Scientific Journal of Electronical & Cyber Defence
Vol. 11, No. 2, Summer 2023, Serial No. 42

A novel way to identify effective test-case in software testing

S. Bejani*, A. H. Keymanesh

* Assistant Professor, Imam Hossein University (AS), Tehran, Iran

(Received: 17/04/2023, Accepted: 06/02/2022)

ABSTRACT

Test data generation is one of the costly parts of the software testing, which is performed according to the designed test cases. The problem of designing test cases and then generating optimized test data is one of the challenges of the software testing, including the mutation testing technique. mutation testing has the ability to measure the test cases quality and determine the adequate test cases. However, to perform mutation testing, you need a test set that provides the maximize Coverage of source code and thus have the ability to identify the program errors. In this work, we use code coverage techniques to design test cases and automatically generate optimized test data using the meta-heuristic FA-MABC algorithm. The results are a test suite that cover and test the maximum number of source code lines. Such test suite is more likely to identify errors and get a higher score in the mutation testing. In the proposed method to obtain effective test cases, first generated test cases are applied to mutation testing and then effective test cases are extracted using the Extinguished mutation table. The results of the evaluation show that the FA-MABC algorithm reduces the time of the test data generation, and "modified condition / decision coverage", increases the mutation score.

Keywords: Automatically Generate Optimized Test Data, Mutation Testing, FA-MABC Algorithm, Code Coverage, Effective Test Cases.

* Corresponding Author Email: Sbejani@ihu.ac.ir

نشریه علمی "پدافند الکترونیکی و سایبری"
سال یازدهم، شماره ۲، تابستان ۱۴۰۲، ص ۱۰۳-۱۱۶

علمی – پژوهشی

ارائه روشی برای شناسایی موارد آزمون مؤثر در آزمون نرم‌افزار

صادق بجانی[1*]، امیرحسین کی‌منش[2]

۱- استادیار، ۲- دانشجوی کارشناسی ارشد، دانشگاه جامع امام حسین (ع)، تهران، ایران

(دریافت: ۱۴۰۱/۰۷/۱۷، پذیرش: ۱۴۰۲/۰۱/۲۸)

چکیده

طراحی موارد آزمون و تولید داده‌های آزمون، یکی از چالش‌های موجود در فرایند آزمون نرم‌افزار، ازجمله فن آزمون جهش است. موارد آزمونی که طراحی می‌شوند باید خطوط کد منبع را به‌صورت حداکثری پوشش دهند و داده‌های آزمون نیز باید بتوانند موارد آزمون را با موفقیت اجرا کنند. آزمون جهش، علاوه‌بر شناسایی خطاهای کد منبع، این توانایی را دارد که کیفیت موارد آزمون را بسنجد و موارد آزمون باکفایت را مشخص نماید. در این پایان‌نامه، از فنون پوشش کد، برای طراحی موارد آزمون و از الگوریتم فرا-ابتکاری FA-MABC برای تولید خودکار داده آزمون بهینه، استفاده می‌شود. در این الگوریتم، ابتدا مجموعه‌ای از داده‌های آزمون، به‌صورت تصادفی تولید می‌شوند و سپس معادله جستجو موجود در الگوریتم، اجرا می‌شود تا در نهایت، داده‌های آزمون بهینه، به‌دست‌آیند. نتیجه این روش، مجموعه آزمونی است که می‌تواند حداکثر خطوط کد منبع را پوشش داده و آزمون کند. چنین مجموعه آزمونی، توانایی بالایی در شناسایی خطاهای برنامه دارد و در آزمون جهش، امتیاز بالایی کسب می‌کند. در روش پیشنهادی، برای رسیدن به موارد آزمون مؤثر، ابتدا موارد آزمون طراحی‌شده، در آزمون جهش اعمال می‌شوند و سپس با استفاده از جدول جهش‌های خاموش‌شده، موارد آزمون مؤثر استخراج می‌شوند. نتایج ارزیابی، نشان می‌دهد که الگوریتم FA-MABC، موجب کاهش هزینه زمانی در تولید داده آزمون می‌شود و معیار پوشش «شرط اصلاح‌شده / تصمیم»، موجب افزایش امتیاز جهش می‌شود.

کلیدواژه‌ها: تولید خودکار داده آزمون بهینه، آزمون جهش، الگوریتم FA-MABC، پوشش کد، موارد آزمون مؤثر

۱- مقدمه

آزمون نرم‌افزار[1]، یکی از مراحل مهم در توسعه نرم‌افزار[2] است که با هدف شناسایی خطاهای برنامه، انجام می‌شود. در فرایند آزمون نرم‌افزار، «طراحی موارد آزمون»، یک گام ضروری برای رسیدن به اهداف آزمون نرم‌افزار می‌باشد. یک مورد آزمون[3]، چگونگی انجام آزمون را مشخص می‌کند [۱]. طراحی موارد آزمون، با توجه به نرم‌افزار تحت آزمون، انجام می‌شود و شامل مجموعه‌ای از اقدامات یا دستورالعمل‌هایی است که آزمونگر به کمک آن‌ها، کارایی یا جنبه خاصی از نرم‌افزار تحت آزمون را ارزیابی می‌کند [۲]. هدف از طراحی موارد آزمون، مقایسه‌ی خروجی دریافتی از نرم‌افزار تحت آزمون و نتایج قابل‌انتظار از عملکرد نرم‌افزار، می‌باشد.

یک مورد آزمون، شامل بخش‌هایی ازجمله گام‌های آزمون، داده آزمون[4]، شرط‌ها و خروجی قابل‌انتظار می‌باشد و فرایند طراحی موارد آزمون، برای مشخص کردن تمام این بخش‌ها، انجام می‌شود [۲]. داده آزمون، از مهم‌ترین نیازهای یک مورد آزمون است که تأثیر مستقیم در نتایج فرایند آزمون نرم‌افزار دارد. داده آزمون، داده‌ای است که به‌عنوان ورودی، برای اجرای مورد آزمون و اعمال در آن نرم‌افزار تحت آزمون، استفاده می‌شود. فرایند تولید داده آزمون، فعالیتی است که داده‌های آزمون را از میان داده‌های موجود، انتخاب یا تولید می‌کند. داده آزمون ضعیف، نمی‌تواند نیازهای فرایند آزمون را برآورده کند؛ بنابراین برای اثربخشی بیشتر فرایند آزمون نرم‌افزار، بهتر است تولید داده آزمون، به‌صورت دلخواه یا تصادفی انجام نشود.

یکی از راه‌های موجود برای ارزیابی داده‌های آزمون، استفاده از یک معیار پوشش[5] کد، مانند معیار پوشش «مسیرهای مستقل[6]» و «شرط اصلاح‌شده / تصمیم[7]» است. معیار پوشش «مسیرهای مستقل»، اطمینان حاصل می‌کند که تمام مسیرهای مستقل برنامه‌ی تحت آزمون، حداقل یک‌بار اجرا شوند. همچنین معیار پوشش «شرط اصلاح‌شده / تصمیم»، اطمینان حاصل می‌کند که تمام خروجی‌های ممکن برای هر شرط[8] و هر تصمیم[9]، حداقل یک‌بار اجرا شود و هر شرط، به‌طور مستقل بر نتیجه‌ی نهایی تصمیم، تأثیر بگذارد.

* رایانامه نویسنده مسئول: Sbejani@ihu.ac.ir

[1] Software Testing
[2] Software Development
[3] Test Case
[4] Test Data
[5] Coverage Criteria
[6] Independent Paths Coverage
[7] Modified Condition/Decision Coverage
[8] Condition
[9] Decision

ناشر: دانشگاه جامع امام حسین (ع)

فرایند آزمون نرم‌افزار، نیازمند یک فن آزمون مناسب برای طراحی موارد آزمون و تولید داده‌های آزمون، است. یکی از این فنون، فن آزمون جهش[1] است. در این فن آزمون، جهش‌هایی از برنامه اصلی تولید می‌شوند و موارد آزمون با هدف شناسایی این جهش‌ها، استفاده می‌شوند. آزمون جهش، علاوه‌بر برآورده کردن هدف آزمون نرم‌افزار (شناسایی خطاها)، توانایی موارد آزمون در شناسایی خطاها را نیز، ارزیابی می‌کند [۳]؛ بنابراین، با استفاده از این فن آزمون، می‌توانیم تمام موارد آزمون را ارزیابی کرده و موارد آزمون مؤثر را از موارد آزمون غیر-مؤثر و اضافی، تفکیک نمود. این ارزیابی، توسط امتیاز جهش، انجام می‌شود که توانایی موارد آزمون را در خاموش کردن جهش‌های برنامه، می‌سنجد. همچنین منظور از موارد آزمون مؤثر، آن دسته از موارد آزمون است که به‌تنهایی، می‌توانند موجب افزایش امتیاز جهش شوند و حذف حتی یک مورد آزمون، تاثیر منفی در امتیاز جهش خواهد داشت.

در این مقاله، از روش آزمون مبتنی بر جستجو[2]، برای تولید داده آزمون، استفاده می‌شود؛ در این روش، مسئله‌ی تولید داده آزمون، به یک مسئله‌ی بهینه‌سازی[3] تبدیل‌شده و با استفاده از الگوریتم‌های بهینه‌سازی، حل می‌شود. فرا-ابتکاری‌ها، یکی از روش‌های حل مسائل بهینه‌سازی هستند که پاسخ مسئله را با جستجو در فضای مسئله، تولید می‌کنند. تحقیقات گذشته نشان می‌دهد که فرا-ابتکاری‌ها، توانایی بالایی در تولید خودکار داده آزمون دارند. الگوریتم FA-MABC[4]، یکی از الگوریتم‌های فرا-ابتکاری[5] است که با ایجاد تعادل بین اکتشاف[6] و بهره‌برداری[7]، توانسته است عملکرد خوبی در مقایسه با سایر الگوریتم‌ها، داشته باشد. این الگوریتم، یک الگوریتم پیوندی[8] مبتنی بر الگوریتم کرم شب‌تاب[9] و الگوریتم کلونی زنبور مصنوعی[10] است. در این الگوریتم پیوندی، ویژگی‌های مثبت دو الگوریتم پایه، استخراج و ترکیب شده‌اند؛ به‌طوری‌که این ترکیب، می‌تواند اثربخشی بهتری در مقایسه با الگوریتم‌های پایه داشته باشد [۴].

ارزیابی انجام‌شده توسط ایوانا و همکاران [۴]، نشان می‌دهد که الگوریتم پیشنهادی به لحاظ دقت راه‌حل، توانمندی و نرخ همگرایی، بهتر از الگوریتم‌های پایه و نسخه‌های بهبودیافته آن‌ها، عمل می‌کند.

تولید خودکار داده آزمون بهینه، می‌تواند موجب صرفه‌جویی در زمان و هزینه شود و همچنین، اثربخشی خوبی در آزمون نرم‌افزار، داشته باشد؛ بنابراین هدف این مقاله، تولید خودکار داده آزمون بهینه و رسیدن به مجموعه آزمون مؤثر به‌منظور شناسایی خطاهای برنامه می‌باشد. ارائه روشی به‌منظور شناسایی موارد آزمون مؤثر، با استفاده از معیار پوشش MC/DC، الگوریتم FA-MABC و فن آزمون جهش» و «ارائه یک تابع تناسب ترکیبی شامل فاصله انشعاب و تابع شباهت»، به‌عنوان نوآوری این مقاله مطرح می‌شود.

در روش پیشنهادی این مقاله، از فن آزمون جهش، جهت آزمون برنامه و ارزیابی کیفیت موارد آزمون، استفاده می‌شود. آزمون جهش معمولاً در مرحلهٔ آزمون واحد[11]، انجام می‌شود. هر واحد برنامه، یک تابع یا یک متد خواهد بود که دارای پارامترهای ورودی است. محدوده این مقاله، آزمون واحد با استفاده از فن آزمون جهش است. همچنین، پارامترهای ورودی به یک واحد برنامه، از نوع داده «اعداد صحیح» خواهند بود که به‌عنوان فرض این مقاله، مطرح می‌شود. علاوه بر این، روش پیشنهادی و برنامه‌های معیار، در زبان برنامه‌نویسی جاوا، توسعه یافته‌اند.

در این مقاله، از الگوریتم FA-MABC، برای تولید خودکار داده آزمون استفاده می‌شود تا مجموعه‌ای از داده آزمون بهینه، به دست آید و برای ارزیابی داده‌های آزمون، از دو معیار پوشش «مسیرهای مستقل» و «شرط اصلاح‌شده / تصمیم» استفاده می‌شود. استفاده از معیار پوشش کد در طراحی موارد آزمون، سبب می‌شود که مجموعه آزمون طراحی‌شده، توانایی بالایی در شناسایی خطاهای موجود در برنامه، داشته باشند.

در مرحلهٔ بعد، موارد آزمون همراه با داده آزمون تولیدشده، در آزمون جهش، اعمال می‌شود. آزمون جهش، علاوه بر شناسایی خطاهای برنامه، می‌تواند برای ارزیابی کیفیت[12] موارد آزمون نیز، مورداستفاده قرار گیرد؛ ازاین‌رو، پس از آزمون جهش، می‌توان موارد آزمون اضافی و غیر مؤثر را از مجموعه آزمون، حذف کرده و درنهایت، موارد آزمون باقی‌مانده را به‌عنوان موارد آزمون مؤثر، ذخیره نمود.

برای ارزیابی روش پیشنهادی، از سه برنامه معیار، استفاده شده و نتایج به‌دست‌آمده از سه الگوریتم فرا-ابتکاری و دو معیار پوشش، با یکدیگر مقایسه شده است. نتایج ارزیابی نشان می‌دهد که الگوریتم FA-MABC در نیمی از اجراها، هزینه زمانی کمتری نسبت به دو الگوریتم دیگر دارد. همچنین، الگوریتم FA در بخشی از اجراها، درصد پوشش و امتیاز جهش بالایی نسبت به دو الگوریتم دیگر دارد؛ این امر، معمولاً به دلیل کاهش حدود متغیرهای مسئله، اتفاق می‌افتد. الگوریتم FA-MABC، حدود متغیرهای مسئله را کاهش نمی‌دهد و از این نظر، نسبت به دو

[1] Mutation
[2] Search-Based Testing
[3] Optimization
[4] Firefly Algorithm and Multi Strategy Artificial Bee Colony
[5] Meta-Heuristic
[6] اکتشاف یا کاوش، معادل با Exploration است.
[7] بهره‌برداری یا استخراج، معادل با Exploitation است.
[8] Hybrid
[9] Firefly Algorithm (FA)
[10] Artificial Bee Colony (ABC)
[11] Unit Testing
[12] Quality

الگوریتم دیگر، برتری دارد. امتیاز جهش در معیار پوشش «شرط اصلاح‌شده / تصمیم»، بیش‌تر از معیار پوشش «مسیرهای مستقل» است.

در بخش دوم، مفاهیم مرتبط با آزمون نرم‌افزار، معیار پوشش، فن آزمون جهش، الگوریتم‌های فرا-ابتکاری و تابع تناسب، تشریح می‌شوند و کارهای پیشین، در بخش سوم، ارائه می‌شوند. بخش چهارم، به ارائه روش پیشنهادی می‌پردازد و ارزیابی‌های انجام‌شده و نتایج آن، در بخش پنجم، ذکر خواهند شد. بخش ششم، شامل نتیجه‌گیری و ارائه پیشنهادهایی برای کارهای آینده می‌باشد.

۲- ادبیات موضوع

آزمون نرم‌افزار، فعالیتی است که در آن، یک سیستم یا مؤلفه تحت شرایط معین، اجرا می‌شود، نتایج آن ثبت می‌شود و برخی از جنبه‌های سیستم یا مؤلفه، ارزیابی می‌شوند [۱, ۵, ۶]. علاوه‌براین، آزمون نرم‌افزار، می‌تواند وجود خطاها را نشان دهد؛ اما هرگز عدم وجود آن را نشان نمی‌دهد [۷].

یک تعریف قدیمی از آزمون نرم‌افزار می‌گوید که «آزمون، فرایند اجرای یک برنامه با هدف یافتن خطاها است». طبق تعریف گالین[1]: «آزمون نرم‌افزار، یک فرایند رسمی است که توسط گروه تخصصی آزمون انجام می‌شود و در آن، یک واحد نرم‌افزاری، چندین واحد یکپارچه‌شده یا یک بسته نرم‌افزاری کامل، با اجرای برنامه‌ها بر روی کامپیوتر، بررسی می‌شود» [۱].

از اهداف و نتایج آزمون نرم‌افزار، می‌توان به (۱) شناسایی و آشکار کردن هرچه بیشتر خطاها در نرم‌افزار تحت آزمون و (۲) رساندن نرم‌افزار تحت آزمون به سطح قابل‌قبولی از کیفیت؛ بعد از اصلاح خطاهای شناسایی‌شده و آزمون مجدد، اشاره نمود [۱]. فقدان کیفیت در نرم‌افزار که به دلیل آزمون نامناسب رخ می‌دهد، می‌تواند منجر به تلفات جانی و مالی شود. یک نمونه اخیر از نقص نرم‌افزاری، هواپیمای Boeing 737 Max است که منجر به جان باختن ۳۴۶ نفر در دو سانحه سقوط هواپیما شد [۸, ۹].

یکی از جنبه‌های مهم در آزمون نرم‌افزار، تعیین زمان خاتمه آزمون است؛ یعنی اینکه بدانیم چه زمانی، توانسته‌ایم به‌اندازه کافی، آزمون انجام دهیم. تعیین زمان خاتمه، یکی از وظایف معیار کفایت است که قوانینی برای پایان دادن به آزمون، تعیین می‌کند. سؤال مطرح در معیار کفایت، این است که آیا آزمون کافی و مناسب، انجام شده است یا خیر. وظیفه دیگر معیار کفایت، بررسی مجموعه آزمون است که خوب بودن یا بد بودن آن را تعیین می‌کند. چنین معیاری، با نام معیار پوشش شناخته می‌شود و درجه‌ای از کفایت را به‌صورت درصدی، نشان می‌دهد. معیار پوشش، به این سؤال پاسخ می‌دهد: چگونه مجموعه آزمونی به دست آوریم که تا حد امکان، خطاهای نرم‌افزار تحت آزمون را شناسایی کند؟ [۱۰-۱۳]

۲-۱- روش‌های آزمون نرم‌افزار

در این طبقه‌بندی، آزمون‌ها بر اساس متدولوژی انجام آزمون و دیدگاهی که نسبت به نرم‌افزار تحت آزمون وجود دارد، طبقه‌بندی می‌شوند. دو روش مهم در این طبقه‌بندی، عبارت‌اند از روش جعبه سفید و روش جعبه سیاه.

روش جعبه سفید، با نام آزمون جعبه شیشه‌ای[2] و آزمون ساختاری نیز شناخته می‌شود. در این روش، کارکرد داخلی نرم‌افزار و کارکرد مجموعه‌ای از مؤلفه‌های نرم‌افزار، آزمون می‌شود. این روش، نیاز به دانش عمیق درباره کد دارد؛ زیرا بخش‌های ساختاری نرم‌افزار را مورد آزمون قرار می‌دهد [۱, ۱۴-۱۶].

وظیفهٔ روش جعبه سفید، انجام آزمون با اجرای کد منبع است؛ بنابراین، تلاش می‌کند که تمام خطوط کد منبع را اجرا و آزمون کند. برای سنجش این تلاش، معیاری به نام معیار پوشش کد، مطرح می‌شود که میزان موفقیت فنون روش جعبه سفید در پوشش کد منبع را می‌سنجد. معیار پوشش، همانند معیار کفایت آزمون است و تعیین می‌کند که طراحی موارد آزمون، چه زمانی به پایان برسد. در ادامه، دو معیار پوشش کد، معرفی و تشریح می‌شوند.

پوشش مسیر پایه[3]:

یکی از فنون انتخاب مسیرها در گراف جریان کنترل، آزمون مسیر پایه است. مبنای این معیار پوشش، مسیرهای مستقل در برنامه است. مجموعه پایه، شامل مسیرهای مستقل در برنامه است که این فن آزمون، سعی در اجرای آن‌ها دارد.

یک مسیر مستقل در گراف جریان کنترل، مسیری است که حداقل یک یال جدید را شامل می‌شود؛ به‌طوری‌که این یال، در مسیرهای تعریف‌شده قبلی، پیمایش نشده باشد. تعداد مسیرهای مستقل، توسط معیار پیچیدگی سائیکلومَتیک[4]، به دست می‌آید. به همین دلیل، تعداد مسیرهای مستقل می‌تواند به‌عنوان حداکثر آزمون‌های موردنیاز برای پوشش کامل برنامه، استفاده شود. مجموعه‌ای از مسیرهای مستقل، با عنوان مجموعه پایه شناخته می‌شود [۱۷, ۱۸].

در این فن آزمون، حداکثر تعداد مسیرهای موردنیاز برای پوشش کد منبع، توسط معیار پیچیدگی سائیکلومَتیک McCabe، محاسبه می‌شود. این معیار، پیچیدگی منطقی برنامه و تعداد مسیرهای مستقل، در گراف جریان کنترل را تعیین می‌کند و

[1] Daniel Galin

[2] Glass Box
[3] Basic Path Coverage
[4] Cyclomatic Complexity

می‌توان از آن برای تعیین حداکثر آزمون‌های موردنیاز برای پوشش کامل برنامه، استفاده کرد. اگر مسیرهای مستقل، پوشش داده شوند؛ تضمین می‌شود که تمام انشعاب‌های گراف جریان کنترل، حداقل یک‌بار اجرا می‌شوند.

فرمول (۱)، نحوه محاسبه‌ی پیچیدگی سایکلومتیک را نشان می‌دهند [۲, ۱۷-۲۰]:

$$V(G) = E - N + 2 \qquad (۱)$$

پوشش شرط اصلاح‌شده / تصمیم (MC/DC)[۱]:

این معیار، تضمین می‌کند که خروجی‌های ممکن برای هر شرط و هر تصمیم و خروجی‌های ممکن برای هر شرطی که به طور مستقل بر نتیجهٔ تصمیم تأثیر می‌گذارد، حداقل یک‌بار اجرا شوند. حداقل تعداد موارد آزمون موردنیاز برای برآورده‌کردن این معیار پوشش، برابر با n+1 می‌باشد که n، تعداد شرط‌های موجود در یک تصمیم است [۲, ۱۳, ۲۰].

برای طراحی موارد آزمون در این معیار پوشش، از جدول درستی[۲]، کمک می‌گیریم. این جدول، توالی‌هایی از True و False است که خروجی نهایی هر دستور شرطی ساده را مشخص می‌کنند. مجموعه‌ای از مقادیر True و False که به یک دستور شرطی، انتساب می‌یابد، خروجی نهایی دستور شرطی را مشخص می‌کنند.

جدول (۱)، برای دستورالعمل‌های شرطی که شامل دو شرط ساده هستند، جدول درستی را ارائه می‌دهد که با توجه به معیار پوشش MC/DC، تولید شده است [۱۳].

جدول (۱). جدول درستی برای دستورالعمل شرطی شامل دو شرط ساده

دستورالعمل شرطی	خروجی C	خروجی Y	خروجی X	مورد آزمون
C = (X and Y)	True	True	True	۱
	False	False	True	۲
	False	True	False	۳
C = (X or Y)	False	False	False	۱
	True	True	False	۲
	True	False	True	۳
C = (X xor Y)	False	True	True	۱
	True	False	True	۲
	False	False	False	۳

روش دیگر برای آزمون نرم‌افزار با نام روش جعبه سیاه، یا آزمون عملکردی، شناخته می‌شود. در این روش، آزمونگر بر روی ورودی‌ها و خروجی‌های مورد انتظار، تمرکز دارد؛ بدون اینکه بداند ساختار داخلی برنامه، چگونه کار می‌کند و این ورودی‌ها چگونه پردازش می‌شوند. روش جعبه سیاه، فرض می‌کند که آزمونگر، کاربر نهایی است و از جزئیات پیاده‌سازی، آگاه نیست. هدف از این روش، بررسی عملکرد نرم‌افزار، اطمینان از کارکرد صحیح و اطمینان از برآورده کردن خواسته‌های کاربران است. این روش، می‌تواند در هر سطحی از آزمون، اعمال شود؛ اما معمولاً در آزمون سیستم و آزمون پذیرش، مورداستفاده قرار می‌گیرد [۱, ۱۴-۱۶].

۲-۲- فن آزمون جهش

یکی از فنون آزمون نرم‌افزار، آزمون جهش است که با نام آزمون مبتنی بر عیب نیز شناخته می‌شود. در این فن آزمون، چندین نسخه خطادار از نرم‌افزار تحت آزمون، توسط آزمونگر و با اعمال تغییراتی در کد منبع، ایجاد می‌شوند. به این نسخه‌های خطادار، جهش گفته می‌شود و هر جهش، تنها شامل یک تغییر در کد منبع است. آزمون جهش، توانایی مورد آزمون را به لحاظ شناسایی خطاهای برنامه، ارزیابی کرده و به هر مورد آزمون، یک امتیاز جهش، نسبت می‌دهد که به‌عنوان معیاری برای ارزیابی کیفیت مورد آزمون، تعریف می‌شود. مقصود اصلی آزمون جهش، آشکار کردن ضعف در مورد آزمون است و به آزمونگر کمک می‌کند که کیفیت مورد آزمون را بسنجد و بهبود بخشد. هدف اعمال مورد آزمون در آزمون جهش، متمایز کردن نسخه جهش‌یافته از نسخه اصلی برنامه است. آزمون جهش، در ابتدا توسط ریچارد لیپتون[۳] و در سال ۱۹۷۱، پیشنهاد شد و سپس در سال ۱۹۷۸، منتشر شد [۲۱-۲۳].

آزمون جهش، بر مبنای دو فرض اساسی است [۱۱, ۱۳, ۲۳-۲۵]: (۱) فرضیه برنامه‌نویس شایسته[۴] و (۲) فرضیه اثر جفتی[۵].

فرضیه برنامه‌نویس شایسته، می‌گوید که برنامه تحت آزمون، توسط یک برنامه‌نویس شایسته، نوشته شده است. چنین برنامه‌هایی، ازنظر درستی و صحت، به حد نهایی خود، نزدیک هستند. اگر عیب‌هایی در برنامه وجود دارد که توسط برنامه‌نویس شایسته، ایجاد شده است؛ فرض می‌کنیم که این عیب‌ها، اشتباهات ساده‌ای هستند که با چند تغییر نحوی کوچک، اصلاح می‌شوند.

فرضیه اثر جفتی، می‌گوید که خطاهای ساده و خطاهای پیچیده، همراه با یکدیگر هستند. طبق این فرضیه، داده آزمونی که می‌تواند خطاهای ساده را شناسایی کند، معمولاً می‌تواند خطاهای پیچیده را نیز شناسایی کند. در اینجا، شناسایی خطا، یعنی خاموش‌کردن جهش‌های غیر معادل. [۳, ۲۲].

[۱] پوشش MC/DC، معادل عبارت Modified Condition/Decision Coverage است و Modify به معنی ایجاد تغییری کوچک برای بهبود چیزی است.

[2] Truth Table

[3] Richard Lipton

[4] Competent Programmer Hypothesis

[5] Coupling Effect Assumption

معیار کفایت جهش:

معیار کفایت جهش، برای سنجش اثربخشی مجموعه آزمون، به لحاظ توانایی در شناسایی عیب‌های نرم‌افزار، استفاده می‌شود. مجموعه آزمون T را در نظر بگیرید. این مجموعه آزمون، در برنامه تحت آزمون اعمال می‌شود و نتایج آزمون نشان می‌دهد که برنامه مطابق با نیازمندی‌ها، کار می‌کند. اکنون می‌خواهیم بدانیم که مجموعه آزمون T چقدر مناسب است؟ آزمون جهش، راهی برای پاسخ دادن به این سؤال، پیشنهاد می‌کند و آن، امتیاز جهش است. امتیاز جهش، عددی بین ۰ و ۱ است و می‌توان آن را به‌صورت درصدی نیز بیان نمود. امتیاز جهش ۱ یا ۱۰۰٪، حد نهایی کفایت برای مجموعه آزمون T است و هدف آزمون جهش نیز، رسیدن به این حد نهایی است [۳, ۱۳, ۲۲].

فرمول (۲)، نحوه محاسبه امتیاز جهش را نشان می‌دهد [۱۳, ۲۵, ۲۶]. در این فرمول: D، معادل تعداد جهش‌های خاموش‌شده؛ L، معادل تعداد جهش‌های زنده؛ M، معادل تعداد کل جهش‌ها و E، معادل تعداد جهش‌های معادل با برنامه اصلی.

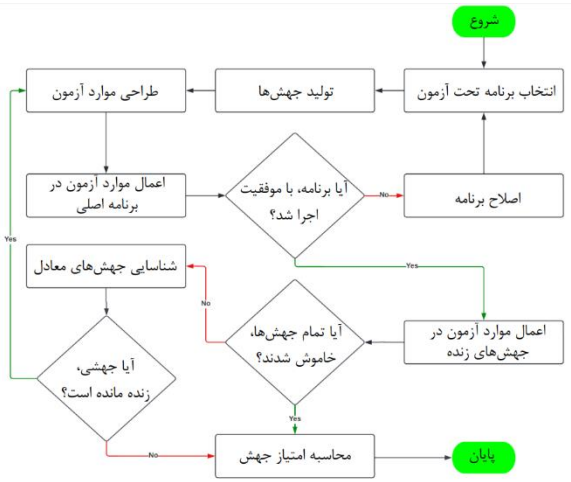
$$\text{Mutation Score} = \frac{D}{M-E} = \frac{D}{L+D} \qquad (۲)$$

مراحل آزمون جهش:

مرحله اول در ارزیابی کفایت مجموعه آزمون، اجرای برنامه تحت آزمون به‌ازای تمام موارد آزمون موجود در مجموعه آزمون است. مرحله دوم، تولید جهش‌های برنامه اصلی است که با اعمال عملگرهای جهش در برنامه اصلی، تولید می‌شوند. مرحله سوم و چهارم، اجرای جهش‌های برنامه، با مجموعه آزمون T است و به‌صورت تکراری انجام می‌شوند. مرحله سوم، انتخاب یک جهش از میان جهش‌های تولیدشده است و مرحله چهارم، اجرای جهش انتخاب‌شده با مجموعه آزمون T است. در مرحله چهارم، جهش انتخاب‌شده، به‌ازای تمامی موارد موجود در مجموعه آزمون، اجرا می‌شود. درنتیجه، این مرحله شامل اجرای چندباره یک جهش است. نتیجه هر اجرا برای مقایسه با نتیجه برنامه اصلی، ذخیره می‌شود و تا زمانی که تمام جهش‌ها، انتخاب و اجرا شوند، این دو مرحله ادامه می‌یابد. در مرحله پنجم، نتایج به‌دست‌آمده از جهش‌های برنامه با نتایج برنامه اصلی، مقایسه می‌شوند. پس از مقایسه نتایج، می‌توانیم جهش‌های موجود را طبقه‌بندی کنیم.

درصورتی‌که نتایج اجرای جهش و برنامه اصلی، متفاوت باشد، نتیجه می‌گیریم که مجموعه آزمون توانسته است خطای موجود در جهش را شناسایی کند. این جهش‌ها را جهش‌های خاموش شده می‌نامیم. درصورتی‌که نتایج، مشابه یکدیگر باشد، نتیجه می‌گیریم که مجموعه آزمون نتوانسته است تفاوت بین جهش و برنامه اصلی را تشخیص دهد. این جهش‌ها را جهش‌های زنده می‌نامیم. مرحله ششم، شناسایی جهش‌های معادل است. مرحله هفتم، محاسبه امتیاز جهش است که با استفاده از فرمول (۲)، محاسبه می‌شود. مرحله هشتم، شامل بهبود مجموعه آزمون است. اگر از مراحل قبل، جهش‌هایی زنده مانده باشند و جهش معادل هم نباشند، بدین معنی است که مجموعه آزمون T توانایی خاموش‌کردن آن‌ها را ندارد؛ بنابراین، مجموعه آزمون باید بهبود یابد تا بتوانیم جهش‌های زنده را خاموش کنیم. پس از اصلاح مجموعه آزمون، به مرحله سوم برمی‌گردیم؛ با این تفاوت که جهش‌های فعلی، تنها جهش‌های زنده‌ای هستند که مجموعه آزمون قبلی قادر به خاموش‌کردن آن‌ها نبوده است.

شکل (۱)، نمایی از فرایند آزمون جهش را نشان می‌دهد [۲۲].



شکل (۱). فرایند آزمون جهش [۲۲]

۲-۳- طراحی موارد آزمون

یک مورد آزمون، مجموعه‌ای از مقادیر ورودی، نتایج قابل انتظار و... است که برای کشف یک نوع خطا یا ارزیابی یک عملکرد خاص، طراحی می‌شود و با وادارکردن برنامه تحت آزمون، به شکست، چنین خطاهایی کشف می‌شوند [۲۷].

مورد آزمون ایده‌آل، تمام ورودی‌های ممکن برای برنامه را در نظر می‌گیرد و در آزمون جامع، استفاده می‌شود. آزمون جامع، در عمل امکان‌پذیر نیست؛ بنابراین، موارد آزمون موردنیاز برای یک برنامه خاص، باید از میان موارد آزمون ایده‌آل، انتخاب شوند. در انتخاب موارد آزمون، می‌توان معیاری برای ارزیابی موارد آزمون تعیین کرد. همچنین می‌توان، موارد آزمون را برای برآورده‌کردن یک معیار مشخص، انتخاب نمود. هر مورد آزمون، باید به طور مناسب و دقیق، طراحی و مشخص شود. فنی که برای طراحی مورد آزمون، استفاده می‌شود، در بهبود کیفیت فرایند آزمون، مؤثر است و کیفیت کلی نرم‌افزار تحت آزمون را بهبود می‌بخشد. درنتیجه، طراحی موارد آزمون، به کمک فن آزمون انجام می‌شود. اگر موارد آزمون، به‌خوبی طراحی شوند، اطمینان حاصل می‌کنیم که هر جنبه از نرم‌افزار، مورد آزمون قرار می‌گیرد و احتمال شناسایی خطاها، افزایش می‌یابد. هر مورد آزمون، می‌تواند شامل

عنوان، توضیحات، هدف آزمون، فرضیات، گام‌های آزمون، داده آزمون، پیش‌نیازهای آزمون، نتایج واقعی و نتایج قابل‌انتظار باشد.

در آزمون نرم‌افزار، بعد از طراحی موارد آزمون، نیاز به تولید داده آزمون[۱]، داریم. تولید داده آزمون، یکی از بحرانی‌ترین مراحل آزمون نرم‌افزار است؛ زیرا تأثیر زیادی در موفقیت یا شکست آزمون دارد. داده آزمون، همان مقادیر واقعی هستند که از دامنه ورودی یک برنامه، به دست می‌آیند [۲۵]. فرایند آزمون نرم‌افزار، نمی‌تواند به‌صورت قطعی نشان دهد که برنامه تحت آزمون، کاملاً درست کار می‌کند؛ اما می‌تواند توسط مجموعه‌ای از داده آزمون مناسب، آزمونگر را قانع کند که برنامه، درست کار می‌کند.

فنون مبتنی بر جستجو[۲] یکی از راهکارهای تولید داده آزمون است که جزء رویکردهای پویا طبقه‌بندی می‌شود. این فرایند، تولید داده آزمون را به یک مسئلهٔ جستجو، تبدیل کرده و از فرا-ابتکاری‌ها برای شناسایی بهترین راه‌حل در فضای جستجوی یک برنامه، استفاده می‌کند. فرا-ابتکاری‌ها، بر تابع تناسب[۳] یا تابع هدف، متکی هستند [۲۸, ۲۹].

۲-۴- الگوریتم‌های فرا-ابتکاری

یکی از روش‌های قابل‌توجه در تولید خودکار داده آزمون، استفاده از الگوریتم‌های فرا-ابتکاری است. کاربرد الگوریتم‌های فرا-ابتکاری، حل مسائل بهینه‌سازی است. فرایند بهینه‌سازی، به یافتن بهترین راه‌حل از میان مجموعه‌ای از راه‌حل‌ها اشاره دارد و در حالت ساده، به حل یک مسئلهٔ کمینه‌سازی یا بیشینه‌سازی می‌پردازد. فرایند تولید داده آزمون نیز مشابه همین تعریف است؛ یعنی یافتن بهترین داده آزمون از میان مجموعه‌ای از داده آزمون. فرا-ابتکاری‌ها، زیرمجموعه‌ای از الگوریتم‌های بهینه‌سازی هستند که برای حل چنین مسائلی، پیشنهاد شده‌اند.

استفاده از الگوریتم‌های فرا-ابتکاری، جزء فنون مبتنی بر جستجو در تولید خودکار داده آزمون است و با نام آزمون مبتنی بر جستجو[۴] نیز شناخته می‌شود. در آزمون مبتنی بر جستجو، مسئلهٔ تولید داده آزمون، به یک مسئلهٔ جستجو، تبدیل می‌شود. الگوریتم‌های فرا-ابتکاری، فضای مسئله را با هدف یافتن بهترین راه‌حل، جستجو می‌کنند. ازجمله الگوریتم‌های فرا-ابتکاری، می‌توان به الگوریتم ژنتیک[۵]، بهینه‌سازی ازدحام ذرات[۶]، کلونی مورچه‌ها[۷]، کرم شب‌تاب، کلونی زنبور مصنوعی و سایر موارد، اشاره نمود.

الگوریتم‌های جستجو، شامل فرا-ابتکاری‌ها، نیازمند پرداختن به مسائلی ازجمله اکتشاف و بهره‌برداری در فضای جستجو هستند. اکتشاف و بهره‌برداری، دو مؤلفه اصلی در هر الگوریتم فرا-ابتکاری هستند که با نام گوناگونی[۸] و تشدید[۹] نیز شناخته می‌شوند [۳۰, ۳۱]. در ادامه، این دو مؤلفه و ویژگی‌های آن‌ها، تشریح می‌شوند.

اکتشاف: به معنی تولید راه‌حل‌های متنوع به‌منظور کاوش فضای جستجو در مقیاس سراسری است. وظیفهٔ این مؤلفه، بازدید از مناطق کاملاً جدید از فضای جستجو است.

بهره‌برداری: به معنی تمرکز بر روی جستجو در یک منطقهٔ محلی با بهره‌برداری از اطلاعاتی که راه‌حل خوب جاری، در این منطقه پیدا کرده است. وظیفهٔ این مؤلفه، بازدید از مناطقی در فضای جستجو است که در همسایگی نقاط بازدید شده قبلی، قرار دارند.

۲-۵- تابع تناسب

یکی از اجزای اصلی در تمام فنون بهینه‌سازی، تابع هدف است که با نام تابع تناسب و تابع هزینه نیز شناخته می‌شود. تابع هدف، می‌تواند همان مسئله‌ای باشد که فرایند بهینه‌سازی، سعی دارد یک مقدار بهینه برای آن، تولید کند. این تابع، داده‌هایی را به‌عنوان ورودی، دریافت کرده و تعیین می‌کند که این داده ورودی تا چه اندازه، مناسب و بهینه است. در زمینهٔ تولید خودکار داده آزمون، معمولاً نمی‌توانیم از برنامه تحت آزمون، به‌عنوان تابع هدف، استفاده کنیم؛ اما راهکارهایی برای پیاده‌سازی تابع تناسب وجود دارد که می‌تواند فرایند جستجو را به سمت داده‌های بهینه، هدایت کند. در ادامه چند تابع تناسب که برای تولید داده آزمون بهینه، ارائه شده‌اند، معرفی می‌شوند.

فاصله انشعاب[۱۰]:

تریسی[۱۱] در سال ۱۹۹۸، برای تولید داده آزمون در روش آزمون ساختاری، روشی برای محاسبه تابع هزینه، ارائه کرد که امروزه به‌عنوان تابع تناسب در الگوریتم‌های فرا-ابتکاری، استفاده می‌شود. در این روش، هزینه‌ای برای دستورات شرطی موجود در کد منبع، محاسبه می‌شود و نتیجه نهایی محاسبات، به‌عنوان هزینهٔ راه‌حل پیشنهادی، در نظر گرفته می‌شود. در توابع هزینه، راه‌حل پیشنهادی که دارای هزینه کمتر باشد، به‌عنوان پاسخ

[۱] داده آزمون، همان چیزی است که در نرم‌افزار تحت آزمون، اعمال می‌شود. خروجی حاصل از این اجرای نرم‌افزار، ذخیره شده و با نتایج مورد انتظار، مقایسه می‌شود.

[2] Search-Based Techniques
[3] Fitness Function
[4] Search-Based Testing (SBT) - Search-Based Software Testing (SBST)
[5] Genetic Algorithm
[6] Particle Swarm Optimization
[7] Ant Colony Optimization
[8] Diversification
[9] Intensification
[10] Branch Distance
[11] Tracey

نهایی انتخاب می‌شود. جدول (۲)، محاسبه هزینه برای دستورات شرطی که خروجی True یا False تولید می‌کنند را ارائه می‌دهد [۳۲].

جدول (۲). محاسبه هزینه برای تابع فاصله انشعاب [۳۲]

عبارت شرطی	محاسبه هزینه
یک متغیر شرطی (A)	$(A == True)\ ?\ 0 : K$
$a = b$	$abs(a-b) = 0\ ?\ 0 : abs(a-b) + K$
$a \neq b$	$abs(a-b) \neq 0\ ?\ 0 : K$
$a < b$	$(a-b) < 0\ ?\ 0 : (a-b) + K$
$a \leq b$	$(a-b) \leq 0\ ?\ 0 : (a-b) + K$
$a > b$	$(b-a) < 0\ ?\ 0 : (b-a) + K$
$a \geq b$	$(b-a) \leq 0\ ?\ 0 : (b-a) + K$
$(a)\ or\ (b)$	$\min(\ cost(a)\ , cost(b)\ )$
$(a)\ and\ (b)$	$cost(a) + cost(b)$

در این جدول، فرمول لازم برای محاسبه هزینه راه‌حل پیشنهادی، ارائه شده است. متغیر K در این فرمول، یک مقدار ثابت و مثبت است که به‌عنوان ضریب مجازات، شناخته می‌شود. مقدار K، می‌تواند صفر نیز باشد. با توجه به عبارت‌های شرطی که در کد منبع برنامه تحت آزمون، وجود دارد، فرمول‌های موردنیاز از جدول (۲)، استخراج می‌شود و با توجه به این فرمول‌ها، هزینهٔ راه‌حل پیشنهادی، محاسبه می‌گردد.

پس از محاسبهٔ هزینه، با استفاده از فاصله انشعاب، می‌توانیم هزینه به‌دست‌آمده را ساده‌سازی کنیم تا مقدار این هزینه در بازه [0,1] را به دست آوریم. برای ساده‌سازی فاصله انشعاب، از فرمول (۳)، استفاده می‌کنیم. ورودی این فرمول (x)، مقداری است که توسط تابع فاصله انشعاب، به‌دست‌آمده است.

$$NBD(x) = 1 - (1.001^{-x}) \qquad (۳)$$

تابع هزینه برای معیار پوشش MC/DC:

هنگام پیاده‌سازی تابع تناسب برای معیار پوشش MC/DC، خروجی هر شرط ساده، باید بررسی شود تا بتوانیم موارد آزمون را پوشش دهیم. در معیار پوشش مسیر، خروجی هر دستورالعمل شرطی در صورت صحیح بودن، بررسی می‌شود؛ اما در معیار پوشش MC/DC، نیاز داریم که خروجی صحیح و غلط را برای هر شرط ساده، بررسی کنیم؛ زیرا موارد آزمون طراحی‌شده در این معیار پوشش، شامل توالی از True و False هستند.

برای محاسبهٔ فاصله انشعاب با توجه به معیار پوشش MC/DC، به تابع هزینه‌ای نیاز داریم که بتواند هزینهٔ مقادیر True و False را تولید کند. برای این منظور، از تابع هزینه پیشنهادی توسط آوِدیکیان[1] و همکاران، استفاده می‌کنیم [۳۳]. جدول (۳)، فرمول موردنیاز برای محاسبه هزینه در معیار پوشش MC/DC را ارائه می‌دهد.

[1] Awedikian

جدول (۳). فرمول محاسبهٔ تابع هزینه برای معیار پوشش MC/DC [۳۳]

عبارت شرطی	هزینه برای مقدار True	هزینه برای مقدار False
$a = b$	$abs(a-b)$	$a = b\ ?\ K : 0$
$a \neq b$	$a \neq b\ ?\ 0 : K$	$abs(a \neq b\ ?\ a-b : 0)$
$a < b$	$abs(a < b\ ?\ 0 : a-b+K)$	$abs(a < b\ ?\ a-b+K : 0)$
$a \leq b$	$abs(a \leq b\ ?\ 0 : a-b)$	$abs(a \leq b\ ?\ a-b : 0)$
$a > b$	$abs(a > b\ ?\ 0 : a-b+K)$	$abs(a > b\ ?\ a-b+K : 0)$
$a \geq b$	$abs(a \geq b\ ?\ 0 : a-b)$	$abs(a \geq b\ ?\ a-b : 0)$
$(a)\ or\ (b)$	$\min(\ cost(a)\ , cost(b)$	$cost(a) + cost(b)$
$(a)\ and\ (b)$	$cost(a) + cost(b)$	$\min(\ cost(a)\ , cost(b)$

فاصله گزاره[2]:

یکی دیگر از راه‌های پیاده‌سازی تابع تناسب، محاسبه اختلاف میان نتیجه دریافتی و نتیجه مورد انتظار است. این تابع به‌عنوان فاصله گزاره، شناخته می‌شود و گزینه مناسبی برای ارزیابی راه‌حل پیشنهادی در زمینهٔ تولید خودکار داده آزمون می‌باشد [۳۴]. برای محاسبه اختلاف میان دو مورد آزمون، تعداد اعضای مشترک در دو مورد آزمون را از تعداد اعضای مورد آزمون هدف، کم می‌کنیم.

تابع شباهت[3]:

یکی دیگر از راه‌های پیاده‌سازی تابع تناسب، محاسبه شباهت میان نتیجه دریافتی و نتیجه مورد انتظار است. این تابع تناسب، اختلاف میان نتیجه دریافتی و نتیجه مورد انتظار را نشان می‌دهد. برای محاسبه این تابع تناسب، هر عضو موجود درنتیجه دریافتی، با عضو معادل درنتیجه مورد انتظار، مقایسه می‌شود. درصورتی‌که این دو عضو، معادل یکدیگر باشند، شباهت آن‌ها صفر است و درصورتی‌که مخالف یکدیگر باشند، شباهت آن‌ها، یک است. این محاسبه، به‌ازای تمام عضوهای موجود، انجام شده و درنهایت، خروجی به‌دست‌آمده، به‌عنوان اختلاف میان نتیجه دریافتی و نتیجه مورد انتظار، مطرح می‌شود. این تابع تناسب، مبتنی بر فاصلهٔ همینگ[4] است [۳۵].

۲-۶ -الگوریتم پیوندی FA-MABC

الگوریتم پیوندی FA-MABC، توسط ایوانا برایویچ و همکاران [۴]، ارائه‌شده و مبتنی بر دو الگوریتم کرم شب‌تاب و الگوریتم کلونی زنبور مصنوعی چند-راهبردی[5] است. این دو

[2] Predicate Distance
[3] Similarity
[4] Hamming
[5] Multi-strategy

الگوریتم که از الگوریتم‌های پرکاربرد در زمینه‌ی بهینه‌سازی هستند، در گروه الگوریتم‌های مبتنی بر هوش جمعی، قرار دارند. ازآنجایی‌که الگوریتم کرم شب‌تاب و الگوریتم کلونی زنبور مصنوعی، ویژگی‌های مثبت خود را دارند، ترکیب آن‌ها می‌تواند یک هم‌افزایی ارزشمند ایجاد کند؛ بنابراین، یک الگوریتم پیوندی مشارکتی، بر اساس کرم شب‌تاب و کلونی زنبورهای مصنوعی توسط ایوانا و همکاران، توسعه یافته است. این الگوریتم برای حل مسائل پیچیده‌ی بهینه‌سازی عددی، ارائه شده است و جزء الگوریتم‌های مبتنی بر همکاری با ساختار چند-مرحله‌ای است. الگوریتم کرم شب‌تاب فضای جستجو را به‌صورت سراسری بررسی می‌کند تا مناطق مطلوب فضای جستجو را بیابد، درحالی‌که نسخه جدید الگوریتم کلونی زنبور مصنوعی، جستجوی محلی را انجام می‌دهد. علاوه‌براین، یک معیار تنوع برای کمک به معیارهای جابه‌جایی استفاده می‌شود؛ بنابراین FA-MABC دارای سه جزء اصلی است، بهینه‌ساز جستجوی سراسری، الگوریتم جستجوی محلی و معیار جابه‌جایی [۴]. نتایج حاصل از ارزیابی انجام‌شده توسط ایوانا و همکاران، نشان می‌دهد که الگوریتم پیشنهادی به لحاظ دقت راه‌حل، توانمندی و نرخ همگرایی، بهتر از الگوریتم‌های کرم شب‌تاب و کلونی زنبور مصنوعی، عمل می‌کند.

۳- پیشینه تحقیق

اُفوت [۳۶] در سال ۱۹۸۸، ایده آزمون مبتنی بر محدودیت (CBT)[1] را برای تولید خودکار داده آزمون، مطرح نمود. در این فن آزمون، کفایت داده آزمون، با محدودیت‌ها مشخص می‌شود و سه خصوصیت برای داده آزمونی که می‌تواند جهش‌ها را خاموش کند، معرفی شده است. فنون عملی در تولید داده آزمون، تلاش می‌کنند که زیرمجموعه‌ای از ورودی‌ها را با توجه به معیارهای آزمون، انتخاب کنند. فرض بر این است که این زیرمجموعه، ویژگی‌های لازم برای یافتن بخش بزرگی از خطاهای یک نرم‌افزار را دارد و می‌تواند سطحی از اطمینان را تضمین کند. تولید داده آزمون مبتنی بر محدودیت، از مفهوم تحلیل جهش برای تولید خودکار داده آزمون، استفاده می‌کند. چنین داده آزمونی، تنها برای خاموش کردن مجموعه‌ای از جهش‌های برنامه، تولید می‌شوند و می‌توان از آن‌ها، به‌عنوان یک مجموعه آزمون مؤثر در سایر آزمون‌ها استفاده نمود.

اُفوت و همکاران [۳۷] در سال ۱۹۹۹، برای فائق آمدن بر مشکلات روش CBT، روش اجرای پویای دامنه (DDR)[2] را پیشنهاد کردند. در این رویکرد، همان سه شرط برای داده آزمون، وجود دارد، اما نحوه رسیدگی به این شرط‌ها، بهبود می‌یابد. در این روش، بر اساس کاهش فضای ورودی در متغیرها، داده آزمون تولید می‌شود. DDR، مسئله‌ی تولید داده آزمون را به‌عنوان یک مسئله‌ی مبتنی بر مسیر پویا، در نظر می‌گیرد و با استفاده از گراف جریان کنترل و بر اساس یک روش جستجوی ابتکاری، داده‌های آزمون را با توجه به دامنه‌ی متغیرهای ورودی، تولید می‌کند.

پاپاداکیس و همکاران [۳۸] در سال ۲۰۱۰، چارچوبی خودکار با استفاده از فنون و ابزارهای متنوع، معرفی کردند که هدف آن، تولید خودکار آزمون مبتنی بر جهش، بود. در این کار، از فنون و ابزارهای موجود، مانند اجرای نمادین و آزمون تکاملی، برای خودکارسازی فعالیت‌های مرتبط با تولید ورودی آزمون، استفاده کردند. همچنین در این کار، از معیار جهش ضعیف، استفاده شده است. در چارچوب پیشنهادی، برای تولید مؤثر داده آزمون مبتنی بر جهش، از آزمون انشعاب استفاده شده است. نتایج روش پیشنهادی، در برنامه‌های جاوا اعمال شده و با سایر ابزارها، مقایسه شده است. رویکرد پیشنهادی در این مقاله، تبدیل خودکار مسئله‌ی خاموش کردن جهش‌ها به یک مسئله‌ی پوشش انشعاب است. این کار می‌تواند به تمرکز بر روی جهش‌های معین، کمک کند تا داده آزمون مؤثر برای خاموش کردن آن جهش‌ها، تولید شود. این رویکرد می‌تواند با اعمال برخی از اصلاحات در فنون و ابزارهای آزمون ساختاری، پیاده‌سازی شود و آزمون جهش را اجرا کند.

میشرا و همکاران [۳۹] در سال ۲۰۱۹، یک روش پیوندی برای تولید داده آزمون با استفاده از الگوریتم ژنتیک، پیشنهاد کردند. روش پیشنهادی، شامل آزمون مسیر و الگوریتم ژنتیک، برای تولید خودکار داده آزمون در آزمون جهش است. در ابتدا، داده آزمون مبتنی بر پوشش مسیر، تولید می‌شوند و سپس این داده آزمون، برای خاموش کردن جهش‌های برنامه، در آزمون جهش اعمال می‌شوند. روش پیشنهادی، می‌تواند داده آزمون اضافی را حذف کرده و اثربخشی آزمون را به لحاظ امتیاز جهش، بهبود بخشد. همچنین، از ماتریس تشخیص خطا، برای حذف داده‌های تکراری که جهش‌های یکسانی را خاموش می‌کنند، استفاده می‌شود.

رانی و همکاران [۴۰] در سال ۲۰۱۹، ایده‌ای برای تولید خودکار موارد آزمون، با استفاده از الگوریتم ژنتیک و آزمون جهش، پیشنهاد کردند که با محدودیت زمانی همراه است. آن‌ها برای کمینه‌سازی هزینه‌های مرتبط با آزمون جهش، از فن جهش انتخابی و عملگر جهش Delete برای تولید جهش‌های برنامه، استفاده کردند. فرایند پیشنهادی، زمانی متوقف می‌شود که به محدوده زمانی معین برسیم. روش پیشنهادی، در هر تکرار، سعی

[1] Constraint-Based Testing
[2] Dynamic Domain Reduction

می‌کند که اندازه مجموعه آزمون را با حذف ورودی‌هایی با تناسب پایین‌تر، کاهش دهد. از پوشش جهش نیز، به‌عنوان تابع تناسب استفاده شده است. نتایج روش پیشنهادی، نشان می‌دهد که بیش از ۹۰٪ جهش‌های برنامه، خاموش می‌شوند. این مقاله، به مسئله‌ی جهش‌های معادل، نپرداخته است.

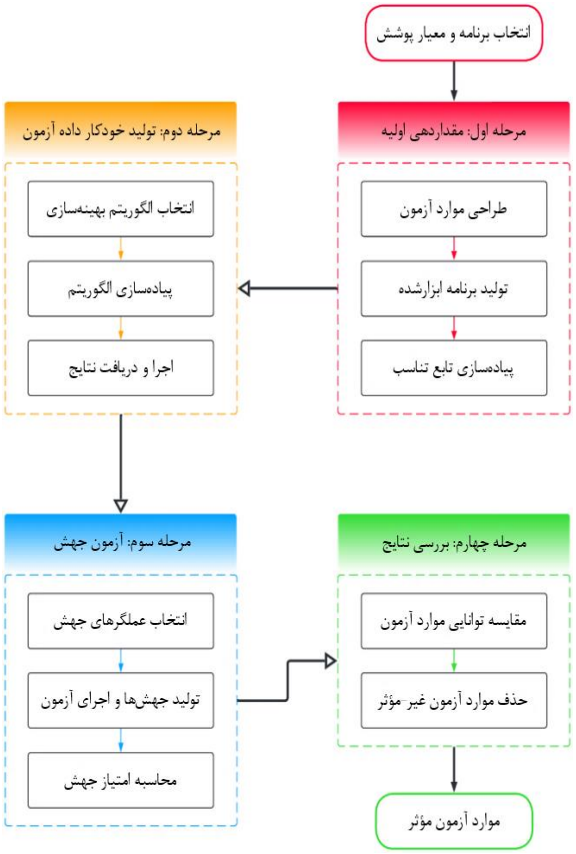
جاتانا و سوری [۴۱] در سال ۲۰۲۰، الگوریتم جستجوی کلاغ بهبودیافته (ICSA)[1] را برای تولید خودکار داده آزمون، پیشنهاد کردند که از مفهوم جهش بهره می‌برد. در این کار، رفتار هوشمندانه‌ی کلاغ‌ها و توزیع Cauchy، شبیه‌سازی می‌شود. ازآنجاکه الگوریتم جستجوی کلاغ، از مشکل گیر-افتادن در بهینه محلی، رنج می‌برد؛ روش پیشنهادی، تلاش می‌کند که توانایی اکتشاف الگوریتم موجود را با استفاده از مفهوم اعداد تصادفی Cauchy، بهبود بخشد. از مفهوم آزمون جهش نیز، برای تعریف تابع تناسب در رویکرد مبتنی بر جستجو، استفاده می‌شود. مشارکت اصلی این مقاله، ارائه یک نسخه بهبودیافته از الگوریتم جستجوی کلاغ، برای تولید داده آزمون در آزمون جهش است. ارزیابی روش پیشنهادی و مقایسه با سایر فرا-ابتکاری‌ها، اثربخشی ICSA را برای تولید مجموعه داده آزمون، نشان می‌دهد.

شرما و پاتیک [۳۴] در سال ۲۰۲۱، برای تولید و بهینه‌سازی داده آزمون، الگوریتم جستجوی کلاغ را به همراه تابع هدف بهبودیافته، پیشنهاد کردند. تابع هدف بهبودیافته، فاصله‌ی انشعاب و فاصله‌ی گزاره را ترکیب می‌کند تا بتواند مسیرهای بحرانی در گراف جریان کنترل را پوشش دهد. از الگوریتم جستجوی کلاغ نیز، برای خودکارسازی آزمون و تولید داده‌های آزمون با حداقل تعداد، استفاده می‌شود. تمرکز این مقاله، طراحی موارد آزمون برای تمام مسیرها، ازجمله مسیرهای بحرانی است و به پوشش مسیر ۱۰۰٪ رسیده است.

۴- روش پیشنهادی

در این بخش، چارچوبی با عنوان «شناسایی موارد آزمون مؤثر»، ارائه می‌گردد. «انتخاب برنامه تحت آزمون» و «تعیین معیار پوشش»، به‌عنوان ورودی‌های اصلی چارچوب پیشنهادی مطرح می‌شوند و مراحل چارچوب پیشنهادی، با توجه به این ورودی‌ها، اجرا خواهد شد. خروجی نهایی چارچوب پیشنهادی، مجموعه‌ای از موارد آزمون است که توانایی خاموش‌کردن حداکثر جهش‌های برنامه را دارند و شامل موارد آزمون غیر مؤثر و اضافی نیستند. شکل (۲)، مراحل و گام‌های چارچوب پیشنهادی را نشان می‌دهد.

[1] Improved Crow Search Algorithm



شکل (۲). چارچوب پیشنهادی

در روش پیشنهادی، برای معیار پوشش «مسیرهای مستقل»، از ترکیب فاصله انشعاب و فاصله گزاره، برای پیاده‌سازی تابع تناسب استفاده می‌کنیم. فاصله انشعاب، از طریق جدول (۲)، محاسبه شده و از طریق فرمول (۳)، ساده‌سازی می‌شود. همچنین از فاصله گزاره، برای محاسبهٔ اختلاف مسیر پیمایش‌شده و مسیر هدف، استفاده می‌کنیم. ترکیب این دو تابع تناسب، می‌تواند روش پیشنهادی را در تولید داده آزمون بهینه، کمک کند؛ بنابراین در معیار پوشش «مسیرهای مستقل»، از فرمول (۴)، برای پیاده‌سازی تابع تناسب، استفاده می‌شود.

$$Fitness\ Value = NBD(Branch\ Distance) + Predicate\ Distance \quad (۴)$$

همچنین برای معیار پوشش MC/DC، از ترکیب فاصله انشعاب و تابع شباهت برای پیاده‌سازی تابع تناسب، استفاده می‌کنیم. فاصله انشعاب، از طریق جدول (۳)، محاسبه شده و از طریق فرمول (۳)، ساده‌سازی می‌شود. همچنین برای محاسبه شباهت توالی اجراشده با توالی هدف، از تابع شباهت استفاده می‌کنیم. فرمول (۵)، نحوه محاسبه تابع تناسب برای معیار پوشش MC/DC را نشان می‌دهد.

$$Fitness\ Value = NBD(Branch\ Distance) + Similarity\ Function \quad (۵)$$

همچنین در این مقاله، از الگوریتم FA-MABC که توسط ایوانا و همکاران [۴] ارائه شده است، برای تولید خودکار داده

آزمون بهینه و از ابزار PITest[1]، برای انجام آزمون جهش استفاده می‌شود.

در روش پیشنهادی، برای شناسایی موارد آزمون مؤثر، توانایی هر مورد آزمون در شناسایی خطاهای برنامه و خاموش‌کردن جهش‌ها، سنجیده می‌شود. ابتدا، جدولی تحت عنوان «جهش‌های خاموش‌شده»، تشکیل می‌شود که در آن، جهش‌های خاموش‌شده توسط هر مورد آزمون، مشخص می‌شود. با بررسی این جدول، مشخص می‌شود که هر مورد آزمون، چه توانایی در خاموش‌کردن جهش‌ها دارد و امتیاز جهش برای هر مورد آزمون، چند است.

۵- ارزیابی روش پیشنهادی

برای ارزیابی روش پیشنهادی، از یک کامپیوتر شخصی با پردازنده Core(TM) i7-8700 و حافظه 8 GB استفاده شده است. همچنین از محیط توسعه IntelliJ و زبان برنامه‌نویسی جاوا، برای پیاده‌سازی نرم‌افزار ATDG و از محیط توسعه Eclipse، برای اجرای ابزارهای مربوط به تولید گراف جریان کنترل و از ابزار PITest، برای انجام آزمون جهش، استفاده شده است.

در ارزیابی روش پیشنهادی و اجرای الگوریتم‌های فرا-ابتکاری، از پارامترهای موجود در جدول (۴)، استفاده می‌شود. در این جدول، برای پارامتر Threshold، دو روش محاسبه، ذکر شده است.

جدول (۴). پارامترهای مورداستفاده در الگوریتم‌های فرا-ابتکاری

Beta 0	1	Theta	1
Beta min	0.2	Limit	Max Iteration × 0.01
Gamma	1	Alpha 0	0.8
Threshold	1		Max Iteration × 0.01
	2		$if(maxIteration < 200)Then$ $T = 2$ $if(200 \leq maxIte < 1000)\ Then$ $T = \frac{maxIteration}{100}$ $if(1000 \leq maxIte)\ Then$ $\frac{maxIteration}{5}$

روش پیشنهادی، با الگوریتم FA و الگوریتم ABC، مقایسه شده است. این دو الگوریتم، از الگوریتم‌های فرا-ابتکاری هستند که برای حل مسائل بهینه‌سازی کاربرد دارند. الگوریتم FA توسط یانگ [۴۲] و الگوریتم ABC توسط کارابوگا [۴۳] ارائه شده است.

جدول (۵)، نتایج روش پیشنهادی در برنامه تشخیص مثلث را نشان می‌دهد که از معیار پوشش «مسیرهای مستقل»، استفاده شده است. در این ارزیابی، ۵ مورد آزمون، طراحی شده است و مقدار پارامتر Threshold از طریق روش اول، محاسبه می‌شود. همچنین حدود متغیرهای مسئله، برابر با [0, 100] می‌باشد.

جدول (۵). نتایج ارزیابی برنامه تشخیص مثلث با معیار پوشش مسیرهای مستقل

الگوریتم	درصد پوشش	زمان اجرا (s)	امتیاز جهش	اندازه جمعیت	حداکثر تکرار
FA-MABC	٪۱۰۰	۰/۰۰۴۴	۷۹	۱۰۰	۵۰
FA	٪۱۰۰	۰/۰۰۸۴	۷۹-۷۵		
ABC	٪۱۰۰	۰/۰۰۴۹	-		
FA-MABC	٪۱۰۰	۰/۰۴۰۳	۷۹	۱،۰۰۰	۵۰۰
FA	٪۱۰۰	۰/۲۷۷۳	۷۹-۷۵		
ABC	٪۱۰۰	۰/۰۱۹۴	۸۳-۷۹		
FA-MABC	٪۱۰۰	۰/۱۸۷۵	۷۹	۵،۰۰۰	۱،۰۰۰
FA	٪۱۰۰	۶/۵۶۵۳	۷۹-۷۵		
ABC	٪۱۰۰	۰/۴۸۵۱	۷۹		
FA-MABC	٪۱۰۰	۰/۴۵۲۸	۷۹	۱۰،۰۰۰	۱،۰۰۰
FA	٪۱۰۰	۲۵/۳۸۱۰	۷۹		
ABC	٪۱۰۰	۲/۰۳۹۲	۷۹		

جدول (۶)، نتایج روش پیشنهادی در برنامه تشخیص مثلث را نشان می‌دهد که از معیار پوشش MC/DC، استفاده شده است. در این ارزیابی، ۱۹ مورد آزمون، طراحی شده است و مقدار پارامتر Threshold از طریق روش اول، محاسبه می‌شود. همچنین حدود متغیرهای مسئله، برابر با [0, 100] می‌باشد.

جدول (۶). نتایج ارزیابی برنامه تشخیص مثلث با معیار پوشش MC/DC

الگوریتم	درصد پوشش	زمان اجرا (s)	امتیاز جهش	اندازه جمعیت	حداکثر تکرار
FA-MABC	٪۸۴-٪۹۴	۰/۰۵۷۶	-	۱۰۰	۵۰
FA	٪۱۰۰	۰/۰۲۳۴	۹۶-۸۸		
ABC	٪۱۰۰	۰/۰۱۲۰	-		
FA-MABC	٪۱۰۰	۰/۱۰۰۵	۹۲-۸۸	۱،۰۰۰	۵۰۰
FA	٪۱۰۰	۱/۲۳۹۶	۹۲-۱۰۰		
ABC	٪۱۰۰	۰/۰۷۴۲	۹۶-۹۲		
FA-MABC	٪۱۰۰	۰/۵۹۰۳	۹۲	۵،۰۰۰	۱،۰۰۰
FA	٪۱۰۰	۲۷/۱۱۴	۱۰۰		
ABC	٪۱۰۰	۱/۹۱۶۶	۹۲-۸۸		
FA-MABC	٪۱۰۰	۰/۸۸۳۱	۹۲	۱۰،۰۰۰	۱،۰۰۰
FA	٪۱۰۰	۱۰۰/۵۶	۱۰۰		
ABC	٪۱۰۰	۶/۱۹۰	۹۶-۹۲		

[1] http://pitest.org

جدول (۷)، نتـایج روش پیشـنهادی در برنامـه تشـخیص مثلـث (کامل) را نشان می‌دهد که از معیار پوشش «مسیرهای مستقل»، استفاده شده است. در این ارزیابی، ۱۴ مورد آزمون، طراحی شده است و مقـدار پـارامتر Threshold از طریـق روش دوم، محاسـبه می‌شود. همچنین حـدود متغیرهـای مسـئله، برابـر بـا [0, 100] می‌باشد.

جدول (۷). نتایج ارزیابی برنامه تشخیص مثلث (کامل) با معیار پوشش مسیرهای مستقل

الگوریتم	درصد پوشش	زمان اجرا (s)	امتیاز جهش	اندازه جمعیت	حداکثر تکرار
FA-MABC	٪۷۸	۰/۰۵۱۵	–	۱۰۰	۵۰
FA	٪۱۰۰	۰/۰۴۹۰	۷۱-۶۹		
ABC	٪۷۸ – ٪۹۲	۰/۰۱۸۹	–		
FA-MABC	٪۷۸– ٪۸۵	۳/۴۸۴۹	–	۱،۰۰۰	۵۰۰
FA	٪۱۰۰	۲/۱۱۹۲	۷۱		
ABC	٪۱۰۰	۰/۰۹۶۳	۷۳-۶۹		
FA-MABC	٪۸۵ – ٪۱۰۰	۷/۸۷	–	۵،۰۰۰	۱،۰۰۰
FA	٪۱۰۰	۴۲/۳۰	۷۱		
ABC	٪۱۰۰	۲/۸۲۲	۷۳-۷۱		
FA-MABC	٪۹۲تا ٪۱۰۰	۵/۱۴	–	۱۰،۰۰۰	۱،۰۰۰
FA	٪۱۰۰	۱۶۴	۷۱		
ABC	٪۱۰۰	۳/۹۶۲	۷۳-۷۱		
FA-MABC	٪۱۰۰	۲/۶۳۳	۷۳-۷۱	۲۰،۰۰۰	۵،۰۰۰
FA	٪۱۰۰	۶۳۴	۷۱		
ABC	٪۱۰۰	۱۴/۱۹۲	۷۳-۷۱		

جدول (۸)، نتـایج روش پیشـنهادی در برنامـه تشـخیص مثلـث (کامل) را نشان می‌دهد که از معیـار پوشـش MC/DC، اسـتفاده شده است. در این ارزیابی، ۳۹ مورد آزمون، طراحی شده اسـت و مقدار پارامتر Threshold از طریق روش دوم، محاسـبه مـی‌شـود. همچنین حدود متغیرهای مسئله، برابر با [0, 100] می‌باشد.

جدول (۹)، نتایج روش پیشنهادی در برنامه محاسبه بزرگ‌تـرین مقسـوم‌علیـه مشـترک را نشـان مـی‌دهـد کـه از معیـار پوشـش «مسیرهای مستقل»، استفاده شده است. در این ارزیابی، ۲ مـورد آزمون، طراحی شده است و مقـدار پـارامتر Threshold از طریـق روش دوم، محاسبه می‌شود. همچنین حدود متغیرهـای مسـئله، برابر با [0, 100] می‌باشد.

جدول (۱۰)، نتایج روش پیشنهادی در برنامه محاسبه بزرگ‌تری مقسـوم‌علیـه مشـترک را نشـان مـی‌دهـد کـه از معیـار پوشـش «مسیرهای مستقل»، استفاده شده است. در این ارزیابی، ۵ مـورد آزمون، طراحی شده است و مقـدار پـارامتر Threshold از طریـق روش دوم، محاسبه می‌شود. همچنین حدود متغیرهـای مسـئله، برابر با [0, 100] می‌باشد.

جدول (۸). نتایج ارزیابی برنامه تشخیص مثلث (کامل) با معیار پوشش MC/DC

الگوریتم	درصد پوشش	زمان اجرا (s)	امتیاز جهش	اندازه جمعیت	حداکثر تکرار
FA-MABC	٪۸۲– ٪۹۲	۰/۰۶۴۹	–	۱۰۰	۵۰
FA	٪۱۰۰	۰/۰۶۲۴	۷۵-۷۳		
ABC	٪۹۲– ٪۱۰۰	۰/۰۲۵۰۲	–		
FA-MABC	٪۹۲– ٪۹۷	۲/۸۷۰۱	–	۱،۰۰۰	۵۰۰
FA	٪۱۰۰	۳/۱۳۴۴	۷۹-۷۷		
ABC	٪۱۰۰	۰/۱۷۱۱	۷۷		
FA-MABC	٪۹۷ – ٪۱۰۰	۲/۶۵۵۷	–	۵،۰۰۰	۱،۰۰۰
FA	٪۱۰۰	۶۸/۶۴	۷۹-۷۷		
ABC	٪۱۰۰	۴/۴۷۸	۷۷		
FA-MABC	٪۹۴– ٪۱۰۰	۶/۹۵۳	–	۱۰،۰۰۰	۱،۰۰۰
FA	۱۰۰	۲۶۱	۷۹-۷۷		
ABC	٪۱۰۰	۱۴/۶۴۲	۷۷		
FA-MABC	٪۱۰۰	۵/۱۵۶	۷۷-۷۳	۲۰،۰۰۰	۵،۰۰۰
FA	٪۱۰۰	۱۰۲۵	۷۹-۷۷		
ABC	٪۱۰۰	۵۰/۲۶۱	۷۷		

جدول (۹). نتایج ارزیابی برنامه محاسبه ب.م.م با معیار پوشش مسیرهای مستقل

الگوریتم	درصد پوشش	زمان اجرا (s)	امتیاز جهش	اندازه جمعیت	حداکثر تکرار
FA-MABC	٪۱۰۰	۰/۰۰۳۵	۶۳	۱۰۰	۵۰
FA	٪۱۰۰	۰/۰۰۲۲	۶۳		
ABC	٪۱۰۰	۰/۰۰۰۶	۶۳		
FA-MABC	٪۱۰۰	۰/۰۰۸۴	۶۳	۱،۰۰۰	۵۰۰
FA	٪۱۰۰	۰/۰۸۷	۶۳		
ABC	٪۱۰۰	۰/۰۰۶۵	۷۵-۶۳		
FA-MABC	٪۱۰۰	۰/۱۳۷	۷۵-۶۳	۲۰،۰۰۰	۵،۰۰۰
FA	٪۱۰۰	۳۲/۷۲	۷۵-۶۳		
ABC	٪۱۰۰	۰/۳۶۹	۷۵-۶۳		

جدول (۱۰). نتایج ارزیابی برنامه محاسبه ب.م.م با معیار پوشش MC/DC

الگوریتم	درصد پوشش	زمان اجرا (s)	امتیاز جهش	اندازه جمعیت	حداکثر تکرار
FA-MABC	٪۱۰۰	۰/۰۰۱۸	۸۸-۶۳	۱۰۰	۵۰
FA	٪۱۰۰	۰/۰۰۲۳	۷۵-۶۳		
ABC	٪۱۰۰	۰/۰۰۱۴	۸۸-۷۵		
FA-MABC	٪۱۰۰	۰/۰۱۷	۸۸-۶۳	۱۰۰۰	۵۰۰
FA	٪۱۰۰	۰/۱۴۴	۸۸-۷۵		
ABC	٪۱۰۰	۰/۰۱۴	۷۵-۱۰۰		
FA-MABC	٪۱۰۰	۰/۳۲۸	۸۸	۲۰۰۰۰	۵۰۰۰
FA	٪۱۰۰	۵۱/۳۰	۸۸-۷۵		
ABC	٪۱۰۰	۰/۴۸۰	۷۵-۶۳		

۵-۱- نتایج ارزیابی

نتایج ارزیابی‌های انجام‌شده، نشان می‌دهد که:

- ✓ الگوریتم FA-MABC در نیمی از اجراها، هزینه زمانی کمتری نسبت به دو الگوریتم دیگر دارد.
- ✓ الگوریتم FA در بخشی از اجراها، درصد پوشش و امتیاز جهش بالایی نسبت به دو الگوریتم دیگر دارد؛ این امر، معمولاً به دلیل کاهش حدود متغیرهای مسئله، اتفاق می‌افتد.
- ✓ الگوریتم FA-MABC، حدود متغیرهای مسئله را کاهش نمی‌دهد و از این نظر، نسبت به دو الگوریتم دیگر، برتری دارد.
- ✓ امتیاز جهش در معیار پوشش «شرط اصلاح‌شده / تصمیم»، بیش‌تر از معیار پوشش «مسیرهای مستقل» است.

با توجه به این نتایج، روش پیشنهادی توانسته است زمان تولید خودکار داده آزمون بهینه، برای پوشش کد منبع را کاهش داده و موجب صرفه‌جویی در هزینه زمانی شود. همچنین، پوشش کد منبع، از طریق دو معیار پوشش «مسیرهای مستقل» و «شرط اصلاح‌شده / تصمیم»، ارزیابی شده است و در هر دو معیار، هزینه زمانی کاهش یافته است.

علاوه بر این، روش پیشنهادی، با طراحی موارد آزمون مبتنی بر معیار پوشش و تولید داده آزمون بهینه، توانسته است آزمون جهش را با موفقیت اجرا کرده و امتیاز جهش بالایی به دست آورد و از این طریق، به شناسایی بهتر خطاهای کد منبع، کمک کند. همچنین، با تولید جدول «جهش‌های خاموش‌شده»، روش پیشنهادی توانسته است موارد آزمون مؤثر برای آزمون جهش را استخراج کرده و از این طریق، موجب کاهش تعداد موارد آزمون و کاهش هزینه زمانی در آزمون جهش شود.

۶- نتیجه‌گیری و کارهای آینده

یکی از بخش‌های پرهزینه و مهم در آزمون نرم‌افزار، تولید داده آزمون است که برای کاهش هزینهٔ آن، روش‌های خودکارسازی استفاده می‌شود. همچنین یکی از موارد مهم در آزمون جهش، طراحی موارد آزمونی است که بتوانند امتیاز جهش بالایی، کسب کنند. در این مقاله، برای طراحی موارد آزمون، از دو معیار پوشش «مسیرهای مستقل» و «شرط اصلاح‌شده / تصمیم»، استفاده شده و به‌منظور کاهش هزینه زمانی در تولید داده آزمون، از الگوریتم FA-MABC استفاده می‌شود.

در این مقاله، چارچوبی به‌منظور شناسایی موارد آزمون مؤثر، ارائه شده و بر اساس آن، روشی مبتنی بر الگوریتم FA-MABC و معیار پوشش «مسیرهای مستقل» و «شرط اصلاح‌شده / تصمیم»، پیشنهاد شده است. ورودی‌های روش پیشنهادی، عبارت‌اند از «انتخاب برنامه تحت آزمون» و «تعیین معیار پوشش» که قبل از شروع، انتخاب می‌شوند و سایر گام‌ها، با توجه به این ورودی‌ها، طی می‌شود. خروجی روش پیشنهادی نیز، مجموعه‌ای از موارد آزمون است که توانایی آن‌ها، در خاموش‌کردن جهش‌های برنامه، ارزیابی شده است و این مجموعه آزمون، شامل مورد آزمون اضافی و غیر مؤثر نیست.

همچنین، برای تولید خودکار داده آزمون بهینه و کمک به انجام آزمون جهش، نرم‌افزاری با نام ATDG، توسعه یافته است. این نرم‌افزار، از سه بخش تشکیل شده است؛ بخش اول، برای تعیین ورودی‌ها و دریافت موارد آزمون؛ بخش دوم، برای تعیین پارامترهای الگوریتم و دریافت نتایج بهینه‌سازی و بخش سوم، برای تولید کد Test مربوط به آزمون جهش، توسعه یافته است. علاوه بر این، این نرم‌افزار شامل سه برنامه تحت آزمون، دو معیار پوشش و سه الگوریتم بهینه‌سازی است.

نتایج ارزیابی روش پیشنهادی، نشان می‌دهد که الگوریتم FA-MABC، قادر است در مدت‌زمان کمتر نسبت به دو الگوریتم دیگر، داده آزمون بهینه را تولید کند. همچنین این الگوریتم در اجراهای بالا نیز، زمان اجرای کمتری نسبت به دو الگوریتم دیگر دارد و از این نظر، در برنامه‌های بزرگ، کارایی مفیدتری خواهد داشت. علاوه بر این، الگوریتم FA-MABC، هنگام بهینه‌سازی داده آزمون، حدود داده آزمون را تغییر نمی‌دهد و داده آزمون بهینه، به حدود مورد تقاضا، نزدیک هستند. ازنظر امتیاز جهش نیز، معیار پوشش «شرط اصلاح‌شده / تصمیم» در مقابل معیار پوشش «مسیرهای مستقل»، امتیاز جهش بالایی کسب می‌کند و امتیاز جهش کسب‌شده توسط هر الگوریتم نیز تقریباً برابر است و اختلاف موجود، عمدتاً تصادفی است. در زمینهٔ استخراج موارد آزمون مؤثر، با استفاده از جدول «جهش‌های خاموش‌شده»، می‌توان موارد آزمون مؤثر را تفکیک کرده و موجب کاهش تعداد

موارد آزمون شود. این موارد آزمون، در آزمون جهش، اعمال می‌شوند و امتیاز جهش آن‌ها، برابر با امتیاز جهش تمام موارد آزمون طراحی‌شده است.

با توجه به چارچوب و روش پیشنهادی، موارد زیر به‌عنوان پیشنهادهایی برای کارهای آینده، مطرح می‌شود.

- ✓ استفاده از سایر الگوریتم‌های بهینه‌سازی در تولید خودکار داده آزمون، با هدف کاهش هزینه زمانی و افزایش دقتِ داده آزمون تولیدشده.
- ✓ استفاده از سایر معیارهای پوشش کد، مانند معیار «پوشش مسیر اصلی»، «پوشش مسیر کامل» و «پوشش شرط چندگانه» برای افزایش تعداد موارد آزمون و اعمال آن‌ها در آزمون جهش.
- ✓ استفاده از معیار پوشش جهش، به‌عنوان تابع تناسب و ارزیابی راه‌حل پیشنهادی توسط الگوریتم.
- ✓ ترکیب موارد آزمون مؤثر که با استفاده از چند معیار پوشش، تولید شده‌اند.
- ✓ تولید خودکار داده آزمون برای برنامه‌هایی که مقادیر ورودی آن‌ها، از نوع دادهٔ double یا String است.

۷- منابع

[1] D. Galin, *Software quality assurance: from theory to implementation*. Pearson education, 2004.

[2] J. L. Mitchell and R. Black, Advanced Software Testing: Guide to the ISTQB Advanced Certification As an Advanced Technical Test Analyst. Vol. 3, 2nd Edition. Rocky Nook, 2015.

[3] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649-678, 2010, doi: 10.1109/TSE.2010.62.

[4] B. Ivona, S. S. Predrag, L. Shuai, and C. Xinwei, "A Hybrid Firefly and Multi-Strategy Artificial Bee Colony Algorithm," *International Journal of Computational Intelligence Systems*, vol. 13, no. 1, pp. 810-821, 2020/06/23 2020, doi: 10.2991/ijcis.d.200612.001.

[5] IEEE Std 730-2014, "IEEE Standard for Software Quality Assurance Processes", 2014. doi: 10.1109/IEEESTD.2014.6835311.

[6] ISO/IEC/IEEE 24765:2017(E), "ISO/IEC/IEEE International Standard - Systems and software engineering--Vocabulary, 2017. doi: 10.1109/IEEESTD.2017.8016712.

[7] *Structured programming*. Academic Press Ltd., 1972.

[8] M. Hamblen. "Killer software: 4 lessons from the deadly 737 MAX crashes." fierceelectronics. https://www.fierceelectronics.com/electronics/killer-software-4-lessons-from-deadly-737-max-crashes (accessed 2022).

[9] J. Herkert, J. Borenstein, and K. Miller, "The Boeing 737 MAX: Lessons for Engineering Ethics," *Science and Engineering Ethics*, vol. 26, no. 6, pp. 2957-2974, 2020/12/01 2020, doi: 10.1007/s11948-020-00252-y.

[10] H. Zhu and P. A. V. Hall, "Test data adequacy measurement," *Software Engineering Journal*, vol. 8, no. 1, pp. 21-30, 1993.

[11] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Comput. Surv.*, vol. 29, no. 4, pp. 366–427, 1997, doi: 10.1145/267580.267590.

[12] B. Choi, "Test adequacy measurement using a combination of criteria," *International Journal of Reliability, Quality and Safety Engineering*, vol. 7, pp. 191-203, 2000.

[13] A. P. Mathur, *Foundations of software testing, 2E*. Pearson Education India, 2013.

[14] V. Sharma. "What is Software Quality Assurance And Types of Testing?" volumetree. https://www.volumetree.com/what-is-software-quality-assurance-and-types-of-testing/ (accessed 2022).

[15] "Quality Assurance, Quality Control and Testing — the Basics of Software Quality Management." altexsoft. https://www.altexsoft.com/whitepapers/quality-assurance-quality-control-and-testing-the-basics-of-software-quality-management/ (accessed 2022).

[16] A. Spillner and T. Linz, *Software Testing Foundations, 5th Edition: A Study Guide for the Certified Tester Exam*. Rocky Nook, 2021.

[17] N. T. "Basis Path Testing." binaryterms. https://binaryterms.com/basis-path-testing.html (accessed 2022).

[18] "Basis Path Testing in Software Testing." geeksforgeeks. https://www.geeksforgeeks.org/basis-path-testing-in-software-testing/ (accessed 2022).

[19] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308-320, 1976, doi: 10.1109/TSE.1976.233837.

[20] P. C. Jorgensen and B. DeVries, *Software Testing: A Craftsman's Approach, Fifth Edition*. Taylor & Francis, 2021.

[21] R. Lipton, "Fault diagnosis of computer programs," *Student report, Carnegie Mellon University*, vol. 2, p. 2, 1971.

[22] A. J. Offutt and R. H. Untch, "Mutation 2000: Uniting the Orthogonal," in *Mutation Testing for the New Century*, W. E. Wong Ed. Boston, MA: Springer US, 2001, pp. 34-44.

[23] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer," *Computer*, vol. 11, no. 4, pp. 34-41, 1978, doi: 10.1109/C-M.1978.218136.

[24] R. A. DeMillo, D. S. Guindi, W. M. McCracken, A. J. Offutt, and K. N. King, "An extended overview of the Mothra software testing environment," in *Proceedings. Second Workshop on Software Testing, Verification, and Analysis*, 19-21 July 1988 1988, pp. 142-151, doi: 10.1109/WST.1988.5369.

[25] K. Naik and P. Tripathy, *Software Testing and*

[35] J.-C. Lin and P.-L. Yeh, "Automatic test data generation for path testing using GAs," *Information Sciences*, vol. 131, no. 1, pp. 47-64, 2001/01/01/ 2001, doi: 10.1016/S0020-0255(00)00093-1.

[36] A. J. Offutt, "Automatic Test Data Generation," *PhD Thesis, Georgia Institute of Technology, Atlanta, Georgia*, vol. Technical report GIT-ICS 88/28, 1988.

[37] A. J. Offutt, Z. Jin, and J. Pan, "The dynamic domain reduction procedure for test data generation," *Software: Practice and Experience*, vol. 29, no. 2, pp. 167-193, 1999.

[38] M. Papadakis, N. Malevris, and M. Kallia, "Towards automating the generation of mutation tests," presented at the Proceedings of the 5th Workshop on Automation of Software Test, Cape Town, South Africa, 2010.

[39] D. B. Mishra, R. Mishra, A. A. Acharya, and K. N. Das, "Test Data Generation for Mutation Testing Using Genetic Algorithm," in *Soft Computing for Problem Solving*, Singapore, J. C. Bansal, K. N. Das, A. Nagar, K. Deep, and A. K. Ojha, Eds., 2019 2019: Springer Singapore, pp. 857-867.

[40] S. Rani, H. Dhawan, G. Nagpal, and B. Suri, "Implementing Time-Bounded Automatic Test Data Generation Approach Based on Search-Based Mutation Testing," in *Progress in Advanced Computing and Intelligent Engineering*, Singapore, C. R. Panigrahi, A. K. Pujari, S. Misra, B. Pati, and K.-C. Li, Eds., 2019 2019: Springer Singapore, pp. 113-122.

[41] N. Jatana and B. Suri, "An Improved Crow Search Algorithm for Test Data Generation Using Search-Based Mutation Testing," *Neural Processing Letters*, vol. 52, no. 1, pp. 767-784, 2020/08/01 2020, doi: 10.1007/s11063-020-10288-7.

[42] X.-S. Yang, "Firefly Algorithms for Multimodal Optimization," in *Stochastic Algorithms: Foundations and Applications*, Berlin, Heidelberg, O. Watanabe and T. Zeugmann, Eds., 2009// 2009: Springer Berlin Heidelberg, pp. 169-178.

[43] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Technical report-tr06, Erciyes university, engineering faculty, computer …, 2005.

Quality Assurance: Theory and Practice. Wiley-Spektrum, 2008.

[26] J. Gao, H.-S. Tsao, and Y. Wu, *Testing and quality assurance for component-based software*. Artech House, 2003.

[27] B. B. Agarwal, S. P. Tayal, and M. Gupta, *Software Engineering and Testing*. Jones & Bartlett Learning, 2010.

[28] F. C. M. Souza, M. Papadakis, V. H. S. Durelli, and M. E. Delamaro, "Test Data Generation Techniques for Mutation Testing: A Systematic Mapping," presented at the Workshop on Experimental Software Engineering (ESELAW), Pucón, Chile, 2014.

[29] L. T. M. Hanh, N. T. Binh, and K. T. Tung, "Survey on Mutation-based Test Data Generation," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 5, no. 5, pp. 1164-1173, 2015.

[30] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, 2003, doi: 10.1145/937503.937505.

[31] M. Črepinšek, S.-H. Liu, and M. Mernik, "Exploration and exploitation in evolutionary algorithms: A survey," *ACM Comput. Surv.*, vol. 45, no. 3, p. Article 35, 2013, doi: 10.1145/2480741.2480752.

[32] N. Tracey, J. Clark, K. Mander, and J. McDermid, "An automated framework for structural test-data generation," in *Proceedings 13th IEEE International Conference on Automated Software Engineering (Cat. No.98EX239)*, 13-16 Oct. 1998 1998, pp. 285-288, doi: 10.1109/ASE.1998.732680.

[33] Z. Awedikian, K. Ayari, and G. Antoniol, "MC/DC automatic test input data generation," presented at the Proceedings of the 11th Annual conference on Genetic and evolutionary computation, Montreal, Québec, Canada, 2009.

[34] M. Sharma and B. Pathik, "Crow Search Algorithm with Improved Objective Function for Test Case Generation and Optimization," *Intelligent Automation & Soft Computing*, vol. 32, no. 2, pp. 1125-1140, 2021.