Journal of Distributed Computing and Systems(JDCS), Vol. 4, Issue 2, 2021, Pages 40-45.

General and technique-independent challenges of Search-based software testing

Sepideh Kashefi Gargari*, Mohammadreza Keyvanpour
Department of Computer Engineering, Faculty of Engineering, Alzahra University, Tehran, Iran.

Article History:

Received: 13 November 2021
Received in revised form: 20 January 2022
Accepted: 15 February 2022
Available online: 17 March 2022

Abstract

In the development of a software system, different factors are involved. Due to the complexity of the work, the occurrence of human errors is inevitable. These errors cause the software to behave unpredictably and destructively in the real execution environment. For example, in some programming languages, there is an error called buffer overflow. It may cause unauthorized grants or access to the system in the network and security applications and even endanger user's privacy or can cause different problems in embedded systems. So software testing is required to increase the reliability and security of the developed software. There are several techniques for testing, among which, the search-based software test (SBST) and the automatic test case generation have a special place. The search-based software test is trying to convert the test problem into an optimization problem and achieve a better solution by searching the problem space. In addition to the benefits of a search-based test, there are challenges. This article provides a comprehensive overview of SBST problems. Understanding challenges is important because it helps scholars find solutions to these challenges or prepare new ideas that will ultimately help improve the quality of future tests and can bring greater security and confidence to the users.

Keywords: SBST, search-based software test, test case generation, security, challenges, problems.

I. INTRODUCTION

Software testing is an essential activity to improve software quality [1]. The test is a way to detect and minimize errors, reduce maintenance, and the total cost of the software [2]. In fact , it refers to the observation of the execution of a software system to validate whether it is operating in accordance with its purpose and to identify potential defects [3]. The interesting point about the test is that this process is difficult and expensive and does not add any efficiency to the software. However, due to the complexity, pervasiveness, and criticality of today's software, its value and importance are increasing more rapidly. Access to a comprehensive test program is not possible [4] because the range of input variables is too large [5]. The large input space makes infinite the number of test cases for a program, but test cases cannot be executed by limited budget and time. As a result, it is not possible to say with certainty that the software developed is faultless. Therefore, it can be concluded that first, the software test is required to ensure the accuracy of the software performance, and secondly, the process is costly. Also if we test the software, it is not possible to guarantee its performance with confidence. It means that errors are not unexpected after testing. The generation of test data is critical to find software failures [6]. .However, question is that among infinite numbers of the test data, which one is useful in the success of the test? Questions like these, push the tester to provide an automated way to produce test cases that have more potential to find faults. With the emphasis on the wide range of program variables, using the concept of searching in the problem space to find a better solution, instead of whole space or manual generation of test cases, has a special place. Search-based software testing is a popular automated testing technique [7] and based on metaheuristic algorithms, such as the genetic algorithm used to automate entire or a part of the test task [8]. In this paper, we try to address the search-based software testing and general challenges presented in it. In fact, we collect, categorize, and present the main problems and challenges of search-based software testing. It also acts as a road map and aids active scholars in the context of SBST to know and improve the challenges of this field and offer solutions to solve them.

The remainder of the paper is organized as follows: Section 2 reviews the related work. The definition of search-based testing is presented in Section 3. The challenges of this area, which is the main part of the article, are discussed in Section 4. In the end, the discussion and conclusion are presented in Sections 5 and 6.

II. RELATED WORK

Harman and Jones (2001) published an article claiming that a new field of research was emerging, called search-based software engineering [9]. SBSE is a subdivision of software engineering that has attracted much attention in the academic community [10]. In the next years, many articles have been published. For example Ref. [11] is a review of the search-based test for non-functional features. Articles [12, 13] provide a framework for generating test data for structural test, And Ref.

*Corresponding Author: Sepideh.kashefi1994@gmail.com

[14] is the description of a method, based on a search for functional testing. Ref. [15], the genetic algorithm has also been used to generate test cases that it is considered as the most important algorithms in this field. Regardless of the details of the published articles, [16-23] are among the most important articles in the search field. Ref. [3] is a review of test achievements and challenges, and [24, 25] are examples of addressing test challenges based on search. The articles mentioned have been selected in such a way that each of them covers a specific topic in the field of search-based testing.

III. SEARCH-BASED SOFTWARE TESTING

The purpose of software testing is to generate an optimal set of test cases/test data that measure software errors according to test adequacy criterion. A test adequacy criterion distinguishes good test cases from bad ones and determines whether the testing process has been finished [26]. In the previous section, it was said that in the software test generating test cases is an important step. A good test case should have a high quality in covering the test objective [4]. The search-based test is no an exception to the rule and tries to find test cases that meet the test criteria. The general idea behind all search-based software testing approaches is that a set of possible inputs for the program constitutes a search space and the search for the solution is carried out in this space. The test adequacy criterion is also encoded as a fitness function. A set of solutions (test cases) is generated by using a search technique from the search space, and the software/program under test is executed by the test cases generated. According to results obtained by the fitness function, the correctness or incorrectness of the software behavior is determined. Because of the complexity of software implementation, the application of search techniques for testing is promising [11]. These methods are very effective in solving software problems in complex and large search spaces [17]. In the search optimization community, there is a lot of interest in results that are human competition [16].

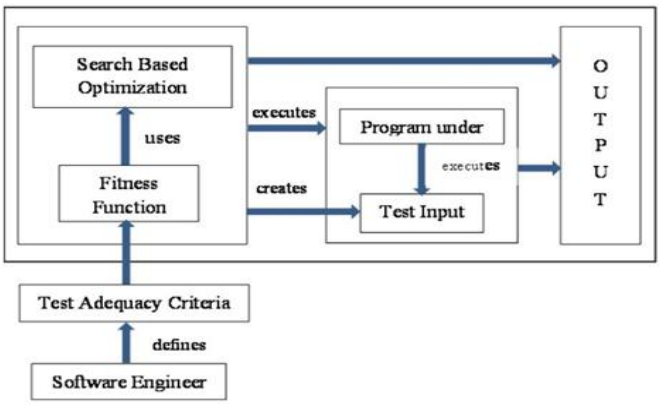

Fig. 1. General overview of search-based test input generation [26]

Figure 1 shows an overview of the generation of search-based test cases. According to figure 1, the software engineer tries to define the adequacy criteria for testing. The adequacy criterion is formulated in the form of a fitness function, and a search algorithm is used to find the solution, depending on the nature of the problem. The test cases generated by these algorithms are evaluated by the fitness function to cover the uncovered targets [27]. In the next step, considering the proportional value obtained in the previous stage, the search is directed to the test cases with greater relevance.

IV. GENERAL CHALLENGES IN SBST

In this section, we intend to have a review of the general challenges and independent of SBST method and classify them in two categories:

- Challenges of the main elements in the SBST process
- Peripheral challenges around SBST

Our main effort in this article is to enumerate some of the most fundamental challenges and categorize them. With the proposed categorization, we tried to provide a clear view of the general challenges to be a road map for future research.

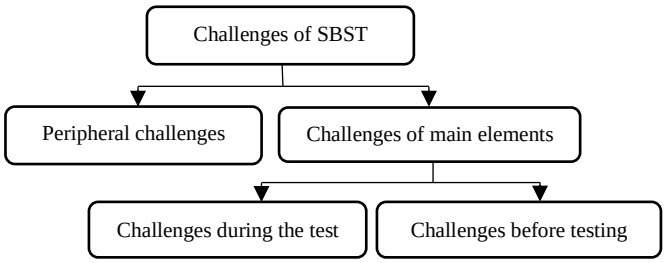

Fig. 2. Classification of SBST challenges

Figure 1 shows that the main elements of SBST include adequate criteria, fitness function, data generation, and so on. Each of these elements is associated with challenges that fall into the challenges of the main elements in the SBST process. Also, other public challenges are mentioned in the name of peripheral challenges. In the following subsections, we will explain in detail all the challenges. Figure 2 shows the classification of SBST challenges.

A. Challenges of the main elements in the SBST process

The main challenges are divided into two categories before and during the test. Pre-test challenges are problems that must be decided before the test. For example, what is the fitness function or when the test ends? But another category, like test execution, is about when we want to start testing.

1) Challenges before testing

a) Problem presentation

To accommodate the search methods to solve problems, one must adopt a method for encoding solutions to achieve them by the search [5]. In other words, choice solutions for the issue of interest should be coded so that they can be manipulated by the search algorithms [9, 16]. Selection of appropriate encoding ensures that similar solutions in the code space, are “neighbors” in the representation space [5]. But the problem is to formulate any new problem that we want to solve by searching, we need to formulate separately that it will be costly and time-consuming.

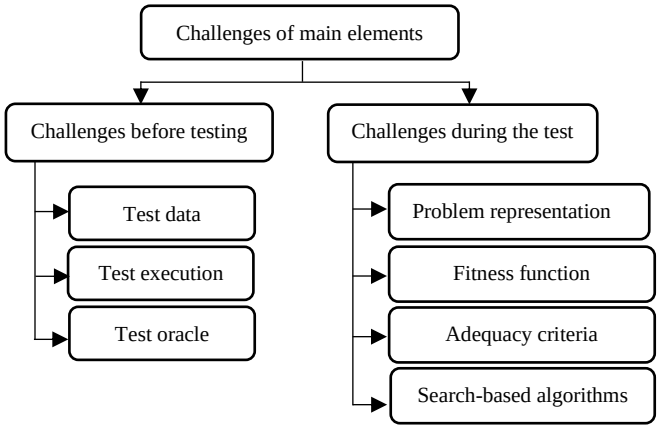

Fig. 3. Challenges of main elements in the SBST process

b) Fitness function

The fitness function guides the search space by evaluating candidate solutions. The fitness function is specific and should be redefined for a new problem [9, 16]. In other words, after coding, the solutions should be measured in terms of fitness. The creation of such a function is difficult to evaluate. Because both the fitness function and the coding of solutions are a dependent problem and should be designed separately. As an example [28], the following formula is used to evaluate each test case.

$$FT = \left(1 - \frac{NC}{APP}\right) + \left(\frac{SEP}{SEP+\beta}\right) \quad (1)$$

[28], presents a structural test method with the prime path coverage criterion using ant colony algorithm. Each of the parameters used in the formula (1) was dependent on the type of approach and criteria chosen by the experimenter. For example, NC is related to the number of targeted prime paths that exist in the traversed path and the number of all the prime paths to be covered is expressed by APP. These parameters are not fixed and changed according to the test properties. SEP is equivalent to the sum of the branch distances for the traversed path. β is a parameter for normalization and between 0 and 1.

c) Adequacy criteria

Officially, the adequacy of the test data is a stopping rule [29, 30]. Many algorithms require a measure to stop the search. The criteria for budget constraints and time constraints are among the usual criteria for this work. It is quite obvious that resources are limited for testing; but where the test and the cost spent are balanced? When can one be sure that the resources are not wasted on the test? Is the goal of the test possible with the cost spent on it? Were we going to have better results if we had more time and expense? There is no definite answer to any of the questions. Besides the ambiguities and uncertainties mentioned about the adequacy criteria, it is applied for many people at regular intervals during the search, so it must be specified at a low price, which is another challenge.

d) Search-based algorithms

Each of these algorithms has its own advantages and disadvantages. An algorithm can't work properly for all optimization problems. For each evolutionary algorithms, balance exploitation and exploration is important to achieve a globally optimal solution. Algorithms such as hill-climbing were faced with a local optimal problem and early convergence to these areas.

In the following other algorithms were proposed to be outsourced from local points. Evolutionary algorithms show potential, flexibility, and efficiency in a wide range of applications. But these algorithms demonstrate a suitable search for solving small or medium optimization problems and when used to solve large-scale optimization, they face serious challenges [32].

2) Challenges during the test

a) Test data

Another challenging issue in the test is related to the amount of data. If the test is to be comprehensive, all data should be considered and if part of the data is not considered, an error may remain undisclosed. So how to create a consistent, large, and beneficial test dataset is the fundamental challenge [33]. Data should be varied enough to be valuable to test, it must match all rules of reference integration and other data model constraints, and it is also possible to calculate the expected results of the tests [33]. During the test phase, the cost can be higher than expected due to the inadequate test cases. These inappropriate test cases waste organizational resources and it is necessary to minimize the cost to achieve an acceptable generation [34].

b) Test execution

Search algorithms such as hill climbing and genetic algorithms are random. Implementing such algorithms twice on a single issue can give different results. It is therefore very important to implement several times of this algorithm to collect the mean results and preventing the propagation of wrong results from a chance of success. However, the question arises, how many times should such implementation be repeated? According to a set of n repeated tests, is n large enough to have the right conclusion? Or should more tests be done [35]?

c) Test oracle

The problem of the oracle test has been well researched, but yet it is a fundamental challenge for SBST. While one can identify the failure of the system, the other Oracles are not as simple as that. Oracles for tools relying on source code (such as EvoSuite) are dependent on a person or something that is extracted from the source code (rather than specifications). In general, automation Oracle test is an open challenge [36].

A. Peripheral challenges around SBST

As mentioned, the second class of search-based test challenges relates to the peripheral problems, which we will explain in the following sections.

1) The challenge of automation

In previous years, many types of research have been done to automate the software test and search-based testing is one of the

methods proposed to automate the software test. But these efforts are limited by the size and complexity of the software. One reason for this limitation is that the use of dynamic memory allocation causes the software behavior to be highly unpredictable [26]. Figure 4 shows the classification of peripheral challenges around SBST.

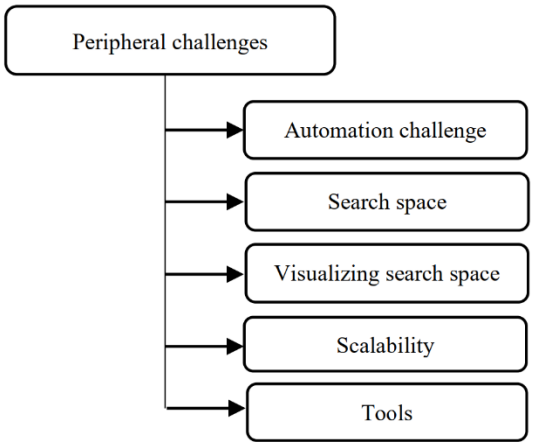

Fig. 4. classification of Peripheral challenges around SBST

3) Search space

The search-based optimization relies on the search space and the search is performed in this space. As we have mentioned range of input variables in the complex programs is large and the search in such space is expensive and time-consuming. So, if the initial search space is reduced or the search is done faster, it can be concluded earlier.

4) Visualizing the search space

In the community of search-based algorithms, it is usual to try to visualize the fitness landscape [37, 38]. The use of fitness function values as a measure of height in a landscape is a natural method where individuals are placed in a search space on a horizontal plane. However, most of the search problems include more than two components [16]. Thus mapping an individual from the search space to a n-dimensional plane is not simple. Of course, the problem is not just the visualization of the search space.

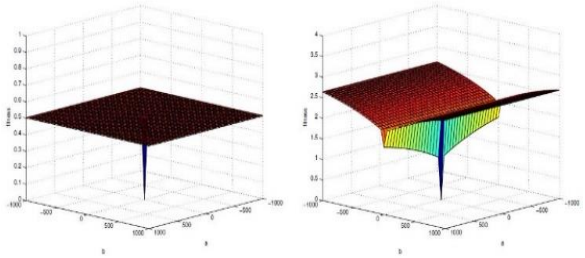

Fig. 5. converting the Search Landscape[39]

Furthermore, selecting the appropriate number of dimensions for search is another challenge. For example, as shown in figure 5 on the left, it is not possible to find the answer to the problem in two dimensions, and it is necessary to move the problem to the 3D space shown on the right of figure 5.

5) Scalability

One of the biggest problems facing software engineers is the scalability of the results. Many of the approaches that are attractive in the laboratory are not scalable. However, saying that a proposed solution must be scalable is easy but construct a scalable solution in many cases is not easy [16]. Although one of the attractions of search-based optimization models is that it naturally has parallelism [40]. However testing of large and complex software is difficult.

6) Tools

Some SBST tools such as EvoSuite are available for java and are currently in use. But some tools are out of date and are not used anymore or not available to the public. However, the lack of sufficient tools to generate automated test data can be expressed as a challenge. For example, in [41], it is stated that there is not sufficient tool for the C language. The standard industrial instrument for the generation of test cases is not at large scale and all of the studies conducted only on small projects have been reviewed and not applied to actual projects [42].

V. DISSCUSION

In this article, search-based software testing, the main concepts and general challenges in this area were discussed. Although reading this article gives a good overview of the general challenges, but it cannot be claimed that all the challenges in this area have been enumerated. As we know, there are different approaches to testing. Such as structural, functional, non-functional, etc. By choosing either of these approaches, the search-based test faces other challenges depending on the type of approach chosen. In general, it can be concluded that in addition to the many benefits that a search-based test provides, there are always problems and issues that we must overcome.

VI. CONCLUTION

SBST is a way of automating software testing which has attracted a lot of attention in recent years and a lot of progress has been made in this field. The main effort in this paper is to investigate some of the challenges and important issues in this field. It is hoped that this paper and proposed classification will provide a better insight into the search-based testing and help in future research.

REFRENCES

[1] J. Choma Neto, "Automatic support for the identification of infeasible testing requirements," in Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2020, pp. 587-591.

[2] S. Abdullahi et al., "Software Testing: Review on Tools, Techniques and Challenges," International Journal of Advanced Research in Technology and Innovation, vol. 2, no. 2, pp. 11-18, 2020.

[3] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in Future of Software Engineering (FOSE'07), 2007: IEEE, pp. 85-103.

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

[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] L. Joffe and D. J. Clark, "A Generative Neural Network Framework for Automated Software Testing," arXiv preprint arXiv:2006.16335, 2020.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] J.-C. Lin and P.-L. Yeh, "Using genetic algorithms for test case generation in path testing," in Proceedings of the Ninth Asian Test Symposium, 2000: IEEE, pp. 241-246.

[16] M. Harman, "The current state and future of search based software engineering," in Future of Software Engineering (FOSE'07), 2007: IEEE, pp. 342-357.

[17] 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.

[18] M. Harman, S. A. Mansouri, and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications," Department of Computer Science, King's College London, Tech. Rep. TR-09-03, p. 23, 2009.

[19] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," ACM Computing Surveys (CSUR), vol. 45, no. 1, pp. 1-61, 2012.

[20] S. Anand et al., "An orchestrated survey of methodologies for automated software test case generation," Journal of Systems and Software, vol. 86, no. 8, pp. 1978-2001, 2013.

[21] 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.

[22] J. Clarke et al., "Reformulating software engineering as a search problem," IEE Proceedings-software, vol. 150, no. 3, pp. 161-175, 2003.

[23] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," IEEE Transactions on Software Engineering, vol. 36, no. 6, pp. 742-762, 2009.

[24] 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.

[25] S. A. Abdallah, "Challenges and Proposed Solutions of Coverage Based Testing Tools," 2015.

[26] 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.

[27] Y. Lin, J. Sun, G. Fraser, Z. Xiu, T. Liu, and J. S. Dong, "Recovering fitness gradients for interprocedural Boolean flags in search-based testing," in Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2020, pp. 440-451.

[28] 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.

[29] D. Gotterbarn, K. Miller, and S. Rogerson, "Computer society and ACM approve software engineering code of ethics," Computer, vol. 32, no. 10, pp. 84-88, 1999.

[30] J. R. Speed, "What do you mean I can't call myself a software engineer?," IEEE software, vol. 16, no. 6, pp. 45-50, 1999.

[31] 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," Bioinspiration & Biomimetics, vol. 15, no. 3, p. 036004, 2020.

[32] S. Kumar, " Automatic Test Data Generation Using Evolutionary," Thesis 2017.

[33] J. Reviewer-Herzog, "Software Architecture in Practice Third Edition Written by Len Bass, Paul Clements, Rick Kazman," ACM SIGSOFT Software Engineering Notes, vol. 40, no. 1, pp. 51-52, 2015.

[34] A. Sharma, A. Jadhav, P. R. Srivastava, and R. Goyal, "Test cost optimization using tabu search," Journal of Software Engineering and Applications, vol. 3, no. 05, p. 477, 2010.

[35] A. Arcuri, "Evaluating search-based techniques with statistical tests," in Proceedings of the 11th International Workshop on Search-Based Software Testing, 2018, pp. 21-21.

[36] M. B. Cohen, "The maturation of search-based software testing: successes and challenges," in 2019 IEEE/ACM 12th International Workshop on Search-Based Software Testing (SBST), 2019: IEEE, pp. 13-14.

[37] Y.-H. Kim and B.-R. Moon, "Visualization of the fitness landscape, A steady-state genetic search, and schema traces," in Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, 2002, pp. 686-686.

[38] H. Pohlheim, "Visualization of evolutionary algorithms-set of standard techniques and multidimensional visualization," in Proceedings of the Genetic and Evolutionary Computation Conference, 1999, vol. 1: San Francisco, CA., pp. 533-540.

[39] P. McMinn, M. Harman, D. Binkley, and P. Tonella, "The species per path approach to searchbased test data generation," in Proceedings of the 2006 international symposium on Software testing and analysis, 2006, pp. 13-24.

[40] K. Mahdavi, M. Harman, and R. M. Hierons, "A multiple hill climbing approach to software module clustering," in International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings., 2003: IEEE, pp. 315-324.

[41] S. Scalabrino et al., "OCELOT: a search-based test-data generation tool for C," in Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018, pp. 868-871.

[42] M. Dave and R. Agrawal, "Search based techniques and mutation analysis in automatic test case generation: A survey," in 2015 IEEE International Advance Computing Conference (IACC), 2015: IEEE, pp. 795-799.

How to cite: S.Kashefi Gargari, M.Keyvanpour. General and technique-independent challenges of Search-based software testing, Journal of Distributed Computing and Systems(JDCS), Vol 4, Issue 2, Pages 40-45, 2021.