع	تابع اصلی	تابع بهینه
ریاضی	$f(x,y) = x^2 + 2xy + y^2$	$f(x,y) = (x+y)^2$
منطقی	$f(x,y) = x.(x+\bar{y})$	$f(x,y) = x$

۱،٤. ساخت جمعیت اولیه

برای ساخت جمعیت اولیه در الگوریتم ژنتیک، ابتدا تعداد متغیرها (ورودی‌ها)، عملگرهای قابل استفاده و حداکثر عمق درخت مشخص می‌شود. سپس به‌صورت تصادفی، تعدادی تابع با مشخصات گفته شده تولید می‌شود. در این مقاله از ساختار داده درختی برای نمایش توابع ریاضی و منطقی استفاده شده است که منجر به سهولت در برنامه‌نویسی می‌شود. جمعیت اولیه مجموعه‌ای از درخت‌های تصادفی است که تنها با در نظر گرفتن محدودیت‌هایی در زمینه تعداد متغیرها، عملگرهای قابل استفاده و عمق درخت ایجاد می‌شوند. این درخت‌ها، راه‌حل‌های اولیه برای رسیدن به راه‌حل بهینه هستند.

از شبه‌کد شکل ۳ برای ساخت جمعیت اولیه در روش پیشنهادی استفاده شده است. در این شبه‌کد، Operation_Set و Terminal_Set به ترتیب مجموعه‌ای از عملگرها و متغیرهای قابل استفاده هستند؛ همچنین Depth_Range محدوده عمق درخت را مشخص می‌کند. Initial_Population_Set مجموعه درخت‌های اولیه تولیدشده بوده و Initial_Population_Length حداکثر تعداد درخت‌ها در جمعیت اولیه است.

برای ایجاد درخت به‌صورت تصادفی، از توابع موجود در کتابخانه Graphviz استفاده شده است. برای این‌که درخت‌های تولیدشده معادل توابع ریاضی یا منطقی باشند، در تابع generate_random_tree این طور در نظر گرفته شده است که برگ‌های درخت متغیرها بوده و گره‌های میانی، عملگرها باشند. در پارامترهای ورودی این تابع، ما تنها مشخص می‌کنیم که برگ‌ها و عملگرها چه مقادیری می‌توانند داشته باشند و عمق درخت حداکثر تا چه مقداری می‌تواند پایین برود. برای مثال اگر مجموعه متغیرهای ورودی شامل

```
Operation_Set is a set of operators that can be used in mathematical functions
Terminal_Set is a set of inputs in mathematical functions
Depth_Range is a depth range of trees
Initialize_Population_Length is the length of set of initial solutions
Initialize_Population_Set is a set of initial solutions

def generate_population():
  i=0
  Initialize_Population_Set={}
  while (i<Initialize_Population_Length)
    Initialize_Population_Set.add(generate_random_tree(Operation_Set, Terminal_Set, Depth_Range))
    i++
```

شکل ۳: شبه‌کد استفاده‌شده برای تولید جمعیت اولیه

x ،y و z باشد و مجموعه عملگرها شامل جمع، تفریق، ضرب و تقسیم باشـــد، و عمق درخت حداکثر بتواند سه باشد، این تابع برای مثال می‌تواند درخت‌هایی که معادل توابع ریاضی x+(y+z)، (y-x)*x، x/(z*y) و غیره هستند را تولید کند.

۲،٤. ارزیابی سطح تناسب و شرط اتمام الگوریتم

از شبه‌کد شـــکل ٤ برای محاســـبه برازندگی هر یک از درخت‌های موجود در جمعیت، کـــه به‌عنوان راه‌حل‌های ما برای بهینه‌سازی تابع ریاضی یا منطقی اصلی در نظر گرفته می‌شـــوند، استفاده شده است. در این شبه‌کد، نتایج به‌دست آمده از هر یک از درخت‌ها با نتایج به‌دست آمده از تابع اصلی مقایسه شده و مقدار برازندگی راه‌حل در متغیر average_fit-ness قرار می‌گیرد. برای این منظور، ابتدا مجموعه مقادیری که متغیرهای ورودی می‌توانند داشته باشند، مشخص شده و نتایج تابع ریاضی یا منطقی اصلی در آن مجموعه مقادیر ورودی محاسبه می‌شود. سپس به ازای هر درخت موجود در جمعیت، که معادل یک تابع ریاضی یا منطقی است، نتایج تابع در هر یک از مجموعه مقادیر ورودی محاسبه شده و با نتایج تابع اصلی مقایسه می‌شود. در صورتی که به ازای هر مقدار ورودی، نتیجه تابع تولیدشده با تابع اصلی برابر باشد، مقـــدار متغیـــر average_fitness افزایش می‌یابد. البته برای نتایج توابع ریاضی می‌توان یک درصد خطایی در نظر گرفت که با متغیر threshold مشـــخص می‌شود. به‌عبارت دیگر، اگر نتیجه تابع بهینه برای یک مقدار ورودی در محدوده حد آستانه نتیجه تابع اصلی در همان مقدار ورودی باشد، نتایج دو تابع برابر در نظر گرفته می‌شوند.

اگر حداقل یکی از average_fitnessهای محاسبه‌شـــده برای درخت‌هـــای موجود در جمعیت برابر با ۱۰۰ باشـــد، به این معنی اســـت که راه‌حل تولیدشده برازندگی یا تناسب ۱۰۰درصدی با مقادیر تابع اصلـــی دارد. بنابراین الگوریتم خاتمه خواهد یافت و راه‌حل مورد نظر به‌عنوان راه‌حل قطعی نمایش داده می‌شود.

۳،٤. الگوریتم انتخاب نمونه‌ها از جمعیت

در این مقالـــه از الگوریتم انتخاب رقابتـــی و ترکیب آن با الگوریتم نخبه‌گرایی فعال برای تحریک بیشـــتر پیشرفت نســـل بعدی استفاده شـــده اســـت. برای این منظور ابتدا N راه‌حـــل به‌طور تصادفی از بین جمعیت انتخاب می‌شـــود و در یک محیط موقت (به‌عنوان پارامتر دیگر الگوریتم) ذخیره می‌شوند. در این محیط موقت، راه‌حل‌هایی که دارای بالاترین سطح تناسب هستند برای انجام فرآیند تقاطع یا ترکیب انتخاب می‌شوند. این روش احتمال تولید راه‌حل‌های جدید که دارای امتیاز بالاتر باشند را بیشتر می‌کند.

٤،٤. الگوریتم تقاطع یا ترکیب نمونه‌های انتخاب‌شده از جمعیت و جهش آن‌ها

پیچیده‌ترین قسمت الگوریتم ژنتیک، سازوکارهای تقاطع یا ترکیب است. بسیاری از عملگرهای ژنتیکی که در این قضیه درگیر هســـتند برای چنین اقداماتی به‌صورت مجزا طراحی می‌شـــوند. از آنجا که در این مقاله از ســـاختار داده درختی اســـتفاده شده است، امکان استفاده از چندین الگوریتم برای ادغام، هرس و اصلاح آن‌ها با ســـطح پیچیدگی $O(n\log n)$ فراهم شده است.

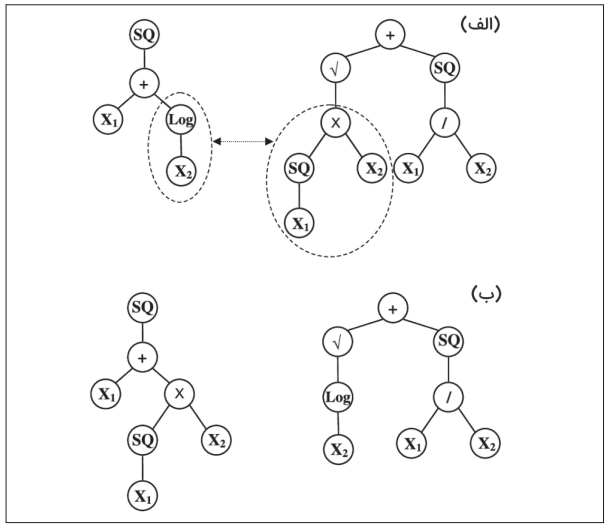
برای انجام این فرآیند، ابتـــدا دو راه‌حل از محیط موقت که دارای بالاترین ســـطح تناســـب هســـتند با نام‌های P_1 و

```
Value_Terminal_Set is a set of values of each input (Terminal)
Org_Result_Set is a set of results of orginal function
Gen_Result_Set is a set of results of each generated function
threshold is an acceptable error range

defcalculate_fitness_for_opulation(population: list):
   for individual in population:
      calculate_fitness_for_individual(individual)


def calculate_fitness_for_individual(tree):
    count=0, total=0
for each terminal in Terminal_Set select one value from Value_Terminal_Set(terminal) as VT:
        total++
        calculate Org_Result_Set(VT)
        calculate Gen_Result_Set(VT)
  if Org_Result_Set(VT) is equal to Gen_Result_Set(VT)±threshold
          count++
   average_fitness=(count/total)*100
```

شکل ۴: شبه‌کد ارزیابی سطح تناسب هر نمونه از جمعیت

P_2 انتخاب می‌شـــوند. سپس از طریق یک فرآیند تصادفی، دو گـــره ســـازگار N_1 و N_2 به‌عنوان نقطـــه تلاقی تولید، علامت‌گذاری شده و زیردرخت آن‌ها بین درختان P_1 و P_2 جابجا می‌شوند. شرط سازگاری دو درخت فرعی این است که در صورت جابجایی آن‌ها، درخت حاصل معتبر بوده و متناسب با پیش‌فرض‌های مسئله باشد. به‌عبارت دیگر، پس از جابجایی درختان فرعی، درختان اصلی باید حتماً یک تابع ریاضی یا منطقی مناسب باشند. وقتی سازگاری گره‌ها تأیید شـــد، می‌توان ترکیب را آغاز نمود. این الگوریتم دو یا چند والد را می‌گیرد و ســـپس درختان فرعی را که از N_1 و N_2 شروع شده‌اند، ترکیب می‌کند. شکل ۵، مثالی از این روند را به تصویر می‌کشد.

پس از هر فرآیند تقاطع یا ترکیب، نســـل جدید با احتمال بسیار کم (حدود یک درصد) جهش داده می‌شود. انجام این کار به این دلیل است که راه‌حل‌های جدیدتری ایجاد شود و امکان رسیدن به راه‌حل‌ها با بالاترین سطح تناسب امکان‌پذیر باشد. اگر فرآیند جهش را در نظر نگیریم، امکان دارد بعضی



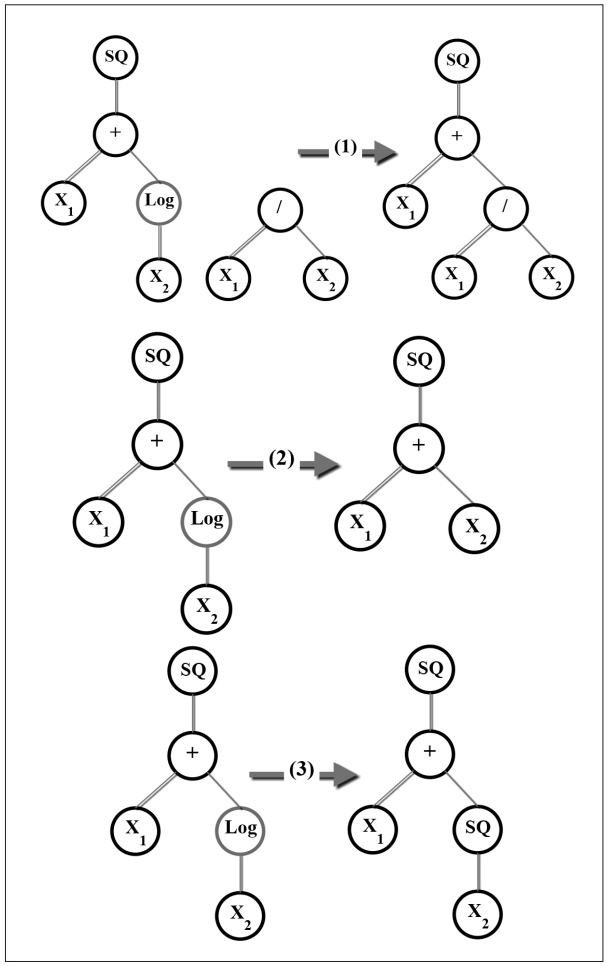
شکل ۵: استفاده از الگوریتم تقاطع یا ترکیب بر روی دو والد و تولید دو فرزند متمایز

از راه‌حل‌ها هرگز تولید نشـــده و مورد ارزیابی قرار نگیرند. راه‌حل جهش‌یافته نیز باید مطابق با پیش‌فرض‌های مسئله، معتبر باشـــد. به‌عبـــارت دیگر، پس از انجـــام فرآیند جهش، درخت جهش‌یافته نیز باید یک تابع ریاضی یا منطقی مناسب باشد. مطابق با شـــکل ٦، برای انجام فرآیند جهش، یک گره تصادفی انتخاب‌شده و یکی از سه فرآیند جهش زیر بر روی آن انجام می‌شود:

۱. افـــزودن: تولید یک درخت تصادفی و قـــرار دادن آن به جای گره تصادفی.
۲. حذف کردن: حذف کردن کل زیرشاخه تصادفی و قرار دادن گره داده به جای آن.
۳. جابه‌جا کردن: تغییر عملگر گره تصادفی

۵- پیاده‌سازی و ارزیابی روش پیشنهادی

در این بخش برای ارزیابی روش پیشـــنهادی، دو مسئله بـــرای بهینه‌ســـازی توابع ریاضی و منطقی مطرح شـــده و پیاده‌سازی می‌شود و خروجی حاصل از آن‌ها مورد ارزیابی قرار می‌گیرد. در مسئله اول (بهینه‌سازی توابع منطقی)، یک محدودیت در اســـتفاده از دروازهٔ XOR مطرح شده و سعی می‌شود فقط با استفاده از دروازهٔ NAND، تابع منطقی مشابه دروازهٔ XOR تولید کرد. در مســـئله دوم (بهینه‌سازی توابع ریاضی)، یک تابع ریاضی پیچیده با تعداد زیادی عملگر مطرح



شکل ۶: فرآیندهای جهش درخت (۱) افزودن (۲) حذف کردن (۳) جابجا کردن

شـــده و سعی می‌شود این تابع را با یک تابع معادل با تعداد کمتری عملگر، پیاده‌سازی کرد.

۱,۵. بهینه‌سازی توابع منطقی

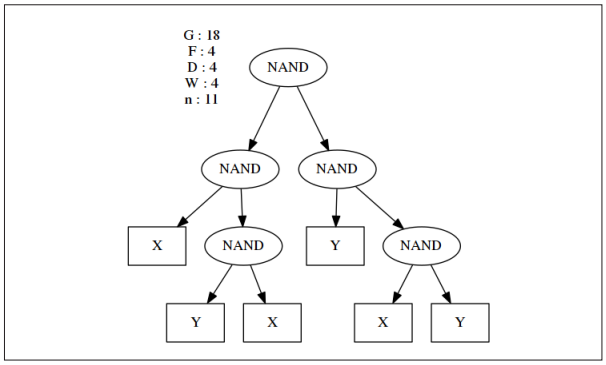
مسئله مطرح‌شـــده برای این بخش، تعریف تابعی برای پیاده‌ســـازی دروازهٔ XOR، فقط با اســـتفاده از دروازه‌های NAND اســـت. در جدول ۱، جدول درســـتی ایـــن دروازه‌ها ارائه شده است. به‌عبارت دیگر، برای حل مسئله XOR با دو ورودی و یک خروجی، تابعی از میان نمونه‌های موجود در جمعیت جســـتجو می‌شود که فقط از دروازهٔ NAND تشکیل شده باشد و جدول درستی دروازهٔ XOR را تأمین کند.

معیار اصلی برازندگی یا تناسب در این مسئله، رسیدن راه‌حل به جدول درســـتی دروازهٔ XOR است. برای ارزیابی تناسب، بررسی میشود که یک راه‌حل به چه میزان توانسته اســـت جدول درســـتی دروازهٔ XOR را تامین کند. بنابراین در مســـئله XOR با دو ورودی، هر چهار ترکیب یک و صفر مورد بررســـی قرار می‌گیرند تا مشـــاهده شـــود چه تعداد از ایـــن ترکیب‌ها با خروجـــی مربوطه مطابقـــت دارند. در صورتـــی که هیچ یک از راه‌حل‌های مورد بررســـی، به‌طور کامـــل جدول درســـتی دروازهٔ XOR را تامین نکنند، از میان راه‌حل‌های موجود، راه‌حلی که ســـطح تناسب بالاتری دارد، برای تولید راه‌حل‌های بعدی انتخاب می‌شود. اگر یک راه‌حل به سطح تناســـب ۱۰۰درصدی برسد، به این معنی است که یک راه‌حل برای دروازهٔ XOR یافت شـــده است که صرفاً از دروازهٔ NAND استفاده کرده است. در این مسئله، مجموعه متغیرهای ورودی شـــامل X و Y بوده و مجموعه عملگرها فقط شـــامل دروازهٔ NAND است. حداکثر عمق درخت برابر با چهار در نظر گرفته شـــده است و مقدار حد آستانه برابر با صفر است، چون مقادیر متغیرهای منطقی فقط می‌توانند صفر یا یک باشند.

در مرحله پیاده‌سازی مسئله دروازهٔ XOR، تنها با گذراندن ۱۸ نسل امکان رسیدن به یک راه‌حل قطعی فراهم شده است. بدون هیچ‌گونه تنظیم دقیـــق و تعدیل بیش از حد پارامترها، مداری از دروازهٔ XOR کشـــف شده که فقط از چهار دروازهٔ NAND ساخته شده است. نتایج به‌صورت شکل ۷ نشان داده شده است. در شکل ۷، شاهد ٤ دروازهٔ NAND (۲ دروازه در عمق ســـوم در واقع یک دروازه هستند که ورودی مشابهی به لایه بالایی خود وارد می‌کنند) هســـتیم که می‌توانند با این چینش ورودی‌ها تمام ٤ پاســـخ مدنظر برای دروازهٔ XOR را تولید کنند. شکل ۷ تنها یکی از فرزندانی است که در نسل ۱۸ الگوریتم ژنتیک تولید شده اســـت اما تنها این فرزند در این نسل توانست به سطح تناسب ۱۰۰درصدی برسد.

۲,۵. بهینه‌سازی توابع ریاضی

مســـئله مطرح‌شـــده برای این بخش، بهنیه‌ســـازی تابع ریاضی $x^2 + 2xy + y^2$ به تابعی اســـت که تعداد عملگر کمتری داشته باشـــد. همان‌طور که مشاهده می شود، تابع اصلی دارای شـــش عملگر است (دو عملگر توان، دو جمع و



شکل ۷: خروجی گرفته‌شده با استفاده از ساختار داده درخت توسط کتابخانه Graphviz

دو ضرب). هدف این است که این تابع را با یک تابع معادل با تعداد کمتری عملگر، تبدیل کنیم.

معیار اصلی برازندگی یا تناسب در این مسئله، رسیدن راه‌حل به نتایج تابع اصلی در مقادیر ورودی مشـــابه است. از آنجایی که متغیرهای ریاضی می‌توانند مقادیر صحیح و اعشاری و همچنین مثبت و منفی داشته باشند، در این مسئله با دو ورودی، چندین ترکیب از مقادیر ممکن مورد بررسی قرار می‌گیرند تا مشـــاهده شود چه تعداد از این ترکیب‌ها با خروجی تابع اصلی مطابقت دارند. در صورتی که هیچ یک از راه‌حل‌های مورد بررسی، به‌طور کامل خروجی تابع اصلی را تامین نکنند، از میان راه‌حل‌های موجود، راه‌حلی که سطح تناســـب بالاتری دارد، برای تولید راه‌حل‌های بعدی انتخاب می‌شود. اگر یک راه‌حل به سطح تناسب ۱۰۰درصدی برسد، به این معنی اســـت که یک راه‌حل برای تابع اصلی پیدا شده است. در این مسئله، مجموعه متغیرهای ورودی شامل X و Y بوده و مجموعه عملگرها شامل جمع، تفریق، ضرب و تقسیم و توان است. حداکثر عمق درخت برابر با چهار در نظر گرفته شده است و مقدار حد آستانه برابر با صفر است.

در مرحله پیاده‌ســـازی مسئله بهینه‌سازی تابع ریاضی $x^2 + 2xy + y^2$، تنها با گذراندن ۳ نسل امکان رسیدن به یک راه‌حل قطعی فراهم شده است. بدون هیچگونه تنظیم دقیق و تعدیل بیش از حد پارامترها، تابع $(x + y)^2$ کشف شده که فقط از دو عملگر ریاضی ساخته شده و نتایج آن کاملاً با تابع اصلی مطابقت دارد.

۶-نتیجه‌گیری

اکثر روش‌هـــای موجود از الگوریتم‌هـــای ژنتیک برای بهینه‌ســـازی مدارهـــای منطقی اســـتفاده می‌کننـــد و برای بهینه‌ســـازی توابع ریاضی کاربردی ندارنـــد. در این مقاله نشان داده شد که چگونه الگوریتم ژنتیک می‌تواند برای تولید توابع ریاضی و منطقی بهینه مورد اســـتفاده قرار گیرد. در این روش، امکان در نظر گرفتن انواع مختلفی از محدودیت‌ها از جمله تعداد متغیرها، تعداد عملگرها و حداکثر عمق توابع وجود دارد. نمایش توابع به‌صورت ساختار داده درخت برای یافتن توابع بهینه بسیار با اهمیت است. اگرچه توابع با ابعاد و پیچیدگی بالاتر برای تقریب کمی وقت‌گیر هستند، اما می‌توان با انجام محاسبات موازی، سرعت الگوریتم را تقویت نمود. به‌عنـــوان کار آینده می‌توان به این موضوع اشـــاره کرد که به‌جای فرآیندهای تصادفی که در روش پیشنهادی استفاده شد، فرآیندهای هوشمندانه تعریف و استفاده شود تا سرعت رسیدن به نتایج دلخواه، بیشتر گردد.

مراجع

[1] “Graphviz,” [Online]. Available: https://graphviz.org/.

[2] X.-S. Yang, “Chapter 6 - Genetic Algorithms,” in Nature-Inspired Optimization Algorithms (Second edition), Academic Press, 2021, pp. 91-100.

[3] P. Wilson and H. A. Mantooth, “Chapter 8 - Event-Based Modeling,” in Model-Based Engineering for Complex Electronic Systems, 2013, pp. 259-303.

[4] C. K. Vijayakumari, P. Mythili, R. K. James and C. V. Anil Kumr, “Genetic Algorithm Based Design of Combinational Logic Circuits Using Universal Logic Modules,” Procedia Computer Science, vol. 46, pp. 1246-1253, 2015.

[5] C. K. Vijayakumari, D. Lukose, P. Mythili and R. K. James, “An improved design of combinational digital circuits with multiplexers using genetic algorithm,” in 2013 Annual International Conference on Emerging Research Areas and 2013 International Conference on Microelectronics, Communications and Renewable Energy, Kanjirapally, India, 4-6 June, 2013, pp. 1-5.

[6] P. Soleimani, R. Sabbaghi-Nadooshan, S. Mirzakuchaki and M. Bagheri, “Using Genetic Algorithm in the Evolutionary Design of Sequential Logic Circuits,” IJCSI International Journal of Computer Science Issues, vol. 8, no. 3, pp. 1-7, 2011.

[7] P. Manna, D. K. Kole, H. Rahaman, D. K. Das and B. B. Bhattacharya, “Reversible Logic Circuit Synthesis Using Genetic Algorithm and Particle Swarm Optimization,” in 2012 International Symposium on Electronic System Design (ISED), Kolkata, India, 19-22 Dec, 2012, pp. 246-250.