IJICTR
International Journal of Information & Communication Technology Research

Volume 14- Number 2 – 2022 (32 -40)

Comparative Analytical Survey on SBST Challenges from the Perspective of the Test Techniques

Sepideh Kashefi Gargari
Department of Computer Engineering
Faculty of Engineering
Alzahra University
Tehran, Iran
sepideh.kashefi1994@gmail.com

Mohammad Reza Keyvanpour*
Department of Computer Engineering
Faculty of Engineering
Alzahra University
Tehran, Iran
keyvanpour@alzahra.ac.ir

Received: 2 April 2022 – Revised: 11 May 2022 - Accepted: 27 June 2022

Abstract—**Despite several decades of working on software quality assurance methods, they still require further investigation. It is paramount to detect a process detecting possible software errors with a minimum budget and effort. Search-based software testing (SBST) is an approach to automated software testing that aims to find a suitable alternative to manual testing. The SBST is to transform the test problem into an optimization problem and obtain better solutions by searching the problem space. Search-based testing has some disadvantages and advantages. The novelty of this paper is that, besides representing the significance and efficiency of SBST in software testing, the search-based test challenges were detected and described from the perspective of the test techniques. Our work is to extract challenges from reliable sources and research and their classification based on test techniques. For this purpose, we considered this framework: 1) A systematic introduction to the most critical metaheuristic optimization algorithms. 2) classifying the test techniques and explaining their advantages and disadvantages. 3) proposing a suitable classification for the challenges of the search-based test area based on the technique used. Our motivation to do this research was to provide complete knowledge about search-based software testing challenges so that new researchers could choose their research fields with prior knowledge and provide a way to improve existing methods. Finally, the results of this paper can be used to compare the existing test techniques *used in SBST*, select the best one, and represent the challenges of each technique.**

Keywords: SBST challenges; problems; test case generation; search-based software testing

Article type: Research Article

Publisher: ICT Research Institute

* Corresponding Author

[DOI: 10.52547/itrc.14.2.32]

I. Introduction

In developing a software system, especially software with high complexity, human mistakes are unavoidable. When the source code is generated, the software must be evaluated in terms of errors. Today, applications are being evidently used in all life aspects. The latest trends in technology and rapid changes in society's demands have led to the introduction of more complex software systems [1]. This has converted software testing to one of the most critical stages of the software development life cycle [2].

Testing mainly aims to find errors occurring during the program's execution processes for all possible inputs. The extensive range of input space makes the number of test cases infinite [3]. In this regard, software cannot be entirely tested with a limited budget and time. Accordingly, we cannot comment on the completeness of the developed software [4]. As a result, the software test is required; however, it does not guarantee the accuracy of the developed software.

Software tests are performed manually and automatically [5]. A manual test is challenging and time-consuming, and it is impossible for software with high complexity. As a result, the test was automated, and the automation process was transferred to machines [6]. If the test process is fully automated, the costs of testing and software development will be remarkably reduced [8 ,7]. SBST is an attempt to automate software testing. Moreover, search-based test data generation is a technique to obtain a test requirement, as an optimization problem with a numerical and heuristic function to be solved [9].

In this paper, after reviewing previous studies, search-based testing is introduced. Then the general testing techniques are classified into three categories of structural testing, functional testing, non-functional testing, whose ideas, advantages, and disadvantages are also described. The idea of using search is proved to provide high potentials in each of the aforementioned techniques. However, in contrast to the achieved accomplishments, there are challenges, in which their expression based on test techniques is the main objective of this study. In Section 5, the search-based testing challenges are presented as the breakdowns of all test techniques.

II. Related Work

Harman and Jones (2001) published an article on software engineering(SE) [10]. This article stated that a new area of research, called search-based software engineering (SBSE), is emerging. SBSE is a branch of software engineering [11] and has a high capability in all software areas, especially testing[12]. SBSE techniques have shown promising results and give us hope that someday it will be possible to automate the tedious and, laborious parts of software development, or at least partially automated software development [13].

Following Ref. [10], much research has addressed the application of searching in testing.

From the perspective of test techniques, some researchers [14-18] examined structural testing based on searching. Ref. is on functional testing[19], and articles [20, 21] are on non-functional testing.

The first search-based algorithms used to automate the generation of test data were descending gradients [22] and algorithms such as hill climbing(HC) [17, 23], tabu search [24-26], and simulated annealing(SA) [27, 28]. Although these algorithms had advantages, they were time-consuming and inefficient and would get caught if there was a local optimization.

Subsequently, other algorithms were introduced that provided a better position than the original algorithms. However, they still faced problems, and there was a possibility of getting caught in the local optimization.

Following the research process, other algorithms, such as genetic algorithms [29-31], were used to generate test data, which provided better conditions. Ref. [32] discusses test data generation for structural testing using genetic algorithms. In Ref. [33], genetic algorithms and reinforcement learning are combined to generate the test data.

As research expanded in later years, algorithms such as particle swarm[29, 34, 35], ant colony[15, 36-38], and bee colony[11, 39] were used, which yielded better results. Some studies have also reviewed previous works [3, 16, 40-42].

Furthermore, some papers have also addressed the challenges in this field [44 ,43 ,11]. In this article, we further explored the challenges in this field.

III. Search Based Software Testing

SBST generates test cases/test data guided by measurements gauging how far tests are from reaching a coverage target [45]. Search algorithms can gradually improve tests to achieve high coverage, using the fitness function as measurement criteria [46, 47]. The value of the fitness function is a numerical value expressing the performance of the candidate solutions according to the current optimal candidate solution for comparison to satisfy the test criterion [3, 48, 49]. Testing mainly aims to generate an optimal set of test cases revealing the software errors [50] according to the test adequacy criterion. The adequacy criteria for testing are also formulated as a fitness function [51]. Test adequacy criteria distinguish acceptable test cases from unacceptable ones, and determining whether or not a test process is complete [4].

The rationale behind all test data generation techniques is that the possible inputs to the program constitute a search space, and the search for the solution is carried out in this space. These techniques have outperformed others in resolving software problems in complex and large search spaces [52]. Figure 1 shows the general view of the search-based test case generation steps and their original elements.

[DOI: 10.52547/itrc.14.2.32]

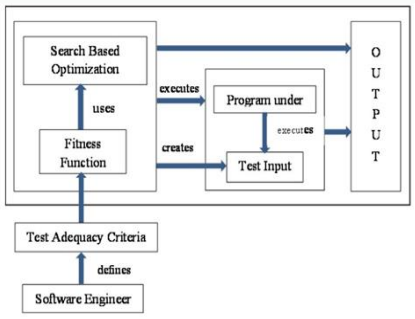

Figure 1. The search-based test input generation scheme [14]

As shown in Figure 1, search-based optimization is one of the main components of search-based testing. Each search-based algorithm can be used in this section to generate new test data according to its logic. The following section will provide more details on the search-based algorithms.

IV. METAHEURISTIC OPTIMIZATION ALGORITHMS

In computer science, the problem of finding the best solution among the possible solutions is called an optimization problem. All optimization problems have a purpose. In such cases, the objective function is often set by the constraints imposed by the nature of the problem. During the search to find the appropriate solution, the value of the objective function takes control of the search.

The objective function is defined so that it must either have a minimum value or a maximum value. An answer space is also evaluated during the problem-solving effort [54 ,53]. So it can be said that optimization problems have the following three key elements [51]:

- A set of restrictions
- There is an answer space to search.
- The objective function should be maximized or minimized during the search depending on the nature of the problem.

In the introduction section, we enumerated several optimization algorithms among the most widely used in test data generation, which we will now describe in more detail. There are various categories for optimization algorithms, so we will skip the review of how to classify them and explain only some of the most famous algorithms in this field. The optimization algorithms are placed into two categories: global and local algorithms. Local algorithms are those algorithms in which the solution is sought locally, and each time we search for a better solution from the neighbors, we may stop at the local optimal point. Nevertheless, global algorithms are looking for answers among all possible solutions and are more likely to succeed. The following section explains the usage and application of some of the most important algorithms in software tests.

A. Hill climbing

Hill-climbing is a local search algorithm to maximize the objective function. The algorithm starts with an initial solution randomly selected from the search space. At each iteration, the neighborhood of the current solution is investigated. It replaces the current solution if it is a better solution [14]. There are two choices in the hill-climbing algorithm:

- The current solution is replaced by the first neighbor who has improved merit.
- In the second case, the current solution is replaced by a neighbor who gives the greatest increase in competition among all the neighbors.

The search continues until no improvement is found in the neighbors to the current solution. When the search ends, the maximum (probably local) is found.

1. Select an initial solution s that belongs to solution space S
2. Repeat
3. Select s' that belongs to neighbourhood N of s such that f(s') > f(s)
4. s = s'
5. Until f(s) ≥ f(s'), for all s' that belongs to neighbourhood of s

Figure 2. Hill climbing algorithm for maximising an objective function f [14]

Figure 3 shows that the hill climbing algorithm is caught in the local optimization and does not achieve global optimization.

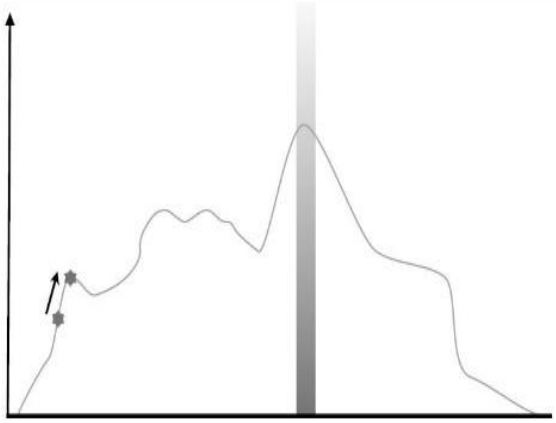

Figure 3. Hill climbing and trapping in to local optimal [55]

B. Simulated annealing

Simulated annealing can be considered a variety of hill climbing that prevents the maximum local problem by allowing people to move with less fitness [51]. If the neighbor's answer is better than the current answer, the algorithm sets it as the current answer and moves towards it. Otherwise, the algorithm accepts that answer with an exp (- ΔE / T) probability as the current answer. In this relation, ΔE is the difference between the objective function of the current answer and the neighboring answer. T is the temperature parameter. At each temperature, several repetitions are performed,

[DOI: 10.52547/itrc.14.2.32]

and then the temperature is slowly reduced. A high temperature is set in the initial steps to make it more likely to accept worse answers. As the temperature gradually decreases in the final steps, the probability of accepting worse answers will decrease, and thus the algorithm will converge toward a good answer.

Figure 4 shows that the algorithm can exit the local optimization

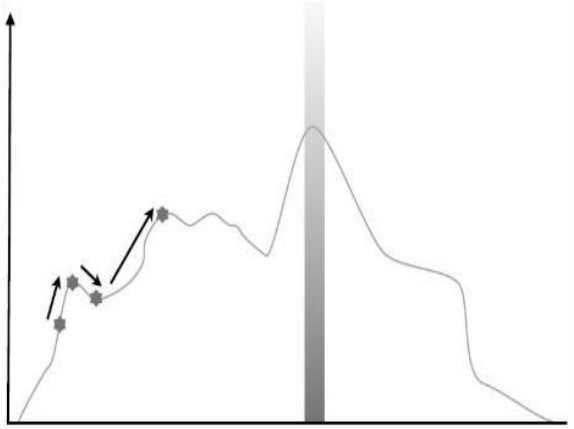

Figure 4. The transition from local to global optimal in SA [55]

C. *Genetic algorithm*

Genetic algorithms use the concepts of population and recombination [56]. Among all optimization algorithms, genetic algorithms are the most commonly applied research technique in SBSE.

First, the answer to the problem is formulated as a gene in this algorithm, and a set of answers is considered randomly. Then, depending on their compatibility and appropriateness, three types of functions called selection, crossover, and mutation are performed, and new sets of answers are created. These answers replace the worst answers in the initial set. Finally, the answer is given by satisfying the condition of stopping, which can be a condition of convergence.

1. Randomly generate or seed initial population P
2. Repeat
3. Evaluate fitness of each individual in P
4. Select parents from P according to selection mechanism
5. Recombine parents to form new offspring
6. Construct new population P' from parents and offspring
7. Mutate P'
8. P = P'
9. Until Stopping Condition Reached

Figure 5. Generic genetic algorithm [14]

V. SOFTWARE TESTING TECHNIQUES

Test case generation is a vital concept used in software testing derived from the user requirements specifications [57]. However, there are different types of tests, and different categories can be imagined. The test type affects the test cases generation. In this regard, we divide the general software testing techniques into three categories, as presented in Figure 6.

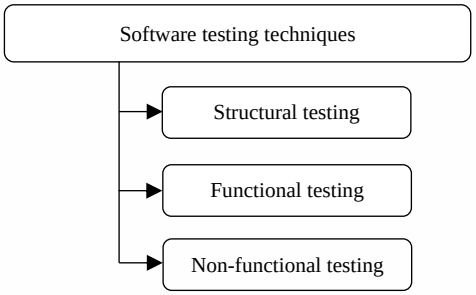

Figure 6. Software test techniques

Table I. presents the ideas, advantages, disadvantages, and examples of each test technique. Table I is partly taken from [4].

The main idea of the structural test is to generate test data/case for the completion of the execution in the prescribed paths. Structural testing plays a fundamental role in the miller and Spooner approach, an application area that has involved the best attention in SBST [20]. In the structural test, the internal structure or the source code is considered, and the program's behavior with its code execution is examined. The test data required to run is derived from program logic and applied to the code. This type of test considers how to act. The main structural tests are statement coverage, branch coverage, path coverage, and data flow coverage [14]. The structural test has been the main focus of SBST so far.

In functional testing, all the system's internal mechanisms are ignored, and the generated output is focused. It is assumed that there is no information about the internal details of the software and the focus of the tests is on different outputs and inputs [58].

Non-functional testing determines how the system or software works and can be performed at all system levels. This type of test is to test the non-functional features of the system never tested by the functional test. Features such as performance, security, portability, scalability, usability, efficiency are part of this test.

TABLE I. THE CLASSIFICATION OF TESTING TECHNIQUES

-In structural test, the internal structure, or same source code, are considered and the application behavior is investigated by executing its code. The required test data are derived from program logic and applied to code	Idea	Structural testing
-Easy implementation ability -Understandable -Comprehensive	Advantage	

[DOI: 10.52547/itrc.14.2.32]

-The generation of test cases is usually done too late in the software development cycle. -Detected errors are very difficult and costly because changes affect a large part of the design, implementation and testing procedures [59].	Disadvantage	
-Statement coverage, branch coverage, path coverage	Example	
-The test cases are generated using system specifications [60]. The purpose of functional testing is testing software performance. In the functional test it is assumed that there is no information on the internal details of the software and the test focus is on different output from different inputs.	Idea	Functional testing
-The test can begin faster in the software design process. -Incompatibilities and ambiguities in the specifications by the testers will be identified sooner . -Errors are openly detected and correction cost is not much	Advantage	
-There is a need for official system specifications that are difficult to identify in the actual program. -The specification is complex and difficult to understand . -Implementations must exactly match the specifications.	Disadvantage	
-Input validation and examining the behavior of the system against the processing of large and heavy queries	Example	
-Non-functional tests indicate how a system works and is a test to measure the characteristics of systems that can be measured in a variable scale.	Idea	Non-functional testing
-Considering other behaviors in spite of the system's logical behaviors.	Advantage	
-It is highly complex as it is dependent on both software and hardware features.	Disadvantage	
-Scalability, Efficiency, Usability, Quality of service, Execution time, Security [21]	Example	

VI. CHALLENGES OF SBST

The general challenge in generating test data is that the test data generated must have the potential to detect program errors, and the result will be better if it takes less time, effort, and cost.

The easiest way to generate test data is to generate it manually. However, manually generating test data is practically impossible for large and highly complex software. Even if the software has a very high degree of expertise in the test, manual production of test data for complex software is unreasonable due to the vast input area. It will waste the examiner's expense, time, and effort.

Also, if we look at the issue from a human resources perspective, the test costs and the nature of the software tests are such that they require a high concentration of the examiner. Most programmers and software engineers are reluctant to test and generate manual test data. For this reason, software quality managers seek to use automatic methods to reduce the cost and time of the test.

Automated testing has its challenges .In this section, the challenges of SBST are explained for each technique.

A. Challenges of structural test

1) Selection of test criteria

There are many criteria for structural testing (e.g., path coverage, branch coverage, statement coverage, and edge coverage). Path coverage is the strongest structural cover criteria, and branch coverage is the standard structural test [36]. The first challenge is choosing the test criteria meeting our requirements with a minimum cost. Selecting a stronger criterion requires higher computational resources and, considering it as a target criterion should be a justification.

2) Nested loops

The presence of nested loops in the program code is directly associated with an increase in the number of targets. As the number of target paths generated for the search increases, finding a solution representing the complete set becomes more difficult. In this case, we may need a better search guide, or we may need to change the algorithm or its fitness function [61].

3) Uncoverd targets

Uncovered targets cannot be easily located in the category of infeasible routes because the source of uncovered targets may be an incomplete search and may be covered by continuing the search. To find out whether or not a target is impractical, more analysis is required [61].

4) Uncertainty after the testing

Automatic test case generation is used to detect errors. The most common characteristic of the correct behavior of a program is that the program does not encounter errors during execution.

If an exception is detected, it indicates an error; however, if the application is executed without errors, the following questions arise [44]:

- Does the program have the proper performance?
- What test cases should be taken to ensure that the next versions of the program preserve current behavior?

5) Insufficient coverage

SBST is suitable for structural coverage and has been studied more frequently [11]. However, the resulting structural coverage is not always as high as expected [62]. Therefore, we may need to rely on insufficient test sets, and all tests require the use of existing tools [63].

[DOI: 10.52547/itrc.14.2.32]

6) Bloat phenomenon

An innovation in some of the SBSE tools, such as evosuite, is to develop all test sets in one place instead of creating a test case in a single time to cover a distinct target so that all cover targets are targeted at the same time. Before addressing the challenges, the technique has several advantages:

- Impractical coverage targets do not spoil the search.
- There is no limit to target selection.
- There is no incidental coverage.

This method can be better than the classical approach (focusing on individual coverage targets); however, as it seems, it is with problems. During the search, the number of test cases in a test set and the length of individual test cases can vary and create additional challenges such as bloat [44] [64] . Bloat is a complex phenomenon in evolutionary computation, in which length grows abnormally over time to the point where search becomes impossible [65].

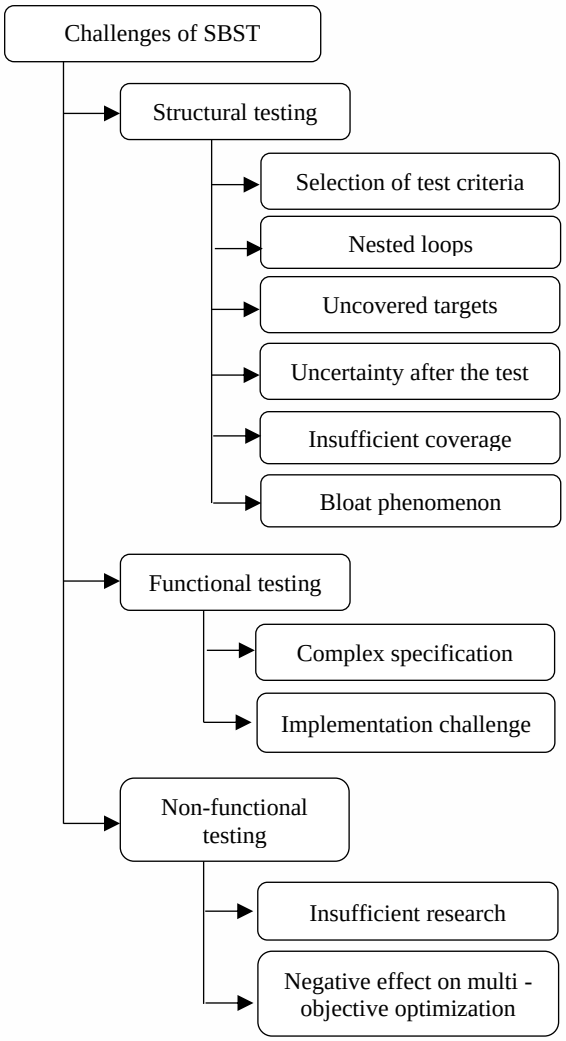

Figure 7. Proposed classification for SBST challenges based on test techniques

B. Challenges of functional testing

In the previous section, we discussed the challenges of structural testing. According to our studies, there is less activity in search-based functional testing than in structural testing.
In this section, we will explain its relevant challenges.

1) Complex nature of system specification

The functional procedures should be taken from the different types of system specifications [3]. However, the system specifications are complex and thereby hard to understand. Not understanding the nature of the problem, its complexity, and the difficulty of formulating the functional issues should be one of the reasons for not addressing the search-based functional testing.

2) Implementation challenge

An implementation must follow the system specifications; however, it is difficult to accurately identify the system specifications in real usage [4]. An existing obstacle to full automation is the fact that mapping must be presented from the abstract model of specifications to the real model to be implemented [3].

C. Challenges of non-functional testing

1) Insufficient research in this area

The lack of studies on non-functional characteristics is surprising due to the increasing importance of non-functional properties. The SBST techniques have a significant advantage and theoretically can be applied to any test problem, in which adequacy criteria can be applied as the work of the fitness function. In essence, testing for execution time, service quality, and energy consumption should not be more severe than the branch coverage, and only a different fit function is required. However, measuring the fitness function for any non-functional feature may be associated with specific challenges [11].

2) Negative effect on multiobjective optimization

One of the reasons indicating why multi-objective techniques have not attracted attention is the lack of field development for non-functional features. Many of the additional goals testers pursue are associated with non-operational features. For example, a tester may be interested in getting more coverage; however, it may also target unusual execution times, security features, or power consumption (or all of these). Because society seems slow to understand the non-functional characteristics, this may significantly affect the application of multi-objective methods [11].

D. Conclusion

In this article, after providing general explanations and the introduction of software testing, some relevant studies were presented. Then we explained the generalities of search-based testing and described how to apply the search-based approach in generating test data. Next, we described the test techniques and classified them into three general categories (namely structural, functional and non-functional tests). According to Table 1, we described the main features

[DOI: 10.52547/itrc.14.2.32]

of each technique. In the last section, which is the most important part of the article, we were to describe the challenges of search-based testing for each of the aforementioned techniques separately.

Given the above, we conducted a study on search-based tests and test case generation in this article and faced a series of challenges expressed in the categories.

All the challenges mentioned in this article have been extracted from credible sources and research and can be cited. Our innovation is the proposed classification for these challenges .The challenges are arranged so that researchers can find complete information about the problems of the field before entering it.

For example, search-based structural testing has its own problems that are different from the search-based functional test, and knowing these issues before conducting further research can benefit researchers.

Also, the search-based test outside the classification based on the test technique has common challenges, which we examined in Ref. [66] and referred to as the general challenges of the search-based test. The study of this paper and Ref. [66] can provide researchers with a good understanding of the general and related challenges related to the search-based test method.

This article can be useful for those interested in the field of software testing, particularly SBST.

REFERENCES

[1] S. Ul Haq and U. Qamar, "Ontology Based Test Case Generation for Black Box Testing," in *Proceedings of the 2019 8th International Conference on Educational and Information Technology*, 2019: ACM, pp. 236-241.

[2] A. P. Mathur, *Foundations of Software Testing: Fundamental Algorithms and Techniques*. Pearson Education, 2008.

[3] P. McMinn, "Search - based software test data generation: a survey," *Software testing, Verification and reliability*, vol. 14, no. 2, pp. 105-156, 2004.

[4] M. R. Keyvanpour, H. Homayouni, and H. Shirazee, "Automatic software test case generation: An analytical classification framework," *International Journal of Software Engineering and Its Applications*, vol. 6, no. 4, pp. 1-16, 2012.

[5] A. M. Bidgoli, H. Haghighi, T. Z. Nasab, and H. Sabouri, "Using swarm intelligence to generate test data for covering prime paths," in *International Conference on Fundamentals of Software Engineering*, 2017: Springer, pp. 132-147.

[6] G. Candea and P. Godefroid, "Automated software test generation: some challenges, solutions, and recent advances," in *Computing and Software Science*: Springer, 2019, pp. 505-531.

[7] S. U. Farooq and S. Quadri, "Identifying some problems with selection of software testing techniques," *Oriental Journal of Computer Science & Technology*, vol. 3, no. 2, pp. 266-269, 2010.

[8] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *2007 Future of Software Engineering*, 2007: IEEE Computer Society, pp. 85-103.

[9] K. Ghani and J. A. Clark, "Automatic test data generation for multiple condition and MCDC coverage," in *2009 Fourth International Conference on Software Engineering Advances*, 2009: IEEE, pp. 152-157.

[10] M. Harman and B. F. Jones, "Search-based software engineering," *Information and software Technology*, vol. 43, no. 14, pp. 833-839, 2001.

[11] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, 2015: IEEE, pp. 1-12.

[12] A. Arcuri and J. P. Galeotti, "Enhancing search-based testing with testability transformations for existing APIs," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 1, pp. 1-34, 2021.

[13] A. Aleti, "On the Effectiveness of SBSE Techniques," in *International Symposium on Search Based Software Engineering*, 2021: Springer, pp. 3-6.

[14] S. Varshney and M. Mehrotra, "Search based software test data generation for structural testing: a perspective," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 4, pp. 1-6, 2013.

[15] C. Mao, X. Yu, J. Chen, and J. Chen, "Generating test data for structural testing based on ant colony optimization," in *2012 12th International Conference on Quality Software*, 2012: IEEE, pp. 98-101.

[16] P. McMinn, "Search-based software testing: Past, present and future," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, 2011: IEEE, pp. 153-163.

[17] M. Harman and P. McMinn, "A theoretical & empirical analysis of evolutionary testing and hill climbing for structural test data generation," in *Proceedings of the 2007 international symposium on Software testing and analysis*, 2007, pp. 73-83.

[18] C. Mao, "Harmony search-based test data generation for branch coverage in software structural testing," *Neural Computing and Applications*, vol. 25, no. 1, pp. 199-216, 2014.

[19] R. Lefticaru and F. Ipate, "Functional search-based testing from state machines," in *2008 1st International Conference on Software Testing, Verification, and Validation*, 2008: IEEE, pp. 525-528.

[20] N. Bala and S. Suhailan, "Effective Search-Based Approach for Testing Non-Functional Properties in Software System: an Empirical Review," *International Journal of Engineering & Technology*, vol. 7, no. 4.28, pp. 368-391, 2018.

[21] W. Afzal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties," *Information and Software Technology*, vol. 51, no. 6, pp. 957-976, 2009.

[22] C. C. Michael, G. E. McGraw, M. A. Schatz, and C. C. Walton, "Genetic algorithms for dynamic test data generation," in *Proceedings 12th IEEE International Conference Automated Software Engineering*, 1997: IEEE, pp. 307-308.

[23] F. C. M. Souza, M. Papadakis, Y. Le Traon, and M. E. Delamaro, "Strong mutation-based test data generation using hill climbing," in *Proceedings of the 9th International Workshop on Search-Based Software Testing*, 2016, pp. 45-54.

[24] P. R. Srivastava, R. Khandelwal, S. Khandelwal, S. Kumar, and S. S. Ranganatha, "Automated test data generation using cuckoo search and tabu search (CSTS) algorithm," *Journal of Intelligent Systems*, vol. 21, no. 2, pp. 195-224, 2012.

[25] K. Perumal, J. M. Ungati, G. Kumar, N. Jain, R. Gaurav, and P. R. Srivastava, "Test data generation: a hybrid approach using cuckoo and tabu search," in *International Conference on Swarm, Evolutionary, and Memetic Computing*, 2011: Springer, pp. 46-54.

[26] E. Díaz, J. Tuya, and R. Blanco, "Automated software testing using a metaheuristic technique based on tabu search," in *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.*, 2003: IEEE, pp. 310-313.

[27] L.-s. LI, X. CAO, and F. WANG, "Test Data Generation Using Simulated Annealing Genetic Algorithm," *COMPUTER TECHNOLOGY AND DEVEI PMENT*, vol. 4, no. 21, p. 4, 2011.

[28] M. Mann, O. P. Sangwan, P. Tomar, and S. Singh, "Automatic goal-oriented test data generation using a genetic algorithm and simulated annealing," in *2016 6th*

[DOI: 10.52547/itrc.14.2.32]

International Conference-Cloud System and Big Data Engineering (Confluence), 2016: IEEE, pp. 83-87.
[29] S. Zhang, Y. Zhang, H. Zhou, and Q. He, "Automatic path test data generation based on GA-PSO," in *2010 IEEE International Conference on intelligent computing and intelligent systems*, 2010, vol. 1: IEEE, pp. 142-146.
[30] [30] A. Rathore, A. Bohara, R. G. Prashil, T. L. Prashanth, and P. R. Srivastava, "Application of genetic algorithm and tabu search in software testing," in *Proceedings of the Fourth Annual ACM Bangalore Conference*, 2011, pp. 1-4.
[31] R. P. Pargas, M. J. Harrold, and R. R. Peck, "Test - data generation using genetic algorithms," *Software testing, verification and reliability*, vol. 9, no. 4, pp. 263-282, 1999.
[32] A. Damia, M. Esnaashari, and M. Parvizimosaed, "Software Testing using an Adaptive Genetic Algorithm," *Journal of AI and Data Mining*, 2021.
[33] M. Esnaashari and A. H. Damia, "Automation of Software Test Data Generation Using Genetic Algorithm and Reinforcement Learning," *Expert Systems with Applications*, p. 115446, 2021.
[34] A. Li and Y. Zhang, "Automatic generating all-path test data of a program based on PSO," in *2009 WRI World Congress on software engineering*, 2009, vol. 4: IEEE, pp. 189-193.
[35] S. Kumar, D. K. Yadav, and D. A. Khan, "A novel approach to automate test data generation for data flow testing based on hybrid adaptive PSO-GA algorithm," *International Journal of Advanced Intelligence Paradigms*, vol. 9, no. 2-3, pp. 278-312, 2017.
[36] A. M. Bidgoli and H. Haghighi, "Augmenting ant colony optimization with adaptive random testing to cover prime paths," *Journal of Systems and Software*, vol. 161, p. 110495, 2020.
[37] H. Sharifipour, M. Shakeri, and H. Haghighi, "Structural test data generation using a memetic ant colony optimization based on evolution strategies," *Swarm and Evolutionary Computation*, vol. 40, pp. 76-91, 2018.
[38] C. Mao, L. Xiao, X. Yu, and J. Chen, "Adapting ant colony optimization to generate test data for software structural testing," *Swarm and Evolutionary Computation*, vol. 20, pp. 23-36, 2015.
[39] S. S. Dahiya, J. K. Chhabra, and S. Kumar, "Application of artificial bee colony algorithm to software testing," in *2010 21st Australian software engineering conference*, 2010: IEEE, pp. 149-154.
[40] A. Ouni, "Search based software engineering: challenges, opportunities and recent applications," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 1114-1146.
[41] M. Harman, "Automated test data generation using search based software engineering," in *Second International Workshop on Automation of Software Test (AST'07)*, 2007: IEEE, pp. 2-2.
[42] M. Khari and P. Kumar, "An extensive evaluation of search-based software testing: a review," *Soft Computing*, vol. 23, no. 6, pp. 1933-1946, 2019.
[43] K. Lakhotia, M. Harman, and P. McMinn, "A multi-objective approach to search-based test data generation," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, 2007, pp. 1098-1105.
[44] S. A. Abdallah, "Challenges and Proposed Solutions of Coverage Based Testing Tools," 2015.
[45] A. Perera, A. Aleti, B. Turhan, and M. Boehme, "An Experimental Assessment of Using Theoretical Defect Predictors to Guide Search-Based Software Testing," *IEEE Transactions on Software Engineering*, 2022.
[46] Y. Lin, Y. S. Ong, J. Sun, G. Fraser, and J. S. Dong, "Graph-Based Seed Object Synthesis for Search-Based Unit Testing," 2021.
[47] M. Harman and P. McMinn, "A theoretical and empirical study of search-based testing: Local, global, and hybrid search," *IEEE Transactions on Software Engineering*, vol. 36, no. 2, pp. 226-247, 2009.
[48] [A. Baughan, N. Hatch, V. Ranganeni, and B. Yang, "Search-Based Test Generation for Robotic Motion Planning Algorithms," 2021.
[49] A. Baresel, H. Sthamer, and M. Schmidt, "Fitness function design to improve evolutionary structural testing," in *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, 2002, pp. 1329-1336.
[50] A. Perera, B. Turhan, A. Aleti, and M. Böhme, "How good does a Defect Predictor need to be to guide Search-Based Software Testing?," *arXiv preprint arXiv:2110.02682*, 2021.
[51] M. Harman, "The current state and future of search based software engineering," in *Future of Software Engineering (FOSE'07)*, 2007: IEEE, pp. 342-357.
[52] M. Harman, P. McMinn, J. T. De Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," in *Empirical software engineering and verification*: Springer, 2010, pp. 1-59.
[53] S. Muthuraman and V. P. Venkatesan, "A comprehensive study on hybrid meta-heuristic approaches used for solving combinatorial optimization problems," in *2017 World Congress on Computing and Communication Technologies (WCCCT)*, 2017: Ieee, pp. 185-190.
[54] "Heuristics in optimisation." homes.ieu.edu.tr/~agokce/Courses/Lecture%201%20introtoOP.pdf (accessed.
[55] Y. Zhu, G. Yang, C. Zhuang, C. Li, and D. Hu, "Oral cavity flow distribution and pressure drop in balaenid whales feeding: A theoretical analysis," (in eng), *Bioinspir Biomim*, Jan 24 2020, doi: 10.1088/1748-3190/ab6fb8.
[56] J. Holland, "Adaptation in natural and artificial systems: an introductory analysis with application to biology," *Control and artificial intelligence*, 1975.
[57] P. Lakshminarayana and T. SureshKumar, "Automatic generation and optimization of test case using hybrid cuckoo search and bee colony algorithm," *Journal of Intelligent Systems*, vol. 30, no. 1, pp. 59-72, 2021.
[58] A. Pachauri and G. Srivastava, "Automated test data generation for branch testing using genetic algorithm: An improved approach using branch ordering, memory and elitism," *Journal of Systems and Software*, vol. 86, no. 5, pp. 1191-1208, 2013.
[59] W.-T. Tsai, D. Volovik, T. F. Keefe, and M. E. Fayad, "Automatic test case generation from relational algebra queries," in *Proceedings COMPSAC 88: The Twelfth Annual International Computer Software & Applications Conference*, 1988: IEEE, pp. 252-258.
[60] M. Alenezi, M. Akour, and H. A. Basit, "Exploring Software Security Test Generation Techniques: Challenges and Opportunities."
[61] I. Hermadi, C. Lokan, and R. Sarker, "Genetic algorithm based path testing: challenges and key parameters," in *2010 Second World Congress on Software Engineering*, 2010, vol. 2: IEEE, pp. 241-244.
[62] K. Lakhotia, P. McMinn, and M. Harman, "Automated test data generation for coverage: Haven't we solved this problem yet?," in *2009 Testing: Academic and Industrial Conference-Practice and Research Techniques*, 2009: IEEE, pp. 95-104.
[63] M. Gligoric, A. Groce, C. Zhang, R. Sharma, M. A. Alipour, and D. Marinov, "Comparing non-adequate test suites using coverage criteria," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, 2013, pp. 302-313.
[64] G. Fraser and A. Arcuri, "Evosuite: automatic test suite generation for object-oriented software," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 416-419.
[65] G. Fraser and A. Arcuri, "It is not the length that matters, it is how you control it," in *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, 2011: IEEE, pp. 150-159.
[66] M. R. K. Sepideh Kashefi Gargari, "General and technique-independent challenges of search-based software testing," presented at the The Second International Conference on Distributed Computing and High Performance Computing (DCHPC 2022), Qom, Iran, 2022.

[DOI: 10.52547/itrc.14.2.32]

Sepideh Kashefi Gargari received her B.Sc. degree in Information Technology from the Urmia University of Technology. She is currently pursuing the M.Sc. degree in Software Engineering at the Alzahra University, Tehran, Iran. Her research interests include Evolutionary Computation and Software Testing.

Mohammad Reza Keyvanpour is an Associate Professor at Alzahra University, Tehran, Iran. He received his B.Sc. degree in Software Engineering from Iran University of Science and Technology, Tehran, Iran. He received his M.Sc. and Ph.D. degrees in Software Engineering from Tarbiat Modares University, Tehran, Iran. His research interests include Image Retrieval and Data Mining.

[DOI: 10.52547/itrc.14.2.32]