Intelligent Automation And Soft Computing, 2020
Vol. 26, no. 4, 725–733
DOI: 10.32604/iasc.2020.010106

The Genetic Algorithm and Binary Search Technique in the Program Path Coverage for Improving Software Testing Using Big Data

Aysh Alhroob[1], Wael Alzyadat[2], Ayad Tareq Imam[1], Ghaith M. Jaradat[3]

[1]Faculty of Information Technology, Isra University, Amman, Jordan
[2]Faculty of Science and Information Technology, Al-Zaytoonah University, Amman, Jordan
[3]Faculty of Computer Science and Information Technology, Jerash University, Jerash, Jordan

ABSTRACT
Software program testing is the procedure of exercising a software component with a selected set of test cases as a way to discover defects and assess quality. Using software testing automation, especially the generating of testing data increases the effectiveness and efficiency of software testing as a whole. Instead of creating testing data from scratch, Big Data (BD) offers an important source of testing data. Although it is a good source, there is a need to select a proper set of testing data for the sake of selecting an optimal sub-domain input values from the BD. To refine the efficiency of software testing, this paper proposes a hybrid Genetic Algorithm and Binary Search (BSGA) technique that is used for detecting the error-prone path in a program. The BSGA combines the Genetic Algorithm (GA) with the Binary Search (BS) algorithm that uses the BD as input values for the program path coverage, and thus enhances the software testing. The BSGA represents a robust nonlinear search technique and a better quality solution, which therefore results in a cost reduction in the software testing industry. The experiments show that the results approved the impact of using the BS to enhance the performance of the GA, in terms of finding optimal test cases and test data for the input Big Data domain values. Whereas, these results minimize the cost of testing.

KEYWORDS: Genetic algorithm, binary search, big data, path coverage, test data, software testing.

1 INTRODUCTION

SOFTWARE Engineering (SE) is a systematic approach that maintains the engineering methods to restrain the process of software development and thus providing authentic work of software on the real environment. As software systems were developed, which is a continued tendency in the next decade, these systems are usually handling huge amounts of data or complex data that conformed with the definition of Big Data (BD). All industries including banking automobile, avionics, telecommunications, hospitals, pharmacy, oil and many other industries intended to produce BD and the implemented software of these systems are dealing with a huge amount of input data (Chen, et al., 2014).

Holding data, data investigates, data transfer, visualization, search, developing privacy of information and storage of data are the main hurdles in BD (Pethuru j, et al., 2015).

The significance of BD is that it is so huge to the point that it cannot be practically contained in any frame, spatially and temporally to attain high expectations. One of the BD characteristics is the volume, which deals with the scale of data. The BD's primary attribute is volume, which can be quantified by counting records, files, tables or transactions.

Thus, BD is viewed as a phenomenon that should be considered while developing a software system to avoid errors occurring while using the system in real life. BD can be of great assistance while developing a software system, especially in the software testing activity of the SE. Software Engineering encompasses an activities set, which are requirements of engineering, designs, implementation, testing and management (Alhroob, et al., 2018). One of the most

CONTACT Aysh Alhroob Aysh@iu.edu.jo

important activities is the software testing activity that affects the software development costs.

Software testing is a software development activity that is used to facilitate and develop the correction, completion, reliability, quality, and security of a developed software system. The possible cost-savings from controlling the software bugs at the stage of the software development cycle instead of other stages had been predicted at exactly forty billion dollars by the National Institute of Standards and Technology (NIST). The amount highlights that the present steps of testing are worthy, hence, removing errors and bugs from the software may be an essential job that results in a considerable income. Software testing automation, especially the generating of testing data increases the effectiveness and efficiency of software testing. Instead of creating testing data from scratch, BD offers an important source of testing data.

Although it is a good source, there is a need to select a proper set of testing data for the sake of selecting optimal sub-domain input values from the BD. This paper proposes a Binary Search Genetic Algorithm (BSGA) that makes use of the data flow dependencies to look for the data, which satisfies certain criteria among the most requested data.

This paper is organized as follows: Section 1 establishes the need for the GABS coverage of testing and to cover the research area. Section 2 shows the manifest of the state of the art in terms of the data flow analysis. Section 3 explains the data flow analysis. Section 4 shows the overall proposed approach in detail. Section 5 shows the implementation of the proposed algorithm using examples of small as well as larger precisions and analyses the results and finds the comparison by calculating the efficiency. Section 6, the conclusion and prospects of this algorithm are explained.

2 STATE OF THE ART

THE software testing design phase specifies the number of test cases that are based on the case of a test generating strategy. However, it is a simple view to estimate the number of test cases instead of calculating the test case number (Naik & Tripathy, 2008) (Chavarría-Báez, et al., 2013). This status achieved attention as old as the work of the demanded test cases of a system. A system that estimates the quantity by the developed tools in which this amount represents the quantity of the necessary path of the whole program to satisfy the criterion. Somehow, this work used the counting approach of the way that is possible and is generated by using the property of backtracking of the Prolog programming language (Bieman & Schultz, 1989).

An approach for maintaining the path of the test and estimating of the number of test cases was suggested by (Beizer, 1990) (Naik & Tripathy, 2008) (Chavarría-Báez, et al., 2013). In this approach, the program that is to be tested is to be divided into numbers and blocks, which are to be gone through while processing the program. This approach encompasses the following steps:

- Define the function points (FPs).
- Divide the type of test (Functional, Performance/Stress, Interoperability, and security) to be used.
- Map the function points to the testing types (FPs). For every FP, the test techniques are specified (BVAEP), Tables Decision, and State Transition.
- Show a maintained table, which contains a rough approximation of the number of test cases for every FP having its testing techniques (FP vs. Testing Techniques vs. estimated No. of Test Cases).
- Make a difference between the data of this table with the Historical Data.

Where every FP represents a node and the approach in sequence at the edge between these pairs of nodes (FPs). This approach creates many ways of testing, especially the reasonable cost of this software. Such situations required time to increase the performance of the testing. It has been confirmed that selecting the optimal path of the test can be a wise solution for decreasing the time of testing. While accepting the coverage of the percentage, selecting the best way to classify an optimization problem was done.

An example of the efforts for calculating the test is the method of synthesis, which utilizes the calculus refinement to define the abstraction rules that are used to calculate the later test case scenarios from the project's requirements (Rauf & A. Aleisa, 2015).

The Genetic Algorithm (GA) based method to select a set of test cases, is illustrated in Figure 1, and was proposed by (Whitten, 1998). This method uses the GA to select a set of test cases to be used in the testing of the software program.

This procedure showed implicit challenges that were revealed later by other researchers, which represents the Generalized Crossing (GC) procedure as a solution for optimizing the block sequences. This GC procedure, which was presented by (Chaudhary & Agrawal, 2015), is an iterative procedure of backtracking. The author of this task claimed that this procedure performed better than the GA based one.

Another modification to the GA based approach for generating testing data was combining the data flow dependencies with the GA to cover the location where a value for a variable is stored in memory (Def) and a location where a variable's value is accessed (Use) (*DU*) for the search data. This approach, which was proposed by (Girgis, 2005) yields new testing data from a previously generated testing data that were evaluated based on the effectiveness of the testing data and searched by the evolving new data generations by the GA. This approach is used for generating test data to the programs with /without procedures and loops.

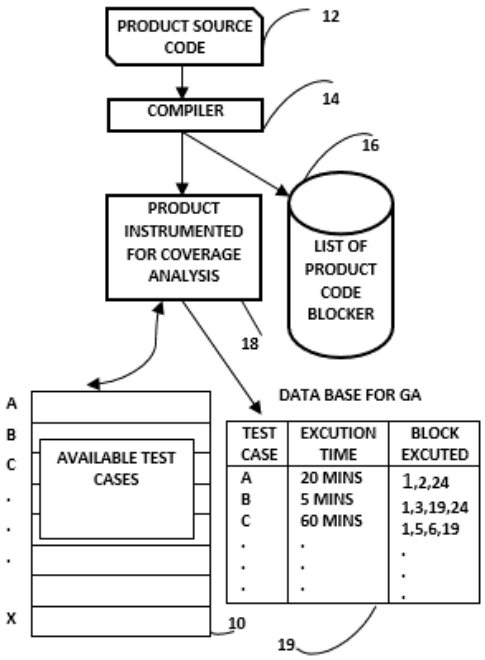

Figure 1. **The Genetic Algorithm (GA) based Method to Select a Set of Test Cases (Whitten, 1998)**

The GA is used to apply the sequential test method that evaluates the efficiency by the Mutation investigation (Odili, et al., 2018). Using the research case without the driver train, the proposed algorithm was being tested. The author suggested that for the investigation, the research could be used by the GA, still it may be used efficiently if there is a huge number of sequential possible investigations (Yousef, et al., 2015).

Authors (Jiang, et al., 2018) produced a novel evolutionary-based approach that achieved generating investigated data for all coverage of explanation-use. In the start, the DU path's subset, through which coverage adequacy can be ensured, is calculated by an algorithm of reduction for the whole path of the DU, then an investigated data is created by applying the GA for a subset path of the DU.

The earlier path based on the coverage-investigated data is created by the proposed achievement by (Mishra, et al., 2019), and onward this is applied to wrap all presented mutants for a particular program in an analysis. The applied method can enhance the effective testing by deleting the test redundant data gained from the testing path in the conditions of an improved score of mutation and matrix of the fault analysis, which is used to refine the residency data and identical mutants (Mishra, et al., 2019).

This work results in a procedure to minimize the number of GA generations, which is the BD related increasing time problem, which is the handling that is needed for performing the system analysis, particularly from the BD's investigated data values that are being derived.

3 THE DATA FLOW ANALYSIS

THE proposed technique labels all-use cases and therefore the data flow analysis procedure is considered as a solution. First, some descriptions are employed for labelling the system under testing. Early, some explanations are hired for the system labels under the testing.

The CF (control flow) usually focuses by mentioning a graph with a set of edges and a set of nodes. Every node focuses on a group of sequential statements that create a primary block. The CF, among the nodes, is being handover. A path is a sequential set of nodes paired by the edges. A whole path can be a path for the first node that is the start node and with the last node. Figure 3 represents the graph flow of the program instance, which is shown in Figure 2.

In a program analysis of the data flow paring among the variable references and definitions (Pradhan, et al., 2019), nodes are associated with the *DU*. Table 1 listed all *DU* for each Node or Edge derived from the example code in Figure 2. Also, this repeats the information in Figure 3, the information is simply repeated, but properly. Further, the same edges information is contained in Table 1.

```
    public static void computeStats (int [ ] numbers)
{
 int length = numbers.length;
 double med, var, sd, mean, sum, varsum;
 sum = 0.0;
      for (int i = 0; i < length; i++)
 { sum += numbers [ i ];
}
med = numbers [length / 2];
 mean = sum / (double) length;
 varsum = 0.0;
 for (int i = 0; i < length; i++)
 { varsum = varsum + ((numbers [ I ] - mean) * (numbers [ I ] - mean));
 }
 var = varsum / ( length - 1.0 );
 sd = Math.sqrt ( var );
 System.out.println ("length: " + length);
 System.out.println ("mean: " + mean);
System.out.println ("median: " + med);
System.out.println ("variance: " + var);
System.out.println ("standard deviation: " + sd);
}

}
```

Figure 2. **Example Program (Ammann & Offutt, 2016).**

The paths of the *DU* are listed for every CGF in Table 2. If a *DU*-pair has two ways or more for the same use, and on multiple rows, they are recorded with subways that end with the same nodes. Automatically done by these analyses. e.g. , [1, 2, 3, 5] (length) is essentially toured by any test that tours the du-path [1, 2, 3, 5, 6, 7], as [1, 2, 3, 5] and is a prefix of [1, 2, 3, 5, 6, 7]. That is why the path that is relative to a double path to the test path shows that the touring team does not reflect the sub-sequential table.

Since the prolonged *DU* route can be infeasible, the prefix is not. One must be a bit cautious with this optimization, and another path can cover and repeat the path of the Prefix. That is why unique paths are shown in Table 3 and the coverage of every variable path is represented.

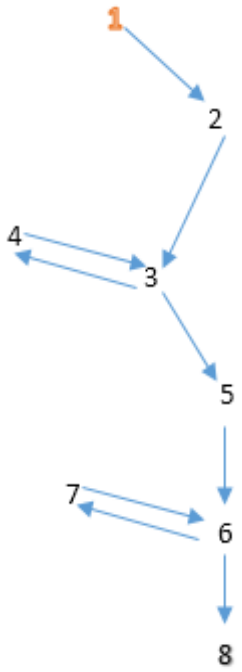

Figure 3. The Flow Graph of the Example Program.

4 THE OVERALL APPROACH

IN this work, the use of the BSGA input variables of the input numbers, paths of the *DU* and the domain and precision of every variable input are as achieve inputs. Further, the parameters of the GA are the maximum number of generations (max numb gen), the population size (pop size), mutation and crossover. The concerns with the test case set, are the set of the DU paths that are covered by every case test.

It is decided that the system of instrumentation and the paths of the *DU* was projected in (Girgis, 2005). The main steps showed in Figure 4 are the main steps that are accomplished to create the test cases for the big data input. This access represents two stages; select the required able sub-domain between the full measurements of the BD domain with the help of the binary search algorithm in this stage, which aims to minimizing the time of searching investigated research making a comparison with the full length of the big data input. The second one is to explain the paths of the *DU* as displayed in Table 3 and then the percentage of the coverage is concluded. The mutation operator of the investigated data and the *DU* crossover operators prove every chromosome coverage of the paths.

Table 1. The *Defs* and *Uses* at Each Node in the CFG (Control Flow Graph).

Node	Def	Use	Edge	Use
1	*Sum, numbers,* and *length*	*numbers*	(1, 2)	
2	*I*		(2, 3)	
3			(3, 4)	{ i, length }
4	*Sum* and *i*	*numbers, i, and sum*	(4, 3)	
5	*med,mean,varsum* and *i*	*length, numbers, and sum*	(3, 5)	{ i, length }
6			(5, 6)	
7	*Varsum* and *i*	*varsum, numbers, i and mean*	(6, 7)	{ i, length }
8	*Var* and *sd*	*varsum, var, mean, length, med, var and sd*	(7, 6)	
			(6, 8)	{ i, length }

4.1 Select Desirable Sub-Domain

A chromosome is a binary vector that represents the variable x for every input data. The domain range and precision are used to determine the vector size. Let us consider a program k, input $x_1 \ldots x_k$, and $D_i = [a_i b_i]$ represents each variable domain. To achieve such accuracy, every domain of every variable D_i should be into $(b_i - a_i) \cdot 10^{di}$ = size ranges.

A binary string of the length $m = \sum_{i=0}^{k} \mathrm{m}$; for every vector test data is shown, the first $m_1$bits map into a value from the range $[a_i, b_i]$ of variable x_i, the last group of m_k bits map into a value from the range of $[a_k, b_k]$ of variable x_k. Such as, a program has one input x, where $50 \leq x \leq 100$, and the precision is 3 decimal places. The x domain has a length of 50. The accurate range implies [50, 100] and should be divided into at least $50 \cdot 1000$ equal size ranges.

As an earlier part of the chromosome, $2^{18} < 50000 \leq 2^{19}$ needs 19 bits. Let us have two variables (x,y) and y for representation requires 15 bits. The final length of a chromosome is then m = 19+15=34 bits; code x will have the first 19 bits and code y will have the remaining 15 bits.

Table 2. The *DU*-paths for Each Variable of the CFG.

Variable	*DU* Pairs	*DU* Paths	Prefix?
numbers	(1,5) (1,4) (1,7)	[1→ 2→ 3→ 5] [1→ 2→ 3→ 4] [1→2→ 3→ 5→ 6→ 7]	Yes
Length	(1,5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))	[1→ 2→ 3→ 5] [1→ 2→ 3→ 5→ 6→ 8] [1→ 2→ 3→ 4] [1→ 2→ 3→ 5] [1→ 2→ 3→ 5→ 6→ 7] [1→ 2→ 3→ 5→ 6→ 8]	Yes Yes Yes Yes Yes
Med	(5, 8)	[5→ 6→ 8]	Yes
Var	(8,8)	-	
Sd	(8,8)	-	
Sum	(1,4) (1,5) (4,4) (4,5)	[1→ 2→ 3→ 4] [1→ 2→ 3→ 5] 4→ 3→ 4] [4→ 3→ 5]	Yes Yes
Mean	(5, 7) (5, 8)	[5→ 6→ 7] [5→ 6→ 8]	Yes Yes
Varsum	(5, 7) (5, 8) (7, 7) (7, 8)	[5→ 6→ 7] [5→ 6→ 8] [7→ 6→ 7] [7→ 6→ 8]	Yes Yes
I	(2, 4) (2, (3,4)) (2, (3,5)) (4, 4) (4, (3,4)) (4, (3,5)) (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))	[2→ 3→ 4] [2→ 3→ 4] [2→ 3→ 5] [4→ 3→ 4] [4→ 3→ 4] [4→ 3→ 5] [5→ 6→ 7] [5→ 6→ 7] [5→ 6→ 8] [7→ 6→ 7] [7→ 6→ 7] [7→ 6→ 8]	Yes Yes Yes Yes Yes Yes Yes

An output from a set of input variables (a chromosome) is generated by a cost function. In some appropriate fashion, the object is to manage the output. By finding an accurate value for the variables input, we do all this without understanding during the selecting limit of the marks of students. The (85-100) rank of excellence and other marks are being ignored and the excellent rank is (50-84). The difference is between the perfect marks limit and the actual student marks. If the variable of the input is the range among (84-100) and that having the highest marks, will be awarded. In this situation, the function's cost is the result of the experiments from the resolved limit than examination of the whole marks among (50-100). Therefore, we observe that finalizing the accurate cost of function and selection, which type of variables to are entirely relative to utilize. The fitness terms are broadly used to assign the output of the work targeted in the literature of the GA. For minimizing the variable's input, a proposed algorithm is used to choose the accurate input limits as displayed in these steps:

Table 3. The *DU* Unique Paths.

DU Paths
[1→ 2→ 3→ 4], [1→ 2→ 3→ 5→ 6→ 7]
[1→ 2→ 3→ 5→ 6→ 8]
[4→ 3→ 4], [4→ 3→ 5]
[7→ 6→ 7], [7→ 6→ 8]
[5→ 6→ 7], [5→ 6→ 8]
[2→ 3→ 4], [2→ 3→ 5]

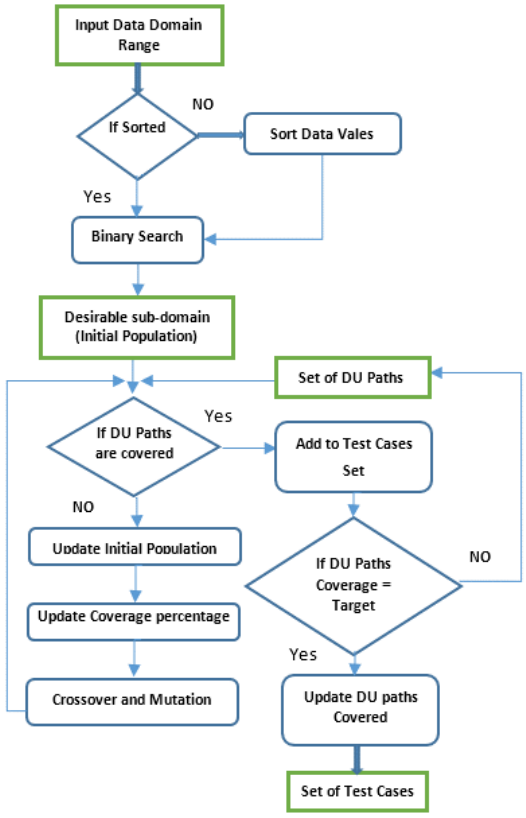

Figure 4. The BSGA Approach.

Step1: Enter variable range A.
Step2: Produce the values among the variable based type range.
Step3: Find Least Index L =0 # first Index.
Step4 Find Last Index R=n-1 # Last Index
While L ≠ R {
M= floor ((L+R)/2)
If C_f=0 { #where, Cf=P_w-E_w;
Inc=M
Return Inc
Inc++
Else If ΔC=0 {
dec=M
Return dec
Dec--
Else
L=M+1
}
}
If L=R and ΔC≠0 {
R=M-1
}

Where;
Pw= $\sum_{=1}^{n}$ Ewk
Ew= no.of covered Edges/Total number of Edges.

As in (Rylander, et al., 2001), the time of convergence is exponential and the linear of the different coverage with the length of a chromosome is done. The actual experiment in (Rylander, et al., 2001) had in an approximate convergence of *3*|b(x)|* (in the binary of *x*, 3 times the length). While the chromosome itself increased at *lg(x)*, this results in that the convergence is in *O (lg(x))*. As a Consequence, it is a small derived representation to make it possible, and we can maintain from a couple of experiments as documented, shows evidence that the complexity of the GA of a maximum of 1's is *€ O(lg(x))*. Further, if the size of the chromosome is not too large, the change in the length chromosome has the same generation numbers. However, using the large variables limit reasoning runtime exponential with the overpowering population probability sizes up to $\mu \leq n\ 1^{\delta-\varepsilon}$ for a few arbitrary small constant ε and problem size n (Oliveto & Witt, 2015). The applied BSGA minimizes the bits of a number that are represented to use in every chromosome variable as shown in the following example:

Assume that *a* and *b* are two variables used to choose the great investigated situation by choosing suitable values of input for everyone, where 0 ≤a ≤100, and 1 ≤b ≤4 are the needed accuracy, which is 4 decimal places for every variable. The variable's domain of *a* has the length of 100; the accurate need means that the range [0,100] should be separated into at least 100.10000 equal ranges of size. This implies that 20 bits are needed as a 1st chromosome's part. Although, variable *y* requires 16 bits for the 2nd chromosome's part. The chromosome's total length is 20+16=36.

Let us examine a parent chromosome example:

P1: 10000001101010000111000111000101111O

P2: 10011001100010000111111010100010101111

Assume the numbers *a's* sub-limit (e.g. 90-100) and numbers *b's* sub-limit (e.g. 3-4) are the suitable inputs. The GABS conclude that the chromosome is a minimized length and sub-range. Respectively, *a* variable requires 10.10000 (17 bits), and the *b* variable requires 1.10000 (14 bits) embedded of 100.10000 and 4.10000 respectively. Currently, the chromosome's total length is 17+14=31.

4.2 Using the GA to Select Optimal Test Cases

Later decreasing the BD domain's volume range in the initial sub-section, the Optimal Test Cases is used by the GA between all possible test studies that could be performed to reduce the period of the testing time. The coverage paths of the *DU* is an impact of choosing those investigating the study, and the percentage of the high coverage based as the target.

The mutation and crossover procedures are used to develop the chances of the path's coverage of the *DU*, which results improvement or non-reflected by the percentage of the coverage path of the *DU*. In this stage, each *TC* (test case) has been developed after the Mutation and Crossover has been calculated to be attached to the lists of test cases or not. The GA evaluates each fitness value TC by the program performing with the TC as the input and listing the paths of the *DU* in the program that are performed by this investigation study. Equation 1 evaluates the value of fitness TCw for each chromosome TCi (i = 1, …., pop_size):

$$\text{TCiw} = \frac{\text{No.DUpaths covered by TCi}}{\text{Total of DU paths}} \qquad (1)$$

The TC_w is the only return from the GA problem. An investigated study, which is indicated by chromosome TC_i which is an active reflection if TC_{iw}> 0.

5 THE EXPERIMENTAL RESULTS

IN the experiments, a group of ranges of the 3-variance domain for the values to be the input is being used as follows: From 1 to 10, and from 1 to 100, then from 1 to 1000 with two and four precisions. Figure 2 shows the efficient expected achievement is used to be demonstrated by the two variables, (*length* and *number*). While the other variables are under control by the input data form the *length* and *number*. The number of variables affects the GA numbers and generations of the BSGA exponentially. The used parameters of the GA were: *maxgen = 100.000*, *pc = 0.7* and *pm = 0.15*.

Figure 5 shows the number of GA generations is reduced by using the Binary Search as a purity technique to select a sub-domain of the original values of the BD domain range. In the Big Data, the domain length can be very lengthy. That is why the number of values of the input is considered, which proposes a high amount of generations of the GA to search the excellent Test Cases.

All experiment conclusions show the effect of the finalizing input sub-domain despite the coverage paths of the *DU*.

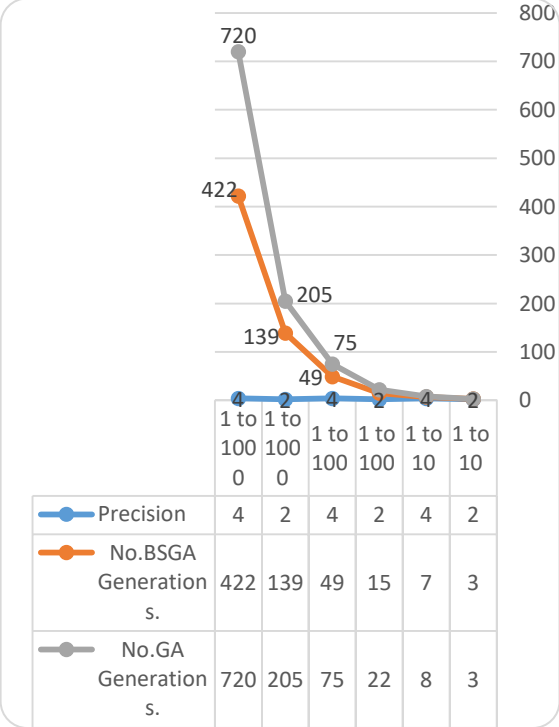

Figure 5. The Experiment Results.

Figure 6 shows the experiment results from the scale of the Logarithmic, because of the assumption of using huge values in the BD and return to the skewness near the lengthy values. The conclusion represents the percentage and gap between the two achievements, although the BSGA is a smaller amount of generation. Figure 5 shows the relationship among the chromosome generation number and the length as the perspective to the logarithm and the BSGA's importance in the usage to restrain the rising generation's number associated with the increasing length of a chromosome.

6 CONCLUSION

THE proposed BSGA is directed by the data flow dependencies in the code to obtain test data for all *DU* criterion. The proposed approach in this work accepts the BD input values, the domain range of the values, the program, and the *DU* paths to be covered. The BD Volume is one of the problems that is covered in this work in terms of testing. Accordingly, the cost of testing is minimized due to the timelessness of the testing.

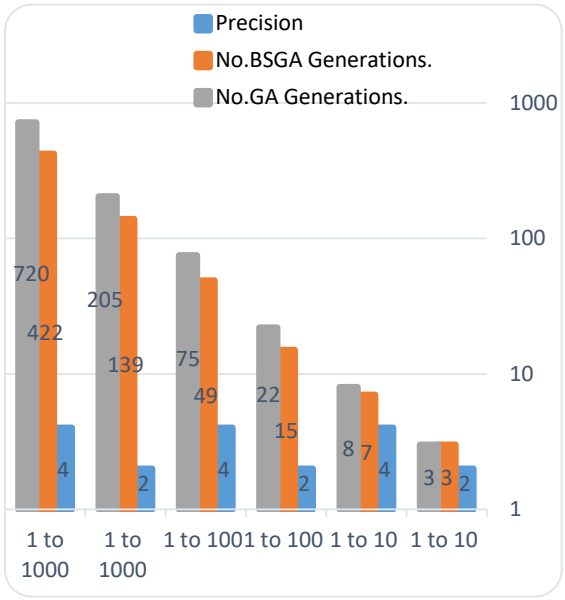

Figure 6. The Results Logarithmic Scale.

The results of the experiments showed that the proposed BSGA was effectively used to minimize the number of iterations compared to the GA to find the optimal test cases for the BD input values. On the other hand, the BSGA shows no large effect in terms of minimizing the number of the GA iterations while the data is not big.

7 REFERENCES

Alhroob, A., Imam, A. T. & Al-Heisa, R., 2018. The use of artificial neural networks for extracting actions and actors from requirements document. *Information and Software Technology*, Volume 101, pp. 1-15.

Ammann, P. & Offutt, J., 2016. *Introduction to software testing*. Cambridge: Cambridge University Press.

Beizer, B., 1990. *Software Testing Techniques*. NY, USA: Van Nostrand Reinhold.

Bieman, J. M. & Schultz, J. L., 1989. *Estimating the Number of Test Cases Required to Satisfy the All-du-paths Testing Criterion*. Key West, Florida, USA, ACM, pp. 179-186.

Chaudhary, R. & Agrawal, A. P., 2015. Regression Test Case Selection for MultiObjective Optimization Using Metaheuristics. *International Journal of Information Technology and Computer Science*, pp. 50-56.

Chavarría-Báez, L., Li, X. & Palma-Orozco, R., 2013. *Estimating the Number of Test Cases for Active*

Rule Validation. Mexico City, Mexico, Springer-Verlag Berlin Heidelberg, p. 120–131.

Chen, M., Mao, S. & Liu, Y., 2014. Big data: A survey. *Mobile networks and applications*, 19(2), pp. 171-209.

Girgis, M. R., 2005. Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm. *Journal of Universal Computer Science*, 11(6), pp. 898-915.

Jiang, S., Chen, J., Zhang, Y., Qian, J., Wang, R. & Xue, M., 2018. Evolutionary approach to generating test data for data flow test. *Software Engineering*, 12(4), pp. 318-323.

Mishra, D. B., Mishra, R., Acharya, A. A. & Das, K. N., 2019. Test Data Generation for Mutation Testing Using Genetic Algorithm. In: *Soft Computing for Problem Solving*. Singapore: Springer, pp. 857-867.

Naik, K. & Tripathy, P., 2008. *Software Testing and Quality Assurance: Theory and Practice*. Hoboken, New Jersey: Wiley.

Odili, J. B., Kahar, M. N. M., Noraziah, A., Zarina, M. & Haq, R Ul 2018. Performance Analyses of Nature-inspired Algorithms on the Traveling Salesman's Problems for Strategic Management. *Intelligent Automation & Soft Computing*, 24(4), pp. 759-769.

Oliveto, P. S. & Witt, C., 2015. Improved time complexity analysis of the simple genetic algorithm. *Theoretical Computer Science*, Volume 605, pp. 21-41.

Pethuru j, R., Anupama, R., Dhivya, N. & Siddhartha, D., 2015. *Big and Fast Data Analytics Yearning for High-Performance Computing*. NewYork: s.n.

Pradhan, S., Ray, M. & Patnaik, S., 2019. Coverage Criteria for State-Based Testing: A Systematic Review. *International Journal of Information Technology Project Management (IJITPM)*, 10(1), pp. 1-20.

Rauf, A. & A. Aleisa, E., 2015. PSO based automated test coverage analysis of event-driven systems. *Intelligent Automation & Soft Computing*, 21(4), pp. 491-502.

Rylander, B., Soule, T. & Foster, J., 2001. *Computational complexity, genetic programming, and implications*. Lake Como, Italy, Springer, pp. 348--360.

Whitten, T. G., 1998. *Method and computer program product for generating a computer program product test that includes an optimized set of computer program product test cases, and method for selecting same*. USA, Patent No. 5,805,795.

Yousef, N., Altarwaneh, H. & Alhroob, A., 2015. Best Test Cases Selection Approach Using Genetic Algorithm. *Computer and Information Science*, 8(1), pp. 1-15.

8 NOTES ON CONTRIBUTORS

Aysh M. Alhroob is an Associate Professor of software engineering at Isra University, Jordan. Aysh received his PhD from the University of Bradford, UK. 2010. Aysh joined Isra University in Jordan as an Assistant Professor in the Faculty of Information Technology, Software Engineering Department 2011. Dr Aysh has a number of published papers in various Computer Science and Software Engineering topics in international journals and conferences. In addition, Aysh has published his first book in software testing, 2010.

Wael Alzyadat is an Assistant Professor at the Software engineering department, Al-Zaytoonah University of Jordan. Wael received his PhD degree in Software Engineering from Universiti Putra Malaysia. Dr Wael published over 20 articles in international journals, the research interest includes several domains area.
Email: wael.alzyadat@ zuj.edu.jo

Ayad Tareq Imam is an Associate Prof. at Al-Isra University / Amman / Jordan. Dr Ayad received his PhD degree in Computer Science from De Montfort University, Leicester, UK in 2010. Dr Ayad has a number of published papers in various Computer Science and Software Engineering topics. Dr Ayad is a reviewer in a number of international journals and conferences of Computer and Information related areas.
Email: alzobaydi_ayad@iu.edu.jo

Ghaith M. Jaradat is an Associate Professor in the Department of Computer Science, Faculty of Computer Science and Information Technology, at Jerash University, Jordan. His research interests are mainly directed to Metaheuristics and Combinatorial Optimization Problems including Course and Exam Timetabling, Vehicle Routing, Travelling Salesman, Knapsack, and Nurse Rostering Problems. He received his bachelor's degree from the Computer Science Department at Jerash University in 2004. He received his master's degree from Intelligent Systems Department at Utara University in Malaysia in 2007.

He received his Ph.D. in Intelligent Research algorithms - Computer Science from the National University of Malaysia in April 2012. He has published a number of high-quality research papers in international journals and conferences.
Email: g.jaradat@jpu.edu.jo