مقاله پژوهشی

بهبود فرایند تولید خودکار داده آزمون نرم‌افزار مبتنی بر معیار پوشش مسیر و با استفاده از ترکیب الگوریتم بهینه‌سازی کواتی و الگوریتم یادگیری Q

مرضیه سپهوند و مجتبی صالحی

چکیده: آزمون نرم‌افزار یکی از مهم‌ترین روش‌های تحلیل میزان اطمینان کیفیت نرم‌افزار است. این فرایند بسیار زمان‌بر و پرهزینه است و تقریباً نیمی از هزینه تولید نرم‌افزار را به خود اختصاص داده است. از این رو به دنبال استفاده از روش‌های خودکار به منظور کاستن هزینه و زمان آزمون هستیم. مسأله عمده در فرایند تولید داده آزمون، تعیین داده‌های ورودی برنامه، به گونه‌ای است که معیار آزمون مشخص‌شده را برآورده سازد. در این تحقیق از روش ساختاری به منظور فرایند خودکارسازی تولید داده آزمون با تمرکز بر معیار پوشش همه مسیرهای متناهی استفاده شده است. در روش ساختاری، مسأله به یک مسأله جستجو تبدیل شده و از الگوریتم‌های فرامکاشفه‌ای برای حل آن استفاده می‌شود. روش پیشنهادی یک الگوریتم ترکیبی است که در آن الگوریتم یادگیری Q به‌عنوان یک روش جستجوی محلی در درون ساختار الگوریتم جستجوی کواتی مورد استفاده قرار می‌گیرد. به طور متوسط، الگوریتم پیشنهادی ما در مقایسه با سایر الگوریتم‌ها حدود ۲۵ تا ۳۰ درصد بهبود را در پوشش نشان می‌دهد که آن را به طور قابل توجهی نسبت به دیگر الگوریتم ها مؤثرتر می‌کند. نتایج آزمایش‌ها نشان می‌دهد که الگوریتم پیشنهادی به‌دلیل رویکرد بهینه در جست‌وجوی مسیرهای آزمون، در مقایسه با سایر الگوریتم‌ها، پوشش مسیر بالاتری حاصل کرده است.

کلیدواژه: **آزمون نرم‌افزار، تولید داده آزمون، آزمون ساختاری، الگوریتم‌های فرامکاشفه‌ای، الگوریتم یادگیری Q.**

۱- مقدمه

مسأله عمده در فرایند تولید داده آزمون نرم‌افزار، تعیین داده‌های ورودی برای پارامترهای برنامه تحت آزمون است. هدف در این مسأله، تولید داده آزمون است به گونه‌ای که معیار آزمون مشخص‌شده را برآورده کند [۱]. این معیارها بر اساس ساختار برنامه تعریف می‌شوند. به عنوان مثال معیار پوشش شاخه از معیارهایی است که هدف در آن، پوشش تمامی شاخه‌های برنامه است. یکی دیگر از این معیارها، پوشش عبارت است که هدف آن اجرای تمامی عبارت‌های برنامه است. از دیگر معیارها، پوشش همه مسیرهای متناهی[1] است که هدف، پوشش تمامی مسیرهای متناهی موجود در برنامه تحت آزمون است [۲]. برای تحقق این معیارها نیاز به گراف جریان برنامه داریم که برای ایجاد این گراف می‌توان به‌صورت دستی و یا از روش‌های خودکار ایجاد گراف جریان استفاده کرد. بعد از ایجاد این گراف می‌توان این مسیرها را از گراف استخراج کرد. روش‌های آزمون نرم‌افزار به دو دسته عملیاتی و ساختاری تقسیم می‌شوند. روش عملیاتی از ویژگی‌های برنامه برای تولید داده آزمون به‌منظور آزمون رفتار منطقی سیستم‌ها بهره می‌گیرد. در روش ساختاری عملیات مورد نظر بر اساس ساختار داخلی برنامه در حال آزمون انجام می‌گیرد. بعد از انتخاب معیار کیفیت می‌توان به تعیین میزان پوشش برنامه توسط داده‌های تولیدشده پرداخت. به طور کلی روش‌های خودکار آزمون ساختاری به دو دسته پویا و ایستا تقسیم می‌شوند. در روش ایستا به‌منظور تولید خودکار داده‌های آزمون، نمی‌توان به‌صورت مستقیم از کد منبع برنامه استفاده کرد و باید آن را به شکلی تبدیل نمود که استاندارد است و حاوی اطلاعات خاصی جهت تولید هر چه مؤثرتر داده‌های آزمون باشد که معمولاً بر پایه اجرای نمادین برنامه بنا شده‌اند [۳]. بنابراین به‌منظور آماده‌سازی، کد منبع برنامه باید مراحلی را طی کرده و به شکل خاصی تبدیل شود. همچنین در روش‌های مبتنی بر ساختار، به‌دلیل پیچیدگی در تحلیل و ردیابی آرایه‌ها و اشاره‌گرها، مشکلاتی در استخراج دقیق مسیرهای اجرایی برنامه وجود دارد [۴]. در روش پویا مشکلات مربوط به روش ایستا حل شده و مسأله تولید داده آزمون به مسأله بهینه‌سازی تبدیل می‌گردد. در بسیاری از روش‌های پویای پیشین از الگوریتم‌های فرامکاشفه‌ای بدین منظور استفاده شده است [۵] و [۶]. الگوریتم بهینه‌سازی کواتی یک الگوریتم جدید، سریع و همچنین دارای قابلیت اکتشاف و قابلیت استفاده خوب است که به منظور تولید خودکار داده آزمون در این مقاله مورد استفاده قرار گرفته شده است [۷]. یکی از عیب‌های عمده الگوریتم‌های فرامکاشفه‌ای این است که هر راه‌حل به تنهایی یک بردار N بعدی را نمایش می‌دهد که معرف یک پاسخ یا راه‌حل برای مسأله است. گاهی امکان دارد که قسمت‌هایی از این بردار به پاسخ‌های صحیح نزدیک شده باشند؛ در حالی که قسمت‌های دیگر بردار از پاسخ صحیح دور باشند. بنابراین در کل این راه‌حل مناسب به نظر نمی‌رسد و باید به موقعیت بهتری برود. امکان دارد که آن قسمت‌هایی از بردار راه‌حل که به جواب نزدیک بوده‌اند طی آپدیت راه‌حل جدید، از پاسخ جدید فاصله بگیرند؛ بنابراین اطلاعات مفید راه‌حل از بین می‌رود. لذا برای

این مقاله در تاریخ ۳ اردیبهشت ماه ۱۴۰۳ دریافت و در تاریخ ۳ مهر ماه ۱۴۰۳ بازنگری شد.

مرضیه سپهوند، گروه کامپیوتر، واحد خرم‌آباد، دانشگاه آزاد اسلامی، خرم‌آباد، ایران، (email: marzieh.sepahvand@gmail.com).

مجتبی صالحی (نویسنده مسئول)، گروه کامپیوتر، واحد خرم‌آباد، دانشگاه آزاد اسلامی، خرم‌آباد، ایران، (email: mo.salehi@iau.ac.ir).

1. All Finite Path Coverage

رفع این مشکل از الگوریتم یادگیری Q به‌عنوان الگوریتم جستجوی محلی در داخل الگوریتم بهینه‌سازی کواتی استفاده شده است [۸]. در این مقاله از این ایده به این صورت استفاده شده که به جزئیات داخل راه‌حل‌ها توجه می‌کند. در الگوریتم بهینه‌سازی کوآتی، تمرکز بر بهترین راه‌حل‌های به‌دست‌آمده است و از طریق اعمال تغییرات جزئی و جابه‌جایی‌های هدفمند در اجزای راه‌حل، کیفیت آن‌ها بهبود داده می‌شود. دستکاری و جابه‌جایی جزئیات داخلی راه‌حل‌ها در روش پیشنهادی از طریق الگوریتم یادگیری Q انجام می‌شود. در واقع، یک عامل یادگیر مسئول بهبود راه‌حل می‌شود. این عامل اقدام‌هایی دارد که از طریق آنها می‌تواند تغییرات کوچک و محلی روی راه‌حل‌ها ایجاد نماید. در یک چرخه یادگیری، آن قدر سعی و خطا انجام می‌گردد تا نهایتاً راه‌حل مزبور بهبود یابد و یا اینکه تعداد تکرار مشخصی از چرخه یادگیری انجام پذیرد. پس از آن، راه‌حل بهبودیافته به چرخه الگوریتم بهینه‌سازی کواتی بازگردانده می‌شود. جهت ارزیابی کارایی روش پیشنهادی، این روش برای تولید خودکار داده آزمون روی تعدادی برنامه مختلف مورد استفاده قرار گرفته و نتایج حاصل از آن با نتایج چندین الگوریتم فراابتکاری معروف مورد مقایسه قرار گرفته شده است، نتایج نشان از برتری مشهود روش پیشنهادی دارند.

۲- کارهای انجام‌گرفته

در روش آزمون پویا برنامه تحت آزمون باید اجرا شود؛ به همین خاطر باید ورودی‌هایی برای برنامه تحت آزمون تولید کنیم. در این روش با استفاده از الگوریتم‌های جستجو می‌توان این ورودی‌ها را تولید کرد. تاکنون الگوریتم‌های مختلفی برای این کار به کار گرفته شده که در این بخش مهم‌ترین این کارها بررسی شده است [۹].

آشوینی و همکاران با استفاده از الگوریتم بهینه‌سازی ذرات و ارتباط بین زمان اجرا و تعداد خطاها تولید داده آزمون و پیش‌بینی خطا را انجام داده‌اند [۱۰]. آودنیکو و سردیکو [۱۱] از دو نسخه الگوریتم ژنتیک با معیار پوشش جملات استفاده کرده‌اند. نصرتی و همکاران از برنامه‌نویسی ژنتیک نیز به‌عنوان یک الگوریتم مکاشفه‌ای برای جستجوی داده‌های آزمون استفاده کردند [۱۲]. به‌منظور بهبود بیشتر توانایی جستجوی سراسری در الگوریتم ژنتیک، روش‌های بهبودیافته مختلفی معرفی شده است. الگوریتم ژنتیک چندجمعیتی در سال‌های اخیر به‌عنوان نوعی الگوریتم ژنتیک بهبودیافته معرفی شده که عملکرد خوبی از خود نشان داده است [۱۳] و [۱۴]. مدل چندجمعیتی می‌تواند از گیرافتادن الگوریتم در مینیمم‌های محلی و همگرایی زودرس جلوگیری کند. از طرف دیگر، مدل چندجمعیتی می‌تواند کیفیت راه‌حل‌ها و سرعت تکامل الگوریتم ژنتیک را بهبود بخشد. در [۱۵] از الگوریتم ژنتیک چندجمعیتی برای به دست آوردن راه‌حل بهینه استفاده شده است. در این روش از دو جمعیت فرزند و یک جمعیت اصلی استفاده شده و جمعیت‌های فرزند به‌صورت موازی اجرا می‌شوند. همچنین در این روش از یک تابع برازندگی جدید استفاده شده است. آزمایش‌ها نشان داده‌اند که این روش در سرعت همگرایی، زمان جستجو و درصد پوشش باعث بهبود شده و کارایی آن نسبت به الگوریتم ژنتیک تک‌جمعیتی و جستجوی تصادفی بهتر است. دو روش مختلف برای تنظیم مقادیر پارامترهای (نرخ بازترکیبی و نرخ جهش) الگوریتم ژنتیک وجود دارد: تنظیم مقادیر پارامترها قبل از فرایند بهینه‌سازی یا تنظیم دینامیکی پارامترها در حین اجرا. در [۱۶] و [۱۷] از الگوریتم ژنتیک بهبود داده شده به‌وسیله نگهداری تنوع جمعیت برای تولید داده آزمون استفاده شده که به‌صورت پویا در حین اجرا نرخ عملگر بازترکیبی و عملگر جهش را با شباهت بین کروموزوم‌ها و مقدار برازندگی کروموزوم‌ها در هر مرحله از الگوریتم به دست می‌آورد. ماکش و پرادیپ تامار در سال ۲۰۱۸ روشی مبتنی بر الگوریتم ژنتیک برای تولید خودکار داده آزمون نرم‌افزار معرفی کردند و نتایج آنها با روش تصادفی مورد مقایسه قرار گرفت. در این مقاله تأثیر جمعیت اولیه در کارایی الگوریتم ژنتیک مورد بررسی قرار گرفت. نتایج آنها نشان داد که روش پیشنهادی آنها نسبت به روش تصادفی مؤثرتر است و به زمان کمتری برای تولید داده آزمون نرم‌افزار نیاز دارد و همچنین با افزایش اندازه اولیه جمعیت، می‌توان فضای جستجوی بیشتری را با افزایش تنوع ایجاد کرد که باعث می‌شود با احتمال کمتری الگوریتم در بهینه‌های محلی قرار گیرد [۱۸]. در [۱۹] از الگوریتم ژنتیک و الگوریتم تبرید شبیه‌سازی‌شده برای خودکارسازی تولید داده آزمون مبتنی بر معیار پوشش مسیر استفاده شده و نتایج آنها با هم مقایسه گردیده است. نتایج آنها نشان داده که الگوریتم ژنتیک با تنظیم درست پارامترها، مؤثرتر از الگوریتم تبرید شبیه‌سازی شده است و در تعداد تکرار کمتری به جواب رسیده و حداکثر پوشش را به دست آورده است. ژیو و همکاران از الگوریتم بهینه‌سازی ازدحام ذرات تطبیقی برای تولید خودکار داده آزمون استفاده کردند. آنها وزن ایستایی را بر اساس مقدار برازندگی هر ذره محاسبه کردند. نتایج آزمایش آنها نشان داد که کارایی روش پیشنهادی آنها از الگوریتم ژنتیک معمولی و الگوریتم ازدحام ذرات معمولی بهتر است [۲۰]. سانجی سینگلا و همکاران با استفاده از ترکیبی از الگوریتم ژنتیک و بهینه‌سازی ازدحام ذرات، رویکردی برای تولید خودکار داده‌های آزمون ارائه دادند. عملکرد روش پیشنهادی در مقایسه با الگوریتم ژنتیک معمولی و الگوریتم بهینه‌سازی ازدحام ذرات برای تولید خودکار داده آزمون حاکی از برتری روش پیشنهادی آنها داشت [۲۱]. جیانگ و همکاران با استفاده از الگوریتم بهینه‌سازی ازدحام ذرات، رویکردی برای تولید داده آزمون ارائه دادند. آنها یک تنظیم انطباقی را بر اساس وزن ایستایی پیشنهاد دادند که قابلیت جستجو بین جستجوی سراسری و جستجوی محلی را متعادل می‌کند. نتایج روش‌های آنها با دیگر نسخه‌های بهبود داده شده الگوریتم ازدحام ذرات مقایسه شد. نتایج آزمایش بهبود سرعت همگرایی و اثربخشی روش پیشنهادی آنها را نشان داد [۲۲]. شریف‌پور و همکاران در سال ۲۰۱۸ یک روش با استفاده از الگوریتم کلونی مورچه‌ها به روش تقلیدی برای تولید خودکار داده آزمون ارائه دادند. نتایج تجربی برتری الگوریتم پیشنهادی آنها نسبت به تکنیک‌های تولید داده آزمون موجود از نظر پوشش شاخه و سرعت همگرایی در مقایسه با الگوریتم تکاملی ژنتیک و الگوریتم تکامل ازدحام ذرات را نشان می‌دهد [۲۳]. دمیا و همکاران در سال ۲۰۲۰ روش جدیدی برای تولید داده آزمون استفاده کردند. آنها از الگوریتم تولید مثل غیرجنسی برای تنظیم تنوع جمعیت و کمک به جستجوی محلی در الگوریتم کرم شبتاب استفاده کردند. نتایج آنها نشان داد که این روش ترکیبی از هر کدام از الگوریتم‌ها به‌صورت جداگانه بهتر است [۲۴]. در [۲۵] از الگوریتم بهینه‌سازی ازدحام ذرات به‌منظور تولید خودکار داده آزمون نرم‌افزار استفاده شده است. در این روش وزن اینرسی به‌صورت دینامیکی در هر دور از الگوریتم با توجه به برازندگی هر ذره محاسبه می‌شود. آزمایش‌ها بر روی برنامه‌های مختلف انجام شده و نتایج آزمایش‌ها نشان می‌دهد که روش پیشنهادی آنها نسبت به چندین روش انجام‌شده توسط انواع دیگر PSO، نرخ هم‌گرایی بهتری دارد. در [۲۶] از یک الگوریتم بهینه‌سازی مبتنی بر یادگیری- آموزش برای برای تولید خودکار داده آزمون استفاده شده است. آنها نتایج خود را با سایر روش‌های پیشرفته بر اساس پوشش مسیر برای ده برنامه جاوا مقایسه کرده‌اند. نتایج به‌دست‌آمده حاکی از برتری روش پیشنهادی آنها دارد. سعادت‌جو، بابامیر

و همکاران در سال ۲۰۱۸ از الگوریتم رقابت استعماری به‌منظور تولید خودکار داده آزمون استفاده کردند. روش آنها با الگوریتم ژنتیک و الگوریتم بهینه‌سازی ازدحام ذرات برای سه برنامه نرم‌افزاری با اندازه‌های فضای جستجوی مختلف پیاده‌سازی مورد مقایسه قرار گرفت. روش پیشنهادی نتایج بهتری نسبت به سایر الگوریتم‌ها در یافتن تعداد مسیرهای غیرتکراری، سرعت همگرایی و زمان محاسباتی با افزایش فضای جستجوی به دست آورد [۲۷]. در [۲۸] یک روش تولید خودکار داده آزمون نرم افزار مبتنی بر الگوریتم تکامل تفاضلی معرفی شده است. نتایج تجربی نشان می‌دهد که در مقایسه با سایر الگوریتم‌های پیشرفته، تکامل تفاضلی با معیار پوشش شاخه می‌تواند به طور قابل توجهی سرعت همگرایی را بهبود بخشد. ساهو و همکاران [۲۹] در پژوهش خود، با بهره‌گیری از الگوریتم بهینه‌سازی جنگل، رویکردی نوین برای تولید داده‌های آزمون ارائه کردند که قادر است در یک اجرای واحد، چندین مسیر از برنامه تحت آزمون را به‌طور هم‌زمان پوشش دهد. این الگوریتم در متلب پیاده‌سازی شده و عملکرد آن با استفاده از شش برنامه ارزیابی شد و نتایج نشان می‌دهند که نتایج خوبی کسب کرده است. بنابراین با توجه به بررسی کارهای انجام‌شده می‌توان گفت که الگوریتم‌های فرامکاشفه‌ای گاهی در بهبود جزئیات راه‌حل‌ها ناکارآمد هستند؛ به طوری که بخش‌هایی از راه‌حل به جواب صحیح نزدیک و بخش‌های دیگر دور می‌شوند. برای رفع این مشکل، از الگوریتم یادگیری q به‌عنوان جستجوی محلی در کنار الگوریتم بهینه‌سازی کوآنتی استفاده شده است. این الگوریتم با تمرکز بر بهترین راه‌حل فعلی و ایجاد تغییرات جزئی در آن، به بهبود راه‌حل کمک می‌کند. نتایج نشان می‌دهند که روش پیشنهادی در مقایسه با سایر الگوریتم‌های فرامکاشفه‌ای عملکرد بهتری دارد.

۳- روش پیشنهادی

۳-۱ مقدمه

مسأله مورد نظر در این مقاله، شناسایی خودکار ورودی‌هایی است که تمامی مسیرهای متناهی موجود در برنامه تحت آزمون را مورد آزمون قرار دهند و هدف، کمینه‌کردن میزان محاسبات لازم برای شناسایی این ورودی‌هاست؛ چرا که با بزرگ‌شدن برنامه و افزایش تعداد توابع در آن، زمان زیادی برای تولید این موارد آزمون نیاز خواهد بود و هرچه راهکار خودکار سریع‌تر و با محاسبات کمتری بتواند تمامی مسیرها را پوشش دهد، بهتر خواهد بود. روش پیشنهادی یک الگوریتم فراابتکاری جدید به نام الگوریتم بهینه‌سازی کواتی (COA) است که در آن یادگیری q به‌عنوان یک روش جستجوی محلی در درون ساختار این الگوریتم مورد استفاده قرار می‌گیرد. تا جایی که نگارنده جستجو کرده است، این رویکرد ترکیبی تاکنون در خصوص خودکارسازی تولید داده آزمون نرم‌افزار استفاده نشده است. ایده اساسی COA شبیه‌سازی دو رفتار طبیعی کواتی ها است: ۱) رفتار آنها هنگام حمله و شکار ایگوانا و ۲) فرار آنها از دست شکارچیان. در این الگوریتم هر عضو جمعیت به‌عنوان یک کواتی شناخته می‌شود. (همان طور که در الگوریتم ژنتیک هر عضو جمعیت یک کروموزوم است.)

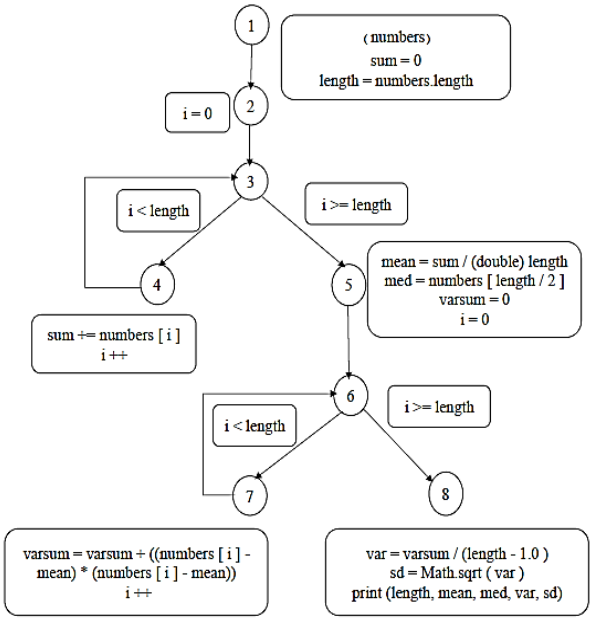
به طور کلی روش انجام کار به این صورت است که ابتدا مسأله مورد نظر فرموله‌سازی می‌شود و سپس گراف جریان برنامه ایجاد می‌شود و مسیرهای این گراف تولید می‌شود. (هدف پوشش تمامی مسیرهای متناهی موجود در متد تحت آزمون است.) بعد از ایجاد جمعیت اولیه به‌صورت تصادفی، مقدار برازندگی هر کواتی محاسبه می‌شود و در نتیجه وارد فاز تکرار می‌شوند. در این قسمت اگر برازندگی بهترین کواتی از یک حد از پیش تعیین‌شده بهتر بود، وارد مرحله جستجوی محلی (عملیات بهبود) می‌شویم. در این مرحله ابتدا یک سری پیش‌پردازش روی مسیرها برای به دست آوردن شبیه‌ترین زیربخش به زیربخش مربوط به مسیر پوشش داده نشده، انجام می‌شود و با به‌کارگیری فرآیند یادگیری تقویتی، مجموعه‌ای از حالت‌ها و اعمال تعریف می‌شود تا عامل یادگیرنده بتواند با استفاده از پاداش‌های دریافتی، زیربخش مربوط به بهبود راه‌حل‌ها را بهینه‌سازی کند. بعد از این مرحله، دو فاز اصلی الگوریتم کواتی انجام می‌شود. در این الگوریتم فاز اول استراتژی شکار و حمله به ایگوانا (مرحله اکتشاف) و فاز دوم فرایند فرار از دست شکارچیان (مرحله بهره‌برداری) است. این چرخه تکرار می‌شود تا الگوریتم به جواب پایانی همگرا شود. شرط توقف در الگوریتم‌ها، رسیدن به یک تعداد مشخص ارزیابی است.

۳-۲ فرموله‌سازی مسأله

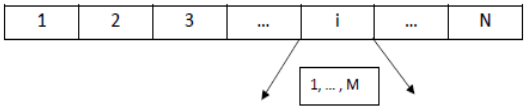
در روش آزمون مسیر سعی می‌گردد تا تمامی مسیرهای کنترلی متد تحت آزمون حداقل یک بار اجرا شوند تا بدین صورت، تمامی دستورهای برنامه اجرا شده باشند. برای انجام این آزمون، ساختار جریان کنترلی متد تحت آزمون به‌صورت گراف جریان در نظر گرفته می‌شود. این گراف جریان کنترلی، منطق درون برنامه را نشان می‌دهد و ساختار برنامه را می‌توان به‌صورت چنین گرافی بیان کرد. در این روش برای استخراج حالات آزمون، با استفاده از طراحی و یا کد منبع گراف، جریان متناظر آن رسم می‌گردد. میزان پیچیدگی منطق درون متد برای گراف جریان تعیین می‌شود و برای هر کدام از مسیرهای مستقل، یک حالت آزمون آماده می‌گردد تا تضمین کند که آن حالت، حتماً اجرا می‌شود. آزمون مبتنی بر پوشش مسیر، جامع‌ترین نوع آزمون ساختاری محسوب می‌شود، زیرا با بررسی تمامی مسیرهای اجرایی برنامه، قادر است خطاهای بیشتری را نسبت به سایر روش‌های پوشش شناسایی کند. تا به امروز، تحقیقات موجود در مورد تولید داده‌های آزمایشی برای پوشش مسیر یا بر اساس رویکردهای تکاملی بوده است یا اجرای نمادین [۳۰]. آزمایش مسیرهای متناهی با در نظر گرفتن کل توالی از عبارات فراتر می‌رود؛ در حالی که دیگر معیارها این گونه نیست. به عنوان مثال در معیار آزمون پوشش شاخه، بر آزمایش هر نقطه تصمیم تمرکز می‌کند و یا همچنین در معیار پوشش جملات بر روی یک جمله تمرکز دارد که ممکن است برخی از جریان‌های منطقی برنامه را از دست بدهد؛ ولی در روش مسیرهای متناهی کل جملات را در نظر می‌گیرد و به طور بالقوه تعاملات پیچیده بین بخش‌های مختلف کد را آشکار می‌کند. در این مقاله از معیار پوشش همه مسیرهای متناهی استفاده شده که هدف، پوشش تمامی مسیرهای متناهی موجود در برنامه تحت آزمون است.

گراف جریان برنامه، جریان کنترل منطقی متد را با استفاده از مسیرها به نمایش می‌گذارد که این مسیرها در حقیقت همان مسیر حرکت ورودی به خروجی هستند. یعنی یک گراف جریان برنامه همه اجراهای یک قطعه کد را با توصیف ساختارهای کنترلی آن مدل‌سازی می‌کند. شکل ۱ گراف جریان برنامه قطعه کد مثال شکل ۲ را نشان می‌دهد.

مسأله‌ای که در اینجا مورد نظر می‌باشد، تعیین‌کردن خودکار مجموعه ورودی‌ها برای آزمون یک متد مشخص است تا اینکه بتواند تمامی مسیرهای N_p را پوشش دهد. بدین منظور از جستجوی روی فضای پارامترهای ورودی متد استفاده می‌شود. یک متد به‌صورت کلی به شکل ([input parameter list]) method_name [return type] معرفی می‌شود. فضای ورودی‌های این متد که باید جستجو روی آن انجام پذیرد



شکل ۱: گراف جریان برنامه قطعه کد مثال ۱.



شکل ۳: نمونه‌ای از کواتی‌های جمعیت.

با توجه به لیست پارامترهای ورودی متد تعریف می‌شود. یعنی اگر متد دارای مثلاً ۲ پارامتر ورودی باشد، فضای ورودی‌ها متشکل از زوج‌های دوتایی $\langle i_1, i_2 \rangle$ خواهد بود که i_1 مقدار منتسب‌شده به پارامتر اول و i_2 مقدار منتسب‌شده به پارامتر دوم را نشان می‌دهد. هدف اصلی در این جستجو کمینه‌کردن میزان محاسبات لازم برای شناسایی ورودی‌هاست؛ چرا که با بزرگ‌شدن برنامه و افزایش تعداد توابع در آن، زمان زیادی برای تولید این موارد تست نیاز خواهد بود و هرچه راهکار خودکار سریع‌تر و با محاسبات کمتری بتواند تمامی مسیرها را پوشش دهد، بهتر خواهد بود.

۳-۳ به دست آوردن گراف جریان برنامه

برای به دست آوردن گراف جریان برنامه می‌توانیم با استفاده از نرم‌افزاری مثل visustin یا با استفاده از کتابخانه pycfg یا staticfg در زبان برنامه‌نویسی پایتون، گراف جریان برنامه تحت آزمون را به دست بیاوریم و سپس مسیرهای این گراف را از گراف مورد نظر استخراج کنیم.

۳-۴ ایجاد جمعیت اولیه

پس از تولید گراف جریان برنامه و سپس به دست آوردن مسیرهای این گراف، یک جمعیت اولیه تصادفی تولید می‌شود. اندازه هر کواتی (راه‌حل) این جمعیت به تعداد متغیرهای متد تحت آزمون و تعداد مسیرها بستگی دارد. به طور کلی در متدی که تعداد متغیرهای ورودی آن M و تعداد مسیرهای گراف جریان آن N است، هر کواتی این جمعیت N زیربخش دارد و اندازه هر زیربخش برابر M است. شکل ۳ یک نمونه از این کواتی‌ها را نشان می‌دهد.

۳-۵ محاسبه برازندگی جمعیت کواتی‌ها

بعد از ایجاد جمعیت تصادفی اولیه، برای تعیین میزان برازندگی هر عضو جمعیت، لازم است که N بار متد تحت آزمون اجرا شود تا مشخص شود که چه مسیرهایی توسط مجموعه آزمون ورودی پوشش

```
public static void compute Stats (int [ ] numbers)
{   int length = numbers.length;
        double med, var, sd, mean, sum, varsum;
        sum = 0;
        for (int i = 0; i < length; i++)
        {    sum += numbers [ i ];
        }
        mean = sum / (double) length;
        med = numbers [ length / 2 ];
        varsum = 0;
        for (int i = 0; i < length; i++)
        {   varsum = varsum + ((numbers [ i ] - mean) *
        (numbers [ i ] -mean));
        }
        var = varsum / ( length - 1.0 );
        sd = Math.sqrt ( var );
        System.out.println ("length:              " +
        length);
        System.out.println ("mean:                " + mean);
        System.out.println ("median:              " + med);
        System out println ("variance :           " + var);
        System.out.println ("standard deviation:  " + sd);
}
```

شکل ۲: قطعه کد گراف جریان برنامه شکل ۱.

داده شده‌اند. مقدار برازندگی هر کواتی طبق (۱) محاسبه می‌شود

$$fitnessCoati(i) = \frac{k_i}{R} \qquad (۱)$$

در (۱)، K_i برابر با تعداد مسیرهای پوشش داده شده توسط کواتی iام و R برابر با تعداد کل مسیرهاست.

۳-۶ عملیات بهبود

در این قسمت بعد از شناسایی بهترین کواتی، مسیرهای این کواتی بررسی می‌شود و یکی از مسیرهای پوشش داده نشده در این کواتی را به‌صورت تصادفی از لیست مسیرهای پوشش داده نشده انتخاب می‌کنیم. سپس شبیه‌ترین مسیر از مسیرهای پوشش داده شده توسط کواتی نسبت به مسیر پوشش داده نشده را با استفاده از معیار شباهت جاکارد[1] پیدا می‌کنیم و زیربخش مرتبط با آن را از کواتی بیرون می‌کشیم. نهایتاً این زیربخش را با استفاده از رویکرد یادگیری q و با تعریف عمل‌هایی روی این زیربخش بهبود می‌دهیم.

۳-۷ پیش‌پردازش مسیرها

معیار شباهت جاکارد، معیاری برای مقایسه شباهت یا تفاوت مجموعه نمونه‌های آماری است. میزان شباهت دو مجموعه نمونه با توجه به معیار شباهت جاکارد، از تقسیم تعداد اشتراک دو مجموعه بر تعداد اجتماع دو مجموعه به دست می‌آید. بنابراین این معیار اطلاعات زیادی راجع به وضعیت مسیرها به ما می‌دهد. شاخص تشابه جاکارد دو مجموعه set_1 و set_2 طبق (۲) محاسبه می‌گردد

$$Jaccard(set_1, set_2) = \frac{|set_1 \bigcap set_2|}{|set_1 \bigcup set_2|} = \frac{|set_1 \bigcap set_2|}{|set_1| + |set_2| - |set_1 \bigcap set_2|} \qquad (۲)$$

1. Jaccard

```
if  x != y  and  x != z  and  y != z :
    print ( " scalene triangle " )
else :
  if  x == y  and  x == z  and  y == z :
    print ( " equilateral triangle " )
else :
    print ( " isosceles triangle " )
return x , y , z
```

شکل ۴: برنامه نمونه.

همچنین فاصله جاکارد که میزان تفاوت دو مجموعه نمونه را می‌سنجد، با کم‌کردن میزان شباهت جاکارد از یک طبق (۳) به دست می‌آید

$$Distance(set_1, set_2) = 1 - Jaccard(set_1, set_2) = \left|set_1 \bigcup set_2\right| - \frac{\left|set_1 \bigcap set_2\right|}{\left|set_1 \bigcup set_2\right|} \quad (3)$$

در این مرحله بررسی می‌شود که آیا مقدار برازندگی بهترین کواتی جمعیت (X_{best}) از یک حد از پیش تعیین‌شده بیشتر است یا نه. اگر بیشتر نبود، الگوریتم به روند عادی خودش ادامه می‌دهد و در غیر این صورت فرایند بهبود کواتی X_{best} شروع می‌شود. به منظور بهبود، ابتدا یکی از مسیرهای پوشش داده نشده در X_{best} را به‌صورت تصادفی از کل مسیرهای پوشش داده نشده توسط این کواتی انتخاب می‌کنیم. فرض کنید که این مسیر $Path_{ns}$ نام دارد. سپس شبیه‌ترین مسیر در بین مسیرهای پوشش داده شده توسط X_{best} را نسبت به $Path_{ns}$ به دست می‌آوریم و بدین منظور از معیار شباهت جاکارد استفاده می‌کنیم. آن گاه زیربخش مربوط به این مسیر را از کواتی X_{best} بیرون می‌کشیم و عملیات بهبود را بر روی این زیربخش انجام می‌دهیم. معیار شباهت جاکارد که در [۳۱] مورد استفاده قرار گرفته است، برای حلقه‌ها دچار مشکل می‌شود و نمی‌تواند به‌صورت دقیق شبیه‌ترین مسیر را به دست آورد. به‌عنوان مثال قطعه کد زیر را در نظر بگیرید

```
Path = [ ]
Path.append (0)
For   i = 1   to   10 :
    Path.append (i)
            .
            .
            .
Path.append (11)
```

در این کد برای به دست آوردن اطلاعات مسیر مثلاً ۰-۴-۱۱ به اطلاعات مسیر ۰-۳-۱۱ نیاز است که معیار شباهت جاکارد این اطلاعات را نمی‌تواند به دست بیاورد و برای هر سه مسیر ۰-۱-۱۱، ۰-۲-۱۱ و ۰-۳-۱۱ مقدار شباهت یکسانی به دست می‌آورد و وجه تمایزی قائل نمی‌شود. برای رفع این مشکل ما برای حلقه‌ها در این مقاله از معیار شباهت منهتن استفاده کرده‌ایم [۳۲].

۳-۸ بهبود زیربخش‌های کواتی با رویکرد یادگیری Q

بعد از اینکه شبیه‌ترین زیربخش نسبت به مسیر پوشش داده نشده به دست آمد، عملیات یادگیری Q شروع به کار می‌کند. در الگوریتم‌های یادگیری تقویتی، قبل از انجام هر کاری باید ما حالت‌ها[1] و عمل‌هایی[2] را که قرار است روی این حالت‌ها به کار روند، تعریف کنیم. در روش پیشنهادی اگر تعداد متغیرهای ورودی برنامه تحت آزمون برابر M باشد، تعداد حالت‌ها هم برابر M است و عمل‌ها به گونه‌ای تعریف شده که تا حد امکان بین همه برنامه‌ها مشترک باشد. هرچند با توجه به تنوع و ساختار بین برنامه‌ها، عمل‌ها می‌تواند بسته به نوع برنامه از این تعداد بیشتر باشد. در واقع عمل‌هایی که تعریف شده عبارت است از:

- کم‌کردن از مقدار متغیر: متد شکل ۴ را در نظر بگیرید. فرض کنید مقدار شبیه‌ترین زیربخش با توجه به مسیرهای پوشش داده شده برابر [۵,۵,۶] است؛ در این صورت اگر مقدار متغیر سوم یعنی ۶ را یک واحد کاهش بدهیم، مسیر جدیدی ایجاد خواهد شد.
- اضافه‌کردن به مقدار متغیر: متد شکل ۴ را در نظر بگیرید. فرض کنید مقدار شبیه‌ترین زیربخش با توجه به مسیرهای پوشش داده شده برابر [۵,۵,۴] است؛ در این صورت اگر مقدار متغیر سوم یعنی ۴ را یک واحد افزایش بدهیم، مسیر جدیدی ایجاد خواهد شد.
- ثابت‌ماندن مقدار متغیر: متد شکل ۴ را در نظر بگیرید. فرض کنید مقدار شبیه‌ترین زیربخش با توجه به مسیرهای پوشش داده شده برابر [۵,۵,۶] است؛ در این صورت اگر در متغیرهای (حالت‌های) اول و دوم، این عمل انجام شود بهتر است؛ چرا که اگر عمل دیگری انتخاب شود، باعث خرابی این زیربخش می‌شود.
- تنظیم مقدار متغیر با مقدار متغیر بعدی: متد شکل ۴ را در نظر بگیرید. فرض کنید مقدار شبیه‌ترین زیربخش با توجه به مسیرهای پوشش داده شده برابر [۵,۵,۶] است؛ در این صورت اگر مقدار متغیر سوم یعنی ۶ را برابر مقدار متغیر بعدی یعنی ۵ قرار بدهیم، آن گاه مسیر جدیدی ایجاد خواهد شد.
- صفرکردن مقدار متغیر: به‌عنوان مثال، در متدی با روابط زیر، اگر مقدار متغیر var ۶ برابر صفر شود، شرط مورد نظر ارضا می‌شود و مقادیر متغیرهای var ۵ و var ۶ متناسب با قواعد یادگیری Q به‌روزرسانی خواهند شد

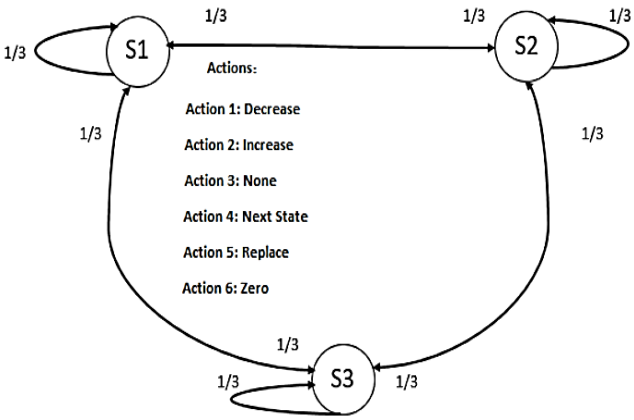
$$f\ var3*2 = var4 + var5*2 + var6$$

اگر در این دستور $var3 = 6$، $var4 = 10$، $var5 = 5$ و $var6 = 2$ باشد و مقدار $var4$ برابر ۰ شود، این دستور ارضا می‌شود. از جهتی دیگر با صفرکردن متغیرهای $var5$ و $var6$ باز هم می‌توانیم به جواب نزدیک‌تر شویم. (یکی از کاربردهایش این است که به جواب نزدیک شویم و هر دو طرف شرط در نزدیکی هم قرار بگیرند و وقتی که در نزدیکی یکدیگر قرار بگیرند با انجام‌دادن سایر عمل‌ها می‌توان به جواب نهایی رسید.)
- جابه‌جایی مقدار متغیر با متغیر بعدی یا قبلی: فرض کنید متدی داریم که دستورات آن به این صورت است

```
If  var1 < var2 :
    Print (var1 )
        Else :
    Print (var2)
```

در این متد دو مسیر وجود دارد. مسیر اول به ازای $var1 = 5$ و $var2 = 10$ پوشش داده می‌شود. اکنون اگر مقادیر این دو متغیر با

1. State
2. Action



شکل ۵: فرایند یادگیری تقویتی برای برنامه مثال شکل ۴.

هم تعویض گردد مسیر دوم هم از روی مسیر اول ایجاد می‌گردد.

شکل ۵ فرایند یادگیری تقویتی برای مثال برنامه شکل ۴ را نشان می‌دهد. به‌عنوان نمونه، در برنامه‌ی ارائه‌شده در شکل ۴، تعداد حالات (M) برابر با ۳ و تعداد اعمال ۶ در نظر گرفته شده است؛ به‌طوری‌که در هر حالت، تمامی شش عمل تعریف‌شده قابلیت اجرا دارند. مثلاً در حالت S_1 با انجام هر عمل و با احتمال $1/M$ به هر حالت ممکن برویم و برای سایر حالات هم همین روند انجام می‌شود. برای برآوردکردن عملیات یادگیری تقویتی از یکی از معروف‌ترین الگوریتم‌های این حوزه که یادگیری‌کیو[1] است، استفاده شده است. اکنون در این مرحله باید جدول کیو[2] را ایجاد کنیم که این جدول M سطر و ۶ ستون دارد.

وقتی که الگوریتم شروع به کار می‌کند با یک جدول با مقدار تصادفی شروع به کار می‌کند و بعد از انجام یک عمل در یک حالت عملی که انتخاب شده باعث تغییری در مقدار آن حالت (متغیر ورودی آن تکه از زیربخش) می‌شود. در هر مرحله این اعمال در حالت‌های مختلف ما رخ می‌دهد و بعد از اعمال‌کردن این عمل‌ها روی حالت‌ها، اگر به هدف رسیدیم جایزه[3] مثبت و در غیر این صورت جایزه منفی می‌گیریم تا زمانی که به یک زیربخش خوب تبدیل شود یا تعداد مرحله‌ها از یک مقدار از پیش تعریف شده بیشتر شود. سپس وقتی که یک زیربخش خوب پیدا شد یا تعداد مرحله‌ها از یک مقداری بیشتر شد، مقدار زیربخش بازگشت داده می‌شود به کواتی و کواتی به جمعیت اصلی انتقال داده می‌شود.

۳-۹ فاز اول الگوریتم کواتی

مرحله اول به‌روزرسانی جمعیت کواتی‌ها در فضای جستجو بر اساس شبیه‌سازی استراتژی آنها هنگام حمله به ایگواناها مدل‌سازی شده است. در این استراتژی، گروهی از کواتی‌ها از درخت بالا می‌روند تا به یک ایگوانا برسند و آن را بترسانند. در طبیعت، تعدادی از کوآتی‌ها در زیر درخت منتظر می‌مانند تا ایگوانا از درخت سقوط کند؛ پس از آن، فرآیند شکار آغاز می‌شود. این رفتار به‌عنوان الگوی الهام‌بخش در طراحی فاز جست‌وجوی الگوریتم COA به‌کار گرفته شده است.

در طراحی COA، موقعیت بهترین عضو جمعیت، موقعیت ایگوانا[4] در نظر گرفته شده است. همچنین فرض بر این است که نیمی از کواتی‌ها از درخت بالا می‌روند و نیمی دیگر منتظر می‌مانند تا ایگوانا روی زمین بیفتد. بنابراین موقعیت کواتی برآمده از درخت به صورت ریاضی با استفاده از (۴) شبیه‌سازی می‌شود

$$X_i^{P_1}: X_{i,j}^{P_1} = X_{i,j} + r.(Iguana_j = I.X_{i,j}) \quad , \text{for } i = 1,2,\ldots,\left\lfloor \frac{N}{2} \right\rfloor \text{ and } j = 1,2,\ldots,m \tag{۴}$$

در (۵) و (۶)، موقعیت‌های جدید اعضای جمعیت (کوآتی‌ها) بر اساس موقعیت ایگوانا و محدوده جست‌وجو محاسبه می‌شود. رابطه موقعیت اولیه ایگوانا در فضای جست‌وجو از طریق (۵) تعیین می‌شود، در حالی که (۶) فرآیند به‌روزرسانی موقعیت کوآتی‌ها را براساس مقایسه برازندگی و رفتار حمله یا تعقیب مدل می‌کند. این دو رابطه در واقع بیانگر فاز اکتشاف و بهره‌برداری الگوریتم بهینه‌سازی کوآتی هستند که موجب تعادل میان جست‌وجوی سراسری و محلی می‌شود

$$Iguana: Iguana_j^G = lb_j + r.(ub_j - lb_j) \quad , \ i = 1,2,\ldots,m \tag{۵}$$

$$X_i^{P_1}: X_{i,j}^{P_1} = \begin{cases} X_{i,j} + r.(-I.X_{i,j}) & , \ F_{Iguana}^G < F_i \\ X_{i,j} + r.(X_{i,j} - Iguana_i^G) & , \text{ else} \end{cases} \quad \text{for } i = \left\lfloor \frac{N}{2} \right\rfloor + 1, \left\lfloor \frac{N}{2} \right\rfloor + 2,\ldots,N \text{ and } j = 1,2,\ldots,m \tag{۶}$$

رابطه (۷) مرحله‌ی انتخاب نهایی را در الگوریتم COA مدل‌سازی می‌کند. در این مرحله، کوآتی‌ها تنها در صورتی موقعیت جدید خود را می‌پذیرند که برازندگی آن از موقعیت فعلی بهتر باشد. این فرآیند باعث می‌شود الگوریتم از افت کیفیت پاسخ‌ها جلوگیری کرده و به‌تدریج به سمت بهترین راه‌حل‌ها همگرا شود. به‌عبارتی، این رابطه نقش «مکانیزم انتخاب طبیعی» در الگوریتم را ایفا می‌کند و تضمین‌کننده‌ی پایداری همگرایی است.

$$X_i = \begin{cases} X_i^{P_1} & , \ F_i^{P_1} < F_i \\ X_i & , \text{ else} \end{cases} \tag{۷}$$

در اینجا $X_i^{P_1}$ موقعیت جدید محاسبه‌شده برای کواتی iام است، $X_{i,j}^{P_1}$ بعد j ام آن است، $F_i^{P_1}$ مقدار تابع هدف آن است، r یک عدد واقعی تصادفی در بازه [۰,۱] است، در الگوریتم COA، بردار Iguana موقعیت بهترین عضو جمعیت را در فضای جست‌وجو نشان می‌دهد که به‌عنوان نقطه‌ی مرجع برای هدایت سایر کوآتی‌ها در فرآیند بهینه‌سازی به‌کار می‌رود، Iguanaj بعد j ام آن است، I یک عدد صحیح است که به طور تصادفی از بین اعداد ۱ یا ۲ انتخاب می‌شود، IguanaG موقعیت ایگوانا روی زمین است که به طور تصادفی تولید می‌شود، IguanaGj بعد j ام آن، FIguanaG مقدار تابع هدف آن و $\lfloor . \rfloor$ تابع کف است.

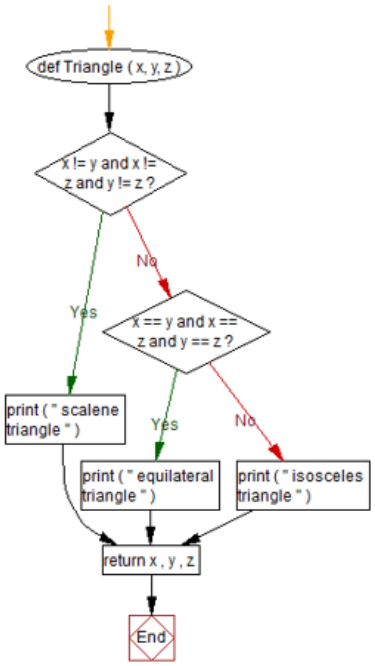
۳-۱۰ فاز دوم الگوریتم کواتی

مرحله دوم فرایند به‌روزرسانی موقعیت کواتی‌ها در فضای جستجو به صورت ریاضی بر اساس رفتار طبیعی کواتی‌ها در هنگام مواجهه با شکارچیان و فرار از شکارچیان مدل‌سازی شده است. هنگامی که یک شکارچی به کواتی حمله می‌کند، این حیوان از موقعیت خود فرار می‌کند. حرکات کواتی در این استراتژی منجر به قرارگرفتن آن در موقعیت امن نزدیک به موقعیت فعلی می‌شود که نشان‌دهنده توانایی بهره‌برداری COA در جستجوی محلی است. برای شبیه‌سازی این رفتار، یک موقعیت تصادفی در نزدیکی موقعیتی که هر پوشش در آن قرار دارد بر اساس (۸)

1. QLearning
2. Qtable
3. Reward
4. Iguana

```
def Triangle ( x, y, z ) :
    if  x != y  and  x != z  and  y != z :
        print ( " scalene triangle " )
    else :
        if  x == y  and  x == z  and  y == z :
            print ( " equilateral triangle " )
        else :
            print ( " isosceles triangle " )
    return x , y , z
```

(الف)



(ب)

شکل ۶. (الف) برنامه دسته‌بندی مثلث و (ب) گراف جریان متناظر آن.

و (۹) شبیه‌سازی می‌شود. موقعیت بهتر طبق (۷) جایگزین می‌شود

$$lb_j^{local} = \frac{lb_j}{t} \ , \ ub_j^{local} = \frac{ub_j}{t} \ , \ \text{where } t = ۱,۲,\ldots,T \qquad (۸)$$

$$X_i^{P۲} : X_{i,j}^{P۲} = X_{i,j} + (۱-۲r).(lb_j^{local} + r.(ub_j^{local} - lb_j^{local})) \qquad (۹)$$
$$, \ i = ۱,۲,\ldots,N \ , \ j = ۱,۲,\ldots,m$$

۳-۱۱ شبه‌کد الگوریتم پیشنهادی

۱) تهیه گراف جریان متد تحت آزمون و به دست آوردن تعداد مسیرهای آن
۲) ایجاد یک جمعیت تصادفی اولیه از راه‌حل‌ها
۳) ارزیابی جمعیت
۴) تا زمانی که به هدف نرسیده‌ایم تکرار کن:
a. انتخاب راه‌حل‌های مناسب‌تر از جمعیت
b. اگر در بهترین راه‌حل به‌دست‌آمده، مقدار برازندگی از حد مشخصی بهتر بود، آن را انتخاب کن و برو به مرحله h
c. فاز اول الگوریتم
d. فاز دوم الگوریتم
e. ترکیب راه‌حل به‌دست‌آمده از مرحله b در صورت اجرای آن و جمعیت فعلی در d
f. ارزیابی جمعیت
g. انجام جستجو بین همه مسیرهای به‌دست‌آمده تکرار فعلی
h. با استفاده از معیار شباهت جاکارد، شبیه‌ترین مسیر به مسیر محقق‌نشده داخل مجموعه مسیرهای راه‌حل را به دست بیاور و زیربخش مربوط به شبیه‌ترین مسیر را انتخاب کن و برو به مرحله i.
i. به کار بردن الگوریتم یادگیری کیو برای بهبود آن زیربخش و بازگشت نتیجه

جدول ۱: تنظیم پارامتر الگوریتم‌ها.

Algorithms	Parameter	Value
AGA IGA	Crossover Rate	Adaptive
	Mutation Rate	
	Selection	
APSO	C۱	Adaptive
	C۲	Adaptive
	W	Adaptive
POA	I	[۱،۲]
	R	۰/۲
PRO	Mutation	۰/۰۶
DO	α	[۰،۱]
	k	[۰،۱]
FA	alpha	۰/۵
	gamma	۱
	β_0	۱

٤- آزمایش و نتایج

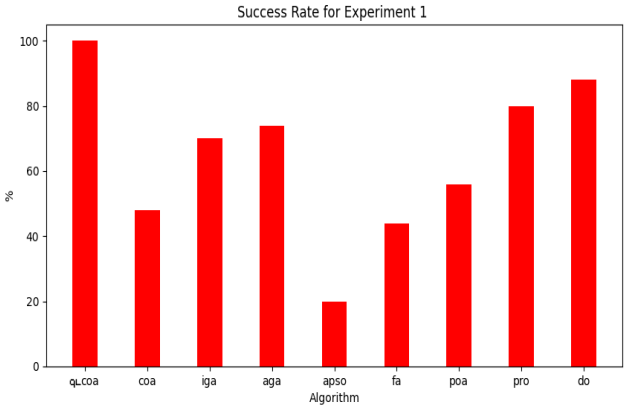
آزمایش‌ها به ازای برنامه‌های معروف و مختلفی که در اکثر مقالات مهم در این حوزه وجود دارد، صورت گرفته و سعی گردیده که تمام ساختارهای برنامه‌نویسی در فرآیند تحلیل ساختار برنامه، اجزای کنترلی مانند حلقه‌ها[1] و شرط‌ها[2] شناسایی شوند تا مسیرهای ممکن اجرای برنامه برای تولید داده‌های آزمون استخراج گردند. هر برنامه ساختارهای شرطی و تودرتویی متفاوتی نسبت به سایر برنامه‌ها دارد. برای انجام ارزیابی‌ها از دو معیار زیر استفاده شده است:

- میانگین تعداد دفعات فراخوانی تابع برازندگی
- نرخ موفقیت[3] که بر اساس (۱۰) محاسبه می‌شود

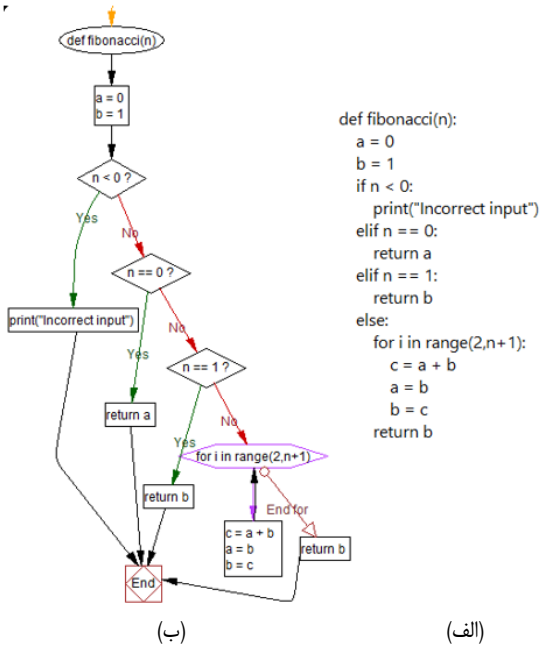
$$SR = \frac{\sum_{i}^{ps} bbci}{ps} \qquad (۱۰)$$

در (۱۰) ps تعداد اجرای الگوریتم و bbc هم یک پرچم[4] بولین است که اگر الگوریتم از یک حد ثابت فراخوانی تابع برازندگی جواب به حداکثر پوشش رسید، این پرچم برابر یک می‌شود و در غیر این صورت برابر صفر است. الگوریتم پیشنهادی[5] (QLCOA) با نتایج حاصل از دو الگوریتم ژنتیک تطبیقی (IGA و AGA) [۱۶] و [۱۷]، (بهبودیافته با به دست آوردن نرخ جهش و بازترکیبی به صورت پویا در حین اجرای الگوریتم)، الگوریتم بهینه‌سازی ازدحام ذرات تطبیقی (APSO) [۳۳]، الگوریتم بهینه‌سازی پلیکان [۳۴]، الگوریتم بهینه‌سازی ضعیف و غنی (PRO)

1. Loop
2. Conditions
3. Success Rate
4. Flag
5. Q-Learning-Based Coati Optimization Algorithm



شکل ۷: نرخ موفقیت الگوریتم‌ها برای برنامه دسته‌بندی مثلث.



(ب) (الف)

شکل ۸: (الف) دنباله فیبوناچی و (ب) گراف جریان متناظر آن.

[۳۵]، الگوریتم بهینه‌سازی قاصدک (DO) [۳۶]، الگوریتم بهینه‌سازی کرم شبتاب (FA) [۳۷] و الگوریتم کواتی استاندارد (COA) مورد مقایسه قرار گرفته است.

تنظیم پارامتر این الگوریتم‌ها در جدول ۱ نشان داده شده است. در این مقاله نمایش اعداد صحیح در بازه ۱۰۰- و ۱۰۰ در نظر گرفته شده و تعداد اجرا برای هر الگوریتم ۵۰ و حداکثر تعداد فراخوانی تابع برازش هم ۱۰۰۰۰۰ در نظر گرفته شده است.

٤-۱ برنامه تحت تست ۱

برنامه دسته‌بندی مثلث که یکی از برنامه‌های معروف در این حوزه است به زبان برنامه‌نویسی پایتون نوشته شده و گراف جریان مربوط به آن در شکل ۶ را در نظر بگیرید. در گراف جریان شکل ۶ تعداد مسیرها برابر ۳ و تعداد متغیرها برابر ۳ است. برای مثال، چند نمونه داده تولیدشده توسط الگوریتم پیشنهادی برای این برنامه به صورت زیر است

[۲۳،۷۸،۱۲،۵،۵،۵،۶۶،۶۶،۸۹]

[۱،۱،۱،۵۵،۵۵،۸،۴۵،۷۹،۱۲]

[۳۴،۳۴،۵،۹۰،۹۰،۹۰،۷،۳۴،۴۱]

[۶۰،۶۰،۶۰،۵،۷۷،۸۲،۴۵،۴۵،۹۸]

[-۳،-۳،۳،۶۷،۸۹،۴۵،۶،۶،۸۰]

جدول ۲: نتایج برنامه دسته‌بندی مثلث.

Algorithms	MNFE	std	p-value	best	worst	pc
IGA	۱۷۷۹٫۵	۱۷۵۸٫۱	۰٫۰	۹۵	۷۱۷۵	۱۰۰٪
AGA	۱۳۶۷٫۹	۱۱۸۶٫۳	۰٫۰	۲۱۵	۵۶۴۵	۱۰۰٪
APSO	۳۸۳۵٫۵۶	۲۰۳۷٫۴	۰٫۰	۵۸۰	۸۵۰۶	۱۰۰٪
POA	۲۱۸۶٫۴	۱۴۳۳٫۴	۰٫۰	۴۴۰	۶۴۲۰	۱۰۰٪
PRO	۱۲۷۸٫۴	۷۷۹٫۷	۰٫۰	۱۸۰	۳۹۰۰	۱۰۰٪
DO	۱۲۳۸٫۸	۵۱۸٫۵	۰٫۰	۳۴۰	۲۹۲۰	۱۰۰٪
FA	۲۹۶۶٫۰۲	۲۵۱۰٫۱	۰٫۰	۵۵۶	۱۱۴۳۵	۱۰۰٪
COA	۲۹۰۵٫۸	۲۶۵۴٫۵	۰٫۰	۱۸۰	۱۲۶۴۰	۱۰۰٪
QLCOA	۶۹٫۱۶	۹٫۰۶	۰٫۰	۴۱	۸۷	۱۰۰٪

جدول ۳: نتایج برنامه فیبوناچی.

Algorithms	MNFE	std	p-value	best	worst	pc
IGA	۶۷۶٫۴	۵۰۵٫۸	۰٫۰	۹۵	۲۳۱۵	۱۰۰٪
AGA	۷۲۵٫۶	۴۹۳٫۱	۰٫۰	۹۰	۱۷۶۰	۱۰۰٪
APSO	۹۹۹٫۲	۹۸۸٫۰	۰٫۰	۱۸۵	۵۲۵۵	۱۰۰٪
POA	۹۱۹٫۷	۶۳۲٫۹	۰٫۰	۱۴۰	۲۷۳۵	۱۰۰٪
PRO	۵۳۵٫۱	۳۵۶٫۶	۰٫۰	۱۱۰	۱۷۰۰	۱۰۰٪
DO	۸۷۵٫۰	۶۰۳٫۴	۰٫۰	۱۵۵	۲۵۲۵	۱۰۰٪
FA	۹۲۰٫۹	۶۷۸٫۹	۰٫۰	۸۰	۲۶۹۰	۱۰۰٪
COA	۶۲۷٫۸	۴۹۷٫۳	۰٫۰	۸۰	۲۲۵۵	۱۰۰٪
QLCOA	۲۰۷٫۶۲	۱۳۳٫۷	۰٫۰	۷۲	۸۴۱	۱۰۰٪

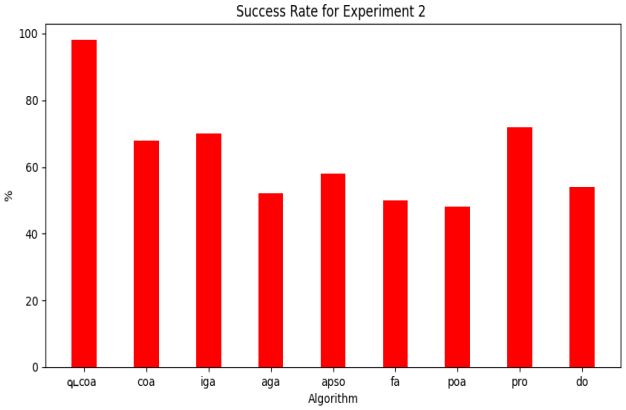
جدول ۴: نتایج برنامه محاسبه ریشه‌های رابطه درجه دوم.

Algorithms	MNFE	std	p-value	best	worst	pc
IGA	۱۳۴۷٫۲	۱۲۴۴٫۵	۰٫۰	۱۴۰	۵۶۱۵	۱۰۰٪
AGA	۱۰۴۴٫۸	۹۴۷٫۳	۰٫۰	۱۴۰	۴۶۱۰	۱۰۰٪
APSO	۸۷۹۷٫۴۸	۶۶۹۴٫۴	۰٫۰	۱۲۷۹	۲۷۲۲۸	۱۰۰٪
POA	۷۲۹٫۲	۵۹۲٫۹	۰٫۰	۴۰	۲۳۸۰	۱۰۰٪
PRO	۳۰۴٫۸	۲۹۰٫۷	۰٫۰	۴۰	۱۴۰۰	۱۰۰٪
DO	۱۲۳۵٫۲	۵۵۹٫۵	۰٫۰	۲۶۰	۲۸۸۰	۱۰۰٪
FA	۷۶۴۹٫۱۶	۷۴۷۳٫۱	۰٫۰	۶۱۲	۳۴۰۴۷	۱۰۰٪
COA	۶۶۳٫۲	۴۸۲٫۰	۰٫۰	۱۰۰	۳۲۲۰	۱۰۰٪
QLCOA	۷۴٫۶	۱۱٫۱۱	۱٫۰	۶۲	۹۹	۱۰۰٪

جدول ۵: نتایج برنامه تحت تست ۴.

Algorithms	MNFE	std	p-value	best	worst	pc
IGA	۱۳۴۷٫۲	۱۲۴۴٫۵	۰٫۰	۱۴۰	۵۶۱۵	٪۱۰۰
AGA	۱۰۴۴٫۸	۹۴۷٫۳	۰٫۰	۱۶۰	۴۶۱۰	٪۱۰۰
APSO	۸۷۹۷٫۴۸	۶۶۹۴٫۴	۰٫۰	۱۲۷۹	۲۷۲۲۸	٪۱۰۰
POA	۷۲۹٫۲	۵۹۲٫۹	۰٫۰	۴۰	۲۳۸۰	٪۱۰۰
PRO	۳۰۴٫۸	۲۹۰٫۷	۰٫۰	۴۰	۱۴۰۰	٪۱۰۰
DO	۱۲۳۵٫۲	۵۵۹٫۵	۰٫۰	۲۶۰	۲۸۸۰	٪۱۰۰
FA	۷۶۴۹٫۱۶	۷۴۷۳٫۱	۰٫۰	۶۱۲	۳۴۰۴۷	٪۱۰۰
COA	۶۶۳٫۲	۶۸۲٫۰	۸×۱۰^{-۶}	۱۰۰	۳۲۲۰	٪۱۰۰
QLCOA	۷۴٫۶	۱۱٫۱	۱٫۰	۵۰	۱۱۰	٪۱۰۰

جدول ۲ نتایج حاصل و شکل ۷ نرخ موفقیت را برای این الگوریتم‌ها نشان می‌دهد. در جداول ۲ تا ۵ ستون اول معرف الگوریتم‌ها، ستون دوم میانگین تعداد ارزیابی‌ها در ۵۰ بار اجرا به ازای هر الگوریتم (MNFE)،



شکل ۹: نرخ موفقیت الگوریتم‌ها برای برنامه فیبوناچی.

ستون سوم انحراف معیار (std)، ستون چهارم مقدار p_value، ستون پنجم بهترین تعداد ارزیابی (best)، ستون ششم بدترین تعداد ارزیابی (worst) و ستون هفتم درصد پوشش مسیرها (pc) را نشان می‌دهد.

٤-٢ برنامه تحت تست ٢

برنامه بعدی که می‌خواهیم مورد آزمون قرار دهیم دنباله فیبوناچی است. این برنامه یک بنچمارک معروف است که در اکثر مقالات مهم به کار رفته است. در این مقاله هدف از آزمون این برنامه، آزمون حلقه است. دنباله یا سری فیبوناچی دنباله‌ای از اعداد به شکل زیر است

۰،۱،۱،۲،۳،۵،۸،۱۳،۲۱،...

در این دنباله عدد بعدی با جمع دو عدد ماقبل خود به دست می‌آید و بقیه اعدد نیز به همین ترتیب محاسبه می‌شوند. این برنامه به همراه گراف جریان متناظر آن در شکل ۸ نشان داده شده است. در شکل ۸ تعداد مسیرها بستگی به متغیر ورودی برنامه یعنی n دارد. در این مقاله هر حلقه حداکثر دو مسیر آن پوشش داده شده است. جدول ۳ نتایج حاصل از این الگوریتم‌ها و شکل ۹ نرخ موفقیت را برای این الگوریتم‌ها نشان می‌دهد.

٤-٣ برنامه تحت تست ٣

در این بخش برنامه محاسبه ریشه‌های رابطه درجه دوم[1] را می‌خواهیم مورد آزمون قرار دهیم. هدف از آزمون این برنامه، آزمودن شروطی است که در شرط آنها عبارت محاسباتی وجود دارد. برای حل رابطه درجه دوم ax^2+bx+c می‌توان از (۱۱) استفاده کرد

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a} \qquad (11)$$

ابتدا باید جذر b^2-4ac را محاسبه کرده و بعد از آن سه حالت زیر برقرار خواهد بود:

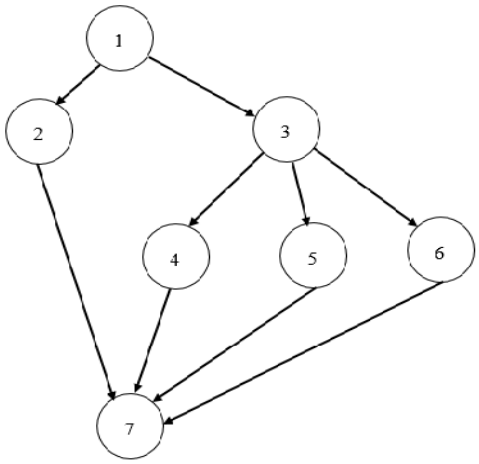
- اگر $b^2-4ac>0$ باشد، آن گاه رابطه دارای دو جواب (ریشه) حقیقی متمایز (۱۲) است

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a} \qquad (12)$$

- اگر $b^2-4ac=0$ باشد، آنگاه رابطه دارای یک جواب حقیقی مضاعف است که با استفاده از (۱۳) به دست می‌آید

```
#  Quadratic equation
# 4 path and 3 variant
def Quadratic_equation(mylist):
    a, b, c=mylist[0],mylist[1],mylist[2]
    d = b * b - 4 * a * c
    list=[]
    list.append(1)
    if a == 0:
        list.append(2)
    else:
        list.append(3)
        if d > 0:
            list.append(4)
        elif d == 0:
            list.append(5)
        else:  # d<0
            list.append(6)
    list.append(7)
    return list
```

(الف)



(ب)

شکل ۱۰: (الف) برنامه محاسبه ریشه‌های رابطه درجه دوم و (ب) گراف جریان متناظر آن.

$$x=\frac{-b}{2a} \qquad (13)$$

- اگر $b^2-4ac<0$ باشد، آنگاه رابطه دارای جواب حقیقی نیست.

برنامه این رابطه به همراه گراف جریان آن در شکل ۱۰ نشان داده شده که این برنامه دارای ۳ متغیر و ۴ مسیر است. جدول ۴ نتایج حاصل از این الگوریتم‌ها را نشان می‌دهد.

٤-٤ برنامه تحت تست ٤

هدف از این آزمون، آزمودن شرط‌های تودرتو و پوشش مسیرهای زیاد است. کد و گراف این برنامه در شکل ۱۱ نشان داده شده است.

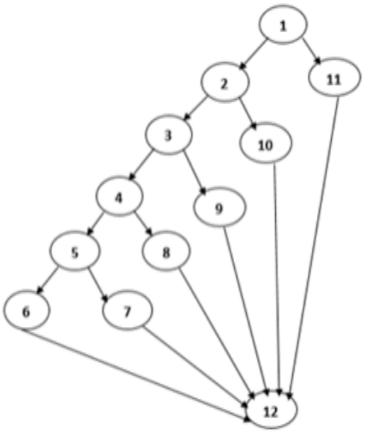
٤-٥ برنامه تحت تست ٥

در این برنامه مسیرهای پیچیده‌تر و مشکل‌تری مورد آزمون قرار گرفته‌اند. با توجه به اینکه یک حد ثابت تعداد ارزیابی برنامه تحت تست در نظر گرفته شده است، پیداکردن این مسیرها برای این الگوریتم‌ها زمانگیر و مشکل است. شکل ۱۲- الف برنامه نوشته‌شده به زبان پایتون این برنامه را نشان می‌دهد. نمودار جریان کنترل (CFG) مربوط به این برنامه در شکل ۱۲- ب ارائه شده است که ارتباط بین بلوک‌های کد و مسیرهای ممکن اجرای برنامه را نشان می‌دهد. در این شکل مسیر ۱-۳-۴-۶ (با رنگ تیره برجسته شده است) یک مسیر سخت و مشکل است؛ چرا که عبارات محاسباتی زیادی در آن وجود دارد و همچنین عباراتی در

1. Quadratic Equation

```
def code ( a, b, c, d, x, y ) :
    if a == b :
            print ( " ********** " )
            if b == c :
                print ( " ********** " )
                if c == d :
                    print ( " ********** " )
                    if d == x :
                            print ( " ********** " )
                            if x == y :
                                    print ( " ********** " )
                            else :
                                    print(' sum')
                    else :
                        print ( " ********** " )
                else :
                    print ( " ********** " )
            else :
                print ( " ********** " )
    else :
            print ( " ********** " )
    return a,b,c,d,x,y
```

(الف)

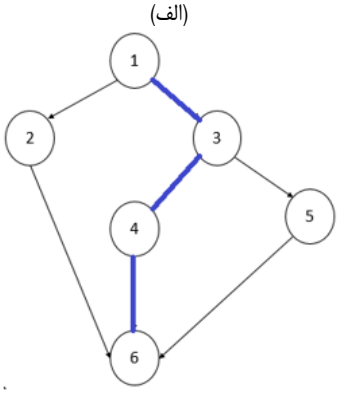


(ب)

شکل ۱۱: (الف) برنامه محاسبه ریشه‌های رابطه درجه دوم و (ب) گراف جریان متناظر آن.

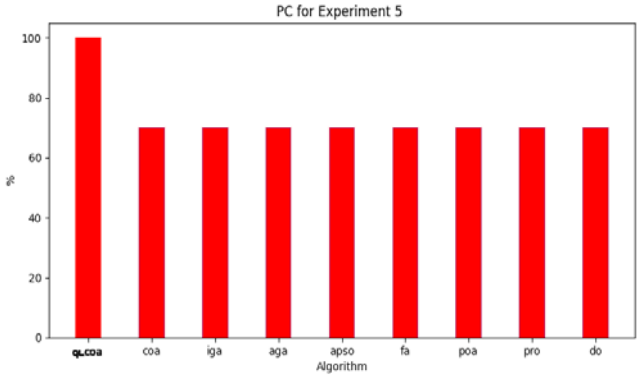
```
def code_test(mylist):
    a, b, c , d ,e= mylist[0], mylist[1],\
                    mylist[2],mylist[3],mylist[4]
    list = []
    list.append(1)
    if (a+c == b-c) and (c == 2) and not (a > 10 or b<0) :
        list.append(2)
    else:
        list.append(3)
        if d == 11 and int(math.gamma(3)) == b and\
                a == (2*3+1) and e == int(math.sqrt(77)):
            list.append(4)
        else:
            list.append(5)
    list.append(6)
    return list
```

(الف)

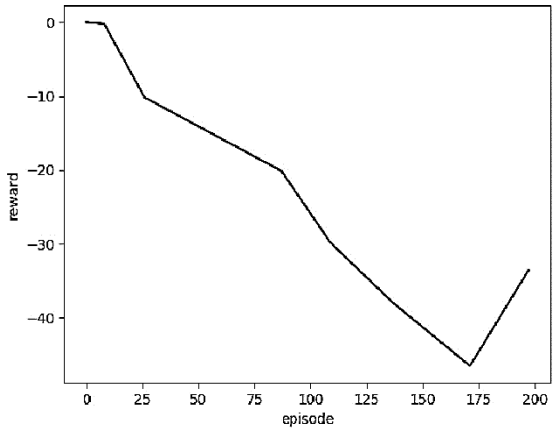


(ب)

شکل ۱۲: (الف) کد برنامه تحت تست و (ب) گراف جریان مربوط به آن.



شکل ۱۳: درصد پوشش مسیر برای الگوریتم‌های مورد مقایسه.



شکل ۱۴: روند همگرایی الگوریتم یادگیری تقویتی.

آن وجود دارد که فقط به ازای یک مقدار ثابت ارضاپذیر هستند. شکل ۱۳ درصد پوشش مسیر حاصل از اجرای الگوریتم پیشنهادی (QLCOA) و سایر الگوریتم‌های مقایسه‌ای را نشان می‌دهد. همان‌طور که دیده می‌شود، QLCOA بیشترین میزان پوشش مسیر را در بین روش‌های بررسی‌شده به‌دست آورده است.

همه الگوریتم‌های مورد مقایسه ۷۰ درصد مسیرها را بعد از ۱۰۰۰۰۰ ارزیابی، پوشش داده‌اند؛ در صورتی که الگوریتم پیشنهادی ۱۰۰ درصد مسیرها را با ۱۹۲۴۰ ارزیابی پوشش داده است. الگوریتم پیشنهادی به دلیل اینکه ساختار درون راه‌حل‌ها را با روش هوشمندانه‌تری به‌روزرسانی می‌کند، از کارایی بیشتر و بهتری نسبت به سایر الگوریتم‌ها برخوردار است.

۵- همگرایی

نمودار همگرایی الگوریتم یادگیری تقویتی نشان می‌دهد که در ابتدا پاداش‌ها نزدیک به صفر بوده و به تدریج با افزایش تعداد اپیزودها کاهش می‌یابد؛ به طوری که تا اپیزود ۱۰۰ به حدود منفی ۲۰ و تا اپیزود ۱۷۵ به حدود منفی ۴۰ می‌رسد. این روند نشان‌دهنده ضعف عملکرد عامل است. اما پس از اپیزود ۱۷۵، افزایش قابل توجهی در پاداش مشاهده می‌شود که نشان‌دهنده بهبود و همگرایی به سمت یک استراتژی بهتر است. به طور کلی، عامل برای همگرایی به یک راه‌حل بهینه به حدود ۱۷۵ اپیزود نیاز دارد. شکل ۱۴ روند تغییر مقدار برازندگی الگوریتم پیشنهادی (QLCOA) را در طول تکرارها نشان می‌دهد. کاهش تدریجی مقدار خطا بیانگر همگرایی مناسب الگوریتم و پایداری فرآیند یادگیری است.

٦- نتیجه‌گیری و کارهای آینده

در این مقاله روشی برای تولید داده آزمون با استفاده از الگوریتم‌های

فرامکاشفه‌ای ارائه شده که هدف در آن پوشش مسیرهای پیچیده است. در این روش از ترکیب الگوریتم بهینه‌سازی کواتی و یادگیری تقویتی به‌عنوان یک الگوریتم ممتیک برای جستجو استفاده شده است. در این روش سعی شده با پیداکردن شبیه‌ترین مسیرها به مسیرهای پیدانشده و با دستکاری زیربخش‌های این مسیرها در کواتی، کواتی را بهبود دهد تا در نتیجه تعداد ارزیابی‌های الگوریتم به حداقل برسد و سرعت این فرایند افزایش یابد. به‌منظور بررسی کارایی، نتایج حاصل از الگوریتم پیشنهادی با سایر الگوریتم‌های فرامکاشفه‌ای مقایسه شد. ارزیابی انجام‌شده نشان می‌دهد کارایی روش ارائه‌شده نسبت به این الگوریتم‌ها بیشتر است. از این رو می‌توان از این روش به منظور تولید داده آزمون نرم‌افزار استفاده کرد. برای کارهای آتی می‌توان روش پیشنهادی را با سایر الگوریتم‌های فرامکاشفه‌ای مانند الگوریتم یادگیری و آموزش، الگوریتم بهینه‌سازی گرگ خاکستری، الگوریتم مار پیتون و سایر الگوریتم‌ها پیاده‌سازی نمود و نتایج الگوریتم‌های مختلف را از نظر تعداد ارزیابی‌ها مقایسه کرد. همچنین می‌توان این روش را برای برنامه‌های شیءگرا و کلاس‌ها بسط داد.

مراجع

[1] P. Ammann and J. Offutt, *Introduction to Software Testing*, Cambridge University Press, 2016.

[2] F. Lonetti and E. Marchetti, "Emerging software testing technologies," *Advances in Computers, Elsevier*, vol. 108, pp. 91-143, 2018.

[3] S. Parsa, "Automatic test data generation symbolic and concolic executions," In S. Parsa (ed.), *Software Testing Automation: Testability Evaluation, Refactoring, Test Data Generation and Fault Localization*, pp. 503-542, Springer, 2023.

[4] G. Zhang, *et al.*,"FDSE: enhance symbolic execution by fuzzing-based pre-analysis (competition contribution)," in *Proc. 27th Int. Conf. on Fundamental Approaches to Software Engineering*, pp. 304-308, Luxembourg City, Luxembourg, 6-11 Apr. 2024.

[5] O. Sahin and B. Akay, "Comparisons of metaheuristic algorithms and fitness functions on software test data generation," *Applied Soft Computing*, vol. 49, pp. 1202-1214, Dec. 2016.

[6] M. Harman and P. McMinn, "A theoretical and empirical study of search-based testing: local, global, and hybrid search," *IEEE Trans. on Software Engineering*, vol. 36, no. 2, pp. 226-247, Mar./Apr. 2009.

[7] M. Dehghani, Z. Montazeri, E. Trojovská, and P. Trojovský, "Coati optimization algorithm: a new bio-inspired metaheuristic algorithm for solving optimization problems," *Knowledge-Based Systems*, vol. 259, Article ID: pp. 110011, Jan. 2023.

[8] N. Soffair and S. Mannor, *\Textit {MinMaxMin} Q-Learning*, arXiv preprint arXiv:2402.05951, 2024.

[9] N. Khoshniat, A. Jamarani, A. Ahmadzadeh, M. Haghi Kashani, and E. Mahdipour, "Nature-inspired metaheuristic methods in software testing," *Soft Computing*, vol. 28, no. 2, pp. 1503-1544, 2024.

[10] P. Ashwini, B. Rajani, and B. Vijitha, "An efficient early software reliability prediction using particle swarm optimization (PSO)," In K. Venkata Murali Mohan, M. Suresh Babu (eds). *Disruptive Technologies in Computing and Communication Systems*, pp. 52-58, CRC Press, 2024.

[11] T. Avdeenko and K. Serdyukov, "Automated test data generation based on a genetic algorithm with maximum code coverage and population diversity," *Applied Sciences*, vol. 11, no. 10, Article ID: 4673, May-2 2021.

[12] M. Nosrati, H. Haghighi, and M. V. Asl, "Test data generation using genetic programming," *Information and Software Technology*, vol. 130, Article ID: 106446, 2021.

[13] M. Angelova, K. Atanassov, and T. Pencheva, "Multi-population genetic algorithm quality assessment implementing intuitionistic fuzzy logic," in *Proc. Federated Conf. on Computer Science and Information Systems*, pp. 365-370, Wroclaw, Poland, 9-12 Sept. 2012.

[14] M. Alshraideh, B. A. Mahafzah, and S. Al-Sharaeh, "A multiple-population genetic algorithm for branch coverage test data generation," *Software Quality J.*, vol. 19, pp. 489-513, 2011.

[15] N. Zhang, B. Wu, and X. Bao, "Automatic generation of test cases based on multi-population genetic algorithm," *Int. J. Multimedia Ubiquitous Eng.*, vol. 10, no. 6, pp. 113-122, 2015.

[16] X. Bao, Z. Xiong, N. Zhang, J. Qian, B. Wu, and W. Zhang, "Path-oriented test cases generation based adaptive genetic algorithm," *PloS One*, vol. 12, no. 11, Article ID: e0187471, 2017.

[17] A. Damia, M. Esnaashari, and M. Parvizimosaed, "Software testing using an adaptive genetic algorithm," *J. of AI and Data Mining*, vol. 9, no. 4, pp. 465-474, Oct. 2021.

[18] M. Mann, P. Tomar, and O. P. Sangwan, "Test data generation using optimization algorithm: an empirical evaluation," in *Proc. of Soft Computing: Theories and Applications*, vol. 2, pp. 679-686, Jaipur, India, 28-30 Dec. 2018.

[19] M. Mann, O. P. Sangwan, P. Tomar, and S. Singh, "Automatic goal-oriented test data generation using a genetic algorithm and simulated annealing," in *Proc. 6th Int. Conf.-Cloud System and Big Data Engineering*, pp. 83-87, Noida, India, 14-15 Jan. 2016.

[20] X. M. Zhu and X. F. Yang, "Software test data generation automatically based on improved adaptive particle swarm optimizer," in *Proc. Int. Conf. on Computational and Information Sciences*, pp. 1300-1303, Chengdu, China, 17-19 Dec. 2010.

[21] S. Singla, D. Kumar, H. Rai, and P. Singla, "A hybrid PSO approach to automate test data generation for data flow coverage with dominance concepts," *International J. of Advanced Science and Technology*, vol. 37, no. 11, pp. 15-26, 2011.

[22] S. Jiang, J. Shi, Y. Zhang, and H. Han, "Automatic test data generation based on reduced adaptive particle swarm optimization algorithm," *Neurocomputing*, vol. 158, pp. 109-116, Jun. 2015.

[23] H. Sharifipour, M. Shakeri, and H. Haghighi, "Structural test data generation using a memetic ant colony optimization based on evolution strategies," *Swarm and Evolutionary Computation*, vol. 40, no. 40, pp. 76-91, Jun. 2018.

[24] A. H. Damia and M. M. Esnaashari, "Automated test data generation using a combination of firefly algorithm and asexual reproduction optimization algorithm," *International J. of Web Research*, vol. 3, no. 1, pp. 19-28, Jun. 2020.

[25] A. Damia, M. Esnaashari, and M. Parvizimosaed, "Automatic web-based software structural testing using an adaptive particle swarm optimization algorithm for test data generation," in *Proc. 7th Int. Conf. on Web Research*, pp. 282-286, Tehran, Iran, 19-20 May 2021.

[26] O. Al-Masri and W. A. Al-Sorori, "Object-oriented test case generation using teaching learning-based optimization (TLBO) algorithm," *IEEE Access*, vol. 10, pp. 110879-110888, 2022.

[27] M. Saadtjoo and S. Babamir, "Optimizing cost function in imperialist competitive algorithm for path coverage problem in software testing," *J. of AI and Data Mining*, vol. 6, no. 2, pp. 375-385, Jul. 2018.

[28] X. Dai, W. Gong, and Q. Gu, "Automated test case generation based on differential evolution with node branch archive," *Computers & Industrial Engineering*, vol. 156, Article ID: 107290, Jun. 2021.

[29] R. R. Sahoo and M. Ray, "Forest optimization-based test case generation for multiple paths with metamorphic relations," *International J. of Applied Metaheuristic Computing*, vol. 13, no. 1, pp. 1-18, Jan. 2022.

[30] F. Feyzi and S. Parsa, "Bayes-TDG: effective test data generation using Bayesian belief network: toward failure-detection effectiveness and maximum coverage," *IET Software*, vol. 12, no. 3, pp. 225-235, Jun. 2018.

[31] M. Esnaashari and A. H. Damia, "Automation of software test data generation using genetic algorithm and reinforcement learning," *Expert Systems with Applications*, vol. 183, Article ID: 115446, Nov. 2021.

[32] M. Malkauthekar, "Analysis of euclidean distance and manhattan distance measure in face recognition," in *3rd Int. Conf. on Computational Intelligence and Information Technology*, pp. 503-507, Chennai, India, 27 Jul. 2013.

[33] Y. Duan, *et al.*, "CAPSO: chaos adaptive particle swarm optimization algorithm," *IEEE Access*, vol. 10, pp. 29393-29405, 2022.

[34] P. Trojovský and M. Dehghani, "Pelican optimization algorithm: a novel nature-inspired algorithm for engineering applications," *Sensors*, vol. 22, no. 3, Article ID: 855, 2022.

[35] S. H. S. Moosavi and V. K. Bardsiri, "Poor and rich optimization algorithm: a new human-based and multi populations algorithm," *Engineering Applications of Artificial Intelligence*, vol. 86, pp. 165-181, Nov. 2019.

[36] S. Zhao, T. Zhang, S. Ma, and M. Chen, "Dandelion optimizer: a nature-inspired metaheuristic algorithm for engineering applications," *Engineering Applications of Artificial Intelligence*, vol. 114, Article ID: 105075, Sept. 2022.

[37] I. Fister, I. Fister Jr, X. S. Yang, and J. Brest, "A comprehensive review of firefly algorithms," *Swarm and Evolutionary Computation*, vol. 13, pp. 34-46, Dec. 2013.

مرضیه سپهوند تحصیلات خود را در مقطع کارشناسی رشته نرم افزار کامپیوتر در سال ۱۳۸۹ از دانشکده تربیت دبیر شریعتی تهران و مقطع کارشناسی ارشد نرم افزار در سال ۱۴۰۲ از دانشگاه آزاد اسلامی واحد خرم آباد ادامه داد ، از دهه ۱۳۸۰ تا کنون دبیر آموزش و پرورش و زمینه تدریس ایشان دروس هنرستان رشته نرم‌افزار می باشد، زمینه‌های تحقیقاتی مورد علاقه ایشان عبارتند از: آزمون نرم‌افزار و مهندسی نرم‌افزار.

مجتبی صالحی در سال ۲۰۰۷ م. مدرک کارشناسی مهندسی نرم‌افزار را از دانشگاه آزاد اسلامی دریافت نمود. وی سپس در سال ۲۰۱۰ م. موفق به اخذ مدرک کارشناسی ارشد در رشته مهندسی کامپیوتر دانشگاه آزاد اسلامی شد. ایشان هم‌اکنون دانشجوی دکتری مهندسی کامپیوتر در دانشگاه آزاد اسلامی می‌باشد. زمینه‌های پژوهشی وی عمدتاً حول محور توسعه و بهبود روش‌ها و تکنیک‌های آزمون نرم‌افزار است.