Tabriz Journal of Electrical Engineering (TJEE), vol. 52, no. 2, Summer 2022 Serial no. 100
DOI: 10.22034/tjee.2022.15427

Minimum Covering Array Generation Using Success-History and Linear Population Size Reduction based Adaptive Differential Evolution Algorithm

Einollah Pira[1,*], Vahid Rafe[2], Sajad Esfandyari[2]

[1]Faculty of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran
[2]Department of Computer Engineering, Faculty of Engineering, Arak University, Arak, Iran
Emails: pira@azaruniv.ac.ir, v-rafe@araku.ac.ir, sajad.a1367@gmail.com

Short Abstract
Exhaustive testing of software systems with a large number of input parameters and combinations between them often causes the problem of combinatorial explosion. Combinatorial *t*-way testing is a technique that generates an array of test cases to maximize combinations covering of between input parameters. Generating a minimum covering array is an optimization problem that many strategies based on metaheuristic algorithms such as teaching and learning based optimization, particle swarm optimization, and genetic and cuckoo search algorithms have been used for solving it. Although these strategies have produced smaller covering arrays, complete minimization has not yet been performed. In this paper, we propose a new strategy based on the success-history and linear population size reduction based adaptive differential evolution algorithm (so-called LSHADE), which is one of winners of IEEE CEC competitions, to generate minimum covering array. The results of Friedman mean rank show that the LSHADE strategy has the first rank in terms of generating the covering array with the lowest size and the lowest average number of algorithmic calls, compared to mathematics based strategies such as TConfig, greedy strategies such as IPOG, Jenny and PICT and meta-heuristics such as GS, TLBO, HC-BAT, PSTG, WOA , BAPSO and GSTG. While, in terms of the average number of fitness function evaluations and the average runtime, this strategy has the first rank after the GS strategy. Moreover, the convergence diagrams confirm the high convergence speed of this strategy compared to the other meta-heuristic strategies.

Keywords
Exhaustive testing, combinatorial explosion, t-way testing, minimum covering array, differential evolution.

1- Short Introduction

In functionality (black-box) testing of a software, the internal structure of the software is ignored and the requirements of the software are considered for test case generation. For the black-box test to be successful, it should be comprehensive. It is possible that the exhaustive testing of complex software systems cause the problem of combinatorial explosion. Combinatorial *t*-way testing is a technique that generates an array of test cases to maximize combinations covering of between input parameters. Due to the NP-hardness of minimum covering array generation, different metaheuristic algorithms-based strategies such as teaching-learning optimization, particle swarm optimization, genetic and cuckoo search algorithms have been proposed to solve it.

2- Proposed Work and Methodology (including comprision, simulation/experimental results and discusion)

Although the proposed strategies have produced smaller covering arrays, complete minimization has not yet been obtained. Success-History based Adaptive DE (SHADE) is an example of algorithms which use a historical memory including successful parameter settings that were previously found during the run, to generate new parameter setting. In SHADE, the paremeters of scaling factor and crossover rate are automatically adjusted, whereas the population size remains fixed throughout the search. Since small population sizes accelerate the convergence of evolutionary algorithms, the new version of SHADE (so-called LSHADE) is proposed in which the population size of SHADE is continuously reduced in accordance with a linear function. In this paper, we propose a new strategy based on LSHADE to generate minimum covering array. The implementation results of this strategy along with other strategies on 73 configurations show that the LSHADE strategy in 80% of cases has the best performance compared to mathematical based strategies such as TConfig (4%), greedy such as IPOG (12%), Jenny (2%), PICT (0%) and metaheuristic such as GS (33%), TLBO (23%), HC-BAT (17%) and PSTG (11%). As a future work, LSHADE can be combined with other metaheuristic algorithms or we can use the one-parameter-at-a-time method instead of the one-row-at-a-time method.

3- Conclusion

Combinatorial *t*-way testing is a technique that generates an array of test cases to manage combinatorial explosion problem. In this paper, we have used the success-history and linear population size reduction based adaptive differential evolution algorithm (so-called LSHADE) to generate minimum covering array. The results of Friedman mean rank show that the LSHADE strategy has the first rank in terms of generating the covering array with the lowest size and the lowest average number of algorithmic calls, compared to mathematics based strategies such as TConfig, greedy strategies such as IPOG, Jenny and PICT and meta-heuristics such as GS, TLBO, HC-BAT, PSTG, WOA , BAPSO and GSTG. While, in terms of the average number of fitness function evaluations and the average runtime, this strategy has the first rank after the GS strategy. Moreover, the convergence diagrams confirm the high convergence speed of this strategy compared to the other meta-heuristic strategies.

4- References

[1] R. Tanabe and A. S. Fukunaga, "Improving the search performance of SHADE using linear population size reduction," in *2014 IEEE congress on evolutionary computation (CEC)*, 2014: IEEE, pp. 1658-1665.
[2] B. S. Ahmed, K. Z. Zamli, and C. P. Lim, "Constructing a t-way interaction test suite using the particle swarm optimization approach," *International Journal of Innovative Computing, Information and Control*, vol. 8, no. 1, pp. 431-452, 2012.

تولید آرایه پوشش کمینه با استفاده از الگوریتم تکامل تفاضلی تطبیقی مبتنی بر تاریخچه موفقیت و کاهش خطی اندازه جمعیت

عین‌الله پیرا
استادیار، دانشکده فناوری اطلاعات و مهندسی کامپیوتر، دانشگاه شهیدمدنی آذربایجان، تبریز، ایران

وحید رافع
دانشیار، دانشکده فنی‌مهندسی، دانشگاه اراک، ایران

سجاد اسفندیاری
دکتری کامپیوتر، دانشکده فنی‌مهندسی، دانشگاه اراک، ایران

چکیده
تست جامع سیستم‌های نرم‌افزاری با تعداد زیادی پارامتر ورودی و ترکیبات بین آنها اغلب باعث وقوع مشکل انفجار ترکیبی می‌شود. تست ترکیبی *t*-ستونی تکنیکی است که با تولید آرایه‌ای از نمونه‌های تست به پوشش حداکثری ترکیبات ما بین پارامترهای ورودی می‌پردازد. تولید آرایه پوشش کمینه یک مساله بهینه‌سازی است که الگوریتم‌های فراابتکاری زیادی از جمله بهینه‌سازی مبتنی بر آموزش و یادگیری، ازدحام توده ذرات، ژنتیک و الگوریتم جستجوی فاخته برای حل آن به کار رفته‌اند. اگر چه این الگوریتم‌ها توانسته‌اند آرایه‌های پوشش با اندازه‌های کوچک‌تر را تولید کنند ولی هنوز کمینه‌سازی کامل انجام نشده است. در این مقاله، یک استراتژی جدیدی برپایه الگوریتم تکامل تفاضلی تطبیقی مبتنی بر تاریخچه موفقیت و کاهش خطی اندازه جمعیت (معروف به LSHADE) که جزو برندگان کنگره IEEE در محاسبات تکاملی است، جهت تولید آرایه پوشش کمینه ارائه می‌کنیم. نتایج آزمون فریدمن نشان می‌دهند که استراتژی LSHADE دارای اولین رتبه از نظر معیارهای تولید آرایه پوشش با کمترین اندازه و کمترین تعداد متوسط فراخوانی‌های الگوریتمی در مقایسه با استراتژی‌های مبتنی بر ریاضی از جمله TConfig، حریصانه از جمله IPOG، Jenny وPICT و فراابتکاری از جمله GS، TLBO،HC-BAT، PSTG، WOA، BAPSO و GSTG است. در حالی‌که، از نظر معیارهای تعداد متوسط ارزیابی‌های تابع محاسبه وزن و متوسط زمان اجرا، این استراتژی بعد از استراتژی GS، دارای اولین رتبه است. ضمناً، نمودارهای همگرایی سرعت همگرایی بالایِ این استراتژی را در مقایسه با استراتژی‌های فراابتکاری دیگر تایید می‌کنند.

کلمات کلیدی
تست جامع، انفجار ترکیبی، آزمون *t*-ستونی، آرایه پوشش کمینه، الگوریتم تکامل تفاضلی.

نام نویسنده مسئول: عین‌الله پیرا
ایمیل نویسنده مسئول: pira@azaruniv.ac.ir

تاریخ ارسال مقاله: ۱۴۰۱/۰۱/۱۵
تاریخ(های) اصلاح مقاله: ۱۴۰۱/۰۴/۱۶
تاریخ پذیرش مقاله: ۱۴۰۱/۰۶/۰۶

۱- مقدمه

تست یکی از مهم‌ترین مراحل چرخه تولید سیستم‌های نرم‌افزاری است که سعی می‌شود خطاهای سیستم مورد نظر با صرف هزینه کم‌تری کشف شوند [۱]. فرآیند تست شامل تولید نمونه‌های تست، اجرای نرم‌افزار بازای این نمونه‌ها و سپس تحلیل نتایج حاصله است. فرآیند تست، سیستم تحتِ ساخت را از دو جنبه ساختاری و عملکردی مورد بررسی قرار می‌دهد. در جنبه ساختاری (آزمون جعبه-سفید)، نمونه‌های تست با توجه به ساختار و کد پیاده‌سازی نرم-افزار طراحی می‌شوند. در حالی‌که، در جنبه عملکردی (جعبه-سیاه)، ساختار داخلی نرم‌افزار نادیده گرفته شده و نیازمندی‌ها و خصوصیات نرم‌افزار مبنای تولید نمونه‌های تست هستند [۲]. در واقع، با توجه به این نیازمندی‌ها می‌توان پارامترهای ورودی به نرم‌افزار و خروجی‌های مدنظر بازای این پارامترها را استخراج کرد. شرط موفقیت آمیز بودن تست جعبه-سیاه این است که بصورت جامع باشد به این مفهوم که نمونه‌های تست شامل همه ترکیبات ممکن از پارامترهای ورودی باشند. مهم‌ترین مشکل تست جامع این است که در سیستم-های نرم‌افزاری بزرگ و پیچیده که دارای تعداد زیادی پارامتر ورودی هستند ممکن است مشکل انفجار ترکیبی رخ دهد.

تست *t*-ستونی تکنیکی است که با تولید آرایه‌ای از نمونه‌های تست به پوشش حداکثری ترکیبات ما بین پارامترهای ورودی می‌پردازد. مقدار *t* در این تکنیک که اصطلاحاً قوه نیز گفته می‌شود مشخص می‌کند که مجموعه تست، به‌جای پوشش تمام ترکیبات متفاوت از پارامترهای سیستم، فقط شامل ترکیبات *t* پارامتر باشد. در [۳] نشان داده شده است که اکثر خطاهای موجود در نرم‌افزار در نتیجه تعامل دو تا حداکثر شش پارامتر ایجاد می‌شوند و این نشان می‌دهد که تست *t*-ستونی با قوه حداکثر t = 6 می‌تواند یک مجموعه تست را تولید کرده که توانایی تشخیص اکثر خطاهای سیستم نرم‌افزاری تحت ساخت را داشته باشد. آرایه پوشش تولید شده توسط تست *t*-ستونی در یک سیستم با تعداد *p* پارامتر ورودی، که هر کدام دارای *d* مقدار متفاوت باشد،

دارای $\binom{p}{t} \times d^t$ نمونه تست متفاوت خواهد بود که یک عدد نمایی برحسب قوه t است. در نتیجه، تولید آرایه پوشش کمینه یک مساله بهینه‌سازی NP-سخت است [۴]. در حالت کلی، اندازه کمینه آرایش پوشش از قبل نامشخص است و تولید آرایه پوشش با اندازه کم‌تر مطلوب است. برای این منظور، الگوریتم‌های متفاوتی ارائه شده‌اند که می‌توان آنها را به دو دسته کلی دقیق و تقریبی تقسیم‌بندی کرد. الگوریتم‌های دقیق که مبتنی بر روش‌های ریاضی هستند می‌توانند جواب بهینه را بصورت دقیق پیدا کنند. بعنوان مثال، الگوریتم‌های Combinatorial Test Services [۵] و TConfig [۶] از روش ریاضی جهت تولید آرایه پوشش بهینه بهره می‌برند. مشکل عمده این الگوریتم‌ها این است که فقط می‌توانند جواب بهینه را برای پیکربندی‌های خاص و کوچک تولید کنند.

الگوریتم‌های تقریبی می‌توانند جواب‌های تقریبا بهینه را در زمان‌های کوتاه تری نسبت به الگوریتم‌های دقیق پیدا کنند. الگوریتم‌های تصادفی از ساده‌ترین نوع الگوریتم‌های تقریبی هستند که نمونه‌های تست را بطور تصادفی تولید می‌کنند [۷]. الگوریتم‌های حریصانه نوع دیگری از الگوریتم‌های تقریبی هستند که از دو روش یک-پارامتر-در-یک-زمان[۱] (OPT) و یک-سطر-در-یک-زمان[۲] (ORT) بهره می‌برند [۸]. در روش OPT، ابتدا آرایه پوشش فقط برای دو پارامتر تولید می‌شود سپس با اضافه کردن یک پارامتر جدید، آرایه پوشش موجود بروزرسانی شده و این مراحل تا موقعی تکرار می‌شوند که پوشش، کامل شود. از الگوریتم‌های مبتنی بر روش OPT می‌توان به الگوریتم‌های IPOG [۹]، IPOG-D [۱۰] و IPOG-F [۱۱] اشاره کرد. روش ORT، در هر مرحله، یک نمونه تست با بیشترین پوشش از ترکیبات ممکن از t پارامتر را تولید کرده و آن را به آرایه پوشش اضافه می‌کند و این روند را تا موقعی تکرار می‌کند که تمام ترکیبات مورد نظر پوشش داده شوند. از مهم‌ترین الگوریتم‌های مبتنی بر روش ORT می‌توان به الگوریتم‌های TVG [۱۲]، GTWay [۱۳]، CTE-XL [۱۴]، PICT [۱۵] و Jenny [۱۶] اشاره کرد. مهمترین مشکل این الگوریتم‌ها عدم توانایی در فرار از بهینگی محلی است.

الگوریتم‌های فراابتکاری نوع دیگری از الگوریتم‌های تقریبی هستند که می‌توانند آرایه‌های پوشش تقریبا بهینه را در زمان معقولی تولید کنند. همچنین، این الگوریتم‌ها مشکل بهینگی محلی ندارند. این الگوریتم‌ها که مبتنی بر روش ORT هستند ابتدا یک مجموعه تصادفی از نمونه‌های تست تولید کرده سپس با استفاده از عملگرهایی آن‌ها را بهبود داده و در نهایت، نمونه تست با بیشترین پوشش را انتخاب می‌کنند و این مراحل را تا موقعی تکرار می‌کنند که تمام ترکیبات ممکن از t پارامتر پوشش داده شوند. الگوریتم ژنتیک نوعی الگوریتم فراابتکاری است که از اصل تکاملی داروین (اصل بقاء شایسته‌ترین‌ها) نشات می‌گیرد. در هر تکرارِ الگوریتم، تعدادی از نمونه‌های تست با بیشترین پوشش انتخاب می‌شوند، سپس عملگرهای ادغام و جهش روی آنها اِعمال شده و جمعیت نسلِ بعدی از میان نمونه‌های تست نسل فعلی و تولید شده انتخاب می‌شوند. از الگوریتم‌های مبتنی بر الگوریتم ژنتیک می‌توان به استراتژی ژنتیک (GS) [۱۷] و استراتژی ترکیبیِ نخبه‌گرای بهینه‌سازی پرندگان مهاجر و ژنتیک (EMBO-GA) [۱۸] اشاره کرد. استراتژی GS، در هر تکرار، به تعداد *CrossRate*N* عمل ترکیب و تعداد *MutRate*N* عمل جهش را بطور مداوم روی هر نمونه تست انجام می‌دهد. پارامترهای *CrossRate* و *MutRate*، به‌ترتیب به نرخ ترکیب و جهش اشاره دارند. نتایج اجرای استراتژی GS روی چندین پیکربندی مختلف نشان می‌دهند که این استراتژی دارای کارایی بهتری نسبت به استراتژی‌های دیگر دارد. همچنین، این استراتژی می‌تواند آرایه پوشش تا قوه t = 20 را تولید کند. استراتژی EMBO-GA قابلیت‌های پویشِ[۳] سراسری و انتفاعِ[۴] محلی الگوریتم مهاجرت پرندگان را با قابلیت تنوع‌بخشیِ الگوریتم ژنتیک ترکیب کرده تا از افتادن در بهینگی محلی اجتناب کند. ویژگی‌های پویش و انتفاع، به‌ترتیب بیانگر توانایی الگوریتم در تولید راه‌حل‌های جدید از قسمت‌های ناشناخته‌ای از فضای جستجو و قابلیت افزایش کیفیت راه‌حل‌های جدیداً تولید شده با توجه به اطلاعات راه‌حل‌های قبلی اشاره دارند.

استراتژی بهینه‌سازی مبتنی بر آموزش و یادگیری[۵] (TLBO) [۱۹] یک الگوریتم فراابتکاری است که دارای دو مرحله آموزش و یادگیری است. در مرحله‌ی آموزش، شایسته‌ترین فرد جمعیت (نمونه تست با بیشترین پوشش) به‌عنوان معلم انتخاب شده و بقیه افراد جمعیت را آموزش می‌دهد و این باعث می‌شود که میزان شایستگی (پوشش) بقیه‌ی افراد جمعیت (نمونه‌های تست) نیز افزایش پیدا کند. در مرحله‌ی یادگیری، همه افراد جمعیت (بجز معلم در مرحله‌ی آموزش) دانشِ خود را به بقیه یاد می‌دهند. از ویژگی برجسته این الگوریتم می‌توان به داشتن کمترین تعداد پارامتر (فقط یک عدد) اشاره کرد. الگوریتم ازدحام توده ذرات[۶] (PSO) الگوریتم فراابتکاری دیگری است که مبتنی بر رفتار گروهی پرندگان در جستجوی غذا می‌باشد. در این الگوریتم، هر نمونه تست معادل یک پرنده در نظر گرفته می‌شود که دارای وزن (مقدار شایستگی) است. بیشتر بودن وزن یک نمونه تست (پرنده) نشان دهنده میزان نزدیکی به طعمه را نشان می‌دهد. هر پرنده دارای یک موقعیت در فضای جستجو و یک سرعت است که موجب هدایت حرکت و تغییر موقعیت آن پرنده می‌شود. ایده نهفته در این الگوریتم این است که همه پرندگان سعی می‌کنند پرنده‌ای را دنبال کنند که به طعمه نزدیک‌تر است. از استراتژی‌های مبتنی بر این الگوریتم می‌توان به PSO [۲۰]، PSTG [۲۱]، DPSO [۲۲] و BAPSO [۲۳] اشاره کرد. اگرچه این استراتژی‌ها می‌توانند آرایه پوشش تا قوه t = 12 را تولید کنند ولی دارای مشکلاتی از جمله تنظیمِ دشوارِ پارامترها و همگرایی زودرس هستند.

الگوریتم مولد تست جستجوی گرانشی[۷] (GSTG) نوعی الگوریتم فراابتکاری است که از برهم‌کنش گرانشی بین اجسام الگو می‌گیرد. در این الگوریتم، هر راه‌حل کاندید (جسم) نشان دهنده یک نمونه تست است و میزان پوشش نمونه تست، جرم جسم را نشان می‌دهد. این اجسام همدیگر را با نیروی گرانشی جذب می‌کنند و این باعث می‌شود که در کل روند اجرای الگوریتم، اشیاء سبک‌تر (نمونه‌های تست با وزن کمتر) آرام آرام به سمت اشیاء سنگین‌تر (نمونه‌های تست با وزن بیشتر) حرکت کنند. این الگوریتم تضمین می‌کند که در نهایت، یک نمونه تست با بیشترین وزن باقی بماند که در واقع نشان‌دهنده بهینه‌ترین نمونه تست است. نتایج اجرای این الگوریتم روی چندین پیکربندی مختلف، توانایی تولید آرایه پوشش تا قوه t = 10 را تایید می‌کنند.

الگوریتم کلونی زنبور مصنوعی [۲۴] یک الگوریتم فراابتکاری است که از رفتار گروهی زنبورهای عسل نشات می‌گیرد. از استراتژی‌های مبتنی بر این الگوریتم، می‌توان به PABC[۸] [۲۵]، ABCS[۹] [۲۶] و HABCSm[۱۰] [۲۷] اشاره کرد. سیستم کلونی مورچه‌ها (ACS) [۲۸] نوعی استراتژی فراابتکاریِ مبتنی

[۱] One-parameter-at-a-time

[۲] One-row-at-a-time

[۳] Exploration

[۴] Exploitation

[۵] Teaching–Learning-Based Optimization

[۶] Particle Swarm Optimization

[۷] Gravitational Search Test Generator

[۸] Pairwise Artificial Bee Colony algorithm

[۹] Artificial Bee Colony Strategy

[۱۰] Hybrid Artificial Bee Colony

بر بهینه‌سازی کلونی مورچه‌ها است که از استراتژی IOR[۱۱] که مبتنی بر خروجی سیستم تحت مطالعه و تعامل بین پارامترهای ورودی خاص است استفاده می‌کند. الگوریتم ACOF [۲۹] استراتژی فراابتکاری دیگری است که مبتنی بر ترکیب بهینه‌سازی کلونی مورچه‌ها و منطق فازی می‌باشد. از استراتژی‌های فراابتکاری دیگر، می‌توان به ترکیب جستجوی تپه‌نوردی و خفاش (HC-BAT) [۳۰]، مولد تستِ کرمِ شب‌تاب (FATG) [۳۱]، سیاه‌چاله چندگانه (MBH) [۳۲] و جستجوی فاخته (CS) [۱] اشاره کرد. اگر چه این استراتژی‌ها توانسته‌اند آرایه‌های پوشش با اندازه‌های کوچک‌تر را تولید کنند ولی هنوز کمینه‌سازی کامل انجام نشده است.

تکامل تفاضلی[۱۲] (DE) نوعی الگوریتم تکاملی است که کارایی آن به پارامترهای اندازه جمعیت، فاکتور مقیاس‌گذاری و نرخ ادغام وابسته است [۳۳]. ویژگی مهم این الگوریتم این است که مقادیر پارامترهای آن به مساله بهینه‌سازی تحت مطالعه وابسته هستند و باید هم‌زمان با پیشرفت جستجوی الگوریتم تنظیم شوند. با توجه به اهمیت این موضوع، مکانیسم‌های تطبیقی مختلفی برای تنظیم مقادیر این پارامترها در حین فرآیند جستجو ارائه شده‌اند. تطبیق مبتنی بر تاریخچه موفقیت یک مکانیسم جدیدی است که از یک حافظه سابقه که شامل تنظیمات موفقیت‌آمیز این پارامترها در اجراهای قبلی است، استفاده کرده تا مقادیر جدیدی برای این پارامترها محاسبه کند. الگوریتم تکامل تفاضلی تطبیقی مبتنی بر تاریخچه موفقیت[۱۳] (SHADE) نمونه‌ای از الگوریتم‌هایی است که از این نوع تطبیق استفاده می‌کنند [۳۴]. در این الگوریتم پارامترهای فاکتور مقیاس‌گذاری و نرخ ادغام بطور اتوماتیک تنظیم می‌شوند و معمولاً اندازه جمعیت ثابت می‌ماند. با توجه به این نکته که اندازه‌های کوچک جمعیت باعث تسریع همگرایی الگوریتم‌های تکاملی می‌شوند در [۳۵] الگوریتم LSHADE ارائه شده است که سرعت همگرایی الگوریتم SHADE را با کاهش خطی اندازه جمعیت در مراحل مختلف اجرا افزایش می‌دهد. در این مقاله، از الگوریتم LSHADE که جزو برندگان کنگره IEEE در محاسبات تکاملی است، جهت تولید آرایه پوشش کمینه استفاده می‌کنیم.

در ادامه مقاله و در بخش ۲، به پیش‌زمینه لازم از جمله تست ترکیبی t-ستونی و الگوریتم LSHADE خواهیم پرداخت. بخش ۳ به ارائه استراتژی پیشنهادی می‌پردازد. در بخش ۴ کارایی استراتژی ارائه شده از طریق مقایسه با چندین استراتژی موجود ارزیابی می‌شود. در بخش ۵ نتیجه‌گیری و نیز پیشنهاداتی برای کارهای آتی مطرح می‌شود.

۲- پیش‌زمینه

۲-۱- تست ترکیبی t-ستونی

برای تست جامع یک سیستم نرم‌افزاری، نمونه‌های تست باید شامل همه ترکیبات ممکن از پارامترهای ورودی باشند که این باعث رخ دادن مشکل انفجار ترکیبی در سیستم‌های نرم‌افزاری با تعداد زیادی پارامتر ورودی می‌شود. تست ترکیبی t-ستونی تکنیکی است که آرایه‌ای از نمونه‌های تست را تولید کرده که بجای پوشش تمام ترکیبات متفاوت از پارامترهای سیستم، فقط ترکیبات t پارامتر را پوشش دهد. فرض کنید یک سیستم فرضی دارای p پارامتر ورودی بصورت $\{x_1, \ldots, x_p\}$ است طوری که هر پارامتر x_i دارای d_i مقدار متفاوت باشد. هر نمونه تست با یک بردار p تایی بصورت $(v_1, \ldots, v_p)$ نشان داده می‌شود که در آن، v_i یکی از مقادیر پارامتر x_i است. بعنوان مثال، یک سیستم نرم‌افزاری تجارت الکترونیکی[۱۴] (ECSS) ساده را در نظر بگیرید [۳۶]. مطابق جدول ۱، این سیستم شامل ۵ پارامتر ورودی ($p = 5$) است که سه پارامتر آن دارای ۲ مقدار متفاوت و دو پارامتر آن دارای ۳ مقدار متفاوت هستند ($d_1 = d_2 = d_3 = 2$, $d_4 = d_5 = 3$). خطاهای یک سیستم معمولاً بازای تنظیم مقادیر چندین پارامتر مختلف ظاهر می‌شوند. در واقع، هر چنین ترکیبی یک نمونه تست را مشخص می‌کند. به‌عنوان مثال، در سیستم ECSS ممکن است خطایی توسط تنظیمات پارامتر تلفن هوشمند بازای مقدار iPhone و پارامتر بانک اطلاعاتی تجاری بازای مقدار Oracle ظاهر شود. برای تشخیص چنین خطایی، نمونه تست (-, iPhone, -, -, Oracle) باید حداقل یکبار در مجموعه تست ظاهر شود (علامت - برای یک پارامتر نشان دهنده این است که پارامتر موردنظر می‌تواند هر مقداری را داشته باشد). برای کشف کلیه خطاها، باید تمام تنظیمات (ترکیبات) متفاوت از پارامترهای سیستم تولید شوند. بنابراین، برای تست جامع این سیستم، مجموعه تست باید شامل ۷۲ = ۳×۳×۲×۲×۲ نمونه تست باشد که این عدد می‌تواند در سیستم‌های با تعداد زیادی پارامتر خیلی بزرگ بوده و این باعث مشکل انفجار ترکیبی بشود.

برای موفقیت تست t-ستونی، مقدار t باید بطور دقیق انتخاب شود؛ مقادیر کوچک ممکن است باعث عدم پوشش بعضی ترکیبات لازم برای تشخیص خطا توسط مجموعه تست تولید شده شود در حالیکه مقادیر بزرگ ممکن است دوباره باعث بروز مشکل انفجار ترکیبی شوند. در این نوع تست، t قوه پوشش نامیده می‌شود و مشخص‌کننده عمق پوشش می‌باشد. به مجموعه تست تولید شده توسط این نوع تست، آرایه پوشش (CA) گفته می‌شود. آرایه پوشش برای سیستم فرضی (بیان شده در بالا) بصورت $CA(N; t, p, d_1, \ldots, d_p)$ در نظر گرفته می‌شود که N تعداد نمونه‌های تست را نشان می‌دهد. ضمناً، اگر همه d_i ها برابر با مقدار ثابت d باشند آرایه پوشش بصورت $CA(N; t, p, d)$ یا $CA(N; t, d^p)$ نشان داده می‌شود.

جدول ۱- سیستم نرم‌افزاری تجارت الکترونیکی [۳۶]

بانک اطلاعاتی تجاری	مرورگر کاربر	سرور وب	تلفن هوشمند	سرور پرداخت
SQL	Chrome	iPlanet	iPhone	Master
Oracle	Explorer	Apache	Blackberry	VISA
Access	Firefox			

تعریف: آرایه پوشش $CA(N; t, p, d_1, \ldots, d_p)$ بصورت یک آرایه دوبعدی $N \times p$ تعریف می‌شود که این آرایه دارای دو ویژگی مهم است: (۱) همه مقادیر ممکن متغیرهای x_i $(1 \le i \le p)$، باید حداقل یک‌بار در ستون i-ام این آرایه دوبعدی دیده شود. (۲) همه ترکیبات t پارامتر دلخواه باید توسط یک زیرآرایه دوبعدی $N \times t$ پوشش داده شوند [۱۷].

مطابق با جدول ۲، در سیستم نرم‌افزاری تجارت الکترونیکی فقط ۹ نمونه تست برای ساختن آرایه پوششی ۲-ستونی نیاز است. در اینجا، آرایه پوشش $CA(9; 2, 5, 2, 2, 2, 3, 3)$ یا بطور خلاصه‌تر $CA(9; 2, 2^3 3^2)$ وجود دارد. طبق تعریف، هر کدام از مقادیر پارامترها باید حداقل یک‌بار در ستون مربوطه دیده شوند. بعنوان مثال، پارامتر مرورگر وب دارای مقادیر Chrome، Explorer و Firefox است که هر کدام حداقل یک‌بار در ستون سوم جدول ۲ ظاهر شده‌اند.

[۱۱] Input-output based relations

[۱۲] Differential Evolution

[۱۳] Success-History based Adaptive Differential Evolution Algorithm

[۱۴] E-commerce software system

همچنین، همه ترکیبات مختلف از ۲ پارامتر دلخواه باید توسط یک زیرآرایه دوبعدی ۲×۹ پوشش داده شوند. بعنوان مثال، پارامترهای سرور پرداخت و مرورگر وب دارای شش ترکیب مختلف (Master, -, -, Chrome, -)، (Master, - -, Explorer, -)، (Master, -, -, Firefox, -)، (VISA, -, -, Chrome, -)، (VISA, -, -, Explorer, -) و (VISA, -, -, Firefox, -) هستند که هرکدام حداقل یک‌بار توسط ستون‌های ششم و سوم جدول ۲ پوشش داده می‌شوند.

۲-۲- الگوریتم LSHADE

الگوریتم تکامل تفاضلی (DE) دارای پارامترهای مهم اندازه جمعیت (N)، فاکتور مقیاس‌گذاری (F) و نرخ ادغام (CR) است که مقادیر آنها وابسته به مساله بهینه‌سازی تحت مطالعه هستند و باید هم‌زمان با روند جستجو تنظیم شوند [۳۳]. با توجه به اهمیت این موضوع، الگوریتم‌های مختلفی برای تنظیم این پارامترها در حین فرآیند جستجو ارائه شده‌اند. الگوریتم تکامل تفاضلی تطبیقی مبتنی بر تاریخچه موفقیت (SHADE) نمونه‌ای از این الگوریتم‌ها است که از یک حافظه سابقه که شامل تنظیمات موفقیت‌آمیز این پارامترها در اجراهای قبلی است، استفاده کرده تا مقادیر جدیدی برای این پارامترها محاسبه کند [۳۴]. ضمناً، بر خلاف الگوریتم تکامل تفاضلی که در آن، مقادیر پارامترهای F و CR برای تمام افراد جمعیت یکسان هستند، در الگوریتم SHADE، این پارامترها می‌توانند متفاوت باشند. الگوریتم LSHADE نسخه بهبودیافته‌ای از الگوریتم SHADE است که علاوه بر تنظیم اتوماتیک پارامترهای فاکتور مقیاس-گذاری و نرخ ادغام، اندازه جمعیت را مرحله به مرحله و بصورت خطی کاهش داده تا سرعت همگرایی افزایش پیدا کند [۳۵].

جدول ۲- آرایه پوشش CA (9; 2, $2^3 3^2$) [۳۶]

شماره	بانک اطلاعاتی تجاری	مرورگر کاربر	سرور وب	تلفن هوشمند	سرور پرداخت
۱	Oracle	Firefox	Apache	Blackberry	Master
۲	Oracle	Explorer	iPlanet	iPhone	VISA
۳	SQL	Chrome	Apache	iPhone	VISA
۴	Access	Chrome	iPlanet	Blackberry	Master
۵	SQL	Explorer	iPlanet	Blackberry	Master
۶	Access	Firefox	iPlanet	iPhone	VISA
۷	Access	Explorer	Apache	iPhone	Master
۸	Oracle	Chrome	iPlanet	Blackberry	VISA
۹	SQL	Firefox	iPlanet	Blackberry	VISA

مشابه با همه الگوریتم‌های تکاملی، الگوریتم LSHADE نیز با یک جمعیت X از بردارها بصورت $x_i = (x_1, ..., x_D)$, $i = 1, 2, ..., N$ سروکار دارد که D به دامنه مساله هدف و N به اندازه جمعیت اشاره می‌کند. در ابتدا، همه بردارهای جمعیت بصورت تصادفی انتخاب می‌شوند. شکل ۱ شبه‌کد الگوریتم LSHADE را نشان می‌دهد. مطابق با این الگوریتم، شش مرحله زیر تا موقعی که شرایط پایان الگوریتم برآورده شوند تکرار می‌شوند. یکی از شرایط پایان این است که الگوریتم به تعداد مشخص شده توسط کاربر تکرار شود. برای خوانایی بیشتر، مراحل به مراه شماره خطوط متناظر با الگوریتم نیز ذکر می‌شوند.

مرحله ۱- تعیین مقادیر جدید برای پارامترهای F و CR با توجه به حافظه سابقه (خطوط ۱۳-۴): این الگوریتم یک آرایه با نام M و بطول H (مشخص شده توسط کاربر) را که هر خانه آن شامل مقادیر F و CR هست را به عنوان حافظه سابقه نگهداری می‌کند. در شروع اجرای الگوریتم، همه این مقادیر ($M[i].CR$ و $M[i].F$ بازای $i = 1,...,H$) به ۰٫۵ مقداردهی اولیه می‌شوند. در هر تکرار الگوریتم، مقادیر پارامترهای CR_i و F_i برای بردار x_i (بازای $i = 1,2,...,N$) بترتیب از رابطه‌های CR_i= *findrnd* ($M[k].CR$, 0.1) و F_i= *findrnd* ($M[k].F$, 0.1) محاسبه می‌شوند که در آن k یک عدد تصادفی صحیح مابین اعداد ۱ تا H است و تابع *findrnd* (p , 0.1) یک عدد تصادفی از توزیع نرمال با میانگین p و انحراف معیار 0.1 تولید می‌کند. اگر مقدار تولید شده برای CR_i خارج از محدوده صفر و یک باشد با نزدیک‌ترین مقدار (صفر یا یک) جایگزین می‌شود. همچنین، اگر مقدار تولید شده برای F_i بزرگ‌تر از یک باشد به مقدار یک تغییر داده می‌شود ولی اگر مقدار آن کوچک‌تر از صفر باشد آنقدر از این رابطه استفاده کرده تا بالاخره یک عدد مثبت تولید شود.

مرحله ۲- اِعمال عملیات جهش و ترکیب روی بردارهای جمعیت فعلی (خطوط ۱۷-۱۴): بعد از تولید مقادیر پارامترهای CR و F برای بردارهای X، از رابطه (۱) استفاده کرده تا بردارهای جهش Y ایجاد شوند. در این رابطه، x_i به بردار i-ام X اشاره دارد و بردار x_{pbest} بطور تصادفی از بین $N \times p$ شایسته‌ترین بردار انتخاب می‌شود (p یک عدد تصادفی بین صفر و یک است). همچنین، بردارهای غیرتکراری x_a و x_b (و متفاوت با x_i) نیز بصورت تصادفی از بین بردارهای جمعیت فعلی انتخاب می‌شوند.

$$y_i = x_i + F_i.(x_{pbest} - x_i) + F_i.(x_a - x_b) \qquad (۱)$$

اگر مقدار بُعد j-ام بردار تولید شده y_i خارج از محدوده $[x_{jmin}, x_{jmax}]$ باشد آنگاه مقدار آن با استفاده از رابطه (۲) تصحیح می‌شود [۳۷].

$$y_i[j] = \begin{cases} (x_{jmin} + x_i[j])/2 & if\ y_i[j] < x_{jmin} \\ (x_{jmax} + x_i[j])/2 & if\ y_i[j] > x_{jmax} \end{cases} \quad (j = 1, 2, ..., D) \qquad (۲)$$

بعد از تولید بردارهای جهش y_i (بازای $i = 1,2,...,N$)، این بردارها با بردارهای x_i از طریق رابطه (۳) ترکیب شده تا بردارهای z_i حاصل شود. در این رابطه، *rnd* و *jrnd* اعداد تصادفی هستند که به ترتیب در محدوده [0 , 1) و [1 , D] قرار دارند.

$$z_i[j] = \begin{cases} y_i[j] & if\ rnd \le CR_i\ or\ j = jrnd \\ x_i[j] & otherwise \end{cases} \quad (j = 1, 2, ..., D) \qquad (۳)$$

مرحله ۳- ایجاد جمعیت جدید (خطوط ۲۷-۱۹): بعد از تولید بردارهای Z، هر بردار z_i (بازای $i = 1,2,...,N$) با بردار x_i مقایسه شده و شایسته‌ترین آنها به جمعیت جدید اضافه می‌شود.

مرحله ۴- بِروزرسانی جمعیت بایگانی (خطوط ۲۲ و ۲۸): بعد از ایجاد جمعیت جدید، بردارهای والد X که کم‌شایسته‌تر از بردارهای Z هستند (طبق رابطه (۳))، بعنوان جمعیت بایگانی نگهداری می‌شوند. ضمنا، بردار x_b در رابطه (۱) از بین بردارهای موجود در جمعیت‌های فعلی X و بایگانی (A) بصورت تصادفی انتخاب می‌شود. نکته قابل ذکر این است که اگر اندازه جمیعت بایگانی از مقدار پیش‌فرض $|A|$ بیشتر شود تعدادی از بردارهای آن بصورت تصادفی انتخاب شده و حذف می‌شوند.

مرحله ۵- بِروزرسانی حافظه سابقه (خطوط ۲۳ و ۲۹): در هر تکرار الگوریتم، مقادیر CR_i و F_i مربوط به بردار x_i (بازای $i = 1,2,...,N$) که باعث شده‌اند تا بردار تولیدی z_i طبق رابطه (۳) از بردار x_i شایسته‌تر بشود در دو مجموعه S_{CR} و S_F جمع‌آوری شده و در پایان تکرار الگوریتم، حافظه سابقه M با توجه به این دو مجموعه بروزرسانی می‌شوند [۳۷].

مرحله ۶- کاهش اندازه جمعیت (خطوط ۳۵-۳۰): اندازه جمعیت (N) در هر تکرار الگوریتم طوری کاهش می‌یابد که مقدار آن در تکرار اول برابر با N_{init} و بعد از آخرین تکرار برابر N_{min} بشود. اندازه جمعیت برای تکرار بعدی با استفاده از رابطه (۴) محاسبه می‌شود که در آن، NFE به تعداد ارزیابی‌های تابع برازندگی تا اجرای فعلی الگوریتم و NFE_{MAX} به حداکثر تعداد ارزیابی‌های تابع برازندگی اشاره می‌کنند.

$$N_{new} = round[(\frac{N_{\min} - N_{init}}{NFE_{MAX}}) \times NFE + N_{init}] \quad (۴)$$

اگر مقدار بدست آمده برای N_{new} کمتر از مقدار فعلی N باشد تعداد $N\text{-}N_{new}$ از کم‌شایسته‌ترین‌های جمعیت فعلی حذف می‌شوند. ضمناً، اندازه جمعیت بایگانی (A) نیز با توجه به مقدار جدید N_{new} بروزرسانی می‌شود.

Algorithm 1 The LSHADE algorithm

Input: N, D, H, N_{min}, NFE_{MAX}, $Archive_{size}$, fit;
Output: A vector $x = (x_1, ..., x_D)$ with the highest fitness;
$N_{init} = N$; Archive $A = \emptyset$;
Initialize the population of X randomly;
CRF M = New CRF[H];
for i =1 **to** H **do**
    $M[i].CR = 0.5$; $M[i].F = 0.5$;
end
CRF $Mnew$ = New CRF[N];
while The termination criteria are not met **do**
    $S_{CR} = \emptyset$; $S_F = \emptyset$;
    for $i = 1$ **to** N **do**
        k = Select from [1,H] randomly.
        $CR = findrnd\ (M[k].CR, 0.1)$;
        $F = findrnd\ (M[k].F, 0.1)$;
        Generate $Y[i]$ from $X[i]$ using Formula (1);
        Update $Y[i]$ using Formula (2);
        Generate $Z[i]$ from $X[i]$ and $Y[i]$ using Formula (3);
        $Mnew[i].CR = CR$; $Mnew[i].F = F$;
    end
    for $i = 1$ **to** N **do**
        if $fit\ (X[i]) \le fit\ (Z[i])$ **then**
            $Xnew\ [i] = Z[i]$;
            $A = A \cup Z[i]$;
            $S_{CR} = S_{CR} \cup Mnew[i].CR$; $S_F = S_F \cup Mnew[i].F$;
        else
            $Xnew\ [i] = X[i]$;
        end
    end
    RemoveExtra (A);
    Update M using S_{CR} and S_F ;
    Compute N_{new} using Formula (4);
    Sort vectors of $Xnew$ based on their fitness;
    if $N_{new} < N$ **then**
        Remove lowest $N\text{-}N_{new}$ vectors from $Xnew$;
        Update the size of A due to N_{new};
    end
    $X = Xnew$; $N = N_{new}$;
end
return the vector of X with the highest fitness;

شکل ۱- شبه‌کد الگوریتم LSHADE

۳- استراتژی پیشنهادی

در همه استراتژی‌های ارائه شده مبتنی بر الگوریتم‌های فراابتکاری از جمله GS، TLBO، WOA، BAPSO و GSTG، این الگوریتم‌ها بازای تولید هر نمونه تست با بیشترین پوشش، یکبار فراخوانی می‌شوند. با توجه به این‌که اندازه آرایه پوشش کمینه از پیش نامشخص است بنابراین این الگوریتم‌ها بطور تکراری فراخوانی شده تا در نهایت، تمام ترکیبات ممکن از t پارامتر پوشش داده شوند. لازم به ذکر است که این الگوریتم‌ها دارای دو تفاوت مهم با نسخه اصلی‌شان هستند: ۱) هر نمونه تست بصورت یک راه‌حل (کروموزوم یا فرد) در نظر گرفته می‌شود که در ابتدا بصورت تصادفی تولید می‌شود. بنابراین، در اینجا با یک جمعیتی از نمونه‌های تست سروکار داریم. ۲) یک تابعِ محاسبهِ وزن بعنوان تابع برازندگی استفاده می‌شود که یک نمونه تست را گرفته و وزن آن (تعداد ترکیبات پوشش داده شده) را محاسبه می‌کند.

مشابه با استراتژی‌های مذکور، استراتژی LSHADE نیز الگوریتم اصلی را با اِعمال دو تغییر ذکر شده در بالا بکار می‌برد. این استراتژی ابتدا یک مجموعه تست TS (با مقدار اولیه تهی) را در نظر گرفته و الگوریتم LSHADE را جهت تولید یک نمونه تست با بیشترین پوشش فراخوانی می‌کند. این الگوریتم نیز جمعیتی از نمونه‌های تست تصادفی را تولید کرده و بعد از اِعمال عملگرهای مربوطه (مطابق با بخش ۲-۲)، نمونه‌های تست با بیشترین پوشش (برازنده) را جهت ایجاد جمعیت نسل بعدی انتخاب می‌کند و این روند را به تعداد مشخص شده توسط کاربر تکرار می‌کند. در پایان الگوریتم، نمونه تست با بیشترین برازندگی انتخاب شده و توسط استراتژی LSHADE به مجموعه تست TS اضافه می‌شود. با توجه به نامشخص بودن اندازه مجموعه تست کمینه از قبل، الگوریتم LSHADE، تا پوششِ تمام ترکیبات ممکن از t پارامتر، بطور تکراری فراخوانی می‌شود. ممکن است استراتژی LSHADE نتواند برای بعضی پیکربندی‌های پیچیده، در زمان‌های کمتر از ۲۴ ساعت به جواب برسد و یا ممکن است بدلیل محدودیت‌های حافظه‌ای با خطا مواجه شود. بنابراین شرطِ تولید مجموعه تست کمینه در کمتر از ۲۴ ساعت را نیز به شرایط ادامهِ اجرای این استراتژی اضافه می‌کنیم. بدیهی است که اگر استراتژی LSHADE بتواند آرایه پوشش کمینه‌ای در کمتر از ۲۴ ساعت تولید کند مجموعه حاصل شده را بعنوان جواب به کاربر اعلام می‌کند در غیراینصورت به کاربر اعلام می‌کند که نتوانسته است مجموعه پوشش کمینه را در کمتر از ۲۴ ساعت تولید کند. در واقع، هیچ موقع حالتی پیش نمی‌آید که قسمتی از مجموعه تست به کاربر اعلام شود.

در هر لحظه، برای اینکه مشخص شود که کدام ترکیبات توسط نمونه‌های تست تولید شده در مجموعه تست پوشش داده می‌شوند و کدام ترکیبات هنوز بدون پوشش مانده‌اند از یک ماتریس پوشش (CM) استفاده می‌کنیم. ضمنا، برای محاسبه میزان برازندگی هر راه‌حل (نمونه تست) از تابع *calcweight* استفاده می‌کنیم که یک نمونه تست را گرفته و وزن (تعداد ترکیبات پوشش داده شده) آن را محاسبه می‌کند. موقعی که نمونه تست تولید شده در پایان اجرای الگوریتم LSHADE به مجموعه TS اضافه می‌شود ابتدا مقدار وزن آن از مقدار ترکیبات پوشش داده نشده $RemCov$ کم می‌شود و همچنین، ماتریس پوشش CM بروزرسانی می‌شود. شکل ۲ فلوچارت استراتژی ارائه شده را نشان می‌دهد. مطابق با این فلوچارت، بعد از مشخص کردن مقادیر t، p و d، ابتدا ماتریس پوشش CM تولید می‌شود. سپس، مقدار ترکیبات پوشش داده نشده $RemCov$ مشخص شده که در ابتدا برابر $\binom{p}{t} \times d^t$ است. در ادامه، الگوریتم LSHADE مطابق با تنظیمات موجود در [۳۵] مقداردهی اولیه شده و جهت تولید یک نمونه تست اجرا می‌شود. در ادامه و در بخش ۱-۳ نحوه ایجاد ماتریس پوشش را توضیح می‌دهیم. در بخش ۲-۳ نیز به نحوه محاسبه وزن یک نمونه تست و نحوه بروزرسانی ماتریس پوشش خواهیم پرداخت.

۳-۱- ایجاد ماتریس پوشش

ماتریس پوشش، ساختار داده‌ای است که وضعیت پوشش تمام ترکیبات ممکن از t پارامتر را در هر لحظه از اجرای الگوریتم نگهداری می‌کند. اگر سیستمی p پارامتر ورودی داشته باشد که هر کدام دارای d مقدار متفاوت باشد تعداد کل ترکیبات برابر $\binom{p}{t} \times d^t$ خواهد بود. واضح است که ماتریس پوشش (CM) باید دارای $\binom{p}{t}$ سطر باشد. ضمناً، هر سطر باید شامل t ستون برای نگهداری اندیس‌های ترکیبات مربوط به این سطر و d^t ستون برای نشان دادن مقادیر ممکن دهدهی معادل ترکیبات t-تایی باشد و یک ستون (آخرین) نشان

دهنده تعداد ترکیباتی از این سطر هست که هنوز توسط هیچ نمونه تستی پوشش داده نشده است. در ابتدای اجرای استراتژی، مقادیر تمام d^t ستون مربوط به همه سطرها به عدد صفر و همچنین مقدار آخرین ستون در همه سطرها به عدد d^t مقداردهی اولیه می‌شود. به‌عنوان مثال، شکل ۳ ساختار ماتریس پوشش را برای آرایه پوشش CA (N; 2, 3^4) نشان می‌دهد (t=2, p=4, d=3). این ماتریس دارای تعداد $\binom{p}{t} = \binom{4}{2} = 6$ سطر و تعداد $t+d^t+1 = 2+3^2+1=12$ ستون دارد. ستون‌های صفر و یک شامل اندیس‌های این ۶ ترکیب متفاوت است (12, 13, 14, 23, 24, 34)، نه ستون بعدی شامل مقادیر دهدهی ممکن از مقادیر متناظر در این دو پارامتر می‌باشد (00, 01, 02, 10, 11, 12, 20, 21, 22) و ستون آخر تعداد عناصر پوشش داده نشده را نشان می‌دهد. بصورت دقیق‌تر، شماره ستون مربوط به اندیس‌های یک ترکیب دلخواه که برابر ij باشد در حالت کلی از طریق فرمول $c = t+ i\times d+j$ محاسبه می‌شود. بعنوان مثال، در شکل ۳، شماره ستون مربوط به اندیس‌های ترکیب ۲۴ که برابر ۱۰ است بصورت $c = 2+(1\times3+0)=5$ محاسبه می‌شود. علاوه بر این، مطابق با این شکل، مقدار *RemCov* (تعداد عناصر پوشش داده نشده) در ابتدا برابر با $\binom{p}{t} \times d^t = \binom{4}{2} \times 3^2 = 54$ خواهد بود.

شکل ۲- فلوچارت استراتژی ارائه شده

۳-۲- محاسبه وزن یک نمونه تست (تابع *calcweight*) و بروزرسانی ماتریس پوشش

وزن یک نمونه تست به تعداد حالات جدیدی گفته می‌شود که توسط این نمونه تست پوشش داده می‌شود. برای این منظور، با توجه به مقادیر موجود در نمونه تست داده شده و همچنین مقادیر t ستون اول هر سطر، ابتدا معادل دهدهی آن محاسبه شده و بعد از اضافه کردن عدد t به این مقدار، عدد (سلول) متناظر با این سطر و ستون بدست می‌آید. یک بودن مقدار این سلول به این معنی است که این ترکیب توسط نمونه‌های تست قبلی (که به مجموعه تست اضافه شده‌اند) پوشش داده می‌شود و نیازی به تغییر مقدار وزن این نمونه تست نیست. ولی اگر مقدار آن صفر باشد یک واحد به وزن این نمونه تست اضافه شده و مقدار این سلول به عدد یک تغییر می‌کند ضمناً، از عدد موجود در ستون آخر سطر موردنظر نیز یک واحد کسر می‌شود. وقتی که همه سطرها بررسی شدند وزن بدست آمده را از مقدار *RemCov* کم می‌کنیم. شکل ۴ نحوه محاسبه وزن نمونه تست $tc = (1,2,0,1)$ را با توجه به ماتریس پوشش شکل ۳ نشان می‌دهد. همانطور که در این شکل دیده می‌شود شماره ستون مربوط به اندیس‌های تمام ترکیبات (سطرها) بصورت ij از طریق فرمول $c = 2+ i\times3+j$ محاسبه شده و سلول‌های موردنظر که در اینجا همه صفر بودند به یک تغییر می‌کنند ضمناً، مقادیر ستون‌های آخر که ۹ بودند به ۸ کاهش یافته و در نهایت مقدار *RemCov* به عدد ۴۸ کاهش پیدا می‌کند.

۱	۲	۰	۰	۰	۰	۰	۰	۰	۰	۰	۹
۱	۳	۰	۰	۰	۰	۰	۰	۰	۰	۰	۹
۱	۴	۰	۰	۰	۰	۰	۰	۰	۰	۰	۹
۲	۳	۰	۰	۰	۰	۰	۰	۰	۰	۰	۹
۲	۴	۰	۰	۰	۰	۰	۰	۰	۰	۰	۹
۳	۴	۰	۰	۰	۰	۰	۰	۰	۰	۰	۹
۰	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱

***RemCov* = ۹ + ۹ + ... + ۹ = ۵۴**

شکل ۳- ساختار ماتریس پوشش برای پیکربندی CA (N; 2, 3^4)

۴- ارزیابی

برای ارزیابی استراتژی LSHADE، آن را در نرم‌افزار مطلب نسخه ۲۰۱۷ پیاده‌سازی کرده و نتایج حاصل شده را با استراتژی‌های مبتنی بر ریاضی مانند TConfig، حریصانه مانند IPOG، Jenny و PICT و فراابتکاری مانند HC-BAT، PSTG، TLBO، GS، WOA، BAPSO و GSTG مقایسه می‌کنیم. برای محاسبه مقادیر مناسب برای پارامترهای استراتژی LSHADE، از روش آزمون و خطا استفاده کرده‌ایم، ضمناً مقادیر پارامترهای موردنیاز بقیه استراتژی‌ها از مقالات مربوطه اقتباس شده‌اند. جدول ۳، مقادیر مناسب برای این پارامترها را نشان می‌دهد.

جدول ۳- پارامترها و مقادیر مناسب برای استراتژی‌های مبتنی بر الگوریتم‌های فراابتکاری موجود (تعداد تکرار همه الگوریتم‌ها برابر ۱۰۰ در نظر گرفته شده است)

الگوریتم	پارامترها و مقادیر مناسب
LSHADE	$H = 5$, $N_{min} = 4$
GS	$N = 120$, $MutRate = 0.4$, $CossRate = 0.4$
TLBO	$N = 60$
WOA	$N = 180$
BAPSO	$N = 100$, $Loudness = 0.9$, $PulseemissionRate = 0.9$, $Minfrequency = 0$, $Maxfrequency = 1$, $Tolerance = 0.025$, $WeightValue = 0.4$, $CycWalkvalue = 1.49$
GSTG	$N = 100$, $ElitistCheck = 1$, $Rpower = 1$, $Rnorm = 2$

برای تولید نتایج توسط استراتژی‌های در نظر گرفته، همه استراتژی‌ها را به تعداد ۱۰۰ بار روی هر کدام از پیکربندی‌های در نظر گرفته اجرا می‌کنیم. محیط اجرایی این استراتژی‌ها عبارت است از: CPU Intel® Core™ i5 و RAM 6GB. در جدول نتایج، اگر نتیجه اجرای استراتژی برای یک پیکربندی خاصی

در دسترس نباشد از عبارت NA استفاده می‌کنیم. علاوه بر این، عبارت "day>" موقعی استفاده می‌شود که استراتژی موردنظر نتوانسته باشد یک آرایه پوشش بهینه در زمان کم‌تر از ۲۴ ساعت تولید کند.

جدول ۴ اندازه آرایه پوشش تولید شده را برای پیکربندی CA (N; t, p, 3) بازای $2 \le t \le 6$ و $t+1 \le p \le 12$ نشان می‌دهد. مطابق با این جدول، تعداد ۴۰ پیکربندی مختلف داریم که کم‌ترین اندازه آرایه پوشش بدست آمده توسط همه استراتژی‌ها را مبنای مقایسه در نظر می‌گیریم. استراتژی TConfig که مبتنی بر ریاضی است توانسته است فقط در یک پیکربندی (t=6) و (p=7) کمترین مقدار را داشته باشد. در بین استراتژی‌های حریصانه، IPOG در ۶ مورد، Jenny در یک مورد و استراتژی PICT در هیچ موردی به کمترین اندازه آرایه پوشش دست یافته‌اند. در نتیجه، استراتژی IPOG در میان استراتژی‌های حریصانه دارای بهترین کارایی و استراتژی PICT دارای بدترین کارایی است. در بین استراتژی-های فراابتکاری، استراتژی LSHADE با ۳۱ مورد دارای بهترین کارایی و استراتژی PSTG با ۶ مورد دارای بدترین کارایی است. استراتژی‌های GS، TLBO و HC-BAT نیز بترتیب با ۱۸، ۱۵ و ۱۰ مورد کارایی متوسطی دارند.

در جدول ۵، اندازه آرایه پوشش تولید شده برای پیکربندی CA (N; t, 7, d) بازای $3 \le d \le 10$ و $2 \le t \le 4$ نشان داده می‌شود. از بین تعداد ۲۴ پیکربندی موجود، استراتژی TConfig فقط در یک مورد کم‌ترین مقدار را تولید می‌کند. استراتژی IPOG با ۳ مورد بهترین کارایی را در بین استراتژی‌های حریصانه دارد. همچنین، استراتژی LSHADE با ۱۹ مورد دارای بهترین کارایی و استراتژی PSTG با یک مورد دارای بدترین کارایی در بین استراتژی‌های فراابتکاری است. بقیه استراتژی های فراابتکاری نیز دارای کارایی متوسطی هستند.

اندازه آرایه پوشش تولید شده برای پیکربندی CA (N; 4, 10, d) بازای $2 \le d \le 10$ در جدول ۶ نشان داده می‌شود. در این جدول که ۹ پیکربندی مختلف وجود دارند استراتژی LSHADE با ۹ مورد دارای بهترین کارایی است در واقع در تمام پیکربندی‌های تعریف شده در این جدول، این استراتژی بهترین نتیجه را تولید کرده است. استراتژی PSTG نیز فقط در یک مورد به جواب کمینه رسیده است و بقیه استراتژی‌ها اصلا در هیچ موردی به جواب کمینه نرسیده‌اند.

شکل ۴: محاسبه وزن نمونه تست tc = (1,2,0,1) با توجه به ماتریس پوشش شکل ۳ و بِروزرسانی آن

جدول ۴- اندازه آرایه پوشش تولید شده برای CA (N; t, p, 3) بازای $2 \le t \le 6$ و $t+1 \le p \le 12$

t	p	LSHADE N.Best	LSHADE N.Avg	GS N.Best	TLBO N.Best	PSTG N.Best	HC-BAT N.Best	Jenny N.Best	TConfig N.Best	PICT N.Best	IPOG N.Best
۲	۳	۹	۹/۹۹	۹	۹	۹	۹	۹	۱۰	۱۰	۹
	۴	۹	۱۰/۴۰	۹	۹	۹	۹	۱۳	۱۰	۱۳	۹
	۵	۱۱	۱۲/۹۸	۱۱	۱۱	۱۲	۱۲	۱۴	۱۴	۱۳	۱۵
	۶	۱۴	۱۴/۸۰	۱۳	۱۳	۱۳	۱۳	۱۵	۱۵	۱۴	۱۵
	۷	۱۴	۱۵/۳۷	۱۴	۱۴	۱۵	۱۴	۱۶	۱۵	۱۶	۱۵
	۸	۱۵	۱۵/۸۲	۱۵	۱۵	۱۵	۱۵	۱۷	۱۷	۱۶	۱۵
	۹	۱۵	۱۶/۷۵	۱۵	۱۵	۱۷	۱۶	۱۸	۱۷	۱۷	۱۵
	۱۰	۱۶	۱۷/۵۶	۱۶	۱۶	۱۷	۱۷	۱۹	۱۷	۱۸	۱۵
	۱۱	۱۶	۱۷/۹۵	۱۶	۱۶	۱۷	۱۷	۱۷	۲۰	۱۸	۱۷
	۱۲	۱۶	۱۸/۸۲	۱۶	۱۷	۱۸	۱۷	۱۹	۲۰	۱۹	۲۱
۳	۴	۲۷	۳۱/۱۲	۲۷	۲۷	۲۷	۲۸	۳۴	۳۲	۳۴	۳۲
	۵	۳۸	۴۰/۴۳	۳۸	۳۸	۳۹	۳۸	۴۰	۴۰	۴۳	۴۱
	۶	۴۲	۴۵/۴۷	۴۳	۴۳	۴۵	۴۳	۵۱	۴۸	۴۸	۴۶
	۷	۴۹	۵۰/۸۱	۴۹	۴۹	۵۰	۴۹	۵۱	۵۵	۵۱	۵۵
	۸	۵۲	۵۵/۰۴	۵۴	۵۳	۵۴	۵۳	۵۸	۵۸	۵۹	۵۶
	۹	۵۷	۵۸/۵۸	۵۸	۵۷	۵۸	۵۷	۶۲	۶۴	۶۳	۶۳
	۱۰	۵۹	۶۱/۸۴	۶۱	۶۲	۶۲	۶۲	۶۵	۶۸	۶۵	۶۶
	۱۱	۶۲	۶۴/۸۲	۶۳	۶۴	۶۴	۶۴	۶۵	۷۲	۷۰	۷۰

	۱۲	۶۶	۶۷/۳۷	۶۷	۶۶	۶۷	۶۶	۶۸	۷۷	۷۲	۷۳
۴	**۵**	۹۳	۹۹/۶۰	۸۸	۹۲	۹۶	۹۳	۱۰۹	۹۷	۱۰۰	۹۷
	۶	۱۳۰	۱۳۵/۱۰	۱۲۹	۱۳۱	۱۳۳	۱۳۲	۱۴۰	۱۴۰	۱۴۲	۱۴۱
	۷	۱۵۳	۱۵۷/۳۰	۱۵۲	۱۵۴	۱۵۵	۱۱۴	۱۶۹	۱۶۹	۱۶۸	۱۶۷
	۸	۱۷۱	۱۷۶/۸۱	۱۷۱	۱۷۵	۱۷۵	۱۷۵	۱۸۷	۱۸۷	۱۸۹	۱۹۲
	۹	۱۸۹	۱۹۴/۴۰	۱۸۷	۱۹۳	۱۹۵	۱۹۵	۲۰۶	۲۰۶	۲۱۱	۲۱۰
	۱۰	۲۰۶	۲۱۰/۵۹	۲۰۶	۲۰۹	۲۱۰	۲۱۰	۲۲۱	۲۲۱	۲۳۱	۲۳۳
	۱۱	۲۲۰	۲۲۵/۶۶	۲۲۳	۲۲۶	۲۲۲	۲۲۲	۲۳۶	۲۳۶	۲۴۹	۲۵۱
	۱۲	۲۳۵	۲۴۱/۵۲	۲۳۶	۲۴۱	۲۴۴	۲۴۲	۲۵۲	۲۵۲	۲۶۹	۲۷۲
۵	**۶**	۳۰۶	۳۱۶/۳	۳۰۵	۳۰۴	۳۱۲	۳۰۷	۳۴۸	۳۰۵	۳۱۰	۳۰۵
	۷	۴۳۷	۴۴۰/۷۰	۴۳۶	۴۴۱	۴۴۱	۴۴۰	۴۵۸	۴۷۷	۴۵۲	۴۶۶
	۸	۵۱۵	۵۲۱/۱۰	۵۲۰	۵۱۵	۵۱۵	۵۲۳	۵۴۸	۵۸۳	۵۵۵	۵۷۵
	۹	۵۹۰	۵۹۴/۱۰	۵۹۹	۵۹۴	۵۹۸	۵۹۴	۶۳۳	۶۸۴	۶۳۷	۶۶۷
	۱۰	۶۶۵	۶۶۶/۶۰	۶۷۷	۶۷۲	۶۶۷	۶۷۴	۷۱۴	۷۷۳	۷۳۵	۷۶۱
	۱۱	۷۳۴	۷۳۶/۳۴	۷۴۸	۷۴۱	۷۴۷	۷۴۶	۷۹۱	۸۵۸	۸۲۲	۸۵۱
	۱۲	۸۰۲	۸۰۴/۱۳	۸۱۲	۸۰۶	۸۰۹	۸۱۱	۸۵۰	۹۳۸	۹۰۰	۹۲۹
۶	**۷**	۹۵۴	۹۷۵/۶۰	۹۷۳	۹۶۸	۹۷۷	۹۷۲	۱۰۸۶	۹۲۱	۱۰۱۵	۹۲۱
	۸	۱۳۹۵	۱۴۰۵/۳۰	۱۴۰۵	۱۳۹۷	۱۴۰۲	۱۴۰۱	۱۴۶۶	۱۵۱۵	۱۴۵۵	۱۴۹۳
	۹	۱۶۸۱	۱۶۹۵/۴۰	۱۶۸۸	۱۶۹۲	۱۶۸۴	۱۶۹۰	۱۸۴۰	۱۹۳۱	۱۸۱۸	۱۸۸۹
	۱۰	۱۹۶۷	۱۹۷۹/۲۰	۱۹۹۰	۱۹۸۰	۱۹۸۰	۱۹۸۶	۲۱۶۰	>day	۲۱۶۵	۲۲۶۲
	۱۱	۲۲۵۳	۲۲۶۳/۶۱	۲۲۶۹	۲۲۶۹	۲۲۵۵	۲۲۷۰	۲۴۵۹	>day	۲۴۹۶	۲۶۰۷
	۱۲	۲۵۲۲	۲۵۳۶/۴۲	۲۵۷۲	۲۵۴۱	۲۵۲۸	۲۵۵۶	۲۷۵۷	>day	۲۸۱۵	۳۶۴۹

جدول ۵- اندازه آرایه پوشش تولید شده برای CA (*N*; *t*,7, *d*) بازای $2 \le t \le 4$ و $3 \le d \le 10$

t	*d*	LSHADE N.Best	LSHADE N.Avg	GS N.Best	TLBO N.Best	PSTG N.Best	HC-BAT N.Best	Jenny N.Best	TConfig N.Best	PICT N.Best	IPOG N.Best
۲	**۳**	۱۴	۱۵/۱۳	۱۴	۱۵	۱۵	۱۵	۱۶	۱۵	۱۶	۱۵
	۴	۲۴	۲۵/۳۱	۲۴	۲۴	۲۶	۲۵	۲۷	۲۸	۲۸	۲۹
	۵	۳۶	۳۷/۹۷	۳۶	۳۷	۳۷	۳۶	۴۰	۴۰	۳۷	۴۵
	۶	۵۱	۵۲/۱۱	۵۲	۵۳	۵۵	۵۳	۵۶	۵۷	۵۵	۵۵
	۷	۶۷	۶۸/۳۱	۶۸	۷۱	۶۸	۷۰	۷۴	۷۶	۷۴	۴۹
	۸	۸۸	۸۹/۴۱	۸۹	۹۲	۹۰	۹۴	۹۷	۱۰۱	۹۳	۹۵
	۹	۱۱۰	۱۱۱/۳۳	۱۱۱	۱۱۷	۱۱۰	۱۱۹	۱۲۱	۱۲۴	۱۱۴	۱۰۷
	۱۰	۱۳۴	۱۳۵/۶۱	۱۳۷	۱۳۹	۱۳۸	۱۴۰	۱۴۸	۱۵۲	۱۵۷	۱۳۹
۳	**۳**	۴۹	۵۱/۳۲	۴۹	۴۹	۵۰	۴۹	۵۱	۵۵	۵۱	۵۵
	۴	۱۱۶	۱۱۷/۸۲	۱۱۶	۱۱۶	۱۱۶	۱۱۶	۱۲۴	۱۱۲	۱۲۴	۱۱۲
	۵	۲۲۰	۲۲۲/۳۶	۲۲۱	۲۲۴	۲۲۵	۲۲۵	۲۴۱	۲۳۹	۲۳۶	۲۳۷
	۶	۳۷۴	۳۷۶/۶۷	۳۷۴	۳۸۳	۳۸۱	۳۸۱	۴۱۳	۴۲۳	۴۰۰	۴۲۰
	۷	۵۸۱	۵۸۲/۹۴	۵۸۹	۶۰۴	۵۸۰	۵۹۸	۶۴۳	۶۵۲	۶۲۴	۶۴۹
	۸	۸۵۸	۸۶۲/۴۵	۸۸۸	۸۹۷	۸۹۹	۸۹۸	۹۵۷	۹۶۴	۹۲۴	۹۳۵
	۹	۱۲۱۲	۱۲۱۴/۶۷	۱۲۳۶	۱۲۷۶	۱۲۵۶	۱۲۷۴	۱۳۴۱	۱۳۷۷	۱۲۸۸	۱۳۶۶
	۱۰	۱۶۴۴	۱۶۴۶/۶۷	۱۶۸۵	۱۷۵۱	۱۶۹۵	۱۷۳۹	۱۸۱۵	۱۸۸۸	۲۱۷۲	۱۸۶۲
۴	**۳**	۱۵۴	۱۵۵/۷۴	۱۵۳	۱۵۴	۱۵۵	۱۵۴	۱۶۸	۱۶۶	۱۶۹	۱۶۷
	۴	۴۸۵	۴۸۹/۶۷	۴۸۶	۴۸۶	۴۸۷	۴۹۰	۵۲۹	۵۶۸	۵۱۷	۶۱۴
	۵	۱۱۶۶	۱۱۶۹/۷۳	۱۱۷۳	۱۱۶۹	۱۱۷۶	۱۱۸۲	۱۲۷۹	۱۳۲۰	۱۲۴۸	۱۲۹۳
	۶	۲۳۴۸	۲۳۸۸/۶۳	۲۴۴۹	۲۴۰۷	۲۴۴۵	۲۴۳۷	۲۶۰۶	۲۷۳۱	۲۵۶۰	۲۶۹۷
	۷	۴۳۴۷	۴۳۵۳/۷۸	۴۴۸۶	۴۴۳۰	۴۵۲۵	۴۴۱۹	۴۷۷۱	۵۰۷۵	۴۶۸۴	۴۹۷۷
	۸	۷۳۲۴	۷۳۲۹/۴۲	۷۶۶۱	۷۵۳۲	۷۶۵۸	۷۵۴۷	۸۰۸۰	>day	۷۸۹۸	۸۲۳۷
	۹	۱۱۶۰۵	۱۱۶۱۰/۴۲	۱۱۷۱۶	۱۱۹۴۱	۱۲۰۲۰	۱۱۸۵۲	۱۲۸۱۳	>day	۱۲۵۳۵	۱۳۵۱۳

۱۰	۱۷۷۱۲	۱۷۷۲۳/۱۳	۱۷۷۶ ۵	۱۸۱۳۶	۱۸۱۱ ۹	۱۸۰۵۴	۱۹۳۶۷	>day	۲۶۶۲ ۴	۲۰۵۸ ۴

جدول ۶- اندازه آرایه پوشش تولید شده برای CA (N; 4,10, d) بازای $2 \le d \le 10$

d	LSHADE N.Best	LSHADE N.Avg	GS N.Best	TLBO N.Best	PSTG N.Best	HC-BAT N.Best	Jenny N.Best	TConfig N.Best	PICT N.Best	IPOG N.Best
۲	۳۴	۳۹/۶۰	۳۵	۳۷	۳۴	۳۷	۳۹	۴۵	۴۳	۴۵
۳	۲۰۷	۲۱۰/۷۰	۲۰۹	۲۱۱	۲۱۳	۲۱۰	۲۲۱	۲۳۵	۲۳۱	۲۳۳
۴	۶۶۳	۶۶۸/۳۰	۶۷۵	۶۷۶	۶۸۵	۶۷۴	۷۰۳	۷۱۸	۷۲۴	۷۱۵
۵	۱۶۰۸	۱۶۱۴/۶۰	۱۶۳۸	۱۶۶۲	۱۷۱۶	۱۶۵۷	۱۷۷۹	۱۸۷۵	۱۸۱۲	۱۸۶۹
۶	۳۳۰۷	۳۳۱۴/۸۱	۳۴۰۳	۳۴۴۲	۳۸۸۰	۳۴۳۱	۳۵۱۹		۳۷۳۵	۳۸۹۱
۷	۶۲۷۶	۶۲۸۴/۶۲	۶۲۸۸	۶۳۶۶		۶۳۴۲	۶۴۸۲		۶۸۸۶	۷۱۵۳
۸	۱۰۷۳۷	۱۰۷۹۴/۷۵	۱۰۷۴۷	۱۰۸۲۴	NA	۱۰۷۸۹	۱۱۰۲۱	NA	۱۱۶۳۰	۱۲۱۰۳
۹	۱۷۲۳۹	۱۷۲۵۳/۸۳	۱۷۲۴۳	۱۷۲۴۹		۱۷۲۵۳	۱۷۵۲۷		۱۸۵۰۷	۱۹۴۱۹
۱۰	۲۶۳۴۸	۲۶۳۵۸/۴۱	۲۶۳۵۲	۲۶۳۶۴		۲۶۳۶۱	۲۶۶۲۴		۲۸۰۲۵	۲۹۵۴۱

با جمع‌بندی نتایج کسب شده در تمام ۷۳ پیکربندی مختلف به این نتیجه می‌رسیم که استراتژی LSHADE در ۵۹ مورد (تقریبا ٪۸۰) دارای بهترین کارایی در مقایسه با استراتژی‌های GS (۲۴ مورد؛ تقریبا ٪۳۳)، TLBO (۱۷ مورد؛ تقریبا ٪۲۳)، HC-BAT (۱۲ مورد؛ تقریبا ٪۱۷)، IPOG (۹ مورد؛ تقریبا ٪۱۲)، PSTG (۸ مورد؛ تقریبا ٪۱۱)، Jenny (۱ مورد؛ تقریبا ٪۲) و PICT (صفر مورد؛ ٪۰) است.

برای ارزیابی دقیق‌تر عملکرد استراتژی LSHADE و مقایسه آن با استراتژی‌های فراابتکاری GS، TLBO، WOA، BAPSO و GSTG، آنها را روی شش پیکربندی بصورت CA (N; 2, 4, d) بازای $11 \le d \le 16$ اجرا می‌کنیم. نتایج، شامل معیارهای اندازه‌های بهترین (N.Best) و متوسط (N.Avg) آرایه‌های پوشش تولید شده، تعداد متوسط ارزیابی‌های تابع محاسبه وزن (NFFEs)، تعداد متوسط فراخوانی‌های هر استراتژی (NACs) و متوسط زمان اجرای (RunTime) استراتژی‌ها در ۱۰۰ بار اجرا می‌باشد. جدول ۷ نتایج حاصل شده را نشان می‌دهد. خلاصه کلی این نتایج که در پنج سطر آخر این جدول نشان داده شده‌اند شامل رتبه استراتژی‌ها در هر کدام از معیارهای فوق‌الذکر می‌باشد. در این مقاله از آزمون فریدمن که یک آزمون فرضیه آماری ناپارامتریک است جهت مقایسه چندگانه نمونه‌های مرتبط استفاده می‌کنیم [۳۸]. مطابق با این جدول، استراتژی LSHADE دارای اولین رتبه از نظر معیارهای تولید آرایه‌های پوشش با اندازه‌های بهترین (N.Best) و متوسط (N.Avg) هست، علاوه‌براین، این استراتژی کم‌ترین تعداد NACs را دارد. از نظر تعداد متوسط فراخوانی‌های تابع محاسبه وزن و متوسط زمان اجرا، استراتژی GS اولین رتبه (کمترین زمان اجرا) را دارا می‌باشد. بعد از استراتژی GS، استراتژی LSHADE دارای کم‌ترین رتبه در این معیارها (NFFEs و RunTime) است.

برای مقایسه سرعت همگرایی استراتژی LSHADE با استراتژی‌های فراابتکاری GS، TLBO، WOA، BAPSO و GSTG، آنها را روی شش پیکربندی مختلف اجرا می‌کنیم. مطابق با نمودارهای همگرایی موجود در شکل ۵، استراتژی LSHADE دارای سرعت همگراییِ بالاتری نسبت به بقیه استراتژی‌ها است.

۵- نتیجه‌گیری

تست ترکیبی t-ستونی تکنیکی است که با تولید آرایه‌ای از نمونه‌های تست، مشکل انفجار ترکیبی را در تست جامع سیستم‌های نرم‌افزاری مدیریت می‌کند. در این مقاله، با توجه به NP-سخت بودن مساله تولید آرایه پوشش کمینه، از الگوریتم تکامل تفاضلی تطبیقی مبتنی بر تاریخچه موفقیت و کاهش خطی اندازه جمعیت (معروف به LSHADE) جهت تولید آرایه پوشش کمینه استفاده کرده‌ایم. برای ارزیابی کارایی، استراتژی LSHADE را بهمراه استراتژی‌های مبتنی بر ریاضی مانند TConfig، حریصانه مانند IPOG، Jenny و PICT و فراابتکاری مانند HC-BAT، PSTG، TLBO و GS روی ۷۳ پیکربندی مختلف اجرا کرده‌ایم. نتایج حاصله نشان می‌دهند که استراتژی LSHADE در ٪۸۰ موارد دارای بهترین کارایی در مقایسه با TConfig (٪۴)، IPOG (٪۱۲)، Jenny (٪۲)، PICT (٪۰)، GS (٪۳۳)، TLBO (٪۲۳) وHC-BAT (٪۱۷) است. همچنین، برای ارزیابی دقیق‌ترِ عملکرد استراتژی LSHADE در مقایسه با استراتژی‌های فراابتکاری GS، TLBO، WOA، BAPSO و GSTG، آن‌ها را روی شش پیکربندی بصورت CA (N; 2, 4, d) بازای $11 \le d \le 16$ اجرا کرده‌ایم. نتایج آزمون فریدمن نشان می‌دهند که استراتژی LSHADE دارای اولین رتبه از نظر معیارهای N.Best و N.Avg هست، علاوه‌براین، این استراتژی کمترین تعداد NACs را دارد. از نظر معیارهای NFFEs و RunTime، استراتژی LSHADE، بعد از استراتژی GS، دارای اولین رتبه است. ضمناً، نمودارهای همگرایی سرعت همگرایی بالایِ این استراتژی را در مقایسه با استراتژی‌های فراابتکاری دیگر تایید می‌کنند. در کارهای آینده می‌توان الگوریتم LSHADE را با سایر الگوریتم‌های فراابتکاری ترکیب کرد و یا می‌توان بجای استفاده از تکنیک یک-سطر-در-یک-زمان، از تکنیک یک-پارامتر-در-یک-زمان استفاده کرد.

جدول ۷- نتایج اجرای استراتژی‌های فراابتکاری برای پیکربندی CA (N; 2,4, d) بازای $11 \le d \le 16$

d		LSHADE	GS	TLBO	WOA	BAPSO	GSTG
۱۱	N.Best	۱۳۷	۱۳۹	۱۳۹	۱۴۰	۱۴۱	۱۳۹
	N.Avg	۱۴۰/۳۵	۱۴۲/۲۰	۱۴۰/۳۳	۱۴۲/۲۱	۱۴۳/۵۰	۱۴۱/۰۱
	NFFEs (avg)	۴۱۲۴۸۷	۲۸۶۱۴۸	۱۶۲۹۴۹۰	۲۶۱۹۰۰۰	۱۴۵۶۰۰۰	۱۳۹۵۶۰۰
	NACs (avg)	۱۴۰/۳۵	۱۴۲/۲۰	۱۴۱/۳۳	۱۴۶/۵۰	۱۴۶/۶۰	۱۴۲/۲۷

	RunTime (avg)	۵/۱۵	۳/۶۷	۳۸/۷۲	۳۰/۹۴	۳۰/۳۶	۱۳۷/۵۳
	N.Best	۱۶۴	۱۶۴	۱۶۴	۱۶۸	۱۶۶	۱۶۵
	N.Avg	۱۶۶/۲۱	۱۶۸/۲۰	۱۶۶/۶۷	۱۷۰/۳۴	۱۷۱/۳۰	۱۶۷/۵۰
۱۲	NFFEs (avg)	۴۷۷۶۲۹	۳۴۲۰۷۱	۲۰۱۰۰۰۰	۳۰۷۸۰۰۰	۱۷۴۲۰۰۰	۱۶۵۳۸۲۱
	NACs (avg)	۱۶۶/۲۱	۱۶۸/۶۰	۱۶۷/۶۷	۱۷۱/۲۳	۱۷۵/۲۰	۱۶۸/۵۶
	RunTime (avg)	۴/۲۱	۳/۲۳	۵۸/۶۸	۳۴/۲۳	۳۳/۸۴	۱۴۰/۴۲
	N.Best	۱۹۱	۱۹۳	۱۹۳	۱۹۷	۱۹۶	۱۹۴
	N.Avg	۱۹۳/۶۰	۱۹۶/۸۰	۱۹۴/۹۳	۱۹۹/۵۹	۱۹۹/۲۰	۱۹۶/۶۰
۱۳	NFFEs (avg)	۵۵۳۶۴۸	۴۰۲۰۷۷	۲۳۳۱۶۳۰	۳۵۸۲۴۶۸	۲۰۵۳۰۰۰	۱۹۴۶۷۲۳
	NACs (avg)	۱۹۳/۷۰	۱۹۷/۳۲	۱۹۵/۵۳	۲۰۰/۰۲	۲۰۶/۳۰	۱۹۷/۶۰
	RunTime (avg)	۴/۷۰	۳/۳۲	۷۸/۳۰	۳۹/۵۱	۴۰/۲۷	۱۷۰/۲۳
	N.Best	۲۲۰	۲۲۵	۲۲۳	۲۲۹	۲۲۹	۲۲۶
	N.Avg	۲۲۳/۲۰	۲۲۶/۸۰	۲۲۴/۳۳	۲۳۰/۵۴	۲۳۱/۳۲	۲۲۸/۳۰
۱۴	NFFEs (avg)	۶۴۴۵۰۴	۴۶۰۸۵۷	۲۶۹۸۳۰۴	۴۱۵۸۰۰۰	۲۳۴۸۰۰۰	۲۲۶۰۱۷۸
	NACs (avg)	۲۲۴/۴۰	۲۲۷/۴۰	۲۲۵/۵۴	۲۳۲/۲۳	۲۳۵/۸۰	۲۲۹/۴۵
	RunTime (avg)	۵/۱۴	۴/۶۳	۸۹/۲۷	۴۷/۰۴	۴۵/۵۹	۱۹۳/۷۴
	N.Best	۲۵۳	۲۵۷	۲۵۵	۲۶۲	۲۶۲	۲۶۱
	N.Avg	۲۵۷/۳۱	۲۵۹/۸۰	۲۵۷/۸۷	۲۶۵/۴۳	۲۶۴/۱۲	۲۶۱/۸۰
۱۵	NFFEs (avg)	۷۳۴۱۴۲	۵۲۹۴۳۵	۳۰۱۵۲۴۵	۴۶۰۸۰۰۰	۲۶۸۹۰۰۰	۲۵۸۱۹۸۰
	NACs (avg)	۲۵۷/۴۰	۲۶۰/۴۰	۲۵۸/۷۸	۲۶۶/۲۳	۲۶۹/۹۰	۲۶۵/۴۵
	RunTime (avg)	۵/۹۶	۵/۵۳	۵۶/۳۴	۵۵/۲۵	۵۱/۸۷	۲۴۰/۳۲
	N.Best	۲۸۸	۲۹۴	۲۸۹	۲۹۹	۲۹۷	۲۹۱
	N.Avg	۲۹۱/۳۰	۲۹۶/۱۰	۲۹۲/۲۰	۳۰۱/۳۰	۳۰۰/۲۰	۲۹۵/۴۰
۱۶	NFFEs (avg)	۸۲۱۳۷۱	۶۰۲۹۱۱	۳۵۲۳۸۴۲	۵۴۲۷۰۰۰	۳۰۴۷۰۰۰	۲۹۲۴۷۳۱
	NACs (avg)	۲۹۱/۳۵	۲۹۶/۴۰	۲۹۳/۶۳	۳۰۲/۵۰	۳۰۵/۷۰	۲۹۶/۴۷
	RunTime (avg)	۷/۶۵	۵/۳۱	۸۷/۳۱	۶۱/۶۵	۵۹/۷۳	۳۱۵/۴۳
Friedman Test	**Rank** N.Best	**۱**	**۲**	**۳**	**۶**	**۵**	**۴**
	Rank N.Avg	**۱**	**۴**	**۲**	**۵**	**۶**	**۳**
	Rank NFFEs (avg)	**۲**	**۱**	**۵**	**۶**	**۴**	**۳**
	Rank NACs (avg)	**۱**	**۳**	**۲**	**۵**	**۶**	**۴**
	Rank RunTime (avg)	**۲**	**۱**	**۵**	**۴**	**۳**	**۶**

شکل ۵- منحنی‌های همگرایی برای استراتژی‌های فراابتکاری در پیکربندی‌های مختلف

مراجع

[1] B. S. Ahmed, T. S. Abdulsamad, and M. Y. Potrus, "Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the cuckoo search algorithm," *Information and Software Technology*, vol. 66, pp. 13-29, 2015.

[2] L. Luo, "Software testing techniques," *Institute for software research international Carnegie mellon university Pittsburgh, PA*, vol. 15232, no. 1-19, p. 19, 2001.

[3] D. R. Kuhn and M. J. Reilly, "An investigation of the applicability of design of experiments to software testing," in *27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002. Proceedings.*: IEEE, pp. 91-95, 2002.

[4] C. Yilmaz, M. B. Cohen, and A. A. Porter, "Covering arrays for efficient fault characterization in complex configuration spaces," *IEEE Transactions on Software Engineering*, vol. 32, no. 1, pp. 20-34, 2006.

[5] A. W. Williams, "Determination of test configurations for pair-wise interaction coverage," in *Testing of Communicating Systems*: Springer, pp. 59-74, 2002.

[6] A. W. Williams and R. L. Probert, "A practical strategy for testing pair-wise coverage of network interfaces," in *Proceedings of ISSRE'96: 7th International Symposium on Software Reliability Engineering*: IEEE, pp. 246-254, 1996.

[7] C. Nie, H. Wu, X. Niu, F.-C. Kuo, H. Leung, and C. J. Colbourn, "Combinatorial testing, random testing, and adaptive random testing for detecting interaction triggered failures," *Information and Software Technology*, vol. 62, pp. 198-213, 2015.

[8] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys (CSUR)*, vol. 43, no. 2, pp. 1-29, 2011.

[9] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: A general strategy for t-way software testing," in *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, IEEE, pp. 549-556, 2007.

[10] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing," *Software Testing, Verification and Reliability*, vol. 18, no. 3, pp. 125-148, 2008.

[11] M. Forbes, J. Lawrence, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Refining the in-parameter-order strategy for constructing covering arrays," *Journal of Research of the National Institute of Standards and Technology*, vol. 113, no. 5, p. 287, 2008.

[12] Y.-W. Tung and W. S. Aldiwan, "Automating test case generation for the new generation mission software system," in *2000 IEEE Aerospace Conference. Proceedings (Cat. No. 00TH8484)*, vol. 1: IEEE, pp. 431-437, 2000.

[13] K. Z. Zamli, M. F. Klaib, M. I. Younis, N. A. M. Isa, and R. Abdullah, "Design and implementation of a t-way test data generation strategy with automated execution tool support," *Information Sciences*, vol. 181, no. 9, pp. 1741-1758, 2011.

[14] B. S. Ahmed, M. A. Sahib, and M. Y. Potrus, "Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing," *Engineering Science and Technology, an International Journal*, vol. 17, no. 4, pp. 218-226, 2014.

[15] J. Czerwonka, "Pairwise testing in the real world: Practical extensions to test-case scenarios," *Microsoft Corporation, Software Testing Technical Articles*, 2008.

[16] B. Jenkins, "Jenny download web page," *Bob Jenkin's website*, 2005.

[17] S. Esfandyari and V. Rafe, "A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy," *Information and Software Technology*, vol. 94, pp. 165-185, 2018.

[18] H. Zakaria, K. Zamli, and F. Din, "Hybrid Migrating Birds Optimization Strategy for t-way Test Suite Generation," in

Journal of Physics: Conference Series, vol. 1830, no. 1: IOP Publishing, p. 012013, 2021.

[19] زهرا عباسی، سجاد اسفندیاری، وحید رافع «ساخت آرایه پوشش با استفاده از الگوریتم بهینه‌سازی مبتنی برآموزش و یادگیری»، مجله مهندسی برق دانشگاه تبریز، دوره‌ی ۴۸، شماره‌ی ۱، صفحه‌ی ۱۶۱ تا ۱۷۱، بهار ۱۳۹۷.

[20] B. S. Ahmed, K. Z. Zamli, and C. P. Lim, "Constructing a t-way interaction test suite using the particle swarm optimization approach," *International Journal of Innovative Computing, Information and Control*, vol. 8, no. 1, pp. 431-452, 2012.

[21] B. S. Ahmed, K. Z. Zamli, and C. P. Lim, "Application of particle swarm optimization to uniform and variable strength covering array construction," *Applied soft computing*, vol. 12, no. 4, pp. 1330-1347, 2012.

[22] H. Wu, C. Nie, F.-C. Kuo, H. Leung, and C. J. Colbourn, "A discrete particle swarm optimization for covering array generation," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 4, pp. 575-591, 2014.

[23] Y. A. Alsariera, A. H. Al Omari, M. A. Albawaleez, Y. K. Sanjalawe, and K. Z. Zamli, "Hybridized BA & PSO t-way Algorithm for Test Case Generation," *IJCSNS*, vol. 21, no. 10, p. 343, 2021.

[24] D. Karaboga, "Artificial bee colony algorithm," *scholarpedia*, vol. 5, no. 3, p. 6915, 2010.

[25] A. K. Alazzawi, A. A. B. Homaid, A. A. Alomoush, and A. A. Alsewari, "Artificial bee colony algorithm for pairwise test generation," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, no. 1-2, pp. 103-108, 2017.

[26] A. K. Alazzawi, H. M. Rais, and S. Basri, "Artificial bee colony algorithm for t-way test suite generation," in *2018 4th International Conference on Computer and Information Sciences (ICCOINS)*, IEEE, pp. 1-6, 2018.

[27] A. K. Alazzawi *et al.*, "HABCSm: a hamming based t-way strategy based on hybrid artificial bee colony for variable strength test sets generation," *arXiv preprint arXiv:2110.03728*, 2021.

[28] N. Ramli, R. R. Othman, and M. S. A. R. Ali, "Optimizing combinatorial input-output based relations testing using Ant Colony algorithm," in *2016 3rd International Conference on Electronic Design (ICED)*, IEEE, pp. 586-590, 2016.

[29] M. Z. Z. Ahmad, R. R. Othman, M. S. A. R. Ali, and N. Ramli, "A Self-Adapting Ant Colony Optimization Algorithm Using Fuzzy Logic (ACOF) for Combinatorial Test Suite Generation," in *IOP Conference Series: Materials Science and Engineering*, vol. 767, no. 1: IOP Publishing, p. 012017, 2020.

[30] وحید رافع, سجاد اسفندیاری، «راهکاری نوین جهت تولید دنباله آزمون کمینه در فرآیند آزمون نرم افزار با ترکیب الگوریتم های جستجوی تپه نوردی و جستجوی خفاش»، مجله مهندسی برق دانشگاه تبریز، دوره‌ی ۴۶، شماره‌ی ۳، صفحه‌ی ۲۵ تا ۳۵، پائیز ۱۳۹۵.

[31] A. A. Alsewari, L. M. Xuan, and K. Z. Zamli, "Firefly combinatorial testing strategy," in *Science and Information Conference*, Springer, pp. 936-944, 2018,

[32] H. N. N. Al-Sammarraie and D. N. Jawawi, "Multiple black hole inspired meta-heuristic searching optimization for combinatorial testing," *Ieee Access*, vol. 8, pp. 33406-33418, 2020.

[33] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341-359, 1997.

[34] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *2013 IEEE congress on evolutionary computation*, IEEE, pp. 71-78., 2013,

[35] R. Tanabe and A. S. Fukunaga, "Improving the search performance of SHADE using linear population size reduction," in *2014 IEEE congress on evolutionary computation (CEC)*, IEEE, pp. 1658-1665, 2014.

[36] B. S. Ahmed and K. Z. Zamli, "A variable strength interaction test suites generation strategy using particle swarm optimization," *Journal of Systems and Software*, vol. 84, no. 12, pp. 2171-2185, 2011.

[37] J. Zhang and A. C. Sanderson, "JADE: adaptive differential evolution with optional external archive," *IEEE Transactions on evolutionary computation*, vol. 13, no. 5, pp. 945-958, 2009.

[38] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *The Annals of Mathematical Statistics*, vol. 11, no. 1, pp. 86-92, 1940.