ISSN 2278-3091
Volume 9, No.4, July – August 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse346942020.pdf
https://doi.org/10.30534/ijatcse/2020/346942020

Optimization of Test Cases: A Meta-Heuristic Approach

Prakash B[1], Saleena B[2], Shravan S[3], Vijayanandh R[4]
[1,2,3]School of Computer Science and Engineering, VIT Chennai
[4]School of Information Technology,IBS University,Papua New Guinea

ABSTRACT

Software testing is considered as one of the most challenging and time consuming activity involved in the software development process. Testing involves the creation of test cases that helps to explore the defects present in the software system. Test cases play a major role in the testing process to uncover the critical errors that exist in the system. Thus, it is necessary to prioritize test cases for effective testing. This paper aims at performing a comparative study for the optimization of test cases involved in software development using Meta-heuristic techniques. The weightage of each test case is determined,which helps in optimizing the test cases. The performance is evaluated by comparing two meta-heuristic algorithms, namely Particle Swarm Optimization (PSO) and Genetic Algorithm (GA).The result shows PSO outperforms GA in terms of accuracy, error rate, and execution time for the chosen test case optimization problem.

Key words: Software Testing, Meta-Heuristic techniques, Optimization.

1. INTRODUCTION

Software Testing is the most critical phase in software development to deliver high quality software. Testing aims at detecting the faults that exist in the software that results in bad quality and poor customer satisfaction. At the same time, testing is the most time consuming and expensive process in software construction [1][2]. Almost 40% of the projects' budget, time, and effort spent on testing the software system [3].The development team faces multiple obstacles and impediments in completing the software project demanded by the customer within the estimated cost and stipulated time. With increased rapid development methods and customer demands, it becomes necessary to deliverthe software project with a minimal defect.

In the early days, testing was performed manually, whereas in recent times the testing process is automated with the help of software tools available. Since automated testing requires less time and resources, it becomes a more preferred approach among testing communities in organizations. The success of testing is highly dependent on the test case generated. Thus, it is very much important to optimize the test case for performing the testing process. Meta-heuristic algorithms are the most widely accepted approach in solving optimization problems in the field of software development. In general, meta-heuristic techniques are used to solve complex problems to obtain near optimal solutions. In recent times, many meta-heuristic algorithms were tried to solve different optimization problems as part of software development[4][5][6]. In this study, one such attempt has been tried to understand the performance of two meta-heuristic algorithms namely, Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) for prioritization of test cases.

The main objective of this study is to:

- Implement GA and PSO algorithms for prioritizing the test cases.
- Evaluate the performance of the algorithms based on three criterion attributes namely, accuracy, error rate, and execution time.

This paper is divided into 5 sections. Section 2 describes the existing literature study, section 3 details about GA and PSO algorithms for optimizing test cases. Section 4 represents the results obtained along with the discussion and finally, section 5 concludes the paper with direction for future study.

2. RELATED WORKS

In recent times, there are various studies focused on applying different meta-heuristic techniques in the field of prioritization. Among those, GA and PSO are the most preferred approaches due to their minimal performance tuning parameters and obtaining better optimal results.

Andreas*et.al* [7] proposed a systematic approach to automate the process of test suite generation and optimization. The authors in this study followed the testing approach ofEvolutionary Structural Testing to automatically generate the test cases to achieve high structural code coverage. A fault-based regression test case prioritization using GA was proposed by Arvinder et al., [8]. This study utilized the evolutionary approach in specific, GA for prioritizing test case suite based on total fault coverage with minimal execution time. Rothermel et al., [9] introduced the regression test case prioritization technique based on user-defined criteria. Priyanka et al., performed a comparative analysis of test case prioritization approaches in the field of regression testing [10]. The various methods of test case prioritization such as total branch coverage and additional branch coverage are utilized to improve fault detection [11].

Srivastava et al., introduced a prioritizationtechnique based on the changes introduced in the program and focused on the objective function of impacted block coverage [12]. The prioritization of test case scenarios derived from Unified Modeling Language (UML) diagrams by applying GA was introduced by Sangeeta et al., [13]. In this study, the author proposed prioritization of test case scenarios derived from the activity diagram and state chart diagram of UML using the concept of basic information flow (IF) metric, stack, and GA.GA and PSO algorithms are applied in several optimization problems for generating test cases for functionality testing [14][15], structural testing [16][17], regression testing [18] [19], and many other areas [20][21].

Based on the existing literature study, it is evident that even though several attempts have been made to prioritize the test cases using evolutionary algorithms, as per the author's knowledge, a comparative study of GA with PSO in the field of test case prioritization has not attempted so far.

3. PROPOSED APPROACH

3.1 Particle Swarm Algorithm (PSO)

This section details our proposed approach for test case prioritization using PSO and GA techniques.PSO is a swarm intelligence algorithm introduced by Kennedy and Eberhart in the year 1995 [22]. PSO is mainly used to optimize a problem through multiple iterations based on a random search method. This means that the algorithm generates candidate solutions randomly and alters these solutions according to the quality measure of that candidate solution as iteration progresses. PSO works by initializing candidate solutions for the chosen test case optimization problem as individual particles, each with their velocity, position, and overall inertial weight. For each iteration, the fitness of the particles is calculated. Based on this fitness score, the position value and the velocity of that particular particle are updated using‘update_position’ and ‘update_velocity’ functions respectively. The updation takes place with regard to the personal best position of that particular particle and the global best position of all particles in the solution space.
The fitness function for update velocity is:

$$\boldsymbol{vi(t+1) = w.vi(t) + A + B} \quad \textbf{(1)}$$
$$\boldsymbol{where\ A = c1.r1(t)(xpBest(t) - xi(t))}$$
$$\boldsymbol{and\ \ B = c2.r2(t)(xgBest - xi(t))}$$

and the formula for update position is

$$\boldsymbol{xi = xi + vi} \quad \textbf{(2)}$$

where **vi (t+1) -** velocity after update (at time t+1), **vi (t) -** velocity of the particle before update (at time t), **w -** inertia factor, set at initialization, **c1 -** constant, confidence of the particle, **c2 -** constant, confidence of the swarm, **r1**, **r2 -** random numbers (at time t), $\mathbf{x_{pBest}(t)}$ **-** pBest value of particle (at time t), **xi(t) -** current position of particle (at time t), $\mathbf{x_{gBest}}$ **-** gBest value of swarm, **xi -** position value of particle, **vi -** velocity of particle.
After ‘n’ iterations, all particles tend to move towards the optimum center of the entire swarm. If the termination constraint is achieved the algorithm comes to a halt. The global best (global optimal center) will be considered as the optimal result for the optimization problem. Figure 1 gives a pseudo code for the PSO algorithm.

```
Start
    1.  Define dimension of problem, solution
        space and population size
    2.  Generate random solutions as particles
        (based on pop, lb, ub)
    3.  Initialize particles with position, velocity.
        Set gBest and pBest
    4.  For each iteration in max_iter:
        4.1 For each particle in pop:
                Evaluate fitness
                Update pBest
        4.2 Find gBest
        4.3 For each particle in pop:
                Update velocity
                Update position
        4.4 Evaluate
        4.5 If termination condition met:
                break
        4.6 Else:
                Return to 4.
    5.  Return global optima
Stop
```

Figure 1: PSO pseudo code

3.2 Genetic Algorithm (GA)

Genetic Algorithm (GA) is a meta-heuristic algorithm inspired by the working of natural selection, belonging to a large list of evolutionary algorithms [23]. The algorithm is based on the concepts of *Darwin's Theory of Evolution*, which proves that the most powerful trait is handed down generation-to-generation. Based on this theory, GA is utilized to solve optimization problems by producing better results after multiple generations of possible individual solutions.

GA works by creating a population of possible solutions for the problem and finding the best result after multiple generations. At initialization, an entire population of possible solutions (which act as the first generation) is created. Each chromosome (or genomes) in a population has its own unique set of traits (or genes). For the first iteration, a set of chromosomes are chosen based on the fitness function score. These chosen pairs of chromosomes are used for the re-population of the solution pool for the next generation. For each iteration, the best chromosomes act as parents to re-populate the next generation. The population of the successive generations is obtained by applying some genetic operations on the parent chromosomes. The genetic operations are crossover and mutation. Crossover is the technique of generating new solutions for the successive generation by crossing over (or) interchanging individual genes between two parent chromosomes. Figure 2 and 3 represents the difference in parent chromosomes before and after crossover.

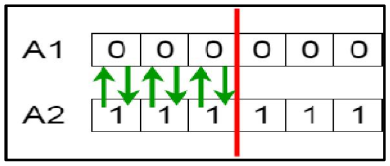


Figure 2: Parent chromosomes before Crossover

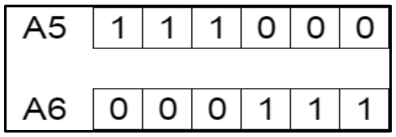


Figure 3: Children chromosomes obtained after crossover

Mutation is a technique of producing new solutions to the successive generation by mutating the solution (or) by altering the genes of the parent chromosome such that a new solution is obtained. Figure4 gives a distinction between the state of the chromosome before and after mutation.

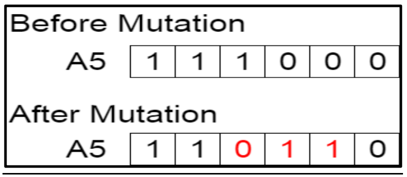


Figure4:Difference in chromosome before and after mutation

There is a continuous *choice-paring-population production* cycle occurring until the adequate size of the population for the next generation is obtained. Once the termination condition is achieved, the algorithm comes to a halt. The result obtained from the algorithm will be the best result from the final generation population with the highest fitness. Figure 5represents a pseudo code of the GA algorithm.

```
Start
    1.  Define the dimension of the problem,
        solution set,and population size
    2.  Generate random solutions as particles
        (based on pop, lb, ub) as population
    3.  Set mutation rate and selective pressure
    4.  For each iteration in max_iter:
        4.1 For each chromosome in pop:
                Evaluate fitness
        4.2 Choose best chromosomes as parents
        4.3 For each pair in parents:
                Crossover
                Mutation
        4.4 Re-populate for next generation
        4.5 If termination condition met:
                break
        4.6 Else:
                Repeat 4.
    5.  Return best chromosome & best fitness
Stop
```

Figure 5: GA Pseudo Code

3.3 Methodology

The system would use the Particle Swarm algorithm to optimize the objective function and hence derive the optimized test suite. The objective function would be the mathematical representation of the “weightage” value found for each test case. The meta-heuristic algorithms consider this objective function to fit the particles or chromosome (in PSO and GA, respectively) and optimize the value of this function. Figure 6 and 7represents the process flow of PSO and GAalgorithms for the test case optimization problem respectively.

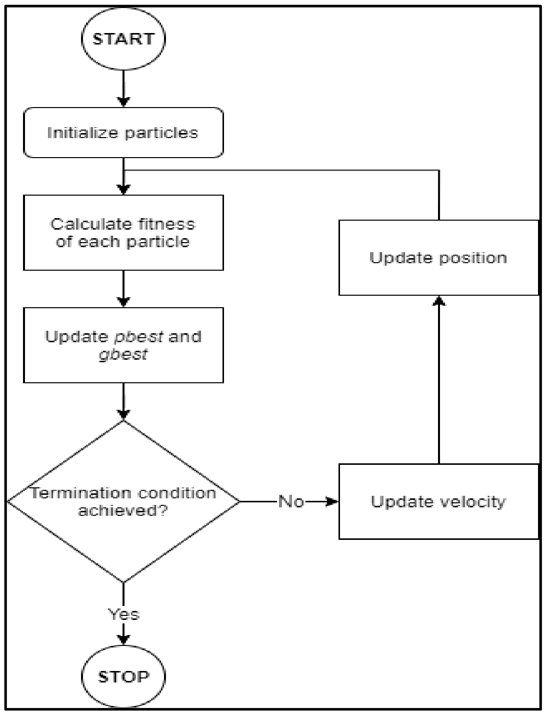

START
Initialize particles
Calculate fitness of each particle
Update *pbest* and *gbest*
Termination condition achieved?
No
Update velocity
Update position
Yes
STOP

Figure 6: PSO Process Flow

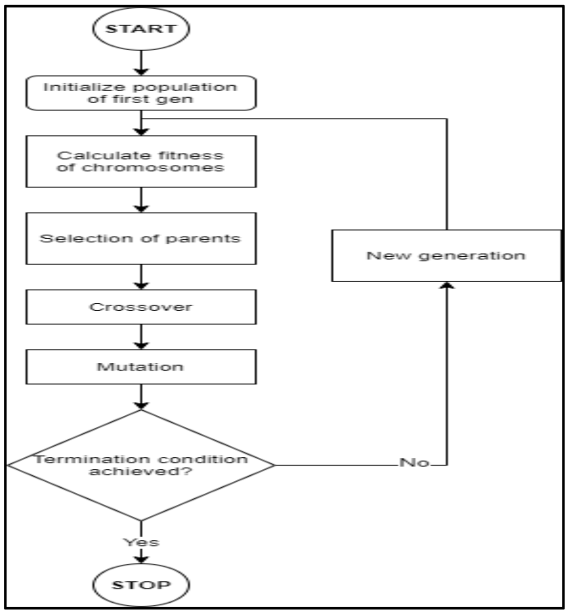

START
Initialize population of first gen
Calculate fitness of chromosomes
Selection of parents
Crossover
Mutation
Termination condition achieved?
No
New generation
Yes
STOP

Figure7:GA Process Flow

Based on the best optimal values obtained, the dataset is re-ordered. The test cases are arranged based on the difference of the weightage value from the optimum obtained from the meta-heuristic algorithm.

4. EMPIRICAL RESULTS

The performance tuning parameters and system configuration used for execution of the system is illustrated in Table1.

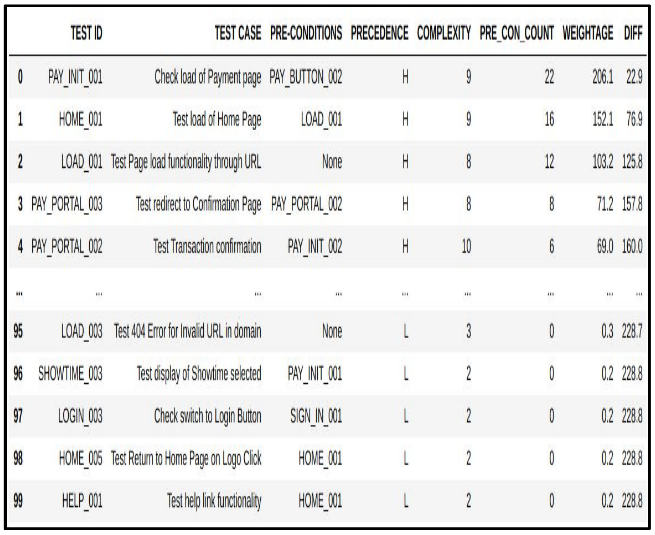
Table 1: System configuration for final results

OS	Linux
Software	Jupyter Notebook, Python
Dimension (objective function)	5
Lower bound	As per dataset
Upper bound	As per dataset
For PSO	
Particles (pop)	100
Max_iter	25
Inertia weight (w)	0.6
c1, c2	0.7
For GA	
Max_gen	25
Population size	100
Parent pair number	4
Selective pressure	1.4
Mutation rate	0.2

4.1 PSO Output

Firstly, the PSO algorithm is used to optimize the test case. The particle swarm optimizer is executed under a number of parameters, which ensure the best possible results obtained.

The output from the algorithm will be the *gBest X* (global best position) and *gBest Y* (global best fitness) values. The *gBest Y* value (representing f(a), where f is the objective function) represents the maximized result that can be obtained as a result from the objective function when using *gBest X* (representing a) as the input. The entire dataset is rearranged based on the difference from the optimal fitness and current weightage and the output will be the test suite optimized by PSO. Figure8 represents the resultant test suite obtained by optimization using Particle Swarm Optimizer.



	TEST ID	TEST CASE	PRE-CONDITIONS	PRECEDENCE	COMPLEXITY	PRE_CON_COUNT	WEIGHTAGE	DIFF
0	PAY_INIT_001	Check load of Payment page	PAY_BUTTON_002	H	9	22	206.1	22.9
1	HOME_001	Test load of Home Page	LOAD_001	H	9	16	152.1	76.9
2	LOAD_001	Test Page load functionality through URL	None	H	8	12	103.2	125.8
3	PAY_PORTAL_003	Test redirect to Confirmation Page	PAY_PORTAL_002	H	8	8	71.2	157.8
4	PAY_PORTAL_002	Test Transaction confirmation	PAY_INIT_002	H	10	6	69.0	160.0
...	...	...	...	...	...	...	...	...
95	LOAD_003	Test 404 Error for Invalid URL in domain	None	L	3	0	0.3	228.7
96	SHOWTIME_003	Test display of Showtime selected	PAY_INIT_001	L	2	0	0.2	228.8
97	LOGIN_003	Check switch to Login Button	SIGN_IN_001	L	2	0	0.2	228.8
98	HOME_005	Test Return to Home Page on Logo Click	HOME_001	L	2	0	0.2	228.8
99	HELP_001	Test help link functionality	HOME_001	L	2	0	0.2	228.8

Figure 8: Output from PSO

4.2 Consistency of PSO

To ensure that the right parameters and attributes are used for the algorithm, the algorithm has been executed for'*m*' instances and the average fitness values and average execution time are recorded. The best possible parameters for PSO found in this study are represented in Table 3.

Table 2 :PSO constants for all instances

Constants for all instances	
Dimensions	5
Particles	10
W	0.6
c1, c2	0.7

Table 3:PSO execution results

S. No	Total no. of executions	Average gBest Y	Average execution time
1	10	229.0	0.00407
2	25	228.9	0.0038
3	50	229.0	0.0038
4	100	229.0	0.0038
5	200	229.0	0.0036

The sub-graphs of Figure 9 represents the execution involved in PSO implementation.

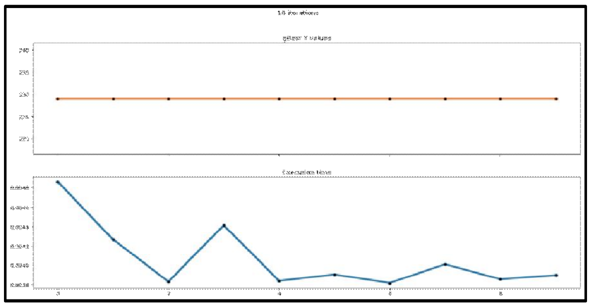


Figure. 9 (a)

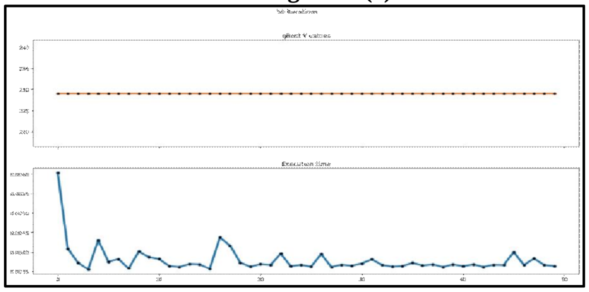


Figure. 9(b)

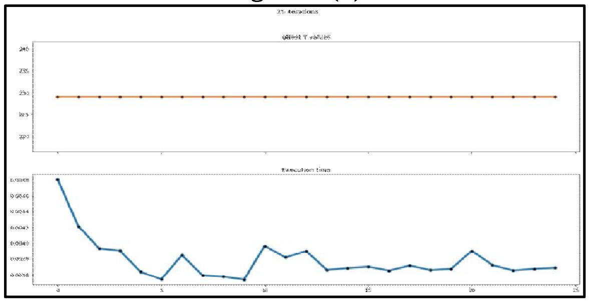


Figure. 9(c)

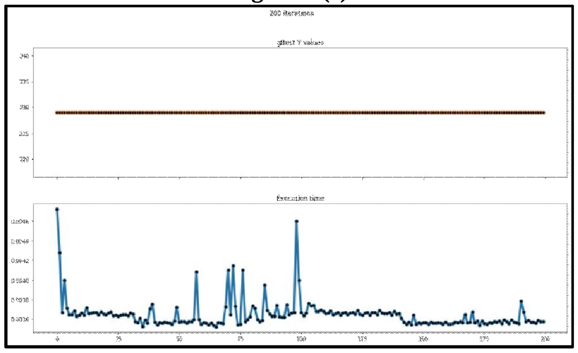


Figure. 9(d)

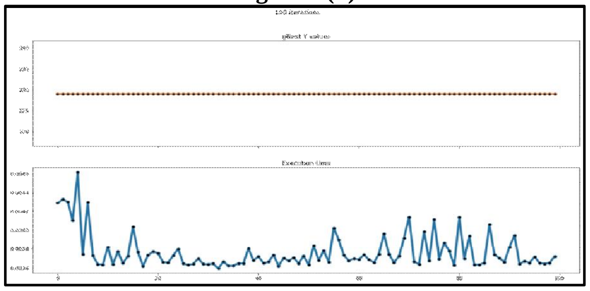
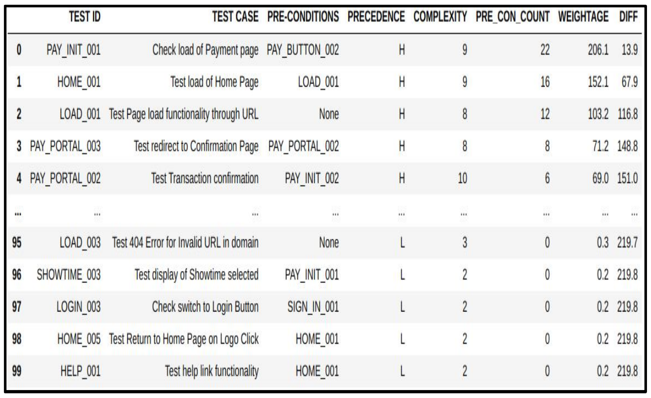


Figure. 9(e)

Figure 9: Execution involved in PSO implementation

4.3 GA Output

The genetic algorithm is executed under certain operating conditions and constraints which preferably provide the best possible results.The algorithm is executed and the results from the execution are recorded. The output from the algorithm will be the ***best*** *genome* and *best fitness* values. The *best fitness* value (representing f(a), where f is the objective function) represents the maximized result which can be obtained as a result from the objective function when using the *best genome* (representing a) as the input. The entire dataset is rearranged based on the difference from the optimal fitness and current weightage and the output will be the test suite optimized by GA. Figure 10 represents the resultant test suite obtained by optimization using the Genetic Algorithm.



	TEST ID	TEST CASE	PRE-CONDITIONS	PRECEDENCE	COMPLEXITY	PRE_CON_COUNT	WEIGHTAGE	DIFF
0	PAY_INIT_001	Check load of Payment page	PAY_BUTTON_002	H	9	22	206.1	13.9
1	HOME_001	Test load of Home Page	LOAD_001	H	9	16	152.1	67.9
2	LOAD_001	Test Page load functionality through URL	None	H	8	12	103.2	116.8
3	PAY_PORTAL_003	Test redirect to Confirmation Page	PAY_PORTAL_002	H	8	8	71.2	148.8
4	PAY_PORTAL_002	Test Transaction confirmation	PAY_INIT_002	H	10	6	69.0	151.0
...	...	...	...	...	...	...	...	...
95	LOAD_003	Test 404 Error for Invalid URL in domain	None	L	3	0	0.3	219.7
96	SHOWTIME_003	Test display of Showtime selected	PAY_INIT_001	L	2	0	0.2	219.8
97	LOGIN_003	Check switch to Login Button	SIGN_IN_001	L	2	0	0.2	219.8
98	HOME_005	Test Return to Home Page on Logo Click	HOME_001	L	2	0	0.2	219.8
99	HELP_001	Test help link functionality	HOME_001	L	2	0	0.2	219.8

Figure 10: Output from GA

4.4 Consistency Of GA

To ensure that the right parameters and attributes are used for the algorithm, the algorithm has been executed form-instances. The best possible parameters for GA found in this study are represented in Table 4.

Table 4 :GA constants for all instances

Constants for all instances	
Genome Length	5
Generations	10
Selective Pressure	1.4
Mutation Rate	0.2

Table 5:GA Execution Results

S. No	**Total no. of executions**	**Average best fitness**	**Average execution time**
1	10	213.6299	0.0680
2	25	213.4600	0.0720
3	50	209.8300	0.0703
4	100	209.7080	0.0671
5	200	212.0204	0.0677

The sub-graphs of Figure 11 represent each of the aforementioned instances.

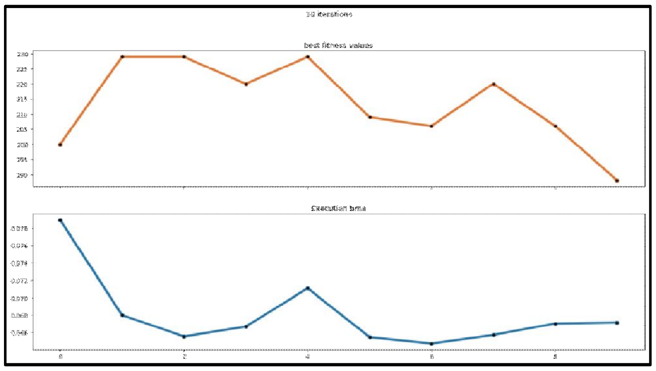


Figure 11(a)

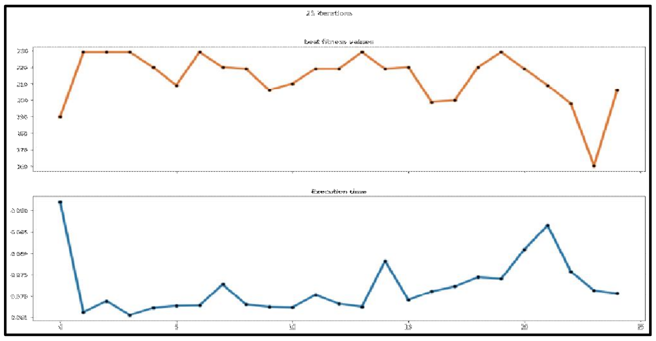


Figure 11(b)

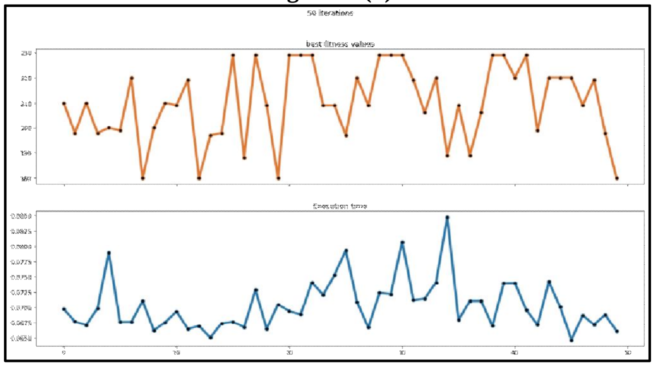


Figure 11(c)

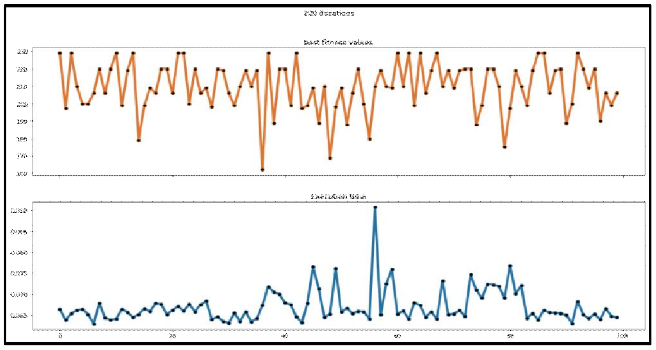


Figure 11(d)

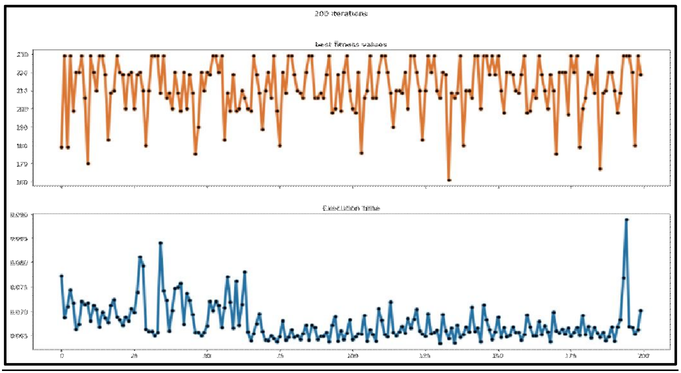


Figure 11(e)

Figure 11: Each of the aforementioned instances.

4.5 Comparative Study

The comparison between the output from the meta-heuristic algorithms takes place as the following steps:

- Check for equality between the two test suites obtained
- Determine the accuracy of the first meta-heuristic algorithm
- Calculate the error rate of the first meta-heuristic algorithm
- Determine the accuracy of the next meta-heuristic algorithm
- Calculate the error rate of the next meta-heuristic algorithm
- Compare the accuracy of the output test suites
- Compare the error rates for the output test suites
- Determine the difference between the times for execution of the algorithm

Firstly, the resultant test suites from both the algorithms are checked if they are equal. This equality might then influence the similar efficiency of the algorithms. Then the accuracy of the output test suites is determined. This is found from the test case ranking obtained from the stakeholders. The formula for accuracy is given by

$$\boldsymbol{Accuracy}\ (\%) = (\boldsymbol{TC_{ranked}} \,/\, \boldsymbol{TC_{total}}) * \mathbf{100} \qquad (3)$$

where $\mathbf{TC_{ranked}}$ - number of test cases which are in the correct rank, $\mathbf{TC_{total}}$ - total number of test cases in the dataset. Based on this accuracy value, the value for error rate is also determined using the formula

$$\boldsymbol{Error\ rate}\ (\%) = \mathbf{100\%} - \boldsymbol{\%\ of\ accuracy} \qquad (4)$$

After the accuracy and the error rates for both the algorithms, their execution time is found. The results are recorded in Table 6.

Table 6: Comparative Study

Iter. No.	Meta-heuristic algorithm	Accuracy %	Error %	Execution Time
1.	PSO	85%	15%	0.41721
	GA	83%	17%	0.41259
2.	PSO	83%	17%	0.41567
	GA	83%	17%	0.40447
3.	PSO	84%	16%	0.399785
	GA	82%	18%	0.412262

These values are used to determine the efficiency of the meta-heuristic algorithm. The relation between the efficiency and values mentioned in Table 6 are given below

System Efficiency ∝ % of accuracy
System Efficiency ∝ 1 / % of error
System Efficiency ∝ 1 / execution time

Where,*System Efficiency* is the efficiency of the sub-system which uses the particular meta-heuristic algorithm to solve the optimization problem.
From Table 6, it is clear that, among the 3 iterations, the PSO algorithm shows higher accuracy and lower error rate. Even though the execution time for GA is less in 2 iterations, PSO remains consistent with its accuracy and low error rate.

5. CONCLUSION

The proposed approach is used to optimize the test cases and finds the associated fitness valuesfor each test case. The maximization of this fitness value is considered as the best optimalsolutionfor the chosen problem. By implementingPSO and GA algorithms, the optimal solution is determined and the test suite is prioritized accordingly. Further, the results of both PSO and GA algorithms are compared in terms of three criterion attributes namely, accuracy, error rate, and execution time.
From the comparative study, it is clear that GA is best suited for lightweight problems which are much faster to optimize whereas PSO works best in solving problems where heavy computation is required. The PSO exhibits more accurate results while the Genetic Algorithm provides faster execution. The result and conclusion depend completely on the sample test case set the input to the system. The user is presented with both the resultant test suites, from which they might choose the most favourable one, taking factors such as testing environment, stakeholder requests, and expectations into consideration.
The future direction might be focused on: (i) Implementing different meta-heuristic algorithms for this test case optimization problem, (ii) a hybrid approach, which is a combination of two meta-heuristic algorithms that can be attempted to understand the behavior on the chosen prioritization problem.

REFERENCES

1. Dr. VelurRajappa, ArunBiradar, Satanik Panda, **Efficient software test case generation Using Genetic algorithm based Graph theory**International conference on emerging trends in Engineering and Technology, pp. 298--303, IEEE (2008). https://doi.org/10.1109/ICETET.2008.79
2. Praveen RanjanSrivastava and Tai-hoon Kim, **Application of Genetic algorithm in software testing**, International Journal of software Engineering and its Applications, vol.3,No.4, pp. 87--96 (2009).
3. ChartchaiDoungsa-ard, KeshavDahal, AlamGirHossain and TaratipSuwannasart, **An automatic test data generation from UML state diagramusing genetic Algorithm**".http://eastwest.inf.brad.ac.uk/ document/publication/Doungsa-ard-KIMA.pdf.
4. Hilary I O., Nkolika J P., Adedotun O A., Chukwuemeka O I., Akunnaya P O. **Smart review of the applications of Genetic Algorithm in Construction and Housing.** Internatinal Journal of advanced trends in Computer Science and Engineering, vol. 9, no. 1, pp. 266 – 273, 2020. https://doi.org/10.30534/ijatcse/2020/40912020
5. Prakash B., Viswanathan V. **A comparative study of meta-heuristic optimisation techniques for prioritisation of risks in agile software development.** Int. J. Computer Applications in Technology, vol. 62, no. 2, 2020.
6. Lucija B., Iztok F., Vili P. **Solving agile software development problems with Swarm Intelligence Algorithms.** International conference in new technologies, development and applications, vol. 76, pp. 298 – 309, Springer, 2019. https://doi.org/10.1007/978-3-030-18072-0_35
7. Ali A. Al-Jadaa (2014). **Software Metrics**,Birzeit University of Engineering and Information Technology, Master of Computing-Software Engineering Course 2014
8. Arvinder K., Shubhra G., **A Genetic Algorithm for Fault based Regression Test Case Prioritization**, International Journal of Computer Applications, vol. 32, no. 8, 2011.
9. Rothermel, G., Untch, R., Chu, C. and Harrold, M.J. **Prioritizing test cases for regression testing,** IEEE Transactions on Software Engineering, vol. 27, No.10, October 2001, pp.929–948.
10. Priyanka P., Shashank S., Debnath B., Taihoon K. **Comparative analysis of test case prioritization approaches in regression testing.** International Journal of advanced trends in Computer Science and Engineering, vol. 8, no. 4, pp. 1260 – 1267, 2019. https://doi.org/10.30534/ijatcse/2019/36842019
11. Elbaum, S., Malishevsky, A. G. and Rothermel, G. **Test case prioritization: A family of empirical studies**. IEEE Transactions on Software Engineering, vol.28, No. 2, 2002, pp.159–182.
12. Srivastava, A. and Thiagarajan, J. **Effectively prioritizing tests in development Environment**, in Proceedings of the ACM SIGSOFT international symposium on Software testing and analysis, USA, 2002, pp. 97–106.
13. Sangeeta S, Ritu S, Chayanika S, **Applying Genetic Algorithm for Prioritization of Test Case Scenarios derived from UML Diagrams**, Internationa Journal of Computer Science, vol. 8, no. 3, 2011.
14. Nirmal Kumar Gupta and Dr. Mukesh Kumar Rohil ,**Using Genetic algorithm for unit testing of object oriented software**, First International conference on Emerging trends in Engineering and technology, IEEE (2008).

15. Francisca Emanuelle, Ronaldo Menezes, Marcio Braga, **Using Genetic algorithms for test plans for functional testing**, ACM (2006).
16. Praveen RanjanSrivastava and Tai-hoonKim ,**Application of Genetic algorithm in software testing**, International Journal of software Engineering and its Applications, vol.3,No.4, pp. 87--96 (2009).
17. Mahaalzabidi, Ajay Kumarand A. D. Shaligram, **Automatic software structural testing by using evolutionary algorithms for test data generations**, IJCSNS International Journal of Computer science and Network Security, Vol.9, No.4 (2009).
18. Robert M .Patton, Annie S. Wu, and Gwendolyn H .Walton, **A Genetic Algorithm approach to focused software usage testing**, Annals of softwareengineering,.http://www.cs.ucf.edu/~ecl/papers/03.rmpatton.pdf
19. Omdev D., Kamna S., Sandeep D. **Comparative analysis of regression test case prioritization techniques.** International Journal of Advanced Trends in Computer Science and Engineering, vol. 8, no. 4, pp. 1521 – 1531, 2019. https://doi.org/10.30534/ijatcse/2019/74842019
20. Nirmal Kumar Gupta and Dr. Mukesh Kumar Rohil, **Using Genetic algorithm for unit testing of object oriented software**, First International conference on Emerging trends in Engineering and technology, IEEE (2008).
21. Jose Carlos, Mario, Alberto, Francisco, **A strategy for evaluating feasible and unfeasible test cases for the evolutionary testing of object-oriented software**, ACM (2008).
22. Kennedy J, EberhartR,**Particle Swarm Optimization**, Proceedings in IEEE International Conference on Neural Networks, Australia, pp. 1942 – 1948, 1995.
23. Goldberg D E, **Genetic algorithms in search, optimization**, Summary the applications of GA – Genetic Algorithm for dealing with some optimal calculations in Economics, Addison Wesley, 1989.