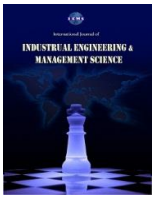
University of Hormozgan

International Journal of Industrial Engineering & Management Science

journal homepage: www.ijiems.com



A New Idea for Search Based Software Testing: Optimization of Test patterns by Using Local Meta-Heuristic Search

Seyed Vahab Shojaedini [a,*], Fahime Delkhoshian [b], Mohsen Khodadoust[c], Mohammad Momenian[d]

[a] *Department of Electrical Engineering and Information Technology, Iranian Research Organization for Science and Technology (IROST)*
[b] *Faculty of Engineering, Alzahra University, Tehran, Iran*
[c] *Department of Electrical, Biomedical and Mechatronics Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran*
[d] *Department of Computer Engineering, Faculty of Engineering, Islamic Azad University, E-Campus, Tehran, Iran*

ARTICLE INFO

Article history:
Received: 2023-12-09
Received in revised form: 2024-05-25
Accepted: 2024-09-01

Keywords:
Search based software testing
Test pattern generation
Local meta-heuristic search
Genetic algorithm
Memetic algorithm
Opposition based learning

ABSTRACT

Automated software testing is the most important method to ensure from reliability of the software during its performance. The most critical part of this procedure is to generate optimal test patterns. Search based software testing methods, try to generate the test patterns in such way that maximizes the detected errors in parallel with minimizing the time of test. In this paper a new search-based idea is introduced for generating optimized test patterns which is based on local meta-heuristic search. Performance of the proposed method is evaluated in parallel with some state of art algorithms in order to compare their effectiveness in generating optimal test patterns. The above comparisons were performed separately in two fast and slow performance classes. Fast performance class comparisons, show the superiority of the proposed scheme against its alternatives in such way that it improves pattern generation procedure maximally 73 percent and minimally 47 percent compared to alternatives. Such superiorities are also observed in slow performance class, by improving pattern generation procedure maximally 47 percent and minimally 27 percent.

1. Introduction

Software test is the evaluation process to ensure the appropriate working of software when confronted with numerous events that it may encounter throughout the application. Based on this fact, the software test is to discover its possible errors to ensure either correct or optimal performance of software [1]-[2].

The stability of the software against various events in this phase means its better performance in the real world which extends from the beginning to the end of the software life cycle. According to official reports, almost 40% of the software sales turnover is spent on software testing and security [3].

However, software tests become a challenging issue due to the growth of information and communication technology over the past decade in parallel with the production of several software applications. The slightest defect in the software testing procedure may lead to considerable quality reduction in software and consequently customer dissatisfaction and loss of market share. Over several years, manual error detection has been applied as conventional method for software test. This method is time consuming and monotonous, as well; the occurrence of human error is also probable. These items hamper the effectiveness of manual error detection in software test [4], accordingly the automated test procedures were substituted [5].

One of the most important issues that affects the accuracy and precision of automated procedure is to generate test patterns.

* Corresponding author.
E-mail address: shojaeddini_va@yahoo.com

These patterns should be along with a maximum diversity to detect maximum probable errors. On the other hand, these patterns should be as minimum possible number to minimize the test time. The above opposite conditions cause that test data production considered as an optimization problem [6].

The primary methods for automated production of test patterns were based on the randomized procedures. Despite the simplicity of these methods, they do not lead to patterns with maximum coverage [7].

Consequently, several techniques in Search Based Software Testing (i.e., SBST) domain has been introduced to accomplish the aims of the aforementioned process [8]. In some researches, several versions of Meta-Heuristic Algorithms based on the ants' behavior have been recommended to address the above problem. Unfortunately, these methods have not been able to achieve the maximum coverage due to the problem of falling in local minima [9]-[10]. In more sophisticated methods the range of patterns is derived from the problem's specifications; accordingly, this range is divided into several sub-bands and finally the test data is generated independently in each sub-range. These methods are called "Partition Testing Methods", and each of above sub-ranges may produce a group of patterns which lead to a specific type of error in software under test [11].

Although the results of several studies show the superiority of partition testing methods against basic random ones, but obtaining sub-bands is still a challenging problem in these techniques which hampers their effectiveness in real world applications. Genetic algorithm has been widely used as an effective idea in various researches to generate optimal software test patterns. In most of these methods, test patterns are modeled as chromosomes and the primary randomized population is converged to the final optimal patterns through standard cycle including parent selection, fitness calculation, crossover and mutation [12]-[15].

A study emphasizing the importance of software quality becoming an important challenge in software engineering processes and the fundamental role of software testing in its measurements was conducted with a review of 65 cases between 2015 and 2022 of meta-heuristic methods based on genetic algorithm, hybrid, based on ant colony optimization, based cuckoo search, based on firefly algorithm, based on artificial bee colony, and other classification meta-heuristic methods. As a result, the strengths and weaknesses of the software test were analyzed with the criteria of mutation score, complexity, and scalability [16].

While achieving optimal solutions includes the precise formulation of the optimization problem and choosing an appropriate approach is one of the main challenges of similar studies, one of the research projects emphasizing the advantages of hybrid search-based software testing (CSST) as an optimization method by presenting a method similar to Black Hole Optimization (BH) shows the superiority of the proposed method over the method Main BH and Particle Swarm Optimization (PSO) approach [17].

In order to determine the appropriate criteria for testing software projects such as code complexity, predicting the project completion date and calculating the efficiency of a research software developer focusing on the descriptive analysis of software algorithms based on machine learning to improve the experience of end users and software quality experts [18].

In some studies, the methods such as classification and clustering and their application in various software testing activities have been examined. Such researches investigated key research gaps and has provided a systematic method for researchers in above area [19].

In other research, referring to one of the major problems in software code testing, especially when testing machine learning models that have been based on SVM was proposed for metamorphic testing of software, which was performed by domain experts whose task is to test the software code and ensure its quality. these models managed to provide suitable solutions, especially for authentic programs with audio data [20].

In another group of researches, machine learning was used to analyze the source code. For this purpose, several categories of software engineering tasks and related machine learning techniques, tools, and datasets were used to solve software testing problems [21].

In this article a new method is proposed to optimize software testing procedure which is based on local meta-heuristic search. In the proposed method the above idea is applied in form of memetic algorithm in order to reinforce it to attain the optimal solution in shorter period of time. The general structure of the article is as follows. The concept based on memetics and the proposed method is reviewed in section 2. In section 3, the performance of the proposed method has been evaluated. Then the proposed method is analyzed to generate patterns responsible for testing a standard code and the results obtained are analyzed and compared with the results obtained from the basic and opposition-based versions of genetic algorithms. This comparison is based on the use of some standard and effective parameters. The conclusion is stated at the end of the article.

2. Proposed Method

As discussed in the previous section, although the genetic algorithm increases the software testing accuracy rather than the random data production method, but it does not necessarily lead into test patterns with minimal number and maximum coverage. In this section the proposed method is described which utilizes memetic concept in order to enhance the generating test patterns.

A. Genetic Algorithm

First, an appropriate portrayal of the problem is demonstrated. A solution (chromosome) is a set of test data (a list of input values) that may be fed to a standard computer procedure to establish how many may satisfy the optimal test data constraints. Hence, a fitness function must be generated to evaluate each pattern. In this study, this function refers to the number of errors that may be discovered by each pattern. Parents (i.e., the current generation patterns with the best fitness values) are subjected to crossover and mutation to produce offspring (i.e., the next generation patterns). It continues until the best generations are produced, which means a set of patterns with maximum coverage. The simple diagram of a genetic algorithm including 4 steps may be demonstrated in Figure 1. Furthermore, Figure 2 shows the pseudo-code of the automatic test data production by using genetic algorithm.

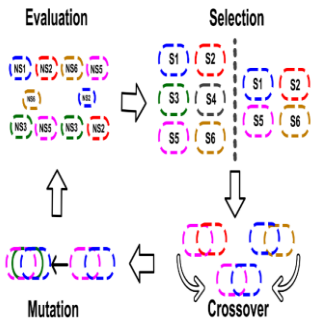


Figure 1. Diagram of genetic algorithm contains steps (Selection, Crossover, Mutation, Evaluation)

```
GA:
►Generate initial population;
►Compute fitness of each individual;
WHILE NOT finished DO
►BEGIN /* produce new generation */
FOR population size / 2 DO
►BEGIN /* reproductive cycle */
Select two of old generation for mating;
►/* biased in favor of the fitter ones */
►Recombine old two to new two of offspring;
►Compute fitness of the two offspring;
►Insert offspring in new generation;
END FOR
END WHILE
```

Figure 2. Test pattern generation using GA

B. Opposition Based Learning Modification of GA

Although in this algorithm, the main structure of the GA is observed, but in parallel with calculating each fitness value, its complementary value is also utilized (e.g., both of A, Ã in Figure2). The pattern associated Ã is marked and its fitness must be calculated. If the new fitness is larger than previous one, Ã will be maintained and replaced instead A; otherwise Ã will be lost and therefore making changes in the new data won't be essential [22]-[24]. Figure 3 demonstrates a basic structure of an opposition-based learning. Immediately, Figure 4 shows the pseudo-code of the opposition-based learning. Eventually, Figure 5 shows how using the concept of opposition-based learning may modify the genetic algorithm in order to produce software test patterns.

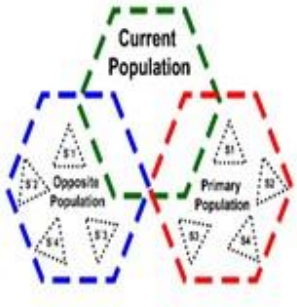


Figure 3. A basic concept of (OBL) includes 3 types of population (Current population, Primary population, and Opposite population)

```
OBL:
►Input: A
►Output: An improved signed A
►Fitness = Compute A;
►Ã=generate opposite of A;
►New Fitness = calculate; fitness of Ã;
►IF new Fitness > fitness; THEN
►Keep Ã;
►Discard A;
►ELSE
►Discard Ã;
END IF
```

Figure 4. Description of Opposition Based Learning Concepts

```
GAOBL:
►Input: Unsigned A and B;
►Output: Sort Number of translocations; ►Generate
the initial population;
►Compute fitness of population;
►Applying BL to initial population;
FOR I = 1 to number Generations DO
►Perform the selection
►Save the best solution found;
►Apply the crossover operator;
►Apply the mutation operator;
►Compute fitness of current population;
►Replace the worst individuals;
►Apply OBL to current population;
►IF entropy Threshold is reached THEN
►►Restart population improved by OBL;
►END IF
END FOR
```

Figure 5. Test pattern generation by using modified version of GA which is based on OBL concept

C. Local meta-heuristic search

Although genetic algorithm-based methods show a considerable ability in finding the solutions, but they are slow to find optimal solutions. The aim of memetic algorithm is to combine population-based searches to improve the above problem by a considerable extent [22-23]. Accordingly, Figure 6 represents the basic structure of a local meta-search algorithm. The pseudo-code of this concept is shown in Figure 7 which has shaped the basis in simulating memetic algorithm in this paper. A local search is done within all existing solutions. The local search is based on the fact that the suitability of each member may be upgraded commensurate to its neighbors. Hence, a better response in less time may be obtained. Subsequently, a subset of the current generation is transmitted to the next generation including a set of current and new solutions and their parents, as well as offspring. This procedure continues until desired test patterns are achieved by satisfying a suitable stopover standard. Memetic Algorithms (MAs) are a type of evolutionary algorithms that use local search instead of global search algorithms and thereby improve the refinement of rolling populations. The result of combining global search and local search is a global search algorithm, which makes it a powerful algorithm [25]. Figure 8 represent methodology of a memetic search. Figure 9 shows the pseudo-code of memetic-algorithm based on the localized search. In similar manner to what have been already discussed for genetic algorithm, the opposition-based learning concept may also be utilized to improve the performance of the memetic algorithm. In the next section the performance of opposition based memetic algorithm is evaluated in parallel with other described schemes.

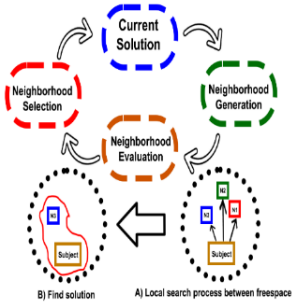


Figure 6. Local meta-heuristic search schematic

```
Local Search:
►Input: A
►Output: An improved signed A
►Fitness = Compute fitness of A;
FOR I to number Iterations DO
►Generate a random position k for A;
►Swap element sign at position k;
►New Fitness = calculate fitness of A;
►IF new Fitness > fitness THEN
►►Update new fitness for A;
►►Break;
►ELSE
►Recover last state of A;
►END IF
END FOR
```

Figure 7. Local meta-heuristic search algorithm

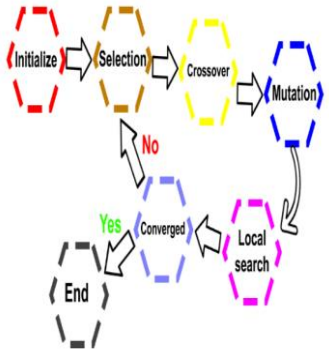


Figure 8. Test pattern generation by using MA-Based on meta heuristic search idea

```
MA:
►Input: Unsigned input A and B
►Output: Number of translocations to sort A
►Generate the initial population of signed genomes;
►Compute fitness of the initial population;
►Improve population by applying Local Search;
FOR I = 1 to number of Generations DO
►Perform the selection and save the best solution found;
►Apply the crossover operator;
►Apply the mutation operator;
►Compute the fitness of the current population;
►Perform replacement of the worst individuals;
►Apply Local Search to the current population;
►IF entropy Threshold is reached THEN
►►Restart population improved by Local Search;
►END IF
END FOR
```

Figure 9. Test pattern generation by using MA-Based on meta heuristic search idea

Considering that the proposed method of this article is based on memetic algorithms, the necessity of explaining the mathematical relations of this algorithm. Furthermore, in the simulation part the performance of the proposed method was compared to an alternative algorithm which is constructed as combination of memetic and opposition-based learning schemes. Therefore, the mathematical relations related to the (MA + OBL) should be considered. Theoretically, the memetic algorithm is a combination of genetic and local search algorithms. To put it better, in fact, this algorithm uses the same cycle as the genetic algorithm, with the explanation that each member of the population can increase its fitness as much as its neighbors (nearby populations). This algorithm calculates the degree of desirability of each solution based on a fitness function and produces new solutions using operators such as intersection and mutation. At the end, every generation, a local search is performed on the sets of solutions of that generation with the aim of intensifying the search process to increase the quality of local optimal solutions [26]. Then the subsets of the current generation (set of current answers, parents and new answers, children) are transferred to the next generation based on survival. The process of production of new generations continues until the conditions of fire are established. One unique feature of the adaptive MAs is to use multiple memes (like genes in the genetic algorithm) in the search and the decision process. In order to describe the different populations that are created by the memetic algorithm, a three-state description model was used, which includes the following. The first state happens if the transition probability is independent of time. The second state occurs when there is an integer duplicate for each pair of generated populations. The last state relates to an irregular state such that for all pairs of populations have an integer. For example, if the input is binary strings of r length which means the number of possible strings such as m is 2^r The total number of these states can be defined as $N = [{}^{n+1-r}_{r-1}]$. In a limited space, by time discretization, we get the following relationship as Equation 1:

$$p_{jk}^{(t)} = P_r\{A^{(t)} = X_k \mid A^{(t-1)} = X_j\} \quad , j,k = 1,\dots \qquad \text{(Equation 1)}$$

Also, the initial probability distribution may be demonstrated as Equation 2:

$$1)\ z_j^{(0)} = P_r\{A^{(0)} = X_j\}, z_j^{(0)} \geq 0$$
$$2)\ \sum_{j=1}^{N} z_j^{(0)} = 1, \qquad j = 1,2,\dots N \qquad \text{(Equation 2)}$$

Then the possible changes in the population generated at each stage of the memetic algorithm are included crossover, and mutation is defined by the probability matrices such as $P_{crossover}$

and $P_{mutation}$ as follows as total $P_{memetic} = P_{mutation} + P_{crossover} + \cdots$.

In the following, we will describe the relations governing an OBL algorithm [27]. Considering $L = [a,\ b]$, the opposite state of L is $L' = [a',\ b']$. Subsequently, considering the state space that occurs for the memetic cycle, including the $Selection = \{S_1,\ S_2, S_3, \dots, S_n\}$, and $Mutation = \{M_1, M_2, M_3, \dots, M_n\}$,spaces, the opposite state of a memetic cycle may be explained as $\sum_{i=0}^{n} S_i' + C_i' + M_i'$, which S_i', C_i', and M_i', denote the opposite state of

selections, crossovers, and mutations respectively. Finally, considering that the proposed algorithm of this article is a combination of two algorithms, memetic (MA) and opposite-based learning (OBL), its final form is defined as Equation 3.

$$MA_{OBL} = p_{jk}^{(t)} + p_{jk}^{(t)'} \qquad \text{(Equation 3)}$$

which $p_{jk}^{(t)} = P_r\{A^{(t)} = X_k \mid A^{(t-1)} = X_j\}, j, k = 1, \dots N$ is the memetic (MA)side and $p_{jk}^{(t)'} = P_r\{A^{(t)'} = X_k \mid A^{(t-1)'} = X_j\}, j, k = 1, \dots N$ is the opposite-based learning (OBL) side of our proposed method.

3. Simulation of Methods

The effectiveness of the proposed method was evaluated by using Java programming language and IntelliJ IDEA software on a MacBook Pro (Retina, 13-inch, Early 2015) computer with 2.7GHZ processor, Intel core i5 and RAM specification of 8GB 1867MHZ DDR3. The alternative methods were implemented consisting of genetic algorithm and its opposition-based versions. Furthermore, the local-meta-heuristic-search idea was simulated in forms of memetic and opposition-based memetic algorithms. Furthermore, a standard code was simulated as test bed which is a simple program that its aim is to determine the largest number among three of them as demonstrated in Figure 10. The reason of choosing this test bed is its several conditions which may challenge each automatic software test procedure. This concept is being understood using the flow control graphs according to Figure 10. In the above graphs, nodes indicate the commands and branches show the control flow among the commands; for example, according Figure 10 branches [5-3], shows that program control may flows from 3 (if command) to 5 (else command). The symbols "T" and "F" indicate how the program routines are transmitted from one state to another. For example, the symbols "T" and "F" in branches [2-3] and [2-6] show that the satisfaction of "if" in line 5 transfers the program to the command of line 3, while failure to satisfy the above condition triggers the routine of line 6. Note that unstructured control transferences, continue, and break may generate more than one aforementioned path. However, the number of these paths is completely low. Also, finding the largest prime number was another challenge that was tested. Finding the largest prime number has always been one of the prominent competitions among supercomputers and it can be considered as a standard test for the efficiency of a coded computer program [28]. Depending on its type, software that is coded with traditional algorithms tries to find the largest prime number compared to itself or other software. Although considerable time has passed since the invention of this test, it can still be cited as a standard test for the quality of software codes. The flowchart of this issue may be demonstrated in Figure 11. Figure 12 demonstrates the general structure belonging to all methods which have been examined in this article. This pseudo-code when it was applied regardless of lines 8 and 9, utilized a conventional genetic algorithm to generate test cases that satisfied the requirements of the test. In case of line 8 execution, program fulfillment based on memetic pattern became significant. In case of line 9 execution, program implementation regards opposition learning modification of genetic algorithm became noteworthy. Simultaneous implementation of these lines also provides a memetic method based on opposition-based learning.

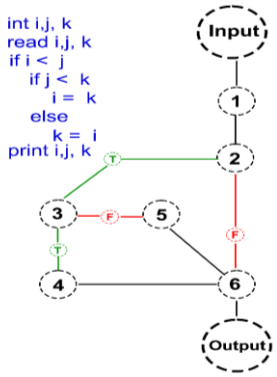


Figure 10. Finding largest number as test bed to examine effectiveness of the proposed method

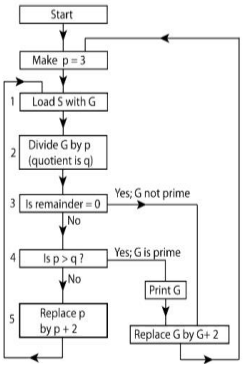


Figure 11. Flowchart of Finding largest prime number as to examine effectiveness of the proposed method [29]

```
BEGIN
Step 1: Initialization and set up*/
Create CDGPaths;
Create and Initialize Scoreboard;
Generate CurPopulation;
/* Step 2: Generate test cases*/
►WHILE ((some (r, unmarked) ∈ TestReq) and not
OutOfTime ()) DO
►►Select unmarked Target from TestReq;
►►WHILE (Target not marked and not MaxAttempts () DO
```

```
►►►Compute fitness values of CurPopulation using CDGPaths;
►►►Improve initial population by applying Local Search
/* For MA   */
►►►Improve initial population by applying OBL Heuristic;
►►►Sort CurPopulation according to fitness;
►►►Select parents of NewPopulation;
►►►Generate NewPopulation from selected members of
CurPopulation;
►►►Execute Program on each member of NewPopulation;
►►►Update Scoreboard and mark TestReq to reflect those tests;
►►END WHILE
►END WHILE
/* Step 3: Clean up and Return*/
Final = test cases that satisfy TestReq;
return (Final, TestReq);
END
```

Figure 12. The structure of the proposed algorithm for automated generation of test patterns

4. Results and Discussion

Figure 13 shows the effectiveness of the four described algorithms when they have been utilized for generating automated patterns test the standard code demonstrated in Figure 10 that denoted the standard flowchart of finding largest number among three of them. In this figure, the horizontal axis shows time intervals required to get results from each algorithm while the vertical axis demonstrates the number of tests that have been completed within each specified time. Based on this figure, each algorithm was indicated by using an exclusive color and the sum of the tests for each of the algorithms was 30 runs.

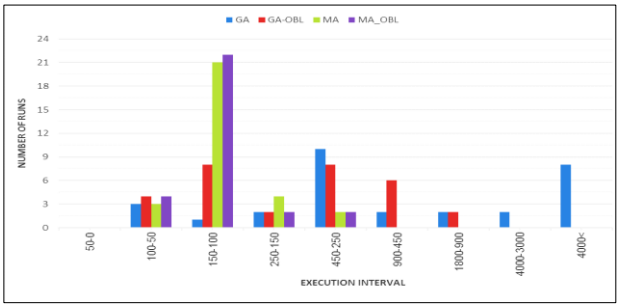


Figure 13. Comparing results obtained for test pattern to find largest number issue based on the proposed and alternative methods

To interpret the results, the levels of the horizontal line have been classified into 4 categories. Therefore, the range of less than 100 seconds, from 100 to 250 seconds, from 250 to 450 seconds and finally greater than 450 seconds were defined as ultra-fast operating range, fast operating range, medium operating range, and slow operating range, respectively. The quality of results was obtained from applying the proposed algorithm and its alternatives were analyzed based on the four above classes. The analysis of the first two levels of this graph showed that only 10% of the tests which have been performed by using genetic algorithm gained to the final results in less than 100 seconds. The performance of opposition based genetic algorithm during the same period of time was 13% which did not show meaningful difference. For this class the proposed idea has not led to any improvement in such way that the memetic and opposition based memetic algorithms obtained the results equal to 10% and 13% respectively. Although the results of proposed and alternative schemes had no significant difference in the case of ultra-fast range, but the effectiveness of memetic based schemes was significantly considerable in the fast-operating class (the execution time between 100 and 250 seconds). Investigating the levels belong to this interval shows that 83% of the tests performed with the memetic algorithm have gained their values within 100 to 250 seconds. Among the tests which have been done by opposition based memetic, 80% belonged to this class. However, only 10% of the tests which have been performed by genetic algorithm and about 33% of those tests which have been done by opposition-based genetic algorithm were completed, within this time range. In the middle operating range) the execution time between 250 and 450 seconds), it may be observed that only 2 runs of memetic and opposition - based memetic algorithms were included which refers to 7% of results. However, 10 runs of the genetic algorithm and 8 runs of the opposition-based genetic algorithm were fallen within this class which means 33% and 27% of results respectively. Finally comparing the behavior of the memetic and genetic-based algorithms in the slow operating range confirmed the effectiveness of memetic paradigm for optimal pattern generation. The levels associated with this range indicated that 47% and 27% of the tests in the genetic algorithm and the opposition-based genetic algorithm were required more than 450 seconds to obtain the responses. However, the proposed method includes memetic and opposition-based memetic algorithms obtained all of their results in less than 450 seconds, therefore none of their tests have been inserted in slow operating range. Based on the above results, it may be observed that the most part (almost 93%) of executions related to the memetic and opposition-based memetic algorithms were classified in ultra-fast and fast execution ranges. In contrast, for genetic and opposition-based algorithms, most of the executions (i.e., 80% and 53% respectively) were categorized as moderate and slow. Also, we scrutinized the results in 8 classes as shown in Figure 14 a-d. The final results informed us that the most important targets demonstrated in range of 100 to 150 (class 2) and range of 250 to 450 seconds (class 4) within all classes. The biggest class belongs to second one taken by range of 100 to 150 seconds. The class three accounted for 4%, 4%, 12%, and 7% for Genetic algorithm, Genetic algorithm plus OBL and MA, and MA_OBL respectively. The GA and GA_OBL methodology gain more results in various ranges including 8 classes. On the other hand, The MA and MA_OBL could obtain less results compared to aforementioned methodologies in 5 classes. Figure 14 a-d demonstrates the plots confirms recent results.

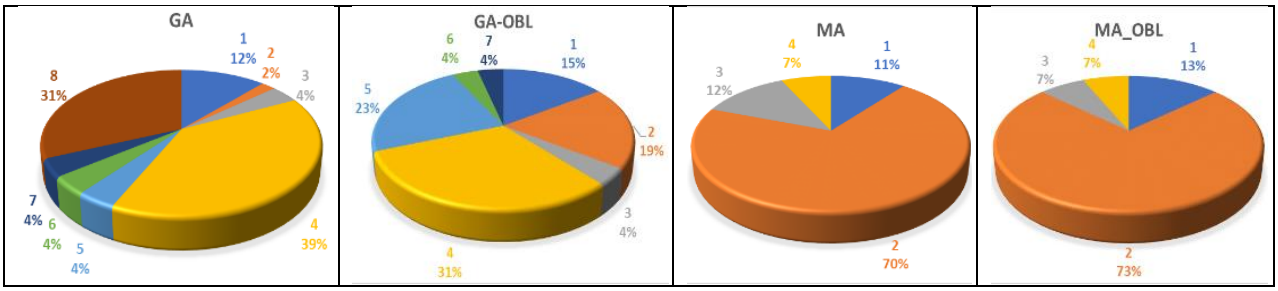


Figure 14a-d. There are four plots including GA, GA_OBL, MA, and MA_OBL methods. Each method bears its special classes from class 1 to class 8 to analyze find largest number problem. Depend on each method and its characteristic, various range of values were assigned to a class

Furthermore, Figure 15 demonstrates the results obtained from applying four described SBST algorithms on standard code shown in Figure 11. Similar to the previous challenge, total number of tests for each algorithm is 30 and the horizontal line levels are classified into 4 categories including ranges of fewer than 100 seconds, from 200 to 300 seconds, from 500 to 900 seconds, and finally more than 900 seconds. The first two levels of this graph showed that only 9% of the experiments that were performed using the genetic algorithm reached the final results in fewer than 100 seconds. The performance of the GA_OBL in the same time period was 14%. Also, the memetic algorithms based on opposition obtained results equal to 9% and 14%. Examining the levels belonging to this interval shows that 79% of the tests performed with the memetic algorithm have obtained their values within 200 to 500 seconds. Among the tests performed by the opposition-based memetic, 78% belonged to this class. However, only 12% of the experiments performed by the genetic algorithm and about 35% of the experiments performed by the adversarial genetic algorithm were completed in this time frame. In the intermediate operating range (running time between 500 and 900 seconds), it may be observed that only 2 enactments of the memetic and adversarial-based algorithms are included, which refers to 8% of the results. Like the previous problem, 9 enactments of the genetic algorithm and 7 enactments of the opposition-based genetic algorithm were included in this class, which means 34% and 28% of the results, respectively. Levels related to this range showed that 47% and 27% of tests in genetic algorithm and opposition-based genetic algorithm needed more than 900 seconds to obtain answers. However, the proposed method includes both memetic and adversarial-based memetic algorithms that obtained all their results in less than 900 seconds, so none of their experiments were included in the slow operating range. To put it better, for genetic and opposition-based algorithms, the majority of executions (78% and 49%, respectively) were not classified rapidly. For more analysis, we analyzed the results in 8 classes from 0 to 8000, as shown in Figure 16 a-d. The final results showed in the range of 200 to 300 (class 2) and the range of 500 to 900 seconds (class 4) in all classes. The largest class belongs to the second category, which is taken with a range of 200 to 300 seconds. The third class received 4%, 4%, 12% and 7% respectively for genetic algorithm, genetic algorithm plus OBL and MA, and MA_OBL. GA and GA_OBL method can achieve more results in different ranges including 8 classes. On the other hand, MA and MA_OBL were able to obtain lower results compared to the mentioned methods in 5 classes which is shown in Figure 16 a-d.

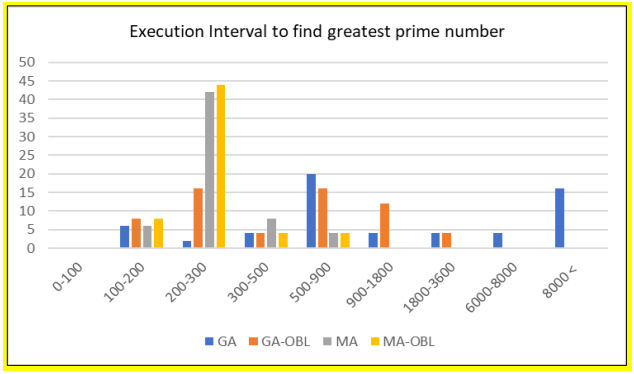


Figure 15. Comparing results obtained for test pattern to find prime number issue based on the proposed and alternative methods

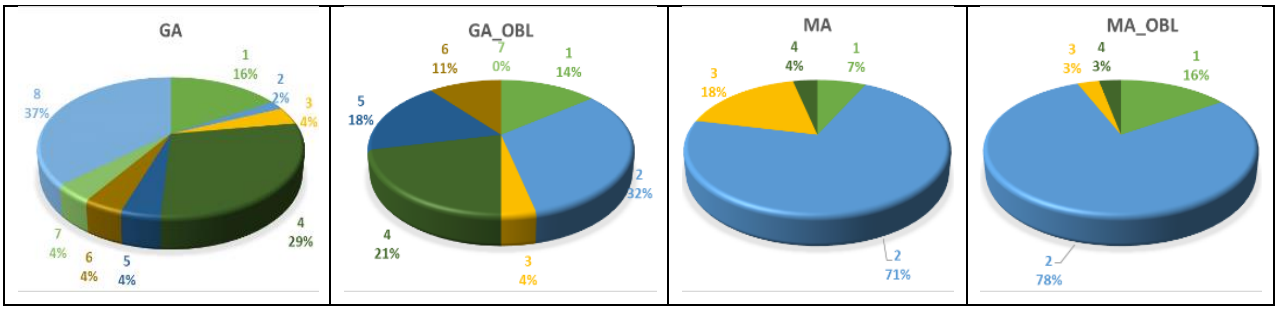


Figure 16a-d. There are four plots including GA, GA_OBL, MA, and MA_OBL methods to analyze finding prime number issue. Each method bears its special classes from class 1 to class 8. Depend on each method and its characteristic, various range of values were assigned to a class

Finally, the performance of some state of art methods that used meta-heuristic base for SBST were evaluated. Therefore, the performance of Cuckoo Search (CA), Bat algorithm (BA), Hill climbing (HCA), Firefly Algorithm (FA), Particle Swarm Optimization (PSO), and Artificial Bee Colony Algorithm (ABC) was investigated in order to solve the problems of finding the largest prime number and finding the largest number. As demonstrated in figure 15, considering the average execution time for both problems of finding the largest number and finding the first number, FA, GA and MA algorithms required the highest time, something around 20 units, which in this respect is in the most unfavorable conditions compared to they had other algorithms. Furthermore CA, PSO, HCA, and GA_OBL algorithms took a time interval of 7 to 15 units, which means that they can be placed in the middle range of algorithms in terms of performance criteria. In the end, MA_OBL and BA algorithm had the best performance compared to other algorithms by allocating a time interval lower than 7 units. Figure 17 shows the general results of the comparison of the alternative algorithms and our proposed scheme.

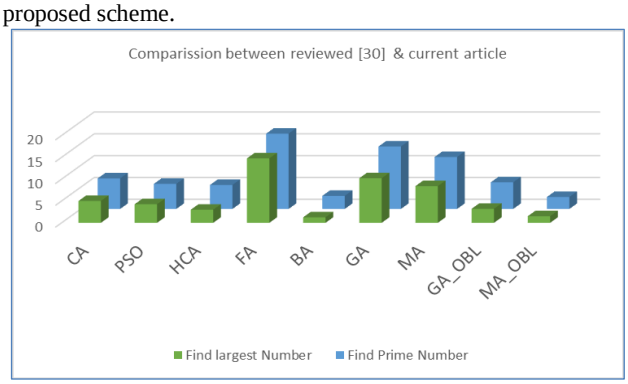


Figure 17. The six algorithms on the left, namely CA, PSO, HCA, FA, and BA are related to reviewed research [30], while the four algorithms on the right include GA, MA, GA-OBL, and MA-OBL related to the current article. Also, blue cubes are related to solving the problem of finding prime numbers and green cubes are related to solving the problem of finding large numbers

5. Conclusion

In this paper, a new method was presented for the optimal test pattern production in search-based software testing paradigm. The proposed method utilized the concept of local meta-heuristic search in two forms including memetic and opposition-based memetic algorithms. To evaluate the effectiveness of this idea, the basic and opposition-based versions of genetic algorithms were simulated and their performances were compared with basic and opposition-based versions of memetic algorithms who used from local meta-heuristic search in their procedures. For better interpretation the obtained results were classified in four categories including ultra-fast, fast, moderate and slow. The analysis of the results showed that the proposed idea may not significantly improve the pattern generation process in ultra-fast category. However, the considerable improvement made by the proposed method in the fast class (i.e., 73% [maximally and 47% minimally) demonstrated the effectiveness of local meta-heuristic search on promoting the automated production of test patterns in this class. The investigation of the slow class confirmed the effectiveness of the proposed idea in such way that almost 47% and 27% of the tests were carried out by genetic and opposition-based algorithms were fallen within this category; while none of the tests which have been performed by using local meta-heuristic search have been included in slow class. Based on these results, it may be concluded that the local meta-heuristic search may be considered as an effective base for developing algorithms which try to produce optimal test patterns in search-based software testing.

References

[1] Lee J. "Survey on software testing practices," IET Software. 6(3): 275-282. 2012.
[2] Garousi V. "A survey of software testing practices in Canada," Journal of Systems and Software. 86(5): 1354-76. 2013.
[3] International Software Testing Qualification Board. Worldwide Software Testing Practices. Online: http://www.istqb.org/documents/ISTQB_Worldwide_Software_Testing_Practices_Report.pdf. 2016.
[4] Spillner A. Linz T. & Schaefer H. "Software testing foundations: a study guide for the certified tester exam," Rocky Nook, Inc. 2014.
[5] Li K. and Wu M. "Calculation Effective software test automation: developing an automated software testing tool," John Wiley & Sons. 2006.
[6] Khalid R. "Towards an automated tool for software testing and analysis," 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST). 461-465. Islamabad. Pakistan. 2017.
[7] Rafi D.M. Moses K.R. Petersen K. & Mäntylä M.V. "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey," Proceedings of the 7th International Workshop on Automation of Software Test. 36-42. Zurich. Switzerland. 2012.
[8] Machado B. Camilo-Junior C. and Rodrigues C. "SBSTFrame: a framework to search-based software testing," IEEE International Conference on Systems, Man, and Cybernetics. 106-11. Budapest. Hungary. 2016.
[9] Sayyari F. and Emadi S. "Automated generation of software testing path based on ant colony," International Congress on Technology, Communication and Knowledge (ICTCK). 435-440. Mashhad. Iran. 2015.
[10] Vats P. Mandot M. and Gosain A. "A comparative analysis of ant colony optimization for its applications into software testing," Computational Intelligence on Power, Energy and Controls with their impact on Humanity (CIPECH). 476-481. Ghaziabad. India. 2014.
[11] Chakrabarti A. and Godefroid p. "Software partitioning for effective automated unit testing," Proceedings of the 6th ACM & IEEE International conference on Embedded software. 262-271. Seoul. Republic of Korea. 2006.

[12] Lijuan W. Yue Z. and Hongfeng H. "Genetic algorithms and its application in software test data generation," International Conference on Computer Science and Electronics Engineering (ICCSEE). 617-620. Hangzhou. China. 2012.
[13] Khan R. and Amjad M. "Automatic test case generation for unit software testing using genetic algorithm and mutation analysis," IEEE UP Section Conference on Electrical Computer and Electronics (UPCON). 1-5. Allahabad. India. 2015.
[14] DONG Y. AND PENG J. "AUTOMATIC GENERATION OF SOFTWARE TEST CASES BASED ON IMPROVED GENETIC ALGORITHM," INTERNATIONAL CONFERENCE ON MULTIMEDIA TECHNOLOGY (ICMT). 227-230. HANGZHOU. CHINA. 2011.
[15] Bouchachia A. "An immune genetic algorithm for software test data generation," 7th International Conference on Hybrid Intelligent Systems (HIS). 84-89. Kaiserlautern. Germany. 2007.
[16] Khoshniat, Niloofar, et al. "Nature-inspired metaheuristic methods in software testing." Soft Computing 28.2 (2024): 1503-1544.
[17] Al-Sammarraie, Hamsa Naji Nsaif, and Dayang NA Jawawi. "Multiple black holes inspired meta-heuristic searching optimization for combinatorial testing." Ieee Access 8 (2020): 33406-33418.
[18] Khilari, Sunil, et al. "Analysis of Strength and Capabilities of Major Machine Learning Algorithms Used in Software Testing and Quality Assurance." NATURALISTA CAMPANO 28.1 (2024): 1414-1421.
[19] Panwar, Arti, and Prasadu Peddi. "Implementation of Software Testing Using Machine Learning: A Systematic Mapping Study." JJTU Journal of Renewable Energy Exchange. ISSN: 2321-1067 Volume 11 Issue 7(2023), PP 58-64.
[20] Nasr, Islam, Lobna Nassar, and Fakhri Karray. "A study of the interactive role of metamorphic testing and machine learning in the quality assurance of a deep learning forecasting application." International Journal of Information Technology 16.1 (2024): 105-120.
[21] Sharma, Tushar, et al. "A survey on machine learning techniques applied to source code." Journal of Systems and Software 209 (2024): 111934.
[22] Iacca G. Neri F. and Mininno E. "Opposition-Based Learning in Compact Differential Evolution," Lecture Notes in Computer Science.6624: 264-273. 2011.
[23] Da Silveira L.A. Soncco-Álvarez J.L. de Lima T.A. and Ayala-Rincón M. "Memetic and Opposition-Based Learning Genetic Algorithms for Sorting Unsigned Genomes by Translocations. Advances in Intelligent Systems and Computing," 419: 73-85. 2015.
[24] Molina D. and Lozano. M. "Memetic Algorithms for Continuous Optimization Based on Local Search Chains," Evolutionary Computation. 18 (1):27-63.2010.
[25] Dutta, Priyom, and B. S. Mahanand. "Affordable energy-intensive routing using metaheuristics." Cognitive Big Data Intelligence with a Metaheuristic Approach. Academic Press, 2022. 193-210.
[26] Ong, Yew-Soon, et al. "Classification of adaptive memetic algorithms: a comparative study." IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 36.1 (2006): 141-152.
[27] Mahdavi, Sedigheh, Shahryar Rahnamayan, and Kalyanmoy Deb. "Opposition based learning: A literature review." Swarm and evolutionary computation 39 (2018): 1-23.
[28] Zagier, Don. "The first 50 million prime numbers." The Mathematical Intelligencer 1. Suppl 1 (1977): 7-19.
[29] Miyazaki, Shintaro. Algorithmic listening 1949-1962 auditory practices of early mainframe computing. AISB/IACAP World Congress 2012: Symposium on the History and Philosophy of Programming, Part of Alan Turing Year 2012.
[30] Khari, Manju, et al. "On the use of meta-heuristic algorithms for automated test suite generation in software testing." Toward Humanoid Robots: The Role of Fuzzy Sets: A Handbook on Theory and Applications (2021): 149-197.