Journal of Fundamental and Applied Sciences

ISSN 1112-9867

Available online at http://www.jfas.info

Research Article

Special Issue

COMPARATIVE EVALUATION OF GENETIC ALGORITHM-BASED TEST CASE OPTIMIZATION

I. M. Ismail*, W. M. N. Wan-Kadir, R. Hassan

Department of Software Engineering Faculty of Computing Universiti Teknologi Malaysia

Published online: 01 February 2018

ABSTRACT

Software testing is a crucial phase in software development process although it consumes more time and cost of software development. Researchers have proposed several approaches focusing on helping software testers to reduce the execution time and cost of the testing process. Test case optimization is a multi-objective approach that has become one of the best solutions to overcome these problems. Test case optimization focusing on reducing the number of test cases in the test suite that may reduce the overall testing time, cost and effort of software testers especially in regression testing. This paper presents the comparative evaluation between test case optimization techniques that are based on Genetic Algorithm (GA). The evaluation is based on five criteria i.e. technique objectives, applied fitness function, contributions, the percentage of the reduced test cases, fault detection capability, and technique limitations. The evaluation results able identify the gaps in the existing GA-based test case optimization approaches and provide insight in determining the potential research directions in this area.

Keywords: Test case optimization, regression testing, multi-objectives, genetic algorithm, software testing.

Author Correspondence, e-mail: izwan4@live.utm.my

doi: http://dx.doi.org/10.4314/jfas.v10i2s.7

1. INTRODUCTION

Software testing is one of the important phases of software development. Its main objective is to identify defects and faults occur in software under development hence ensure the correctness of the product itself [1]. In addition, it is also needed to ensure that the product meets customer's specific requirements as well as free from bugs. Due to inevitable changes in software systems, it is important to retest the changed software. This type of testing is called regression testing. It aims at retesting the entire software once changes occurred in the software such as modification of GUI, insertion of new function or removal of any function from previous version [2]. It is important to ensure that the modification in the software did not affect other parts of the software. However, due to cost and time-consuming factors which resulted from a redundant and large amount of test cases that need to be re-tested, researchers have introduced test case selection, reduction, and prioritization in order to manage the testing process in regression testing [3, 4]. These techniques may reduce the overall cost and time testing process since it is the most crucial factors in regression testing [5]. Test case reduction, which is also known as minimization or optimization, works by eliminating test cases from test suite in regression testing [6]. The aim of test case optimization is to reduce the number of test cases in the test suite so the execution of regression testing for a particular software can be faster than before.

In the early introduction of test case optimization, it is treated as a single objective optimization problem which only considering reducing the overall cost of software testing as their main objective. However, due to multiple factors and variables, it can be considered as multi-objective optimization problem which provides trade-off among solutions [7]. In a multi objective optimization problem, it is hard to satisfy all objectives since there are many optimal solutions for every objective [8]. Pareto optimal solution is a well-known technique in order to solve multi objective optimization problem by finding as many optimal solutions available in a particular problem [9]. Moreover, multi objectives optimization has been introduced by considering multiple objectives in test case optimization such as reducing total cost, number of test cases, number of faults detected and time taken for overall software testing life cycle (STLC) [10].

This study presents a comparative analysis of several selected test case optimization techniques with different approaches and concepts focusing on techniques that implemented

the concept of genetic algorithm (GA). This comparative analysis and evaluation are needed to identify the most effective and suitable technique in solving the test case optimization problem in software testing. This paper is divided into several sections as follow. Section 2 describes the overview of test case optimization and several techniques that have been introduced by researchers. Section 3 briefly explains the concept of test case optimization technique with genetic algorithm approach implemented in each method. Section 4 is divided into two subsections which explaining comparison criteria which describing the criteria to evaluate each technique in this study in the first part. Discussion on the evaluation results is explained in the second part of Section 4. Lastly, Section 5 concludes the result of this study and describes future work.

2. TEST CASE OPTIMIZATION

Generally, test case optimization works by eliminating redundant test cases in a test suite and also finding the best set of test cases by considering the coverage of the test cases. Full coverage of software testing with less number of test cases has become a major concern in test case optimization technique. In addition, test case optimization purposes to find the subset from the set of test cases which contain the most optimized set of test cases by eliminating redundancy in test cases and selecting the best and have good criteria declared in particular test suite [11]. The problem concept of test case optimization can be derived as T = {T_1, T_2, T_3,….T_n}, where T is the original test suite consists of a larger number of test cases. Meanwhile, T’ = {T_1’, T_2’, T_3’,….T_n’}, where T’ consist of the most efficient test cases that optimized from original test suite [9]. Throughout the years, many researches have been conducted in order to help software testers to fully optimize their test cases in software testing by implementing multiple types of algorithms and framework. Several popular algorithms such as Simplified Swarm Optimization (SSO), Artificial Bee Colony (ABC), Cuckoo Search (CS) Algorithm, and Genetic Algorithm (GA) has been implemented as optimization algorithm for test cases minimization and reduction.

2.1. Simplified Swarm Optimization (SSO)

Simplified Swarm Optimization (SSO) algorithm is an enhancement from the conventional Particle Swarm Optimization (PSO). The main objective of this algorithm is to overcome convergence speed problem in GUI testing as the speed is decreasing whenever number of

iteration in testing process increase [12]. This approach focusing on GUI testing procedure that has a larger test suite size when it uses conventional PSO. SSO is combined with Covering Array (CA) to produce an effective test case for s particular software under test (SUT). The advantages of this algorithm are increasing convergence speed in GUI testing for particular SUT as the number of iteration increase. It also overcomes non-deterministic (NP) problem and increasing the rate of fault detection for particular SUT. However, the results of this study are arguable since the number of the case study is relatively small. In addition, the algorithm itself is complex since it consists of many steps and combination with CA increases its complexity for an optimization technique.

2.2. Artificial Bee Colony (ABC)

Artificial Bee Colony (ABC) is one of the most popular optimization algorithms that supports test case optimization. This algorithm is based on the natural behaviour of bee which constantly looking for their food source in the best way. It was introduced by Dervis Karaboga in 2005. In general, there are three main types of bees in ABC algorithm namely scout bees, onlooker bees and employee bees [13]. The purpose of scout bees is to look for the best path and food source which are then presented to onlooker bees. Onlooker bees will calculate the possibility of the food source then pass back the information to scout bees which mean the food source can be used for further exploration. The employee bees search for new food source by doing an exhaustive search. ABC algorithm provides rapid test data generation and fault detection which contribute much to software testing process. The behaviour of this algorithm increases efficiency and effectiveness of the whole software testing process. It is also found that test suite generated and optimized using ABC achieves complete coverage and able to maintain the consistency in the testing process [14].

2.3. Cuckoo Search (CS) Algorithm

Cuckoo Search (CS) algorithm, which is based on the natural behaviour of cuckoo birds, has been introduced to overcome complex optimization problem [15]. The combination of CS with Covering Array (CA) and Levy Flight algorithms may optimize test cases by reducing a number of test cases in a particular SUT. This combination still unable to overcome the inherent non-deterministic problem in test case optimization. Therefore, CS algorithm may be better than conventional PSO but still not the best solution for test case optimization. It is

also due to many loops involves in the iteration to calculate the fitness values which causes some delay in producing the results.

2.4. Genetic Algorithm (GA)

Genetic Algorithm (GA) is one of the available techniques in order to find the most optimized set of test cases. The basic approach of GA in optimizing test cases start with random generation of chromosomes that represent the population. Next, the fitness value of each chromosome in the population is calculated in order to continue with the selection, crossover and mutation operators to generate new population which more fit and optimized [16]. Stopping criteria are applied to the population to determine whether the new population is achieved targeted fitness value.

3. GENETIC ALGORITHM-BASED TEST CASE OPTIMIZATION

Genetic Algorithm (GA) is one of the most popular meta-heuristic algorithm implemented to solve the optimization problem. It is also based on the natural behaviour of chromosomes that in any living body. GA is all about the population of chromosomes which consist of parents and children produced for the next generation and also about finding the fittest chromosomes in particular population [1]. In term of test case optimization, population represents test suite while chromosomes represent test cases in the test suite [17]. The genetic algorithm consists of three main operations that responsible for the generation of the new population. The operations are selection, crossover, and mutation. Selection operation will be executed first in the algorithm in order to choose the chromosomes that have the probability and capability for surviving in the current population [18]. Next, the crossover operation will take place by combining two chromosomes from the selected list to generate a new population. Finally, mutation operation is applied to the new population as a mutant to produce the fittest and high surviving rate of chromosomes in the population. The final number of chromosomes in the population will become the powerful chromosomes compared to the previous population. This iteration will continue same as a human being that continues reproducing their next population as shown in Figure 2.

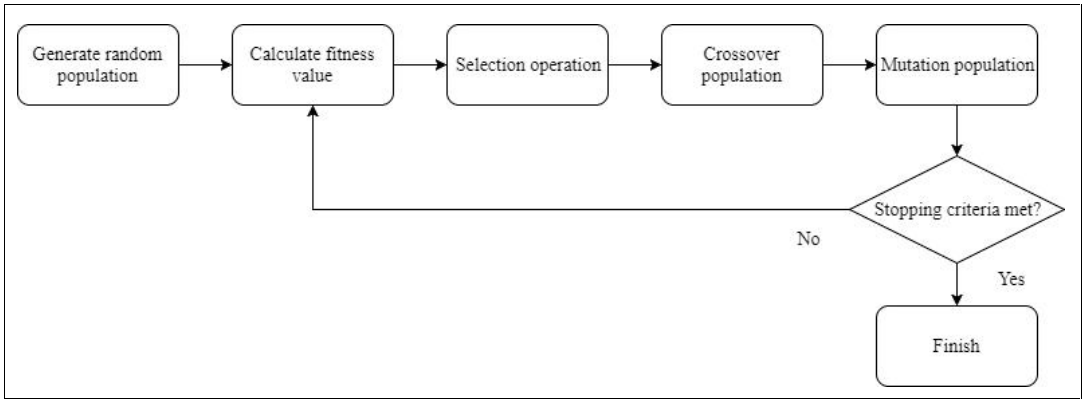

Fig.2. Basic Operation of Genetic Algorithm

Multiple researches have been conducted in order to implement and prove the effectiveness of GA in helping minimize the number of test cases in software testing process. Throughout the years, implementation of GA in test case optimization has been improved by implementing different type of fitness function and even hybridization of GA with another optimization algorithms. This section discusses various implementation of GA in test case optimization.

3.1. Simple Genetic Algorithm

One of the works conducted was using an improvised version of conventional GA to find the most optimize test cases in the test suite. The experiment conducted starting by the generation of a random population of test cases then applying some mutants to the population. The purpose of the mutants is to find the most optimize test cases in the test suite as the mutants itself contain some errors [17]. Next, stopping criteria is applied whereas if the number of test cases with errors found is less than the minimum number of mutants, then it is failed to find errors and vice versa. This research didn't consider the important evaluation metric for optimizing test cases such as number of reduced test cases, execution time and overall fault detection capability thus make it harder to study the overall efficiency of GA implementation.

In addition, Genetic Algorithm also has been implemented in safety critical control testing process to find optimize test cases and also overcome benchmark problem [19]. The implementation of GA in the research purposes to find errors and faults in the code by injecting the mutants in the code to achieve most optimize block coverage metrics. The authors using conventional GA provided in MATLAB tools and defined several parameters

to find the fitness function such as population size, population range, elite count, generation, stopping criteria and plot functions. The result shows that fitness value decreasing as number of generation increase after a single run of the experiment. In addition, implementation of GA also seems to be effective since minimum test case with 100% test coverage can be achieved which increase the effectiveness of software testing.

3.2. Weight-Based Genetic Algorithm (WBGA)

In this approach, the weight-based concept is used to evaluate fitness function of test cases whereas WBGA is a fixed weight of test cases which is predefined by the user for particular test cases in test suite [20]. Several parameters have been considered in order to show the optimization of test cases which are feature pairwise coverage and fault detection capability. The output of the experiment conducted in this research shows that the fault detection capability for particular test suite after optimized using WBGA is lower compared to another algorithm. Fixed-weight algorithm performance is less than a dynamic-weight algorithm which is Random Weighted Genetic Algorithm (RWGA) hence shows that WBGA is not the best algorithm for test case optimization.

Moreover, Genetic Algorithm also has been applied for test case reduction alongside with Weighted Control Flow Graph (CFG) as its fitness function evaluation [21]. The main objectives of this research are minimizing test cases and also finding the most critical solution for particular SUT. In order to do so, authors used weighted CFG and Fitness Scaling to calculate fitness function for chromosomes and to reduce premature convergence of the fitness value itself [21]. By using this hybrid approach, authors claimed that the fitness value of chromosomes has increased significantly and reducing the premature convergence problem from simple GA implementation. This research also concluded that implementation of fitness scaling can overcome the problem from simple GA. It shows that fitness scaling technique can be implemented in variants of Genetic Algorithm in order to help in optimizing a number of test cases and overall testing process.

3.3. Fuzzy-Based Genetic Algorithm

Genetic Algorithm also has been implemented with fuzzy logic approach resulting Fuzzy-Based Age Extension of Genetic Algorithm (FAexGA) [22]. In this research, the fuzzy aging approach has been implemented to chromosome's probability by dividing it into three stages, young, old and middle-aged. This algorithm used to generate effective test cases focusing on

GUI of particular system under test hence becoming one of their limitation since effective testing process should cover external and internal structure of SUT. In this approach, crossover probability of particular chromosomes is controlled by Fuzzy Logic Controller (FLC) which consist of age and lifetime of parents and the current population of chromosomes. The result of this research shows that FAexGA performs higher in term of runs with solution compared to another algorithm. In addition, FAexGA found more number of solutions for test cases after injected with mutants compare to simple GA. The overall performance of FAexGA is also much better if it is compared to other GA-based technique. However, FAexGA's performance decreasing in term of the final generation of test cases and test data which may not be good to overall testing process. Another drawback of FAexGA is the implementation of the fuzzy technique in the algorithm require many variables which make it more complicated to implement to the small and medium size of test suite but can be efficient for the large-scale test suite.

3.4. Non-dominated Sorting Genetic Algorithm (NSGAII)

Non-dominated Sorting Genetic Algorithm (NSGAII) is one of the variants in GA. Pareto ranking and crowding distance approach also implemented in the research to choose a possible solution of test cases [1]. The implementation starting with the generation of random test cases which indicates as the chromosomes. Next, the test cases are organized in a non-dominated form which ranked by minimum test cases and maximum branch coverage. Based on the experiment conducted, authors claimed that entire software testing's time execution reduced to 4 hours and an average of 15% efficiency compared to original test cases in the experiments [1]. This result shows that implementation of GA can help finding the most optimized test cases in software testing process. In another research conducted using NSGAII, the result obtained shows that the percentage of test cases reduction is 20% from original test cases and 84% of fault detection capability which seems that reduction of test cases didn't perform well and the test cases cannot detect more fault in the application [9]. This technique can be improvised by implementing mutation hypothesis to the algorithm as it is the common technique and more accurate result of the optimization process.

4. COMPARATIVE EVALUATION

This paper presents the comparative evaluation in test case optimization techniques focusing on GA-based techniques. This section describes the evaluation criteria and evaluation results of each technique presented in the previous section.

4.1. Evaluation Criteria

This paper evaluates each test case optimization technique based on GA according to several criteria. Each criterion used in the evaluation is briefly described in this section.

a. Objectives of Proposed Technique: This criterion refers to overall purpose and concept applied in each technique since it is differed from each other.

b. Fitness Function: Fitness function is needed in optimization algorithm especially in each technique to assign the fitness value and determining divergence and inequality of test cases in particular test suite [23]. This criterion describes a different type of fitness function applied to each technique to determine the impact of optimized test cases in the optimization process. Different fitness function may give a different result depending on the implementation of each technique.

c. Contributions: It refers to the overall achievement of the technique proposed in term of effectiveness in optimizing number of test cases.

d. Percentage of Reduced Test Cases: This criterion refers to optimized number of test cases after optimization process occurs in term of percentage reduced from the original number of test cases. It is also can be considered as the result of experiments in each technique proposed.

e. Fault Detection: This criterion refers to the capability of fault detection in an optimized set of test cases between each technique which also may know as the capability of the proposed technique in optimizing the overall software testing process.

f. Limitations: It refers to drawbacks and disadvantages of each proposed technique which can reduce the effectiveness of optimizing a particular set of test cases.

4.2. Evaluation Result

Table 1 shows the summary of test case optimization techniques based on Genetic Algorithm.

Table 1. Comparison of Test Case Optimization Techniques

Criteria / **Approach**	**Objective**	**Fitness Function**	**Achievement**	**Percentage of Reduced Test Cases**	**Fault Detection**	**Limitation**
Conventional Genetic Algorithm [17], [19]	Better optimization approach, overcome benchmark problem	×	Well optimized test cases during test cases generation	55% of original test cases	×	Focusing more on test case generation, no additional fitness function
WBGA [20], [24]	Applying weight to test cases to find optimize set of test cases, reduce pre-mature convergence problem	Fitness Scaling	WBGA threshold did not achieve, Fitness value increase, convergence problem decreases	22% reduced from original test cases	0.85 below threshold	Performance of WBGA (fixed-weight) less than RWGA (random weight)
Fuzzy-based Genetic Algorithm [22]	Find most optimize set of test cases by applying fuzzy aging technique	Fuzzy Aging	Number of runs increase but decreasing of final generation of	0.99 % solution in final set of test cases	Higher compared to simple GA	Focusing on test cases for GUI

			test cases and data			
NSGAII [1], [9]	Minimized and optimized test cases in test suite	Pareto ranking and crowding distance	Execution time of testing process reduce, efficiency slightly increase	Overall 29 test cases, 82% of original test cases	×	Redundant technique for selection of test cases, basic operation of GA, small size of case study

Based on Table 1, each technique implemented different concepts of optimization technique. A previous study in [17, 19] aiming in better optimization approach and also reducing number of test cases in safety critical control system which consists of many redundant and unnecessary test cases during regression testing. Furthermore, it is also aiming to overcome benchmark problem since this study is implementing Taguchi method's experiment, which requires the concept of benchmarking between two algorithms. On the other hand, WBGA implemented in [20, 24] proposed the concept of weight of particular value to each test cases in the test suite to obtain the optimized set of test cases. The weight assigned to each test case has a close relationship with fitness function in each technique. In addition, another aim of [24] is to overcome premature convergence problem which resulted from selection of a powerful set of test cases. In another research, fuzzy aging is applied to each test cases in the set to obtain the optimized set of test cases [22]. This approach applies the fuzzy-based technique to get test cases that are more efficient.

In term of the fitness function, proposed techniques in [17, 19] did not apply any additional fitness function. These approaches only depending on pure genetic algorithm operation such as selection, crossover, and mutation to select the most optimize test cases from the large set of chromosomes. The continuous operation in basic GA may take a longer time to obtain the optimized set of test cases. Meanwhile, weight-based test cases and fitness scaling is applied in [20, 24] respectively. WBGA is a fixed weight applied to the test cases and the selection of potential test cases to be in the optimized set is depending on the higher value of weight

while fitness scaling is applied alongside with Weighted Control Flow Graph (CFG) to assist the selection of an optimized set of test cases [24]. The main purpose of fitness scaling is to scale fitness values in each test cases to avoid premature convergence. Another study applied fuzzy aging techniques as their fitness function [22]. Fuzzy aging technique introduces three stages to each test cases and classifies them to 'young', 'old' and 'middle-aged'. Test cases with 'young' and 'middle-aged' have a higher potential to be chosen as the optimized set of test cases since it has more efficiency in software testing. Next, Pareto ranking and crowding distance are applied in NSGAII as its fitness function [1, 9]. Combination of these two-fitness function produces higher efficiency in the set of test cases after undergoing a basic operation in GA.

The overall result and achievement in most of the proposed techniques show that a set of optimized test cases can be achieved. In conventional GA, the final set of test cases generated is a well-optimized test case. However, a proposed technique in [20] cannot achieve threshold that is set at the beginning of an experiment that shows a final set of optimized test cases cannot be achieved. Meanwhile, the fitness value of test cases in [24] increase and premature convergence problem is decreasing due to the implementation of CFG and Fitness Scaling as its fitness function. The final set of test cases produced is more efficient and reliable to continue with the entire testing process. The result from [22] shows that number of runs is increasing but the final generation of test cases and data to obtain final set is decreasing which may affect the overall efficiency of software testing. In another research, it shows that execution time of the testing process is reduced and the efficiency of software testing is slightly increasing [1, 9]. It shows that implementation of NSGAII really helping in optimizing test cases and increase the efficiency of entire software testing process. The details of experimental result can be justified in term of reduced number of test cases. Implementation of basic GA in [17, 19] shows that almost half of the initial number of test cases is reduced and eliminated. The number of test cases is still large and it may affect the time taken to complete the testing process if depending on this set of test cases. Likewise, a number of reduced test cases in [20, 24] shows that only 22% decreasing of test cases. it is clearly shown that implementation of weight-based GA did not produce a good set of test cases although it overcoming pre-mature convergence problem. In a different study, fuzzy-based GA seems to have higher number of solutions in the final set of test cases compared to

other proposed technique which proven that implementation of fuzzy aging technique contributes more in optimizing test cases [22]. However, the authors did not specifically mention the reduced number of test cases in their discussion. Another research conducted using NSGAII shows that the reduced number of test cases is up to 82% from 29 test cases. The final set produced is very small since it is only applied and tested to a small case study.

Next criterion in this comparative evaluation is fault detection capability. Fault detection capability or coverage is used to measure the error and bugs detected in particular software testing by using optimized set of test cases. Theoretically, large coverage and a higher number of fault detected by using a reduced set of test cases indicate that a particular algorithm and technique is reliable to optimize overall software testing process. Based on Table 1, authors in [1, 9, 17, 19] did not mention specifically the result of fault detection based on optimize test cases. the authors may not consider the importance of fault detection capability in their experiment which may lead to inefficient testing process. Meanwhile, faults detected in [20, 24] only 0.85 below the threshold which means it did not achieve threshold set since WBGA's performance is slightly lower compared to random-weight GA. Next, fault detection capability in fuzzy-based GA is higher compared to simple GA although the final generation of test cases is decreasing [22]. Authors claimed that FaexGA is maintaining the diversity in test cases generation which makes the fault detection is higher.

Finally, several limitations and drawbacks of each proposed technique are listed. Firstly, conventional GA more focusing on test cases generation with less optimizing it. It also has no additional fitness function which makes optimizing process more accurate [17, 19]. Secondly, the drawback of WBGA is the performance of fixed -weight GA is less compared to random-weight GA (RWGA). Based on the experiment conducted in [20, 24], random-weight is more efficient and applicable to genetic algorithm since it suits the concept of GA itself, which is the random population at the beginning of the experiment. Thirdly, the authors in [22] only focusing on optimizing test cases for black box testing which only considering on GUI parts of the software. It is needed for test case optimization technique to be applicable to the black box and white box testing to produce better testing process. Lastly, multiple techniques in selecting test cases and small case study has become drawback and limitation in NSGAII [1, 9]. Although NSGAII is proven as one of good optimization

technique in the various field, it still can be improved in term of implementation of the different fitness function to give a variety of approaches in NSGAII.

5. CONCLUSION AND FUTURE WORK

This paper presented a comparative evaluation of test case optimization techniques based on genetic algorithm. The evaluation is based on six criteria i.e. technique objectives, fitness function, contributions, the percentage of reduced test cases, fault detection, and limitations of each technique.

Variety of GA-based optimization techniques shows that implementation of correct and suitable fitness function is compulsory to support the general operation in the genetic algorithm itself. The result of a final set of optimized test cases can be more accurate and reliable in optimizing the overall testing process by reducing the total cost and time taken for the entire testing process. Based on discussion stated previously, authors need to consider the usage of the fitness function and also considering several evaluation variables such as a number of reduced test cases, execution time and fault detection capability. These evaluation variables are needed since test case optimization is a multi-objective problem which it can show the trade-off between variables in optimizing the test cases and entire testing process. Besides that, these evaluation variables also can become experimental results to show the efficiency and effectiveness of particular optimization algorithm.

A genetic algorithm is a good optimization and emerging technique which require more researches and study to make it better compared to existing techniques. The basic operation of the genetic algorithm is reliable to produce a good set of optimized test cases but with the addition of fitness function, it makes the overall operation more accurate and produces a better result. Potential future work of this study is to conduct deeper study including empirical experiments and evaluation on each genetic algorithm-based for test case optimization techniques stated in this paper to provide more concrete evidence as in this paper limited due to time constraint. In addition, comparative evaluation between available techniques also can be conducted to obtain more specific and concrete result for most efficient techniques in test case optimization.

6. ACKNOWLEDGEMENTS

The authors would like to express their deepest gratitude to Ministry of Higher Education Malaysia (MOHE) and Universiti for their financial support under Fundamental Research Grant Scheme (Vot number Q.J130000.2528.16H73 and Q.J130000.2516.11H71).

7. REFERENCES

[1] S. Jeyaprakash and K. Alagarsamy, A distinctive genetic approach for test-suite optimization, Procedia Computer Science., vol. 62, pp. 427–434 **(2015)**.

[2] S. Elbaum, G. Rothermel, and J. Penix, Techniques for improving regression testing in continuous integration development environments, Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE 2014), pp. 235–245, **(2014)** 16-21 November; Hong Kong

[3] S. Yoo and M. Harman, Regression testing minimization, selection and prioritization: A Survey, Softw. Test. Verif. Reliab., 24, 8, pp. 591–592 **(2014)**.

[4] M. Rava and W.M.N. Wan-Kadir, A review on prioritization techniques in regression testing, International Journal of Software Engineering and its Applications, 10, 1, p. 221-232 **(2016)**

[5] A. Schwartz and H. Do, Cost-effective regression testing through Adaptive Test Prioritization strategies, J. Syst. Softw., 115, C, pp. 61–81 **(2016)**.

[6] P. Kandil, S. Moussa, and N. Badr, Regression Testing Approach for Large-Scale Systems, 2014 IEEE Int. Symp. Softw. Reliab. Eng. Work., pp. 132–133, **(2014)** November 3-6; Naples, Italy

[7] V. Savsani and M. A. Tawhid, Non-dominated sorting moth flame optimization (NS-MFO) for multi-objective problems, Eng. Appl. Artif. Intell., 63, pp. 20–32 **(2017)**.

[8] L. Cheng, C.-S. Tsou, M.-C. Lee, L.-H. Huang, D. Song, and W.-S. Teng, "Tradeoff analysis for optimal multiobjective inventory model," J. Appl. Math., vol. 2013, no. i **(2013)**.

[9] N. Chaudhary and O.P. Sangwan, Multi Objective Test Suite Reduction for GUI Based Software Using NSGA-II, I.J. Information Technology and Computer Science, 8, 8, pp. 59–65 **(2016)**.

[10] W. Zheng, R. M. Hierons, M. Li, X. Liu, and V. Vinciotti, Multi-objective optimisation for regression testing, Inf. Sci. (Ny)., vol. 334, pp. 1–16, **(2016)**.

[11] R. Singh, Test Suite Minimization using Evolutionary Optimization Algorithms:

Review, 3, 6 pp. 2086–2091 **(2014)**.

[12] B. S. Ahmed, M. A. Sahib, and M. Y. Potrus, Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing, I.J. Eng. Sci. Technol., 17, 4, pp. 218–226 **(2014)**.

[13] S.S.B. Lam, H.P. Raju, U. Kiran, Swaraj, and P.R. Srivastav, Automated Generation of Independent Paths and Test Suite Optimization Using Artificial Bee Colony, Procedia Eng., 30, pp. 191–200 **(2012)**.

[14] AdiSrikanth, N. J. Kulkarni, K. V. Naveen, P. Singh, and P. R. Srivastava, Test Case Optimization Using Artificial Bee Colony Algorithm, Adv. Comput. Commun., 333031, pp. 570–579, **(2011)**.

[15] B. S. Ahmed, T. S. Abdulsamad, and M. Y. Potrus, Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the Cuckoo Search algorithm, Inf. Softw. Technol., vol. 66, pp. 13–29 **(2015)**.

[16] A. Singhal, S. Chandna, and A. Bansal, Optimization of Test Cases Using Genetic Algorithm 1, 2, 3, pp. 367–369 **(2012)**.

[17] A. Mateen, Optimization of Test Case Generation using Genetic Algorithm (GA), 151, 7, pp. 6–14 **(2016)**.

[18] T. L. Holst, Genetic Algorithms Applied to Multi-Objective Aerospace Shape Optimization, J. Aerosp. Comput. Information, Commun., 2, 4, pp. 217–235 **(2005)**.

[19] K. Samatha, S. Chokkadi, and J. Yogananda, A Genetic Algorithm Approach for Test Case Optimization of Safety Critical Control, Procedia Eng., 38, pp. 647–654 **(2012)**.

[20] S. Wang, S. Ali and A. Gotlieb, Minimizing test suites in software product lines using weight-based genetic algorithms, 15th Genet. Evol. Comput. Conf., pp. 1493–1500 **(2013)**.

[21] G. Kumar and P. K. Bhatia, Software testing optimization through test suite reduction using fuzzy clustering, CSI Trans. ICT, vol. 1, no. 3, pp. 253–260 **(2013)**.

[22] M. Last, S. Eyal, and A. Kandel, "Effective Black-Box Testing with Genetic Algorithms," pp. 134–148, **(2006)**.

[23] C. Sharma, S. Sabharwal, and R. Sibal, "A Survey on Software Testing Techniques using Genetic Algorithm," Int. J. Comput. Sci. Issues, vol. 10, no. 1, pp. 381–393, **(2013)**.

[24] G. Kumar and P. K. Bhatia, "Software Test Case Reduction using Genetic Algorithm : A Modified Approach," vol. 3, no. 5, pp. 349–354, **(2016)**.

How to cite this article:
Ismail I M, Wan-Kadir W M N, Hassan R. Comparative evaluation of genetic algorithm-based test case optimization. J. Fundam. Appl. Sci., 2018, *10(2S), 74-90.*