Goal-Oriented Testing for Pointer Data Type

ABDALLAH ALHAMEEDYEEN
Department of Computer Science
University of Jordan
Amman
JORDAN
Abdallah.alhameedyeen@gmail.com

MOHAMMAD ALSHRAIDEH
Department of Computer Science
University of Jordan
Amman,
JORDAN
mshridah@ju.edu.jo

HAZEM HIARY
Department of Computer Science
University of Jordan
Amman,
JORDAN
hazemh@ju.edu.jo

Abstract: - Software testing is an important phase in software development. Faults can cause serious and costly problems if they are neglected in software development, such as programs used in the fields of medicine, aviation, and military operations. A genetic algorithm (GA) is an evolutionary algorithm that can help to generate test data very quickly and accurately, generating test cases that fit the software under test. in this research, we generate test data for software that contains pointers using GA where these test data are valid for the software regardless of which path to use. The results of the experiments demonstrate that the Genetic Algorithm gives good results once used as test data generators to test pointer data type; such that the test target in all programs under test is reached which means that the percentage of the coverage was (100 %). Also, it shows the effect of using pointers in the source code, where the results were less in terms of execution time and the6 same in terms of the number of generations for a program that does not contain pointers than the same program which contains pointers.

Key-Words: -Software testing (SWT), genetic algorithm (GA), pointers, automated software test data generation, metaheuristic search.

1 Introduction

The production of reliable programs on a large scale is one of the fundamental requirements in the application of computers to the difficult challenges of the present.
In a software development project, bugs can happen at any stage of development. The final code, in addition to the faults that arise during coding activities, is likely to contain some requirements and design errors that need to be fixed to ensure the quality of the final software.

Software testing is one of the main methods in practice to increase the programmer's confidence in the reliability and correctness of the finished software. It may happen that the incorrectly tested program runs perfectly for a while before some input data reveals the presence of problems. Therefore, it is necessary to test software in a professional manner so that any error can be detected during runtime and corrected before delivery. Software testing is a process in which a program is executed with the objective of finding errors. The goal is to detect the bugs in the program by running the software with some inputs and evaluating the quality of the program activity and outputs against the desired outputs. Existing test data that achieves high code coverage provides high confidence in the reliability of the program under test [1].
The most important analytical quality assurance measure for software is testing. Often more than 50 % of the total development resources are used for testing without contributing to the functionality of the product [2]. The critical activity for test quality is test case design to cover structural test coverage criteria such as branch coverage. Manual generation of test data is laborious and costs time and resources. Nevertheless, this activity is fundamental and unavoidable for any organization , as a sufficient level of testing is increasingly required or recommended by internationally recognized standards for security and quality assurance. Systematic and automated testing is required to subsequently increase the efficiency and effectiveness of testing and reduce the overall cost of software system development. Automated software

testing can significantly reduce the cost of software development and test runs would be much faster [3].

In general, searching for an input datum in a search space (domain/set) of possible input datum is treated as an optimization and improvement problem. Therefore, we use the genetic algorithm as an optimization technique to test a program that contains pointer data types by having GA automatically generate test data that satisfy a certain objective within these programs.

This paper investigates and evaluates the performance of GA for generating test data for a variable of data type Pointer in different cases (i.e. multiple programs with different structures). The results show that the proposed algorithm has high performance in terms of coverage and runtime.
First, an overview of the current research on test data generation in software testing is given. Then, the proposed heuristics-based algorithms are presented. Subsequently, the experimental setup and the programs used for testing under the proposed algorithms are presented and evaluated. Finally, new insights are gained and future recommendations for action are derived.

2 Related Work
Software testing is considered one of the most critical phases in the software development life cycle and the efficiency of a software test is directly related to the code coverage. The degree of code coverage is strongly influenced by the test data, which is why the provision of efficient techniques for the automatic generation of test data is a significant aspect .Despite the sizable literature on software quality assurance techniques, testing remains one of the most widely practiced and studied approaches for assessing and improving software quality [24]. There are many types of software testing techniques; each act toward a different purpose. These techniques can be categorized into black-box testing (functional) [15], white–box testing (structural), and grey-box testing as in [16-18], or into random and dynamic test data generation as in[19, 20, 23]. Dynamic test data generation has been a popular way for generating test cases based on the execution of a specific program to obtain the information needed to build an acceptable test case. In black-box testing, known as functional testing, test cases are built only on functional requirements of the system under test without taking into account the internal structure of the program[15]. The goal of this kind of testing is to notice when the program's input/output behaving in disagreement with its specification. Black box testing is inexpensive, and no implementation knowledge is required. White-box testing considers the internal structure of the program. The structure of the software is examined by executing the code. In this type of testing, a tester must have full knowledge about source code [21]; Grey-Box testing integrate both white box and black-box testing. The program in this technique is tested with poor knowledge of the internal structure as well as a basic understanding of the system [4].

Random test data generation involves randomly selecting test data until suitable data is found. This is a simple method that only searches the search space by randomly selecting solutions and evaluating their suitability. Although it is a relatively unspecific strategy, it can be implemented with little effort [21].

Hill Climbing is a well-known local search algorithm where a solution is improved, with a beginning point, a randomly picked initial solution from the search space, this solution's surroundings are explored and evaluated. Once a more suitable solution is identified, it replaces the current solution. The neighborhood of the new solution is then investigated. If a better solution is found, the current solution is replaced again, until no improved neighbors can be found for the current solution. Hill climbing is simple and yields quick results. However, the search can easily lead to suboptimal results if hill climbing leads to a solution that is locally but not globally optimal [5].
Simulated Annealing (SA) is premised on the idea of the chemical process of annealing technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce its defects. The structural properties of the cooled solid depend on the rate of cooling [6]. SA extends Hill Climbing such that it accepts poor solutions with low probability. SA allows for more freedom of movement inside the search space. As the search advances, the chance of accepting an inferior solution (p) changes, and is expressed as [6,7] :

$$p=e-\delta/t \qquad (1)$$

Where t is a control parameter known as the temperature, and δ indicates the difference in objective value between the current solution and the neighboring inferior solution being considered. a cooling schedule is used to regulate the temperature. The temperature is first set high to allow unfettered mobility about the search space and to eliminate reliance on the starting solution. The temperature decreases as the search goes. If cooling

occurs too quickly, however, not enough of the search space will be examined, increasing the possibilities of the search becoming stuck in local optima.
Swarm Intelligence it is simulating the natural phenomenon of bird flocking or fish schooling. The particle swarm optimization algorithm starts with a population of random potential solutions, which can be considered as particles, each particle is given a random velocity and transported through the problem space iteratively. It is drawn to the site where the particle has achieved the best fitness so far, as well as the location where the best fitness has been attained across the entire population [8].

3 Proposed Methodology

In order to create test data to test code which contains pointer data type, GA was used; more specifically, this section describes fitness function design, structural testing criteria, pointer data type, hardware and system software, and finally, optimization tool in MATLAB [14].
In terms of fitness function design, the fitness value for an individual (test data) is computed using the number of levels covered by the individual b local distance at the control dependent node where control flow diverging away from the target node [10,11]. Therefore, the fitness function consists of two main components as follows:

3.1 Approximation level calculation (AL)

Considering Equation 2 which presents AL calculation [10,11], where Dn is the number of dependent nodes and En is the number of executed nodes. Dependent nodes present the number of control dependent nodes for the target node, and executed nodes present the number of control dependent nodes successfully executed in the manner we want.

$$AL = (Dn - En) - 1 \qquad (2)$$

3.2 Local distance calculation (LD)

Referring to Equation 3 presents the local distance (LD) calculation, and Equation 4 which presents the normalized PD, respectively [10,11] and [12]. Further, the PD will be computed according to Korel's distance function in Table 1. The normalized PD is between 0 and 1. Also, Equation 5 presents the final fitness function (FFF).

$$LD = Normalize\ (PD) \qquad (3)$$
$$Normalize\ (PD) = 1 - 1.001\text{-}PD \qquad (4)$$

$$FFF= AL + LD \qquad (5)$$

In terms of structural testing design, node–oriented criteria we will use in this paper to cover branches for if statement, and controlling number of iteration for while statement. The branch criteria goal is to cover specific nodes of the control flow graph (CFG). Considering every possible result of all decisions or branches to be covered at least once; means that all control flows are executed. It implicitly means statement coverage, since every statement is reached if all branches in a program are executed once [13].

Regarding the pointer data type, this study will test code that contains pointers; more precisely, pointers to an integer are thoroughly tested within the if statement and while statement, so we will consider several cases as follows:

IF – conditional statement: IF statement, IF-ELSE, and IF nested in multiple levels will be tested.
WHILE – statement: where the test target is finding out a test data such that the total number of iterations executed equals the total number of expected iterations, this is used in WHILE and nested WHILE.
The combination between WHILE and IF statement: the test target exits in a while – statement and also in if – statement together.
In terms of hardware and system software, the experiments were implemented using an integer vector. The characteristics of the device are presented as follows: Windows 10 Pro as an operating system, 2.8GHz core-i7 CPU, 8GB RAM, and the test programs are implemented in MATLAB version R2018a.
In terms of optimization tool in MATLAB, it is a toolbox used to implement a variety of algorithms (solvers) in MATLAB, which uses its matrix functions to build a set of versatile tools for implementing a wide range of solver methods, in our case, it will be GA [14].

Table 1 The Korel's Distance Functions [22]

Branch Predicate	**Branch Function**
A = B	**ABS(A-B)**
A ≠ B	**K-ABS(A-B)**

A < B	**(A-B) + K**
A ≤ B	**(A-B)**
A > B	**(B – A)+K**
A ≥ B	**(B – A)**
X OR Y	**MIN(Distance(X), Distance(Y))**
X AND Y	**MAX(Distance(X), Distance(Y))**

4 Experimental Results

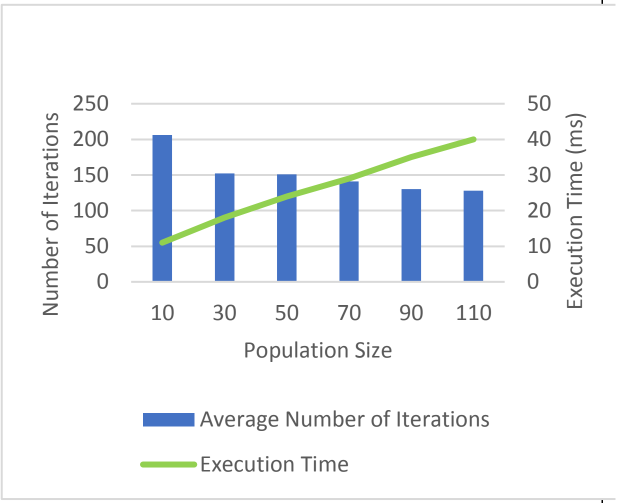
Considering the experiment design, six C++ programs were selected as benchmarks, Table 2. below presents a description of programs under test, to evaluate the performance of GA as a test data generator in order to test code that contains pointers, each program has its unique set of attributes. For each program, corresponding CFG is constructed and the test target is selected. Then, an instrumented version is developed of the considered programs in MATLAB.

Table 2 Programs under Test

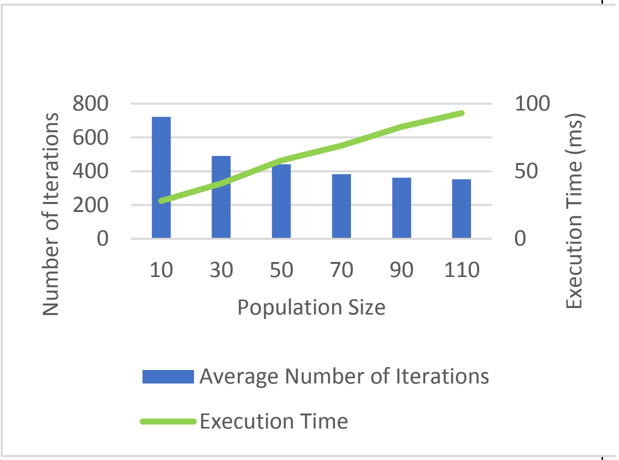
Program name	Description
Exception	A single if-statement in which the condition (predicate) is a simple predicate. The target exits inside the if-statement.
Evaluation:	A single if-else-statement in which the condition is a simple predicate. The else-part contains the target.
Bonus	Two while-statements which are placed within a if-statement where all of the conditions are straightforward predicates
Checkclass	A single if-statement inside a while-statement where all of the conditions are straightforward predicates. The test target exits inside the if-statement.
Minmax	One if-statement and while-statement contain two sequential if-statements. In which all the conditions are simple/primitive. The target is see whether the number of executed iterations equal to the expected iterations number or not.
Triangle	Three sequential if-statements and three nested if-statements in which all the conditions are compound predicates. The target exits inside the second nested if-statement.

Considering evaluation parameters, the GA's performance when used to create test data automatically was evaluated by the complexity that is represented by the amount of time, it takes to generate a test case as well as the average number of required iterations and the accuracy represented by branch coverage. The averaging is carried out after the algorithm has been executed 20 times and the values are calculated for each and every program unit. Each simulation trial is performed six times, wherein a new population size each time is selected to be either 10, 30, 50, 70, 90, or 110. After each execution, in addition to the coverage percentage, we recorded the required average number of iterations and average execution time.. It is required from the GA algorithm to guarantee a high branch coverage (accuracy) favourably a 100%, and to have the minimum average iterations as well as the shortest execution time to maintain a complexity as low as possible.
Figure 1 shows the resulting number of iterations and execution time, respectively, both measured against various population sizes. We can easily see the proportional effect of increasing the population size on the execution time and its advantage when it comes to improving the number of required

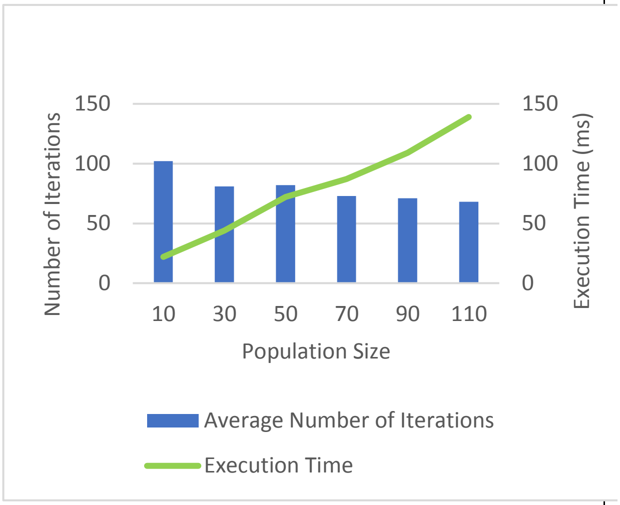
iterations, that because if the population size has a few chromosomes, the genetic algorithm has a few possibilities to perform crossover and only a small part of the search space is explored. On the other hand, if the population size is too high, a huge part of the search space is explored, so more time is needed to find the solution.



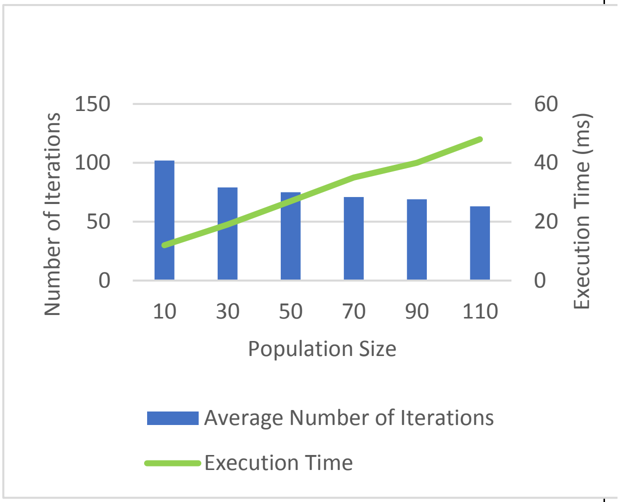
(a) Exception Program



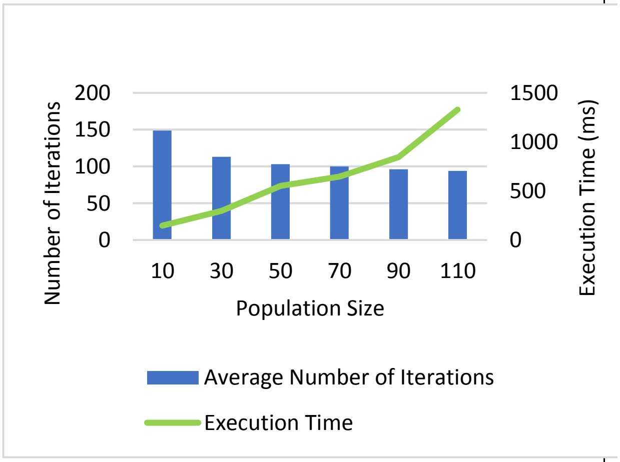
(b) Evaluation Program



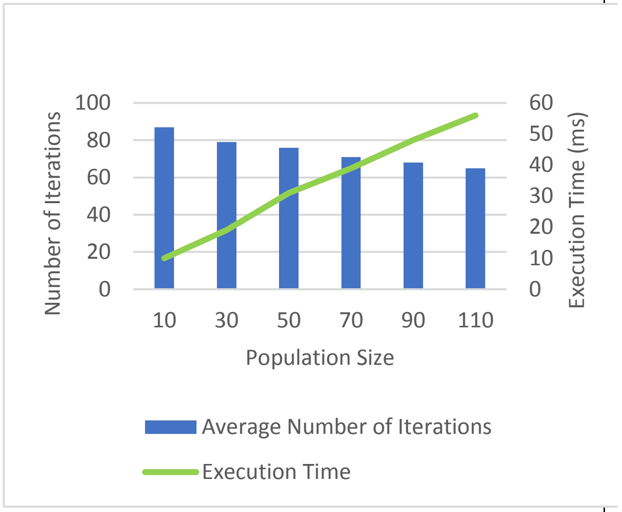
(c) Triangle Program



(d) Bouns Program



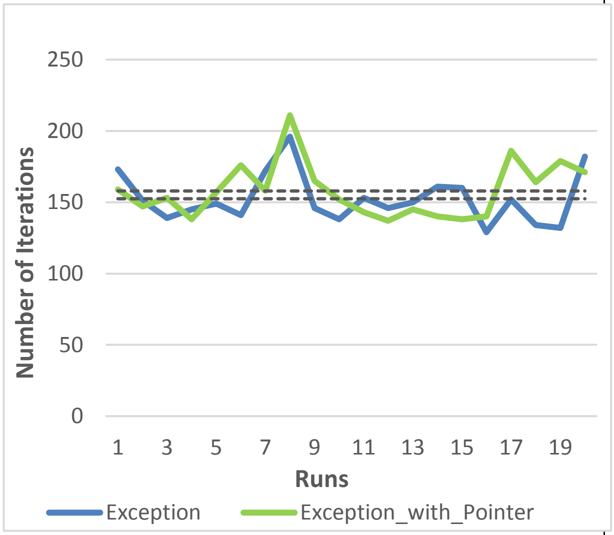
(e) check class Program



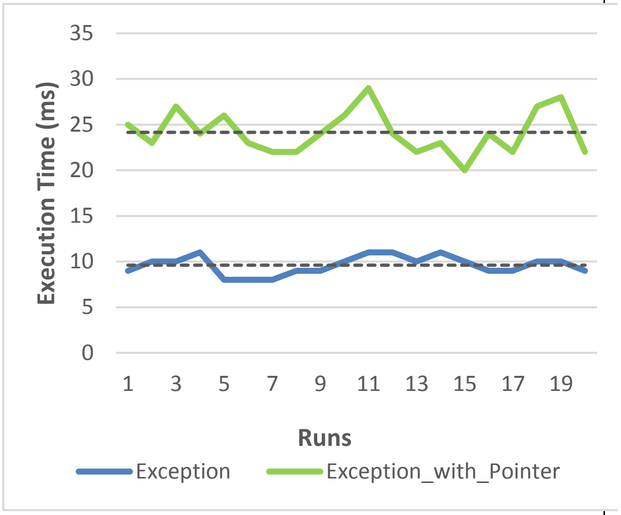
(f) Minmax Program

Fig.1 Average No. of Iterations GA.

Figure 2 shows the behavior of GA from generation to generation for the Exception program in two cases with and without pointers included in the source code. As we can see, the presence of a pointer does not affect the number of iterations needed to reach the solution, while it increases the execution time.



(a) GA Average Number of Iterations for Exception Program with and without pointers in source code



(b) GA Execution Time for Exception Program with and without pointers in source code

Fig.2 Average No. of Iterations and Execution Time for GA Code with pointers (solid) and Code with pointers (dotted)

5 Conclusion and Further Research

In this paper, GA is introduced as a search technique to determine the required test data by branching criteria to test the pointer data type. The experimental results show that this was achieved in such a way that the test objective is met in all the programs under test. This means that the coverage rate is (100%). Furthermore, when the GA uses an integer vector to search for the input domain for the required test sets among several test programs, it gives good results; the number of generations was an average number for all crossover types (single-point, double-point) because the number of possible crossover points integer vector is too small.

Experiments have shown that the single point and double-point crossover in the case of the integer vector give similar results according to the generations' average number in all programs under test. Moreover, the comparison between a program that contains pointers and the same program without pointers shows that the presence of pointers does not affect the number of generations, but it affects the execution time. Further, The fundamental advantage of utilizing the GA as a search technique is its power; because it starts the search from a crowd of points rather than a single point, the chances of being trapped at a local optimum are reduced. Besides, it can also be used for wide range of optimization problems, it provides a good technique for multi-modal problems because GA returns a suite of solutions.

Our future work includes enhancing the fitness function that guides the GA. Also, attempting to develop new operators such as new crossover operators make the search and optimization process easier and faster.
Finally, the execution of pointers for complex types such as linked list will be considered.

References:

[1] Hailpern B. and Santhanam P. (2002), Software debugging, testing and verification,IBM Systems Journal, Volume 41, Number 1, Pages 4-12.

[2] Beizer B. (1990), Software Testing Techniques, 2nd edition, London: Thomson Computer Press.

[3] Korel B. (1992), Dynamic method for software test data generation. Software Testing, Verification, and Reliability, Volume 2, Number 4, Pages 203-213.

[4] Duran J. and Ntafos S. (1984), An Evaluation of Random Testing, IEEE Transactions on Software Engineering, Volume 10, Number 4, Pages 438-444.

[5] Harman M. and McMinn P. (2007), A theoretical &empirical analysis of evolutionary testing and hill-climbing for structural test data generation, Proceedings of the 2007 international symposium on Software testing and analysis, London, United Kingdom, July 9-12,Pages 73-83.

[6] McMinn P. (2004),Search-based Software Test Data Generation: a Survey, Software Testing, Verification & Reliability, Volume 14, Number 2, Pages 105-156.

[7] Kirkpatrick S., Gellat C. D. and Vecchi M. P. (1983), Optimization by Simulated Annealing, Science, Volume 220, Number 4598, Pages 671-680.

[8] Trelea, I. C. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. Information processing letters, Volume 85, Number 6, 31 March 2003, Pages 317-325

[9] Mitchell M. (1996), An Introduction to Genetic Algorithms, Cambridge, London: Massachusetts Institute of Technology, 1st edition.

[10] Wegener J., Baresel A. and Sthamer H. (2001), Evolutionary test environment for automatic structural testing, Information and Software Technology, Volume 43, Number 14, Pages 841-854.

[11] Wegener J., Buhr K. and Pohlheim H. (2002), Automatic test data generation for structural testing of embedded software systemsby evolutionary testing, In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), New York, USA, July 9-13, Pages 1233-1240.

[12] Wang Y., Bai Z., Du W, Qin Y.and Liu X.(2008), Fitness calculation approach for the switch-case construct in evolutionary testing,Proceedings of the 10th annual conference on Genetic and evolutionary computation, Atlanta, Georgia, USA, July 12-16, Pages 1767-1774.

[13] Bottaci L. (2005), Use of branch cost functions to diversify the search for test data, Proceedings of the UK Software Testing Workshop, University of Sheffield, UK, September 5-6, Pages 151-163.

[14] The MathWorks (2009),Optimization with MATLAB and the Genetic Algorithm and Direct Search Toolbox, Version 7.4 (R2007a),

[15] Almasri, N; Tahat, L.; Alshraideh, M., (2016) Maintenance-Oriented Classifications of EFSM Transitions.Journal of Software 11(1), pp. 64-79.

[16] Alshraideh, M., Mahafzah, B., Al-Sharaeh, S. (2012).A multiple-population genetic algorithm for branch coverage test data generation. Software Quality Journal, 19(3), pp. 489-513.

[17] Alshraideh, M. .(2008), A complete automation of unit testing for javascript programs. Journal of Computer Science, 4(12), pp. 1012-1027.

[18] Alshraideh, M., Bottaci, L., Mahafzah, B. (2010). Using program data-state scarcity to guide automatic test data generation. Software Quality Journal, 18(1), pp. 109-144.

[19] Boyapati, C., Khurshid, S., & Marinov, D. (2002, July). Korat:Automated testing based on Java predicates. In ACM SIGSOFT Software Engineering Notes (Vol. 27, No. 4, pp. 123-133).

[20] Hashim J., Alshraideh, M., Mahafzah, B. (2013). Branch coverage testing using theanti-random technique. Journal on Software Engineering, Oct-Dec.

[21] Khan, M. E., & Khan, F. (2012). A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. Editorial Preface, 3(6),12-15.

[22] Korel B. (1990). Automated Software Test Data Generation, IEEE Transactions on Software Engineering, 16(8), 870-879.

[23] Allawi H., Al Manaseer W., Al Shraideh M.. (2020).A greedy particle swarm optimization

(GPSO) algorithm for testing real-world smart card applications,International Journal on Software Tools for Technology Transfer 22(2), 183- 194.

[24] Yenigün, H., Yilmaz, C., & Ulrich, A. (2016). Advances in test generation for testing software and systems.