مجله علمی پژوهشی «پدافند الکترونیکی و سایبری»
سال ششم، شماره ۴، زمستان ۱۳۹۷؛ ص ۲۲-۱۳

ارائه روشی جهت تشخیص بهینه مسیرهای آزمون نرم‌افزاری با استفاده از الگوریتم‌های فراابتکاری

داود اکبری[۱]، صادق بجانی[۲*]، محمدرضا حسنی‌آهنگر[۳]

۱- دانشجوی کارشناسی ارشد، ۲- استادیار، ۳- دانشیار، دانشگاه جامع امام حسین(ع)

(دریافت: ۱۳۹۶/۰۶/۲۷ ، پذیرش: ۱۳۹۷/۰۳/۰۶)

چکیده

در طول تاریخ مهندسی نرم‌افزار، وجود عیب‌های نرم‌افزاری در قلب یک سامانه و عدم پوشش مناسب آن‌ها قبل از استفاده عملیاتی، اکثر مواقع منجر به وقوع حوادث ناگوار جانی و مالی شده است. آزمونی با پوشش مناسب در سطح کد نرم‌افزار می‌تواند از وقوع بسیاری از این حوادث جلوگیری کند. آزمون مسیر مبنا به عنوان قویترین معیار پوشش در آزمون جعبه‌سفید نرم‌افزار محسوب می‌شود. پیش‌نیاز انجام این آزمون، داشتن مجموعه‌ای از مسیرهای آزمون است. هرچه تعداد مسیرهای آزمون بیشتر باشد، سطح بیشتری از کد منبع نرم‌افزار تحت پوشش قرار گرفته و عیوب نرم‌افزاری بیشتری کشف خواهد شد. در نتیجه یک چالش اساسی قبل از انجام آزمون مسیر مبنای نرم‌افزار عبارت است از شناسایی حداکثری مسیرهای آزمونی که قابلیت پیمایش داشته باشند. تاکنون کارهایی برای حداکثر نمودن تعداد مسیرهای آزمونی قابل پیمایش از جمله روش GSO انجام گرفته است، اما بررسی نتایج نشان می‌دهد تعداد مسیرهای آزمونی می‌تواند بیشتر از آن باشد که در حال حاضر به‌دست آمده است. یک راه برای این مهم، استفاده از راه‌حل ترکیبی مبتنی بر دو الگوریتم تکاملی ژنتیک و پرندگان موسوم به EGSOاست که در این مقاله پیشنهاد شده است. نتایج ارزیابی‌ها نشان می‌دهد که استفاده از EGSO موجب افزایش ۹۱ درصدی تعداد مسیرهای آزمون نسبت به روش GSO شده است.

واژه‌های کلیدی: آزمون مسیرمبنای نرم‌افزار، الگوریتم ژنتیک، الگوریتم پرندگان، الگوریتم EGSO

۱- مقدمه

عدم توجه کافی به وجود عیب‌های نرم‌افزاری در بخش کنترلی یک سامانه می‌تواند منجر به وارد آمدن خسارات سنگین جانی و مالی گردد. ازجمله این خسارات می‌توان به وقوع انفجار در ماموریت‌های فضایی، ایجاد اختلال و از کارافتادگی وسیع در صنایع زیرساختی مثل نفت و گاز، انتشار بدافزارهای خطرناک که برخی اوقات یک کشور خاص را مورد هدف قرار می‌دهند و امثال آن اشاره کرد. یک آزمون جامع که نرم‌افزار را قبل از استفاده عملیاتی در یک سامانه، محک زده و در صورت وجود هرگونه عیب اخطارهای لازم را صادر نماید، می‌تواند از بروز بسیاری از حوادث ذکرشده جلوگیری کند.

در آزمون جعبه سفید نرم‌افزار مواردی همانند تشخیص بهینه مسیرهای آزمون و تولید داده آزمون از اهمیت بالایی برخوردار است. در تشخیص بهینه مسیرهای آزمون، هدف شناسایی مسیرهایی از آزمون است که حداکثر پوشش آزمون را فراهم کند؛ درحالی‌که در تولید داده آزمون به دنبال تهیه ورودی‌های مورد نیاز آزمون نرم‌افزار هستیم.

آزمون مسیر[۱] به عنوان قویترین معیار پوشش[۲] در آزمون جعبه‌سفید[۳] نرم‌افزار محسوب می‌شود. این آزمون از روش‌های آزمون ساختار معروف است که در آن کد منبع برنامه به منظور یافتن داده آزمون مناسب با این شرط جستجو می‌شود که بعد از اجرای برنامه با استفاده از داده آزمون ورودی، یک مسیر از پیش تعریف شده پیمایش گردد. یکی از چالش‌های اساسی در آزمون مسیر، یافتن مسیرهای اجرایی هدف است به‌طوری‌که پس از استخراج مسیرها بتوان برای هر مسیر، داده آزمون مناسب را تولید نمود. هرچه تعداد این مسیرها بیشتر باشد، تعداد خطوط بیشتری از برنامه را می‌توان تحت پوشش آزمون قرار داد که این خود منجر به افزایش میزان کشف عیب‌های نرم‌افزاری خواهد شد. استخراج همه مسیرهای اجرایی از یک برنامه غیرممکن است [۱]. در نتیجه، چالش اصلی که در این تحقیق با آن روبرو هستیم انتخاب زیرمجموعه‌ای با فراوانی مطلوب از بین کلیه

*رایانامه نویسنده پاسخگو: sbejani@ihu.ac.ir

1- Basis path testing
2- Coverage criterion
3- White box testing

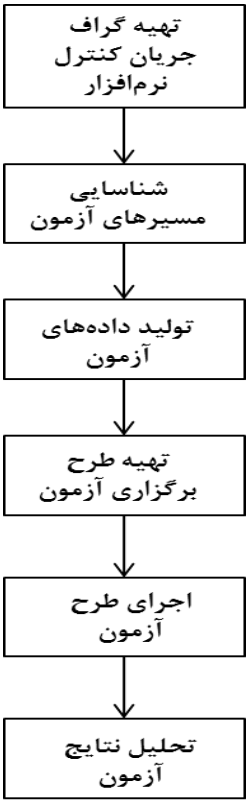
مسیرهای ممکن طبق یک معیار انتخاب است که مسئله مورد نظر را به یک مسئله بهینه‌سازی تبدیل می‌کند. مسیرهای استخراج‌شده باید از نظر اجرایی امکان‌پذیر باشند. اجرایی بودن یک مسیر به معنای قابل پیمایش بودن آن در جریان اجرای نرم‌افزار است که با تولید داده آزمون مناسب میسر می‌گردد [۱۶].

از آنجا که در یک برنامه نرم‌افزاری به تعداد بی‌شماری مسیر اجرایی می‌تواند وجود داشته باشد، استخراج این مسیرها به‌منظور استفاده در آزمون مسیر به یک مساله NP کامل[1] تبدیل می‌شود. در نتیجه استفاده از روش‌های دقیق برای حل آن در یک زمان چندجمله‌ای امکان‌پذیر نخواهد بود. لذا در راه‌حل پیشنهادی از الگوریتم‌های فرا ابتکاری[2] که راه‌حل غیردقیق ارائه می‌دهند استفاده شده است. در تحقیق حاضر برای شناسایی مسیرهای آزمون با یک مسئله بهینه‌سازی مواجه هستیم. الگوریتم ژنتیک[3] و الگوریتم پرندگان[4]، دو نمونه از الگوریتم‌های معروف در حل اینگونه مسائل هستند. محققین در [۵-۲]، نشان داده‌اند که استفاده ترکیبی از دو الگوریتم بهینه‌سازی ژنتیک و پرندگان می‌تواند در حل مساله شناسایی مسیرهای آزمون راه‌گشا باشد. تاکنون در ترکیب این دو روش توجه یکسانی به هریک از الگوریتم‌های ژنتیک و پرندگان شده است، درحالی‌که نتایج تحقیقات نشان می‌دهد الگوریتم پرندگان می‌تواند نسبت به الگوریتم ژنتیک در حصول نتیجه مطلوب موثرتر باشد. لذا در روش پیشنهادی از دو الگوریتم ژنتیک و پرندگان به‌صورت ترکیبی با شرط بیشتر در نظر گرفتن سهم الگوریتم پرندگان استفاده شده است.

۱-۱- آزمون مسیر مبنای نرم‌افزار

آزمون نرم‌افزار به سه دسته آزمون جعبه سفید، آزمون جعبه خاکستری و آزمون جعبه سیاه تقسیم می‌شود. در آزمون جعبه سفید دسترسی به کدهای نرم‌افزار وجود دارد. آزمون جعبه سفید مطابق شکل (۱) انجام می‌گیرد:

یک کد نرم‌افزاری را می‌توان به‌صورت یک گراف در نظر گرفت که رئوس آن را قطعه‌کدهایی تشکیل می‌دهند که هیچ عمل پرشی در آن‌ها انجام نشده باشد. یال‌های این گراف نیز جریان کنترل برنامه تشکیل می‌دهد. با توجه به گفته‌های بالا و داشتن یک مدل گرافی از کد منبع نرم‌افزار، شکست در اجرای نرم‌افزار را می‌توان حاصل پیموده نشدن برخی از مسیرهای این گراف که به گراف جریان کنترل[5] معروف است دانست. خطاهای نرم‌افزاری همیشه در دل کد منبع نرم‌افزار پنهان هستند. طراحی یک آزمون نرم‌افزار که بتواند قبل از مورد استفاده قرار گرفتن نرم‌افزار بصورت عملیاتی تمام شاخه‌های اجرایی نرم‌افزار را مورد پیمایش قرار دهد از اهمیت زیادی برخوردار است. به آزمونی که به منظور پیمایش مسیرهای گراف کنترلی یک برنامه طراحی می‌گردد آزمون مسیر مبنای نرم‌افزار گفته می‌شود. آزمون مسیر، یک روش آزمون ساختاری است که شامل استفاده از کد منبع یک برنامه نرم‌افزاری به منظور پیدا کردن کلیه مسیرهای اجرایی ممکن در آن است. این آزمون در پیدا کردن خطاهای نهفته در کد منبع نرم‌افزار بسیار کمک‌کننده است.



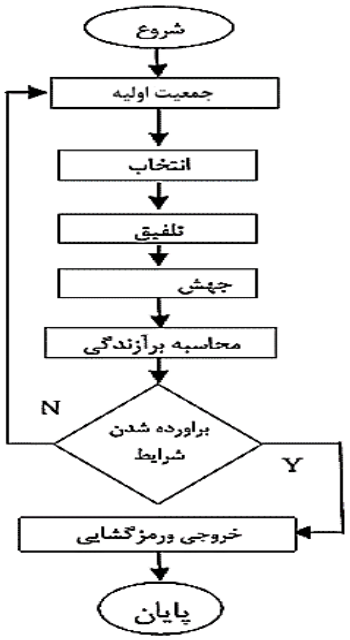
شکل (۱): مراحل اجرای آزمون جعبه سفید نرم‌افزار

۱-۲- معرفی الگوریتم بهینه‌سازی ژنتیک

الگوریتم ژنتیک یکی از الگوریتم‌های پویا است که در سال ۱۹۷۰ توسط جان هالند[6] به‌وجود آمده است [۶]. الگوریتم ژنتیک یک مدل محاسباتی شبیه‌سازی شده از فرایند تکامل زیست‌شناسی تئوری داروین است. نظریه داروین در قرن نوزدهم میلادی مطرح شد و ایده اساسی این الگوریتم انتقال خصوصیات موروثی توسط ژن‌هاست.

1- NP-Complete
2- Metahuristic algorithms
3- Genetic algorithm
4- Particle swarm optimization
5- Control Flow Graph
6- John Holland

بر اساس [۷]، چارچوب کلی الگوریتم ژنتیک در شکل (۲) آمده است.



شکل (۲): چارچوب کلی الگوریتم ژنتیک [۷]

الگوریتم ژنتیک با تولید یک جمعیت اولیه از افراد با توجه به دامنه مسئله کار خود را آغاز می‌کند [۱۷]. هرکدام از افراد با یک رشته بیت باینری که به صورت تصادفی تولیدشده و کروموزوم نام دارد نشان داده می‌شود.

در هر تکرار هر یک از رشته‌های موجود در جمعیت رشته‌ها رمزگشایی شده و مقدار تابع هدف برای آن‌ها به‌دست می‌آید. بر اساس مقادیر به‌دست‌آمده تابع هدف در جمعیت رشته‌ها به هر رشته یک عدد برازندگی نسبت داده می‌شود. این عدد برازندگی احتمال انتخاب را برای هر رشته تعیین خواهد کرد. بر اساس این احتمال انتخاب، مجموعه‌ای از رشته‌ها انتخاب شده و با اعمال عملگرهای ژنتیکی روی آنها رشته‌های جدید جایگزین رشته‌هایی از جمعیت اولیه می‌شوند تا تعداد جمعیت رشته‌ها در تکرارهای محاسباتی ثابت باشد. معمولا جمعیت جدید برازندگی بیشتری دارد این بدان معناست که از نسلی به نسل دیگر جمعیت بهبود می‌یابد. هنگامی جستجو نتیجه‌بخش خواهد بود که به حداکثر تعداد نسل‌های ممکن رسیده باشیم یا همگرایی حاصل شده باشد و یا معیارهای توقف برآورده شده باشد.

۱-۳- معرفی الگوریتم بهینه‌سازی پرندگان

الگوریتم پرندگان یک نوآوری هوشمند است که برای اولین بار در سال ۱۹۹۵ توسط کندی و ابرهارت[1] ارائه شد [۸] و بر اساس رفتار اجتماعی دسته‌های پرندگان عمل می‌کند. الگوریتم پرندگان با تعدادی پاسخ اولیه (ذرات) شروع به‌کار می‌کند و با حرکت دادن این ذرات طی تکرارهای متوالی به دنبال یافتن جواب بهینه برای مسئله است. در هر تکرار دو مقدار pbest و gbest مشخص می‌شوند که به ترتیب عبارت‌اند از محل بهترین مقدار هزینه‌ای که هر ذره در طول حرکت خود به آن رسیده است و محل بهترین ذره از لحاظ هزینه در جمعیت فعلی. پس از یافتن مقادیر فوق، سرعت حرکت ذرات مطابق رابطه (۱) و موقعیت بعدی هر ذره نیز مطابق رابطه (۲) از حاصل جمع موقعیت فعلی و سرعت ذره محاسبه می‌شود.

$$V_i^{k+1} = K*[\ V_i^k + C_1 * r_1 * (pbest - p_i^k) + C_2 * r_2 * (gbest - p_i^k)] \quad (۱)$$

$$K = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|}$$

$$P_i^{k+1} = P_i^k + V_i^{k+1} \quad (۲)$$

در رابطه (۱) مقادیر r_1 و r_2 اعدادی تصادفی بین صفر و یک هستند و ضرایب c_1 و c_2 هم که ضرایب یادگیری نامیده می‌شوند را معمولا برابر ۲ مقداردهی می‌کنیم. عدد K، فاکتور محدودیت است که برای ایجاد اطمینان از همگرایی الگوریتم در نظر گرفته شده است.

۲- پیشینه تحقیق

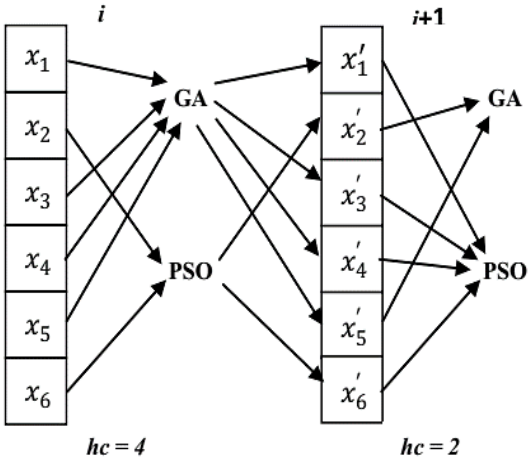
در این بخش چند نمونه از کارهای مرتبط با شناسایی مسیر در آزمون نرم‌افزار را مورد بررسی قرار می‌دهیم.

آقای احمد جیدوک [۹]، برای شناسایی مجموعه مسیرهای آزمون جهت استفاده در فرایند تولید داده برای استفاده در آزمون مسیر از الگوریتم ژنتیک با طول کروموزوم متغیر استفاده کرده است. او در کار خود به‌منظور سازگار کردن الگوریتم ژنتیک برای استفاده در شناسایی مسیر آزمون، تعاریف جدیدی از مفاهیم تقاطع، جهش و تابع تناسب ارائه کرده است. همچنین وی برای اثبات امکان‌پذیری مسیرهای اجرایی، الگوریتم جداگانه‌ای را ارائه کرده است. جیدوک برای ارزیابی کار خود از چند برنامه نمونه در مقیاس کوچک استفاده کرده و نتایج مطلوبی را به‌دست آورده است. ولی چالشی که در کار مذکور مشاهده می‌شود این است که ارزیابی انجام گرفته چندان دقیق نیست زیرا از کدهای نمونه بسیار کوچکی جهت تجزیه و تحلیل روش پیشنهادی استفاده شده است.

1- Kennedy and Eberhar

همان‌گونه که آقای جیدوک در مقاله‌اش به آن اشاره کرده است به خاطر نحوه پیاده‌سازی روش پیشنهادی مطرح‌شده، در مجموعه مسیرهای آزمون شناسایی‌شده ممکن است برخی از مسیرها وجود داشته باشند که قابلیت پیمایش ندارند ولی در عین حال هنگام به‌دست آوردن تعداد مسیرهای آزمون شناسایی‌شده در محاسبات دخالت داده می‌شوند.

جیدوک و همکاران [۲]، برای شناسایی مسیرهای آزمون علاوه بر استفاده از الگوریتم ژنتیک از الگوریتم پرندگان نیز استفاده کرده‌اند. الگوریتم پرندگان یک روش بهینه‌سازی تصادفی است که از حرکت و هوش ذرات الهام گرفته است. هر ذره به‌عنوان یک نقطه در فضای جستجوی n بعدی است که جهت حرکت خود را طبق تجربه حرکت قبلی خود و همچنین تجربه حرکت سایر ذرات تنظیم می‌کند. نهایتا تعدادی از ذرات که تشکیل یک گروه را می‌دهند، در فضای جستجو حرکت می‌کنند تا بهترین راه‌حل را پیدا کنند. مطابق شکل (۳) در روش GSO که ترکیب دو روش GA و الگوریتم پرندگان است در هر تکرار، جمعیت به دو قسمت تقسیم شده و هر قسمت توسط یکی از الگوریتم‌های ژنتیک یا الگوریتم پرندگان به‌طور جداگانه تکامل می‌یابد. سپس دو قسمت با یکدیگر ترکیب شده و تشکیل یک جمعیت واحد جدید را می‌دهند. در تکرار بعدی، دوباره جمعیت طبق یک معیار جداکننده به دو بخش تقسیم شده و GSO به‌کار خود ادامه می‌دهد.



شکل (۲): نمای الگوریتم ترکیبی GSO [۲]

روش GSO نیز فقط در مقیاس برنامه‌های کوچک کاربردی است و برای برنامه‌های پیچیده جوابگو نیست. در این روش توجهی یکسانی به دو الگوریتم ژنتیک و پرندگان شده است. به گونه‌ای که در فرایند تکامل جمعیتی دو بخش ژنتیک و پرندگان روش پیشنهادی سهم یکسانی دارند. در روش GSO جمعیت اولیه به دو قسمت تقسیم می‌گردد ولی هیچ‌گونه مبنایی در نحوه تقسیم آن وجود نداشته و به‌صورت تصادفی انجام می‌گیرد در حالی‌که دو الگوریتم بهینه‌سازی ژنتیک و پرندگان تفاوت‌های زیادی با یکدیگر دارند و در برخی موارد الگوریتم پرندگان توانسته است بهتر از الگوریتم بهینه‌سازی ژنتیک عمل کند. در نتیجه به نظر می‌رسد نگاه یکسان به دو الگوریتم بهینه‌سازی ژنتیک و پرندگان در روش پیشنهادی نمی‌تواند ایده مناسبی به نظر برسد.

در سال ۲۰۱۳ جهت شناسایی مسیرهای آزمون یک روش مبتنی بر الگوریتم کرم شب‌تاب توسط آقای ریواتساوا[1] و همکارش پراوین[2] ارائه شد [۱۰]. این الگوریتم الهام‌گرفته از الگوی ریتمیک تولید نور توسط کرم شب‌تاب است. الگوی نوری که روشن و خاموش می‌شود را می‌توان در یک قالب ریاضی درآورده و به عنوان یک تابع هدف جهت بهینه‌سازی تعریف کرد. طبق نظر یانگ [۱۱]، سه فرض در الگوریتم کرم شب‌تاب وجود دارد:

الف) همه کرم‌های شب‌تاب تک‌جنسه هستند و هر کرم شب‌تاب، کرم شب‌تاب دیگر را جذب می‌کند و یا بدان جذب می‌شود.

ب) میزان جذابیت یک کرم شب‌تاب ارتباط مستقیمی با میزان روشنایی آن دارد (هرچه مسافت طی‌شده بیشتر باشد میزان روشنایی کاهش می‌یابد).

ج) یک کرم شب‌تاب اگر کرم شب‌تاب جذاب‌تر دیگر را نتواند در مناطق مجاور پیدا کند، به‌صورت تصادفی حرکت خواهد کرد.

بابی و همکاران [۱۲]، برای شناسایی مسیرهای بهینه از الگوریتم بهینه‌سازی کلونی مورچگان که مبتنی بر رفتار مورچگان است استفاده کرده‌اند. در این روش، مجموعه‌ای از قوانین مشخص برای یافتن کلیه مسیرهای بهینه مورد استفاده قرار می‌گیرند. در این روش برای پوشش حداکثری مسیرها سعی شده است که تعداد مسیرهای شناسایی شده برابر با پیچیدگی سیکلوماتیک باشد. هدف استفاده از الگوریتم بهینه‌سازی کلونی مورچگان این است که کلیه مسیرهای بهینه موجود در گراف کنترلی برنامه برای حداقل یک‌بار تحت پوشش قرار بگیرند. انتخاب مسیر وابسته به احتمال آن است. هرچه مقدار احتمال یک مسیر بیشتر باشد، شانس انتخاب آن بالاتر است.

آقای ویندیچ[3] و همکاران در [۱۳]، نشان داده‌اند که چگونه الگوریتم بهینه‌سازی پرندگان در بسیاری از موارد به منظور

1- Srivatsava
2- Praveen
3- Windisch

پوشش کد نرم‌افزار می‌تواند نسبت به الگوریتم ژنتیک کاراتر عمل نماید.

در سال ۲۰۱۷ آقای ریواستاوا[14] به‌منظور شناسایی خودکار مسیرهای آزمون از ترکیب دو روش جستجوی فاخته و الگوریتم خفاش استفاده کرده است [۱۴].

در سال ۲۰۱۴ آقای جیدوک [۱۵]یک روش مبتنی بر الگوریتم ژنتیک ارائه کرد که می‌توانست تا حدی میزان پوشش مسیرهای آزمون را بیشینه سازد ولی ایشان با مطالعات بیشتری که انجام داد، با همکاری چند تن دیگر در سال ۲۰۱۵ با ارائه یک مقاله نشان داد که در آزمون مسیر مبنای نرم‌افزار استفاده از الگوریتم بهینه‌سازی پرندگان نتایج بهتری را نسبت به الگوریتم ژنتیک به همراه دارد. به همین خاطر ایشان در کار خود سعی کرد علاوه بر الگوریتم ژنتیک از الگوریتم پرندگان نیز استفاده کند.

نکته قابل توجه در روش ترکیبی مورد نظر این است که اهمیت یکسانی به دو الگوریتم ژنتیک و پرندگان داده شده و سهم هرکدام در حل مسئله یکسان است؛ درحالیکه دو الگوریتم بهینه‌سازی ژنتیک و پرندگان تفاوت‌های قابل توجهی با یکدیگر داشته و در برخی از موارد مثل سرعت همگرایی به راه‌حل بهینه و نیز دقت پاسخ‌های به‌دست‌آمده الگوریتم پرندگان نسبت به الگوریتم ژنتیک بهتر عمل می‌کند. در نتیجه به نظر می‌رسد با توجه به عملکرد بهتر الگوریتم پرندگان نسبت به الگوریتم ژنتیک در حل مسئله مورد بحث، می‌توانیم با بیشتر کردن سهم مورد اول و دادن وزن بالاتر به آن به نتایج بهتری هم از نظر میزان پوشش مسیرهای آزمون و هم از نظر سرعت همگرایی به راه‌حل بهینه دست یابیم.

۲-۱- تحلیل کارهای مرتبط

در سال ۲۰۱۴ آقای جیدوک یک روش مبتنی بر الگوریتم ژنتیک ارائه کرد که می‌توانست تا حدی میزان پوشش مسیرهای آزمون را بیشینه سازد ولی ایشان با مطالعات بیشتری که انجام داد، با همکاری چند تن دیگر در سال ۲۰۱۵ با ارائه یک مقاله نشان داد که در آزمون مسیر نرم‌افزار استفاده از الگوریتم بهینه‌سازی پرندگان نتایج بهتری را نسبت به الگوریتم ژنتیک به همراه دارد. به همین خاطر ایشان در کار خود سعی کرد علاوه بر الگوریتم ژنتیک از الگوریتم پرندگان نیز استفاده کند.

نکته قابل توجه در روش ترکیبی مورد نظر این است که اهمیت یکسانی به دو الگوریتم ژنتیک و پرندگان داده شده و سهم هرکدام در حل مسئله یکسان است. نقص عمده در کارهای پیشین این است که وزن متناسبی به هریک از دو الگوریتم یادشده داده نشده است. در حالی‌که با توجه به تحقیقات قبلی انجام‌گرفته، دو الگوریتم بهینه‌سازی ژنتیک و پرندگان تفاوت‌های قابل توجهی با یکدیگر داشته و در برخی از موارد اثربخشی الگوریتم پرندگان از نظر "سرعت همگرایی به راه‌حل بهینه" نسبت به الگوریتم ژنتیک بیشتر است. در نتیجه به نظر می‌رسد با توجه به عملکرد بهتر الگوریتم پرندگان نسبت به الگوریتم ژنتیک در حل مسئله مورد بحث، می‌توانیم با بیشتر کردن سهم الگوریتم پرندگان و دادن وزن بالاتر به آن به نتایج بهتری هم از نظر میزان پوشش مسیرهای آزمون و هم از نظر سرعت همگرایی به راه‌حل بهینه دست یابیم.

۳- روش EGSO جهت شناسایی خودکار مسیرهای آزمون نرم‌افزار

روش EGSO یک روش مرکب از دو الگوریتم تکاملی ژنتیک و پرندگان و نسخه‌ای توسعه‌یافته از روش GSO است که در بخش دوم مورد بررسی قرار گرفت. این روش به منظور ترکیب مکانیزم‌های موثر هریک از بعد از اتمام کار بخش ارزیابی، نوبت به بخش آخر از معماری EGSO که همان بخش انتخاب است می‌رسد. در این بخش، راه‌حل‌های ارزش‌گذاری‌شده طبق یک معیار، گلچین شده و جمعیت نسل جایگزین را برای شرکت در چرخه بعدی تشکیل می‌دهند. معیار مورد نظر برای این کار نیز میزان امکان‌پذیر بودن راه‌حل تولیدی است. این چرخه تا زمانی ادامه خواهد داشت که یا به مجموعه‌ای از راه‌حل‌ها به عنوان مسیرهای بهینه آزمون دست پیدا کنیم و یا این که به حد آستانه از قبل تعیین‌شده از نظر تعداد نسل‌های تولیدشده برسیم. دو روش بهینه‌سازی ژنتیک و پرندگان برای شناسایی حداکثری مسیرهای آزمونی که قابلیت پیمایش داشته باشند ارائه شده است.

هدف از طراحی روش EGSO ارائه یک الگوریتم ترکیبی است که توانایی بالاتری را نسبت به روش ترکیبی ژنتیک و پرندگان (GSO)، از طریق وزن‌دهی مناسب به هر بخش از الگوریتم در جریان یافتن راه‌حل بهینه برای حل مساله شناسایی مسیرهای آزمون داشته باشد.

۳-۱- معماری کلی روش EGSO

شکل (۴) معماری کلی روش EGSO را که از دو بخش اصلی ژنتیک و پرندگان تشکیل شده است نشان می‌دهد. ورودی الگوریتم EGSO، گراف جریان کنترل نرم‌افزار است که برای آغاز به کار چرخه شکل (۴) لازم است در قالب مجموعه‌ای از رشته‌های عددی مدل شود. در آغاز چرخه شکل (۴)، این

14- Srivastava

رشته‌های عددی به عنوان مجموعه راه‌حل‌های اولیه به‌صورت تصادفی تولید می‌شوند. تولید این راه‌حل‌ها به‌گونه‌ای است که هر عدد در یک رشته، بیانگر شماره یال گراف جریان کنترل قطعه‌کد ورودی بوده و می‌توانیم روی آن اعمال ریاضی انجام دهیم.

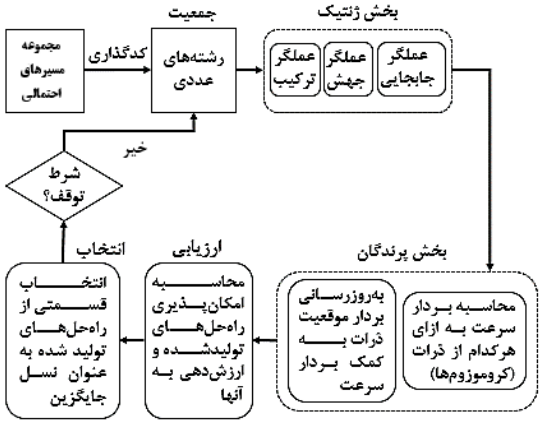
بخش ژنتیک روش EGSO از سه عملگر ترکیب، جهش و جابجایی تشکیل شده است. لازم به ذکر است در اینجا منظور از کروموزوم، یک مسیر از گراف جریان کنترل و منظور از ژن، یک یال از این مسیر است. در الگوریتم پیشنهادی، مسیرهای گراف جریان کنترل برنامه به مجموعه‌ای از رشته‌های عددی تبدیل می‌شود.

کار عملگر ترکیب، شکستن رشته‌های عددی به دو قسمت و ترکیب قطعات تولیدشده با یکدیگر است که منجر به تولید مسیرهای جدید می‌گردد. عملگر جهش نیز دو رقم از رشته مورد نظر را به‌صورت تصادفی انتخاب کرده و تغییر می‌دهد. عملگر جابجایی به عنوان یک عملگر جدید، عملگری است که دو رقم از یک رشته را به‌صورت تصادفی انتخاب کرده و با یکدیگر جابجا می‌کند.

بعد از اتمام کار بخش ژنتیک وارد بخش پرندگان روش EGSO می‌شویم. همانند آنچه که در بخش ژنتیک روش EGSO بیان شد در بخش پرندگان این روش، دو کار اصلی مطرح است. کار اول، محاسبه بردار سرعت به‌ازای هریک از مسیرهای گراف جریان کنترل است که در اینجا با نام ذره شناخته می‌شوند. کار دوم نیز به‌روزرسانی بردار موقعیت ذرات است. منظور از بردار موقعیت در بخش پرندگان، مجموعه اعداد موجود در یک مسیر از گراف جریان کنترل است. بردار موقعیت جدید ذره از حاصل‌جمع بردار سرعت و بردار موقعیت قبلی ذره به‌دست می‌آید.

ویژگی ممتاز روش EGSO نسبت به روش GSO این است که علاوه بر استفاده از عملگر جدید جابجایی سهم بیشتری در فرآیند تکامل جمعیت به بخش پرندگان داده شده است و این بخش می‌تواند مدت زمان بیشتری نسبت به بخش ژنتیک در روش EGSO فعال باشد.

خروجی دو بخش ژنتیک و پرندگان مجموعه‌ای از رشته‌های عددی است که برخی از آنها می‌توانند راه‌حل مورد نظر ما باشند. برای تشخیص این راه‌حل‌ها، مجموعه مورد نظر وارد بخش ارزیابی شده و در ابتدا میزان امکان‌پذیری هریک از اعضای مجموعه محاسبه می‌شود. منظور از امکان‌پذیری یک عضو از مجموعه راه‌حل‌های تولیدشده، میزان قابلیت پیموده شدن آن مسیر در جریان آزمون مسیر مبنای نرم‌افزار است. حال با توجه به میزان امکان‌پذیری هریک از اعضای مجموعه می‌توانیم روی راه‌حل‌ها ارزش‌گذاری کنیم. یک راه‌حل که ارزش مطلوبی را کسب کند به عنوان مسیر بهینه تلقی می‌شود.



شکل(۴): معماری کلی روش EGSO

۳-۲- تشریح بخش ژنتیک در EGSO

فضای جستجو شامل مجموعه کلیه راه‌حل‌هایی است که راه‌حل بهینه نیز جزئی از آن است. هر نقطه در فضای جستجو یک راه‌حل ممکن را نشان می‌دهد. فرض کنیم گراف DDG، نشان‌دهنده گراف جریان کنترلی برنامه تحت آزمون باشد. در این صورت فضای جستجوی الگوریتم ما D است و مانند رابطه۳ نمایش داده می‌شود:

$$D = \{\forall e \mid e \in DDG \text{ and } e \text{ is reached by entry and reaches exit}\} \quad (۳)$$

رابطه (۳)، مجموعه کلیه یال‌هایی از یک گراف جهت‌دار را نشان می‌دهد که از یک گره ورودی می‌توانیم به آنها برسیم و نیز این یالها در نهایت به یک گره خروجی منتهی می‌شوند.

تابع هزینه مورد استفاده در بخش ژنتیک راه‌کار ارائه‌شده همانند [۹] مبتنی بر مفهوم تعداد یال‌های مجاور در گراف جریان کنترل نرم‌افزار تحت آزمون است. یال‌هایی از گراف جریان کنترل، مجاور محسوب می‌شوند که به همدیگر راه داشته باشند. این تابع، هزینه مسیرهایی از گراف را که توسط الگوریتم ژنتیک تولید شده‌اند با محاسبه احتمال مجاور بودن یال‌های آن به‌دست می‌آورد.

بخش ژنتیک راه‌حل پیشنهادی به منظور تبادل اطلاعات بین کروموزوم‌های مختلف و تولید جفت کروموزوم‌های جدید از ترکیب یکنواخت[1] با احتمال PX برابر ۰/۷ استفاده می‌کند.

1- Uniform crossover

مقادیر احتمال هریک از عملگرهای ژنتیکی به کمک آزمون و خطا و انجام آزمایش‌های متعدد تنظیم شده‌اند. نحوه انجام عمل ترکیب نیز همانند [۹] است.

عمل جهش بعد از اعمال عملگر ترکیب و روی دو ژن از کروموزوم‌هایی که با احتمال PM برابر ۰/۲ انتخاب شده‌اند صورت می‌گیرد و نحوه انتخاب ژن‌ها نیز به‌صورت تصادفی است. در جریان عمل جهش دو ژن انتخاب شده از داخل یک کروموزوم دچار تغییر می‌گردند. از آنجا که آزمون مسیری که قرار است انجام پذیرد در مقیاس یک واحد نرم‌افزاری بوده و همه برنامه‌های نمونه تحت آزمون اندازه تقریبا یکسانی دارند، محدوده تغییرات ژن‌ها را بعد از انجام آزمون‌های مختلف مقداری ثابت و از بازه [۲+ , ۲-] انتخاب کرده‌ایم.

عملگر جابجایی[1]، یک عملگر جدید است که در کارهای تحقیقاتی قبلی نبوده است. با اضافه شدن عملگر جابجایی به مجموعه دو عملگر قبلی تنوع ژنتیکی بیشتری ایجاد می‌گردد. کار عملگر جابجایی، جابجا کردن مقادیر دو مورد از ژن‌هایی است که بصورت تصادفی از داخل یک کروموزوم انتخاب شده‌اند. احتمال انجام عملیات جابجایی که با PE نشان می‌دهیم و از روی آزمایش تجربی به‌دست‌آمده، برابر ۰/۲ است. استفاده از عملگر جابجایی با ایجاد تنوع ژنتیکی، منجر به کشف حوزه‌های جدید جستجو گشته که به نوبه خود منجر به افزایش احتمال یافتن راه‌حل‌های بهینه جدید می‌گردد.

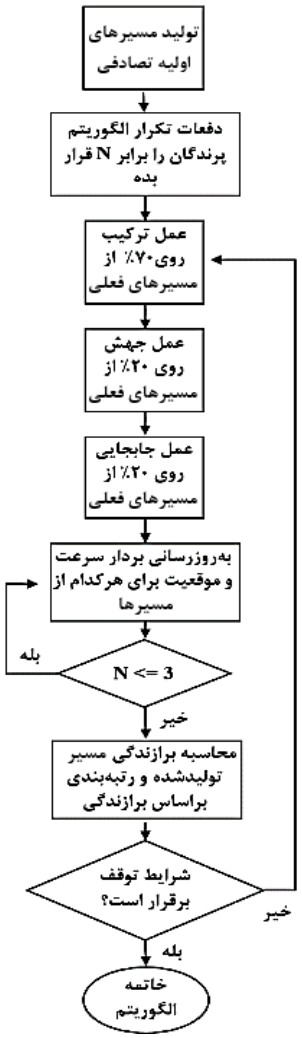
۳-۳- تشریح بخش پرندگان در EGSO

نحوه به‌روزرسانی بردار سرعت طبق رابطه (۱) است. با توجه به گسسته بودن نوع مساله و وجود محدودیت‌های مربوط به آن، مقدار ضریب K در این رابطه برابر ۱ گرفته شده است. ضرایب یادگیری C_1 و C_2 نیز برابر مقدار ۲ گرفته شده‌اند. بعد از به‌دست آوردن بردار سرعت، موقعیت جدید ذره را می‌توان طبق رابطه (۲) به‌دست آورد. پس از به‌دست آوردن بردار موقعیت ذره، لازم است هزینه مربوط به موقعیت جدید ذره نیز محاسبه گردد. تابع هزینه در الگوریتم پرندگان همانند الگوریتم ژنتیک محاسبه می‌گردد. پس از محاسبه هزینه جدید توسط الگوریتم پرندگان، این هزینه با هزینه به‌دست‌آمده توسط الگوریتم ژنتیک مقایسه شده و هرکدام که بهینه‌تر بود انتخاب می‌گردد. در نتیجه بدین ترتیب می‌توانیم از قابلیت‌های دو الگوریتم ژنتیک و پرندگان به‌صورت ترکیبی استفاده کنیم.

۴-۳- الگوریتم EGSO

این بخش به توضیح گام‌به‌گام روش EGSO به منظور شناسایی خودکار مجموعه مسیرهای آزمون اختصاص داده شده است.

نمودار گردشی شکل (۵) به‌خوبی نحوه عملکرد الگوریتم EGSO را نشان می‌دهد.



شکل (۵): الگوریتم EGSO

همان‌طور که در شکل (۵) معلوم است بعد از تولید نسل آغازین ابتدا عملگرهای ژنتیکی روی جمعیت اعمال می‌شوند. بعد از یکبار پردازش جمعیت توسط الگوریتم ژنتیک، نوبت به الگوریتم پرندگان می‌رسد تا جمعیتی را که به‌عنوان خروجی الگوریتم ژنتیک محسوب می‌شود پردازش نماید.

تعداد تکرارهای الگوریتم پرندگان و در نتیجه پردازشی که توسط این الگوریتم روی جمعیت فعلی انجام می‌دهد بیشتر از الگوریتم ژنتیک است. به‌گونه‌ای که پس از انجام آزمایشات متعدد و بررسی نتایج تجربی به‌دست‌آمده از نمودارها، به‌ازای هر نسل جدیدی که تولید می‌شود نسبت تعداد تکرارهای الگوریتم پرندگان به تعداد تکرارهای الگوریتم ژنتیک سه به یک انتخاب شده است. یعنی جمعیت فعلی بعد از آنکه یکبار توسط الگوریتم

1- Exchange operator

ژنتیک پردازش شد، سه بار توسط الگوریتم پرندگان پردازش می‌گردد.

در پایان پس از محاسبه برازندگی جمعیت تولیدشده، چنانچه تعداد نسل‌های تولیدشده به عنوان شرایط توقف از حد معینی گذر کرده باشد الگوریتم متوقف و در غیر این‌صورت کل مراحل دوباره تکرار می‌شود.

۴- ارزیابی روش EGSO

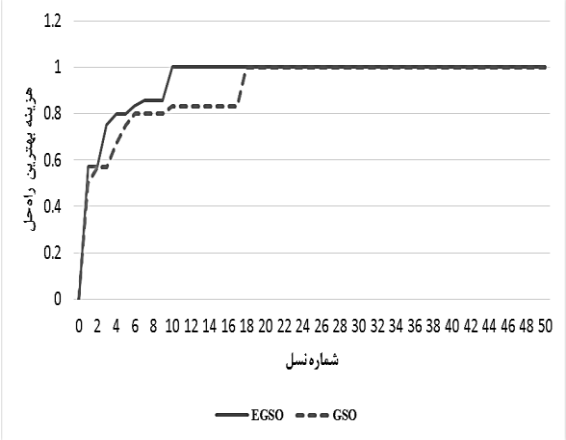
برای ارزیابی اثربخشی روش EGSO سه معیار سرعت همگرایی به مسیر بهینه، تعداد مسیرهای آزمون شناسایی‌شده و تعداد نسل‌های تولیدشده تا رسیدن به شرایط بهینه در نظر گرفته شده است. در این راستا از مجموعه برنامه‌های معیار پرکاربرد در آزمون نرم‌افزار که دقیقا در مقالات مرتبط نیز مورد استفاده قرار گرفته‌اند بهره گرفته شده است [۹ و ۱۹-۱۵]. جدول (۱) مشخصات این برنامه‌ها را نشان داده است.

جدول (۱): مشخصات برنامه‌های معیار

شماره برنامه	تعداد خطوط	تعداد گره	تعداد یال	توضیح برنامه
P#1	۳۲	۴۴	۴۹	پیدا کردن نوع مثلث
P#2	۳۷	۳۹	۴۳	پیدا کردن مقدار میانه
P#3	۲۷	۲۹	۳۱	به‌دست آوردن توان
P#4	۳۸	۴۰	۴۴	به‌دست آوردن مقدار باقیمانده
P#5	۳۴	۳۸	۳۹	ترکیب if و while
P#6	۳۶	۴۲	۴۴	حلقه‌های تکرار تودرتو
P#7	۴۰	۳۶	۴۲	ترکیب حلقه do-while و if
P#8	۲۱	۲۳	۲۵	پیدا کردن بزرگترین و کوچکترین عنصر آرایه
P#9	۲۰	۲۲	۲۴	اعداد زوج یک آرایه
P#10	۲۵۳	۱۲۲	۱۷۰	پیدا کردن تعداد روزهای بین دو تاریخ

۴-۱- سرعت همگرایی

منظور از سرعت همگرایی، شیب نمودار هزینه بهترین راه‌حل تولیدشده به ازای هر نسل تولیدشده است. هدف از مقداریابی برای این کمیت این است که نشان دهیم روش EGSO می‌تواند مجموعه مسیرهای آزمون بهینه را در مدت زمان کمتری شناسایی کند. با اجرای الگوریتم EGSO روی برنامه‌های ده‌گانه جدول (۱) نمودار هزینه بهترین راه‌حل تولیدشده به‌ازای هر نسل مطابق شکل (۶) به‌دست آمده است. طبق شکل (۵) روش ترکیبی EGSO از نظر سرعت همگرایی به نقطه بهینه که همان هزینه ۱ است توانسته است به میزان ۴٪ بهتر از روش GSO عمل نماید. علت این امر این است که روش EGSO با یافتن مجموعه جواب‌های دارای شرایط مناسب برای تکامل یافتن در همان نسل‌های اولیه توانسته راه صحیح رسیدن به شرایط بهینه را پیدا کرده و سریع‌تر از روش GSO به نقطه بهینه برسد.

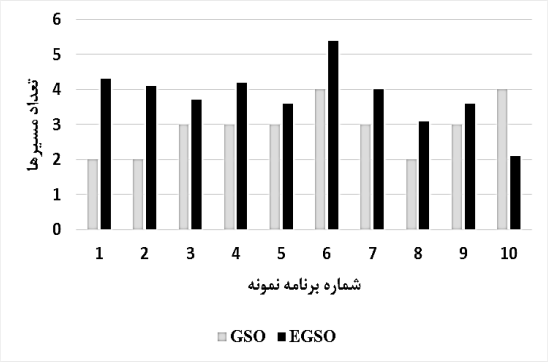


شکل (۶): مقایسه نمودار همگرایی پاسخ‌های دو روش GSO و EGSO به‌نقطه بهینه

۴-۲- تعداد مسیرهای آزمون شناسایی‌شده

هرچه تعداد مسیرهای آزمون شناسایی‌شده بیشتر باشد، سطح وسیعتری از گراف جریان کنترل برنامه در جریان آزمون مسیر تحت پوشش قرار خواهد گرفت.

طبق شکل (۷)، روش EGSO به جز در مورد برنامه معیار دهم که یک برنامه پیچیده است در بقیه موارد توانسته است تعداد مسیرهای بیشتری را نسبت به روش GSO شناسایی کند. با محاسبه میانگین اختلاف نقطه به نقطه دو نمودار، میزان بهبود ۹۱٪ به‌دست‌آمده است. علت این‌که روش EGSO توانسته در مورد اغلب برنامه‌های نمونه نسبت به روش GSO از نظر تعداد مسیرهای شناسایی‌شده بهتر عمل کند این است که ما در روش پیشنهادی خود با توجه به قابلیت‌های الگوریتم پرندگان سهم بیشتری را در فرایند تکامل جمعیتی به این الگوریتم فراابتکاری نسبت به الگوریتم ژنتیک داده‌ایم. این امر منجر به افزایش چابکی و دقت روش EGSO حین جستجو در فضای مساله شده و در نتیجه شاهد کشف مسیرهای بیشتری نسبت به روش GSO هستیم.

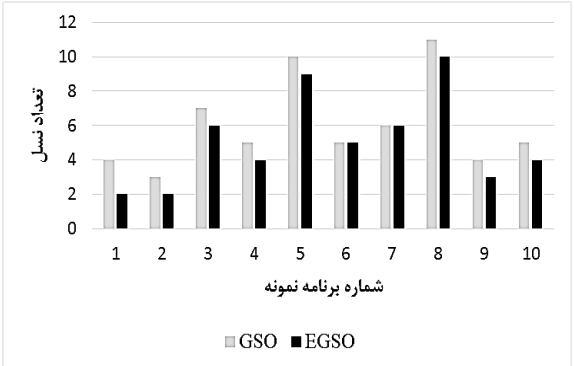


شکل (۷): مقایسه تعداد مسیرهای آزمون شناسایی‌شده در دو روش GSO و EGSO

۴-۳- تعداد نسل‌های تولیدشده تا رسیدن به شرایط همگرایی

شرایط همگرایی عبارت است از شرایطی که در آن منحنی هریک از نمودارهای هزینه مسیر و یا اجراپذیری حالت نوسان نداشته و بصورت یکنواخت با شیب صفر باشد. هرچه با تولید نسل‌های کمتر بتوانیم به شرایط همگرایی برسیم، به همان اندازه هزینه انجام آزمون مسیر کاهش خواهد یافت. در نتیجه هر روشی که پس از تولید تعداد نسل‌های کمتر به شرایط بهینه همگرا شود نسبت به روش دیگر بهینه‌تر خواهد بود.

طبق شکل (۸) روش EGSO توانسته به ازای اکثر برنامه‌های معیار جدول (۱) بهتر از روش GSO عمل کرده و با کاهش تولید تعداد نسل‌ها به میزان ۹٪ به شرایط بهینه برسد.



شکل (۸): مقایسه دو روش EGSO و GSO از نظر تعداد نسل‌های تولیدشده تا رسیدن به شرایط بهینه

۵- نتیجه‌گیری

روش EGSO یک روش مرکب از دو الگوریتم فراابتکاری ژنتیک و پرندگان است که نسخه‌ای توسعه‌یافته از روش GSO محسوب می‌شود. یک جزء از روش مرکب EGSO الگوریتم فراابتکاری پرندگان است. الگوریتم پرندگان دارای مزایایی مثل توانایی تبادل اطلاعات با دیگران، به‌خاطر سپردن اطلاعات گذشته، توانایی استفاده از تجربیات گذشته برای تصمیم‌گیری در آینده، آسانی پیاده‌سازی و کم بودن تعداد پارامترهای تنظیم است که باعث شده این الگوریتم در مقایسه با الگوریتم بهینه‌سازی ژنتیک در پاره‌ای از مسائل موفق‌تر عمل کند. روش GSO علی‌رغم وجود مزایای مذکور در الگوریتم پرندگان، هیچ تفاوتی بین این الگوریتم و الگوریتم بهینه‌سازی ژنتیک قائل نشده و در مراحل مختلف تکامل جمعیتی، به‌صورت تصادفی از هریک از دو الگوریتم استفاده کرده است.

در این مقاله با وزن‌دهی بیشتر به الگوریتم پرندگان در جریان تکامل جمعیتی و همچنین استفاده از عملگر جدید جابجایی توانستیم پوشش بهینه‌تری نسبت به روش GSO روی مسیرهای آزمون نرم‌افزاری ارائه دهیم.

از محدودیت‌های روش پیشنهادی می‌توان به عدم کارایی آن در برنامه‌های با مقیاس بزرگ اشاره کرد که دلیل این امر نیز پیچیدگی بالای گراف جریان کنترل برنامه‌های مقیاس بزرگ است.

پیشنهاد می‌شود در راستای ادامه تحقیق به منظور بهبود روش EGSO از قابلیت روش جستجوی A* که جزء روش‌های موثر در هوش مصنوعی محسوب می‌شود استفاده گردد.

۶- منابع

[1] S. Saurabh Srivastava, "Optimal Path Sequencing in Basic Path Testing," International Journal of Advanced Computational Engineering and Networking, vol. 1, no. 1, pp. 2320-2106, 2013.

[2] A. Moheb and R. Girgis, "Automatic Data Flow Test Paths Generation using the Genetical Swarm Optimization Technique," International Journal of Computer Applications, vol. 1, pp. 975 – 8887, 2015.

[3] D. Sanjay Singla, "A Hybrid PSO Approach to Automate Test Data Generation for Data Flow Coverage with Dominance Concepts," International Journal of Advanced Science and Technology, vol. 37, 2011.

[4] Z. Li, "Breading software test data with Genetic-Particle swarm mixed Algorithms," Journal of computers, pp. 258-265, 2010.

[5] C. Juang, "A hybrid of genetic algorithm and particle swarm optimization for recurrent network design," IEEE transactions on systems management and cybernetics, pp. 997-1006, 2004.

[6] W. F. Lammermann, "Using evolutionary algorithms for unit testing of object oriented software," ACM, 2005.

[7] Y. Dong, "Automatic Generation of Software Test Cases Based on Improved Genetic Algorithm," IEEE-International Conference on Multimedia Technology, pp. 227-230, 2011.

[8] J. Kennedy, "Particle Swarm Optimization," in IEEE International Conference on Neural Networks, 1995.

[9] A. S. Ghiduk, "Automatic generation of basis test paths using variable length genetic algorithm," Information Processing Letters, no. 114, pp. 304–316, 2014.

[10] Y. Srivatsava, “Optimal test sequence generation using firefly algorithm,” Swarm and Evolutionary Computation, pp. 44-53, 2013.

[11] Y. XS, “Firefly algorithms for multimodal optimization,” in Proceedings fifth symposium on stochastic algorithms, foundations and applications, 2009.

[12] R. Srivastava, “An Approach of Optimal Path Generation using Ant Colony Optimization,” IEEE, 2009.

[13] S. Andreas Windisch, “Applying Particle Swarm Optimization to Software Testing,” in Proceedings of the conference on Genetic and evolutionary computation, 2007.

[14] P. Srivastava, “Path Generation for Software Testing: A Hybrid Approach Using Cuckoo Search and Bat Algorithm,” Nature-Inspired Computing and Optimization, pp. 409-424, 2017.

[15] M. Ahmed and S. Ghiduk, “Using genetic algorithms and dominance concepts for generating reduced test data,” Informatica, pp. 377–385, 2010.

[16] G. Michael, “Generating software test data by evolution,” IEEE Trans. Softw. Eng., pp. 1085–1110, 2001.

[17] M. Pargas, “Test data generation using genetic algorithms,” J. Softw. Test. Verif. Reliabil, pp. 263–282, 1999.

[18] M. Girgis, “Automatic test data generation for data flow testing using a genetic algorithm,” J. Univers. Comput. Sci., pp. 898–915, 2005.

[19] J. Diza, “A tabu search algorithm for structural software testing,” J. Comput. Oper. Res., pp. 3052–3072, 2008.