‘Ambha’: A Metaheuristic Framework for Test Case Suite Optimization

Bhawna Jyoti[1], Aman Kumar Sharma[2]
[1]Research Scholar, Computer Science Department, Himachal Pradesh University, Shimla, India
[2]Professor- Computer Science, Himachal Pradesh University, Shimla, India.

Abstract
Background: To deal with resource and time constraints in retesting of modified software, it is really vital as well as a significant issue to look for regression testing strategies that keeps up with performance as well as quality of modified software. To meet this objective, regression test case suites should be optimized to check functionalities of modified software.
Objectives: In regression testing scenario, this study aims to identify an optimized test case suite in order to maximize its fault revealing potential as well as reduction in its execution time.
Methods: Authors developed a metaheuristic framework that consists of three phases, i.e., (i) test case suite reduction using ant system-based classification rule discovery (ii) test case suite selection using Whales as search agents (iii) test case suite prioritization using grey wolf optimization algorithm. The ‘Ambha’ framework is implemented on 12 test case suites taken from GitHub and TravisCI code repositories.
Results: The results reveal that ‘Ambha’ framework has significantly improve five performance metrices. Specifically, Average Percentage of Fault Detection metric is improved up to 56.4% and execution time is reduced by 19.7%.
Conclusions: The proposed metaheuristic framework ‘Ambha’ has shown promising results for regression testing problems as compared to framework (ACO+WOA+GWO).
Keywords:grey wolf optimization algorithm, ant colony optimization algorithm, whale optimization algorithm, test case suite prioritization, test case suite reduction, test case suite selection, regression testing.

NOMENCLATURE
Average Percentage of Fault Detection (APFD), Fault Detection Ratio (FDR), Classification Accuracy (CA). Nondeterministic Polynomial (NP), Test Case suite Prioritization (TCP), Test Case suite Selection (TCS), Test Case suite Reduction (TCR), test case suite reduction ratio (TSRR)

1. Introduction

Software testing is a significant action that includes huge efforts to recognize deficiencies, faults and errors by observing whether the maintained software produces right result and works on the nature of programming. It is the costliest practice since resources of creating and keeping up with updating software are connected with the testing system for guaranteeing correct functioning of software. In present-day period, researchers put their enormous efforts to deal with time and resources concerned with issues of regression testing problems. At the point when updated software spread its aspects, its experiment regression suite should be refreshed as well as cope up with the everchanging needs of user expectations [1][2][3][10].
In regression suite optimization procedures, heuristic algorithms-based calculations are very tedious, require more memory depending on how to address problems of large test case suites. Local optimal solutions incorporate local search space and are not able to exploit global search space for finding optimal solutions. There is a strong need to invest more amounts of energy to develop a framework based on metaheuristic algorithms for optimizing the test case suite. This paper is organized as: Section 2 gives research gaps in the existing literature. Section 3 highlights concept of proposed algorithm and Section 4 provides experimental study. Further, Section 5 provides statistical test measures to validate study followed by discussion in Section 6. Lastly, Section 7 concludes the study.

2. Research Gaps

Indumathi and Madhumathi [42] proposed a prioritization technique that is implemented based on most frequent test selection criteria. Cardinality number is assigned to prioritize test case suite and genetic algorithm prioritizes selected test cases by ranking them according to cardinality number. Promising results are obtained in APFD metric as well as reduction in execution time is observed. But small test case suite size is considered for their empirical study.

Siddik and Sakib [13] proposed a prioritized framework based on software requirements covered, source code and test case suites. Service layer contains processing units of software requirement specification and design-based diagrams are used to get prioritized test case suite. APFD metric is improved. Academic Time Synchronization Project (ATSP) is taken for study. This project consists of small test case suite having only ten test cases that leads to a research gap.

Mittal and Sangwan [14] used metaheuristic algorithm -Genetic, Cuscuta and Artificial bee colony for test case prioritization problem. Global optima are obtained with good variations in results in these three metaheuristic algorithms. Genetic algorithm outperforms the other two algorithms in terms of APFD metric. But only Nine test cases in a suite are taken for implementing metaheuristic techniques.

Singh et al. [15] considers Prioritization with Ant colony optimization. Ants foraging behaviour is responsible for searching best optimal solution of test case selection problem. Selected test cases are prioritized by assigning a cardinality number to them. Good values of APFD metric are obtained.Only eight test cases (having ten faults) are chosen for their empirical study that leads to research gap.

Ahmad et al. [16] used test factors-based optimization technique. Ant colony algorithm is used in addition to time complexity, volatility, code coverage and fault coverage (defined as test vectors) for optimizing test case suite. Requirement matrix is prepared to cover test case requirements. Good algorithm is provided by considering important test factors. Implementation work is not discussed by authors.

Mohapatra and prasad [17] bestowed prioritization with Genetic algorithm. Two fitness functions are used for measuring test adequacy and Execution time in implementing Genetic Algorithm. Crossover operation is used to get better candidate solutions. Emma tool is used for code coverage. APFD metric has good values as compared to random order of test cases execution. Only three java programs with five faults are considered for their empirical study.

Carlson et al. [18] used clustering technique in regression testing problem. Prioritization is being done on multi-coverage criteria – historical information of test cases, code complexity parameters and code coverage. Reduction in execution time and increase in fault detection effectiveness of test case suite is obtained. Financial subsystem of Microsoft Dynamics project having small number of test cases (ten) is used for this study.

Harrold et al. [43] presented ‘H’ (heuristic) test suite reduction algorithm. This algorithm is based on requirement coverage matrix that is formulated set of test cases corresponds to particular test cases. There is reduction in size of test case suite. No attention is paid on fault revealing capability of reduced test case suite.

Chen and Lau [44] presented 'GRE' heuristic-based test case suite reduction. The Greedy Redundant essential heuristic defines that test cases are classified by using the greedy algorithm and then redundant test cases are identified to remove them from the test case suite. Essential test cases are given importance and included in reduced test case suite. Test case size is significantly reduced. This heuristic is not able to find global optimal solution and stuck to local optimal solution. No attention is paid on fault detection potential of test case suite.

Tallam and Gupta [45] highlights A clustering technique is used by authors to reduce test cases in a test case suite. Smaller test case suite size is obtained. Single criterion is used to remove redundant test cases without using second criteria may lead to the removal of some important test cases.

Lin and Huang [46] presented enhanced tie-breaking algorithm for test case suite reduction. This technique is based on tie braking of equal priority test cases. Those test cases which cover less requirements will be removed from resultant test case suite. Fault detection rate is improved. Time complexity is increased, small test case suite is considered.

Black et al. [47] propositioned model-based test suite reduction technique. This technique is based on integral programming constraints equations that are set to minimize function to get desired reduced test case suite. Less execution time of reduced test case suite. Test case suite size is small.

Bhawna et al. [51] highlights genetic algorithm TCS in regression testing. Genetic operators are used to assign cardinality number to most frequent occurring test cases which are designed to reveal faults in a source code. APFD metric is increased. Test case suite considered for study is of small size.

Mohapatra and Pradhan presented genetic algorithm TSR [48]. Genetic algorithm is designed for test case reduction problem. Test case size is decreased. No focus on fault revealing the potential of reduced test case suite.

Mirarab and Tahvildari [49] used prioritization with Bayesian network. Bayesian network-based prioritization involves code change modifications, fault-proneness as well as coverage of test data. APFD metric is improved. High execution time is observed due to long interference time taken.

Hla et al. focused on prioritization with Particle Swarm based optimization [50]. Test cases are prioritized based on position vectors used by search agents. Software code units is used as a primary criterion for test case selection. Execution time is reduced. A little experiment suite having 20 test cases is picked for their work.

None of these investigations have had the option to propose a framework for test case suite optimization. In this way, there is a need put forward a framework based on metaheuristic algorithm.

3. Proposed work

The proposed framework consists of three phases:

3.1 Phase 1: Enhanced 'Ambha_ACO' algorithm for TSR

Ant system is used to find classification rules which decides whether a particular test case will be included in reduced test case suite or not. Construction of classification rules depends on foraging behaviour of ants and best rule is constructed by traversing all test cases present in a test case suite. We have used 'requirement coverage by test case' as a primary criterion. Additional criteria include test case software running platform, test data dependency for executing a particular test case, missing test case, non-relevant test case and duplicate test case. Probability to select a particular test case by ants to create an empty RuleList from source m to destination n having concentration pheromone τ_{mn} is given below:

$$Prob_{mn} = \frac{x_{mn}.\tau_{mn}}{\sum_{m=1}^{a} x_m . \sum_{n=1}^{b} x_{mn}.\tau_{mn}} \quad (1)$$

We determine quality of rule created by using following function:

$$Fun = \frac{true\_pos}{true\_pos + false\_neg} + \frac{true\_neg}{false\_pos + true\_neg} (2)$$

```
Start of algorithm
Initialize: No_ants and Conc_pheromone
Input: Suite_testcase
testcaseSelect_List ⟶ Φ (Initial empty list)
while (untraversed test_case)> set of test cases in suite
do
calculate HValue ();
BestRuleCreated ⟶ Φ (Initial value empty)
antsProduced ();
RuleProduced for TestCaseList ();
 Class_identify ();
qualRule_Fun ();
Verify quality (current_Rule for selecting test case)        Best (rule);
LRules_created ⟶ LRule+Best(rule);
Train_testSuite= Train_testSuite -Test case traversed in rule creation;
End
Output: Red_test_case_suite.
```

Fig. 1 Pseudocode of phase 1: 'Ambha_ACO'

2. Phase 2: Enhanced 'Ambha_WOA' algorithm for TCS

To solve real world optimization problems, Lewis and Mirjalili in 2015 coined whale optimization algorithm'. The basic equations involved in it are explained as under:

Table I: Equations Used in 'Ambha_WOA'

Encircling prey Phase	
$\vec{D} = \lvert\vec{c}.\vec{X}1(t) - \vec{X}(t)\rvert$	here collective behaviour is determined by $\vec{D}$
$\vec{X}\ (t+1) = \vec{X}1(t+1) - \overrightarrow{A1}.\vec{D}$	$\vec{X}$ represents position vector of whale and t gives iteration number

$\overrightarrow{A\,1} = 2.\vec{a}.\overrightarrow{r1} - \vec{a}$	$A\,1$ represents coefficient position vector.
Intensification and exploration Phase	
$\vec{X}(t+1) = \begin{cases} X1(t) - \overrightarrow{A1}.\vec{D}, if p \leq 0.5 \\ D.e^{bl}.\cos(2\pi l) + Y1(t)\ if p \geq 0.5 \end{cases}$	spiral mechanism is followed by whales for attacking prey.
$\vec{D} = \lvert\vec{C}.\vec{X}1random - \vec{X}\rvert$	$X1\,random$ is used to update position of whales.
$\vec{X}\ (t+1) = \vec{X}1(random) - \overrightarrow{A1}.\vec{D}$	next position of whale is updated by using this equation.

Our enhanced 'Ambha_WOA' includes mutation and crossover operators which increases probability of exploiting search space globally. APFD metric of each

Initialize whales' population P_i (i=1,2,3.... n); set object_fun= APFDmetric
while(x<maxIter)
for every candidate_sol s,calculate m_rate (m),
amend a, A1, c1 and pr values,
if (pr<0.5 && |A1| <1)
put on mutation functionto get Xw^* (best solution)and usem_rate (m) to get $Xw^{mutation}$, make crossover_fun
elseif2(|A1|>1); selectcandidate solution (random whale) Xw,
Apply mutation operator on Xw; get $Xw^{mutation}$
complete crossover_fun and give new position number of Xwfor crossover output; end if2,elseif1(pr>=0.5), end if1,end for,
evaluate object_fun for each candidate solution,
compare candidate solution and update Xw^*,
x=x+1, end while, yield X^{best}
Output: select_testCaseSuite.

Fig. 2 Pseudocode of phase 2: 'Ambha_WOA'

3. Phase 3: An enhanced 'Ambha_GWO' algorithm for TCP:

Mirjalili coined term 'Grey Wolf optimization' in 2014. Grey Wolves encircle prey using its vector position. The basic mathematical equations for GWO algorithm are described below:

Table II: Equations Used in 'Ambha_GWO'

$\overrightarrow{A1'}(t+1) = \overrightarrow{A1p}(t) + \overrightarrow{B1}.\overrightarrow{Z1}$	$\overrightarrow{A1'}$ denotes wolf position, $\overrightarrow{A1'p}$ denotes prey position and $Z1$ store variable position from time to time.
$\overrightarrow{Z1} = \vec{G}.\overrightarrow{A1p}(t) - \vec{A}1(t)$, $\overrightarrow{B1} = 2.\overrightarrow{B1}.\overrightarrow{r_1} - \vec{a}$, $\vec{g} = 2.\overrightarrow{r_2}$	$\overrightarrow{B1}$ and $\vec{g}$denotes coefficient vectors, $\vec{a}$ has its decreasing value in random range $[r_1, r_2]$.
$\overrightarrow{Y_\alpha} = \lvert\overrightarrow{G_1}.\overrightarrow{A_\alpha} - \vec{A}\rvert$, $\overrightarrow{Y_\beta} = \lvert\overrightarrow{G_2}.\overrightarrow{A_\beta} - \vec{A}\rvert, \overrightarrow{Y_\delta} = \lvert\overrightarrow{G_3}.\overrightarrow{A_\delta} - \vec{A}\rvert$	$\overrightarrow{Y_\alpha}$, $\overrightarrow{Y_\beta}$ and $\overrightarrow{Y_\delta}$ represents values of three wolves involved in hunting process.
$A(t+1) = crossover(X_1, X_2, X_3)$	X_1, X_2, X_3 are three moving-position vector representations of wolves involved in crossover operation.
$X_1^d = \begin{cases} 1, if(A_\alpha^d + bstep_\alpha^d) \geq 0 \\ 0,\ 0\ otherwise \end{cases}$	X_1^d represents new position-vector of alpha wolf in dimensional space d and $bstep_\alpha^d$ represents binary step of alpha wolf towards hunting of prey.
$X_2^d = \begin{cases} 1, if(A_\beta^d + bstep_\beta^d) \geq 1 \\ 0,\ 0\ otherwise \end{cases}$ $X_3^d = \begin{cases} 1, if(A_\delta^d + bstep_\delta^d) \geq 1 \\ 0,\ 0\ otherwise \end{cases}$	X_2^d represents new position-vector of beta wolf in dimensional space d and $bstep_\beta^d$ represents binary step of beta wolf towards hunting of prey.

$A^d(t+1) = \begin{cases} Y_1^d, if r < \frac{1}{3} \\ Y_2^d, if \frac{1}{3} < r < \frac{2}{3}, \\ Y_3^d otherwise . \end{cases}$	Output of crossover operator showing best three values in search space d.

Grey wolves are used as search agents and basic code involved in it is represented as:

Initialization: population of Grey Wolves and position vector parameters B1, Z1 and g.
Set objective function(x) =APFD
While loop to traverse all test cases until priority number is assigned to each test case.
do
calculate $\overrightarrow{Y_\alpha}, \overrightarrow{Y_\beta}, \overrightarrow{Y_\delta}$
Search for new position-vectors of prey and hunting wolves.
Evaluate objective function for allocating search positions of wolves and prey.
Assign p (priority number) and if (p==already allocated to test case)
set l (low priority); elseset h (higher priority)
End while loop
output: A Prior_testcase_suite

Fig. 3 pseudocode of phase 3: 'Ambha_GWO'

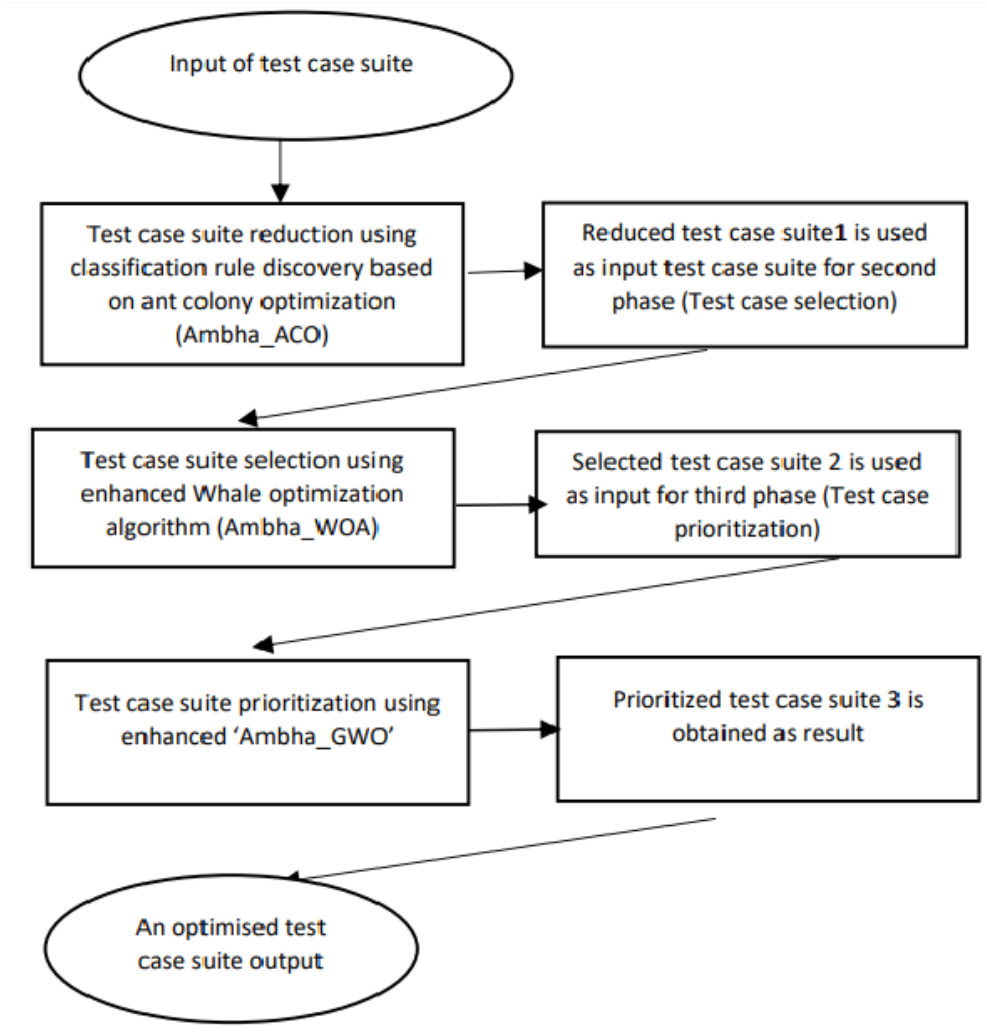


Fig. 4 Flow diagram of 'Ambha' Framework

4. Results

We considered junit test case suites (having test cases range between 100-400) and for source code coverage EclEmma tool is taken. Test cases are executed to know the statements covered as well as faults detected and converted to feature vector space notation (instances*features). The proposed framework is implemented on junit test case suites of open-source code repositories present on GitHub, TravisCI and Spring [23][24][25].

Table III: Dataset description

program	test case classes	test cases in original test case suite	versions	faults
kan_bank	74	156	2	16
kafta_shop	55	165	2	14
delta_lib	56	188	2	15
reg_prt_wes	96	159	2	14
stud_api	81	170	2	16
oryx_banking	89	178	2	15
omega_lib	90	220	2	14
max_hosp	91	262	2	17
ap_axi	15	350	2	12
studentApi	28	322	2	18
u_bank	35	359	2	19
oryx_kafta	42	358	2	12

Table IV shows different parameters taken for test case suite reduction, test case suite selection and test case suite prioritization for regression testing problems.

Table IV: study of parameters

Phases	Parameters taken

TSR by using Ambha_ACO.	Populationof ants = [50,500]
	total number of iterations = 500,
	Evaporation factor(pheromone) = 0.90,
	pher_value (tau) =1,
	heur_ probability(eta) =1,
	cont_ pheromone(alpha) =1,
	cont_ heuristic(beta) =0.1;
	pher_ trail decay =0.2;

TCS by using Ambha_WOA.	Number of whales=10, total number of iterations =100 Lower bound=10 Upper bound=10
TCP by using Ambha_GWO.	Number of wolves as candidate solutions taken (10), Maximum number of iterations (100) Lower bound for wolves used as search agents (10), Upper bound for wolves used as search agents(10)

We considered five performance measurement metrics for the assessment of our proposed framework 'AMBHA' used by various researchers [27,29,34]. In the study, the performance metrics, i.e., Fault detection ratio, test case suite reduction ratio, APFD metric, execution time and F-measure are used to deal with test case suite regression testing problems.

A. Fault Detection Ratio (FDR): This metric considers about number of faults identified by optimized test case suite as compared to original test case suite. Table 5 describes comparative study of framework (ACO+WOA+GWO) and Ambha.

Table V: Description of Fault detection ratio (FDR)

Sr. No.	program	Fault detection Ratio (%) framework (ACO+WOA+GWO)	Fault detection Ratio (%) Ambha	Increase

1	kan_bank	76	100	24
2	kafta_shop	67	90	23
3	delta_lib	64	90	26
4	reg_prt_wes	63	88.8	25.8
5	stud_api	62	87.5	25.5
6	oryx_banking	89	100	18
7	omega_lib	61.5	93.7	32.2
8	max_hosp	72.5	93.7	21.2
9	ap_axi	71.6	91.6	20
10	studentApi	82.6	92.8	10.2
11	u_bank	73.5	92.8	19.3
12	oryx_kafta	72.9	93.7	20.8

It is seen in Table V, FDR metric values showed good results in kan_bank (100), oryx_banking (100) by using Ambha framework. Graphically, FDR metric comparison of these two frameworks can be represented as shown in Figure 5:

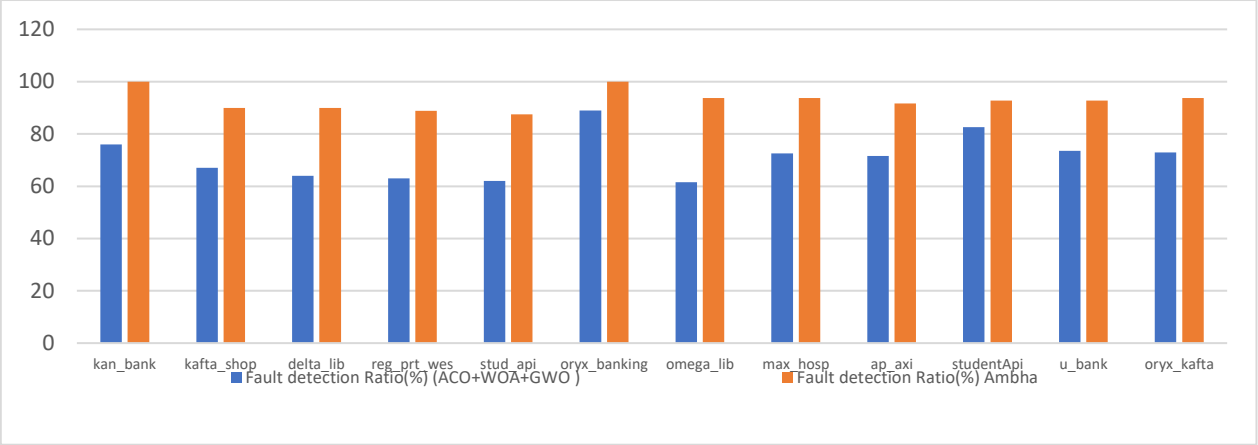


Fig. 5Comparative study of fault detection ratio metric of two frameworks.

B. Test case suite Reduction Ratio (TSRR)

It depicts size reduction of optimised test case suite to the original test case suite. Table VI describes comparison between two frameworks for TSR metric.

Table VI: Description of Values of TSRR

Sr No.	Program	Test case suite reduction ratio (%) using framework (ACO+WOA+GWO)	Test case suite reduction ratio (%) using Ambha	Increase
1	kan_bank	38.1	56.8	18.7
2	kafta_shop	39.3	66.6	27.3
3	delta_lib	59.5	80.4	20.9
4	reg_prt_wes	51.9	79.6	27.7
5	stud_api	49.8	76.1	26.3
6	oryx_banking	39.5	67.2	27.7
7	omega_lib	31.5	54	22.5
8	max_hosp	25.4	53.6	28.2
9	ap_axi	32.8	53.2	20.4
10	studentApi	25.5	50.5	25
11	u_bank	28.2	50.9	22.7
12	oryx_kafta	59.5	83.3	23.8

Datasets i.e., oryx_kafta (83.3), delta_lib (80.4), reg_prt_wes (79.6) and stud_api (76.1) show good values of test case suite reduction in case of Ambha framework in contrast to framework (ACO+WOA+GWO). Further, it is perceived that there is increase in TSR value of max_hosp (28.2) to a great extent. It is addressed graphically as displayed in Figure 6:

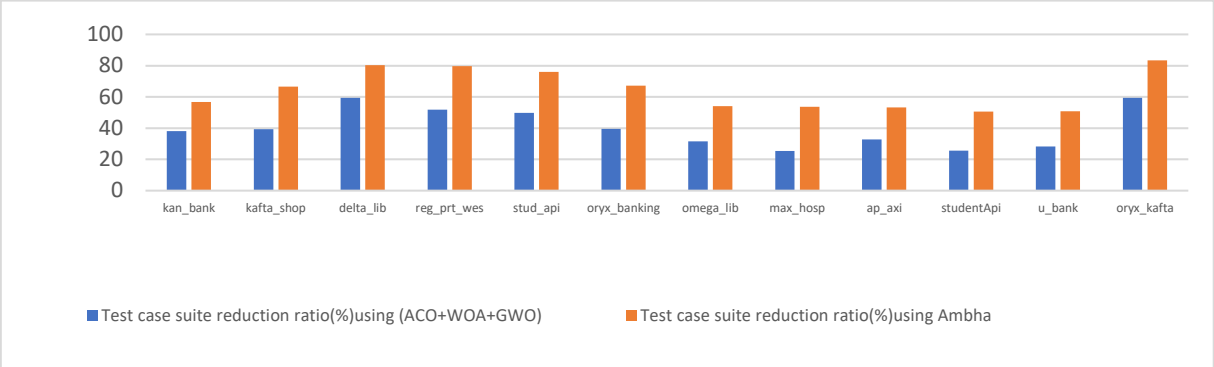


Fig. 6Comparative study of test case suite reduction ratio metric of two frameworks.

C. APFD metric

This metric is very important as well useful to detect average percentage of faults detected by a given test case suite.

$$APFD\ metric = 1 - \frac{1}{m\ (faults) * n(test\ cases)} \sum_{i=n}^{m} \mathrm{TF} + \frac{1}{2n}$$

The following Table 7, represents comparative study of APFD metric between framework (ACO+WOA+GWO) and Ambha for test case suite optimization.

Table VII: Description of Average Percentage of Fault Detection (APFD) metric

Sr No.	Program	APFDusing random order of execution	APFDusing framework (ACO+WOA+GWO)	APFDusing Ambha
1	kan_bank	0.412	0.613	0.911
2	kafta_shop	0.431	0.812	0.924
3	delta_lib	0.424	0.721	0.931
4	reg_prt_wes	0.456	0.652	0.921
5	stud_api	0.457	0.564	0.916
6	oryx_banking	0.458	0.654	0.909
7	omega_lib	0.432	0.753	0.912
8	max_hosp	0.468	0.632	0.965
9	ap_axi	0.498	0.611	0.928
10	studentApi	0.412	0.564	0.919
11	u_bank	0.431	0.599	0.916
12	oryx_kafta	0.424	0.623	0.922

The values of APFD metric for programs delta lib (0.931), max_hosp (0.965), ap_axi (0.928) and oryx_kafta (0.922) had significantly improved by using Ambha as compared to framework (ACO+WOA+GWO). The pictorial representation is given in following Figure 7:

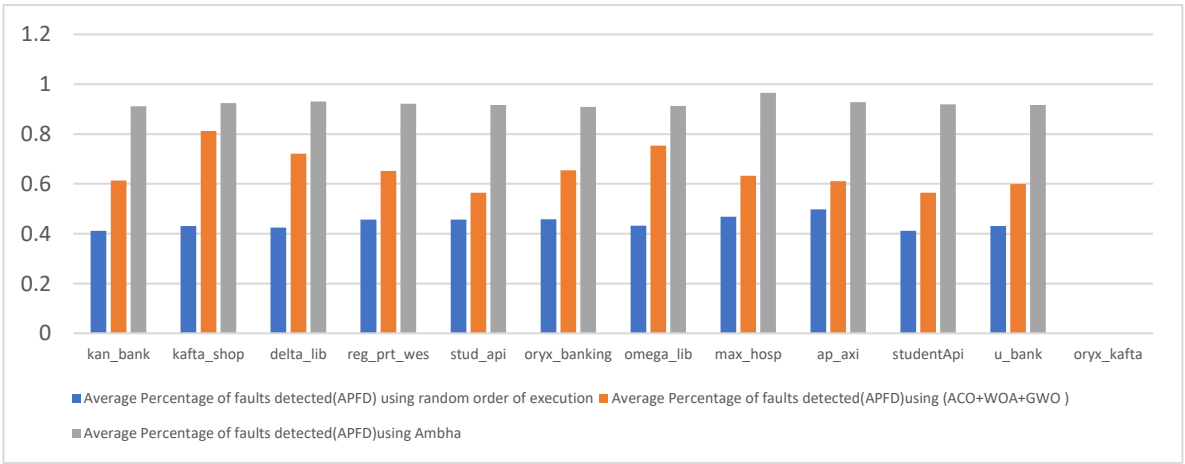


Fig. 7Comparative study of APFD metric

D. Execution Time

It is time taken for executing test case suite of source code program. Table 9 depicts comparative values of execution time in Ambha and framework (ACO+WOA+GWO).

Table VIII: Description of execution time values

Sr No.	Program	Execution time in sec (ACO+WOA+GWO)	Execution time in sec (Ambha)	Decrease
1	kan_bank	28.93	16.54	12.3
2	kafta_shop	26.01	15.21	10.8
3	delta_lib	19.18	13.84	5.34
4	reg_prt_wes	38.1	18.9	19.2
5	stud_api	23.8	17.8	6.0
6	oryx_banking	21.1	11.8	9.3
7	omega_lib	26.4	17.8	8.6
8	max_hosp	25.1	19	6.1
9	ap_axi	29.8	22.3	7.5
10	studentApi	31	23.4	7.6
11	u_bank	29.8	24.5	5.3
12	oryx_kafta	29.1	26.7	2.4

it is introduced graphically as shown in Figure 8:

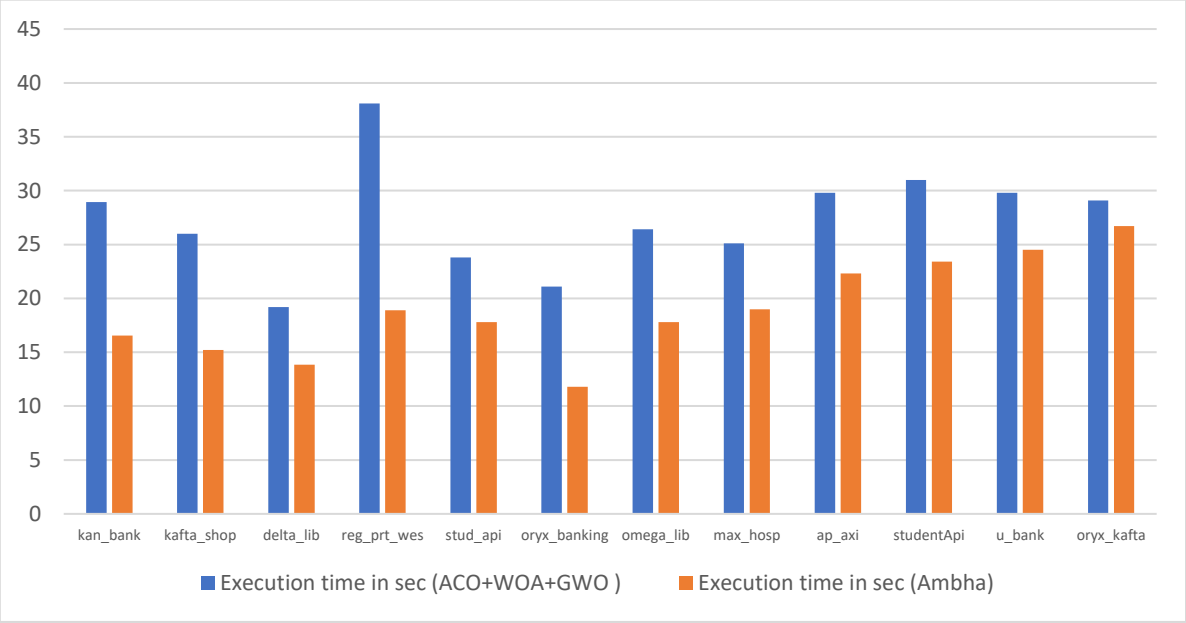


Fig.8 Comparative study of time metric

E. F-measure

It very well may be characterized as

$$F - measure = harmonic\ mean\ of\ recall\ and\ precision = \frac{2 * recall * precision}{recall + precision}$$

The dataset kan_bank (0.821), reg_prt_wes (0.871), oryx_banking (0.891), ap_axi (0.821), u_bank (0.871) and oryx_kafta (0.891) have good F-measure values of Ambha as compared to framework (ACO+WOA+GWO) as depicted in the Table IX.

Table IX: Description of F-measure values

Sr No.	program	F-measure (ACO+WOA+GWO)	F-measure Ambha	Increase

1	kan_bank	0.671	0.762	0.19
2	kafta_shop	0.621	0.821	0.2
3	delta_lib	0.665	0.761	0.10
4	reg_prt_wes	0.778	0.871	0.10
5	stud_api	0.665	0.761	0.10
6	oryx_banking	0.761	0.891	0.13
7	omega_lib	0.501	0.782	0.28
8	max_hosp	0.512	0.762	0.25
9	ap_axi	0.698	0.821	0.13
10	studentApi	0.512	0.761	0.25
11	u_bank	0.776	0.871	0.10
12	oryx_kafta	0.778	0.891	0.12

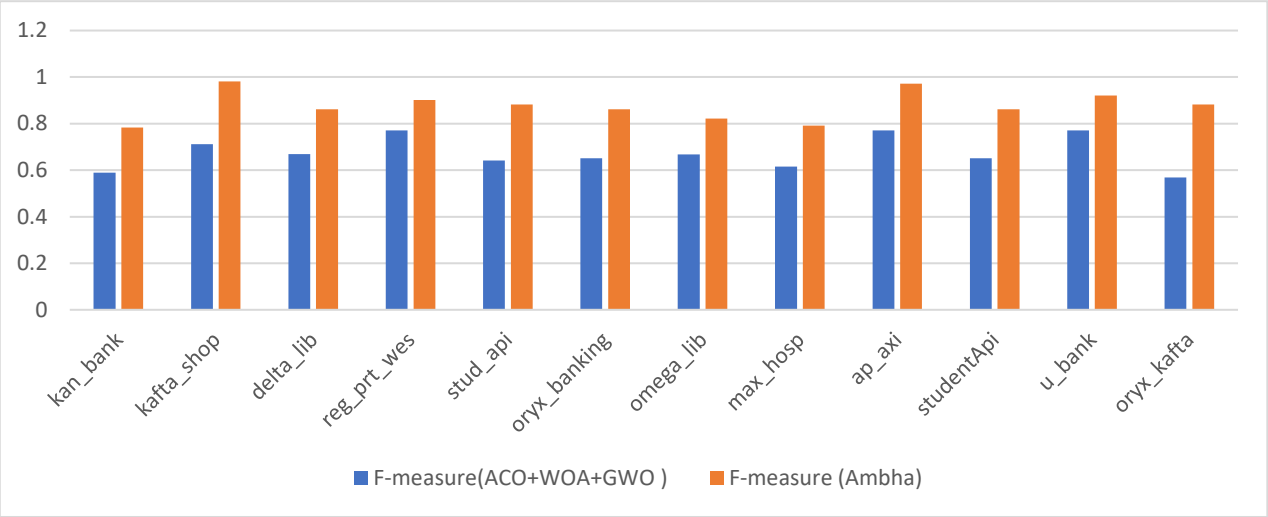
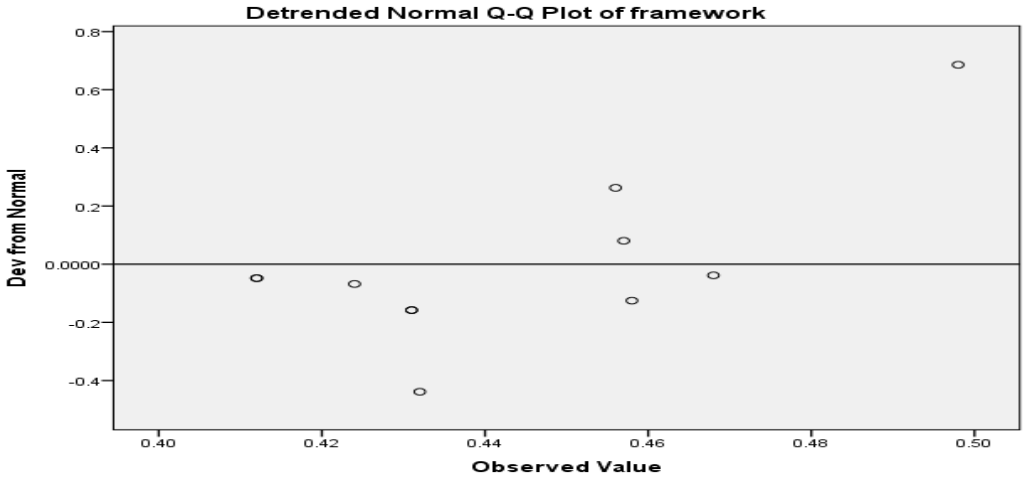


Fig. 9 Comparison of F-measure

5. Non-parametric Statistical tests

To validate our proposed work, NP statistical tests are conducted using IBM SPSS 22 (statistical software) on framework (ACO+WOA+GWO) and Ambha for test case suite optimization. Firstly, normality test is performed to check dataset deviations from normal line.



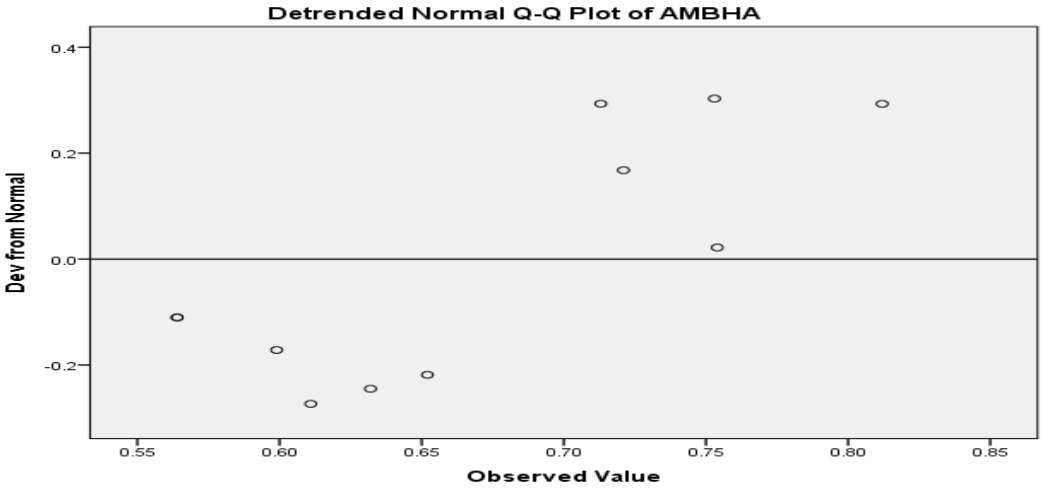


Fig. 10 Deviations from normal leads to non-parametric test

Hypothesis formulation is done as shown in Figure 11:

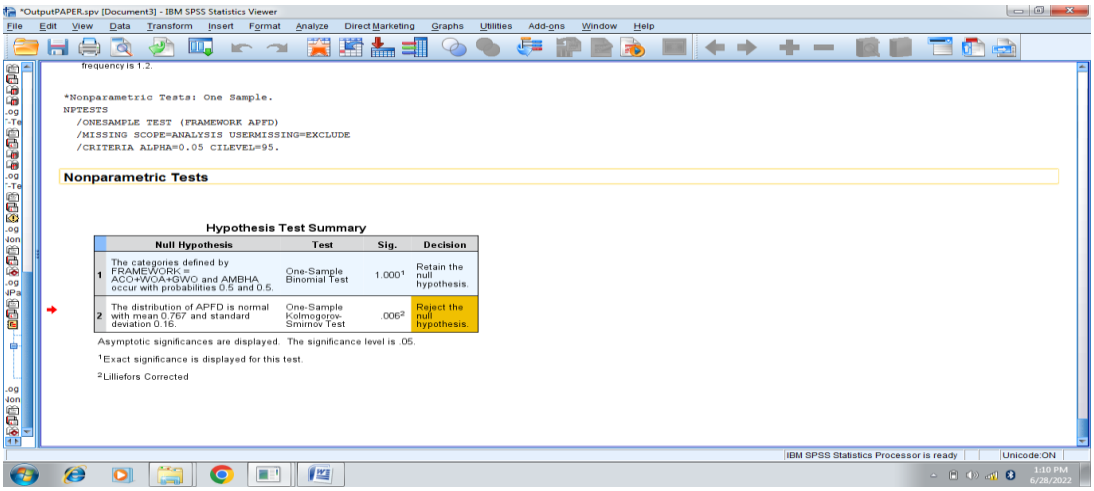


Fig. 11hypothesis test summary

A. Wilcoxon sign rank tests

This non parametric test is used for comparing framework (ACO+WOA+GWO) and Ambha for regression testing problems. We considered α=0.05, xi is difference of scores between two datasets.

Rank_positive=∑rank (xi >0) + 1/2∑rank (xi =0);

Rank_negative=∑rank (xi <0) + 1/2∑rank (xi =0);

Mean Rank_positive value is 4.71 and Rank_negative value is 7.61. Standard stat table is considered to compare values (t-values are 0.241,0.42) and observed that Ambha outperforms framework (ACO+WOA+GWO) for regression testing problems.

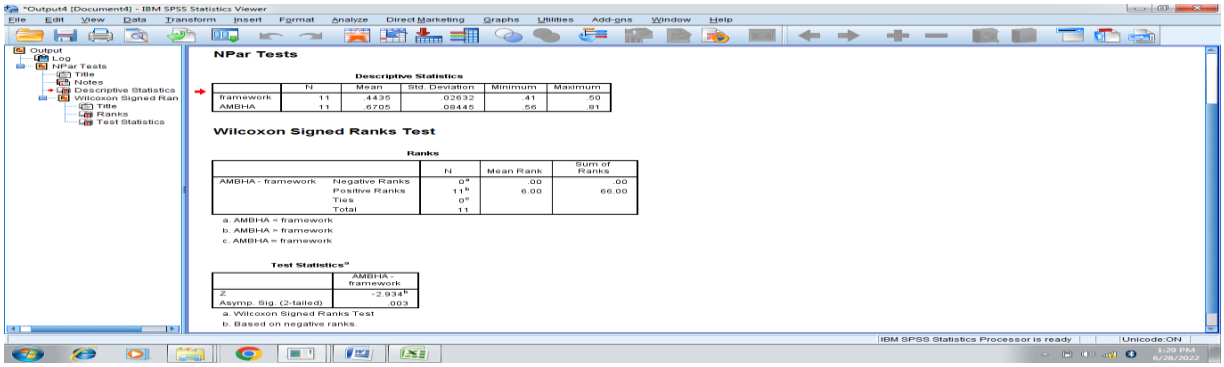


Fig. 12 Screenshot of test environment

B. Friedman test

F test is based on ranks assigned to APFD values of Ambha and framework(ACO+WOA+GWO). Average is calculated by using following equation:

$$(chi)^2 = \frac{12n}{d(d+1)}\sum(rank)^2 - \frac{d(d+1)}{4}$$

here (d+1) signifies'degree of freedom' and *chi* give average value of ranks.

It is seen at $\alpha = '0.05', (chi)^2 = '11.63'$ lie in rejection area.Values are matched with standard stat table which shows that Ambha outperforms framework (ACO+WOA+GWO)

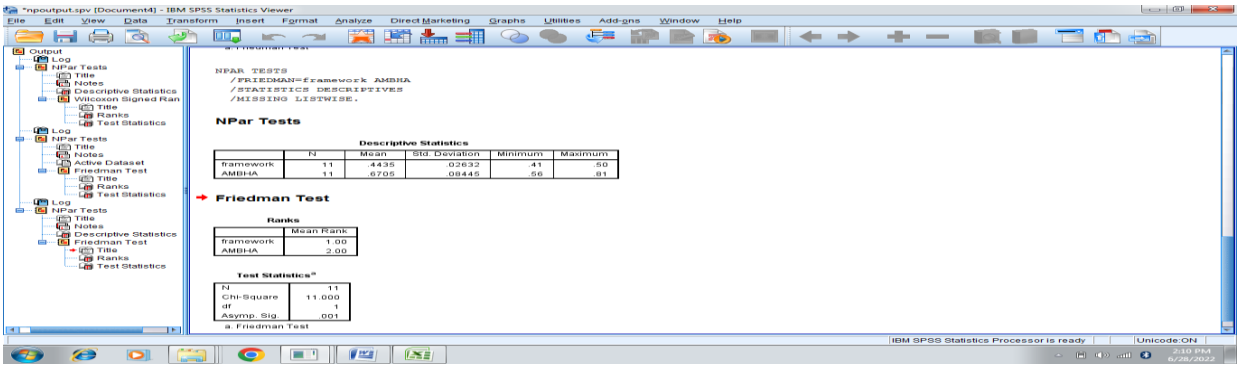


Fig. 13Screenshot of test environment

6. Conclusion

It is concluded that regression testing techniques can efficiently provide optimal solutions to NP-hard real word optimization problem. We proposed a metaheuristic framework 'Ambha' which gave promising results as compared to framework (ACO+WOA+TCS). We considered 12 junit test case suites taken from GitHub, TravisCI and EclEmma code coverage tool in java Eclipse IDE environment. Five evaluation performance measures are used to compare performance of two frameworks and it is seen that 'Ambha' outperformed framework (ACO+WOA+GWO). Further, non-parametric statistical tests are carried out to validate our results. As a future work, we will implement this framework in real-time software testing environment.

References

[1] Panichella, A., Oliveto, R., Di Penta, M., & De Lucia, A. (2014). Improving multi-objective test case selection by injecting diversity in genetic algorithms. *IEEE Transactions on Software Engineering*, *41*(4), 358-383.

[2] Ma, B., Wan, L., Yao, N., Fan, S., & Zhang, Y. (2021). Evolutionary selection for regression test cases based on diversity. *Frontiers of Computer Science*, *15*(2), 1-3.

[3] Kaur, A., & Agrawal, A. P. (2017, January). A comparative study of bat and cuckoo search algorithm for regression test case selection. In *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence* (pp. 164-170). IEEE.

[4] Souza, L. S., de Miranda, P. B., Prudencio, R. B., & Barros, F. D. A. (2011, November). A multi-objective particle swarm optimization for test case selection based on functional requirements coverage and execution effort. In *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence* (pp. 245-252). IEEE.

[5] Harikarthik, S. K., Palanisamy, V., & Ramanathan, P. (2019). Optimal test suite selection in regression testing with testcase prioritization using modified Ann and Whale optimization algorithm. *Cluster Computing*, *22*(5), 11425-11434.

[6] Zhang, L. (2018, May). Hybrid regression test selection. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)* (pp. 199-209). IEEE.

[7] Harrold, M. J., Jones, J. A., Li, T., Liang, D., Orso, A., Pennings, M., & Gujarathi, A. (2001). Regression test selection for Java software. *ACM Sigplan Notices*, *36*(11), 312-326.

[8] Rothermel, G., & Harrold, M. J. (1996). Analyzing regression test selection techniques. *IEEE Transactions on software engineering*, *22*(8), 529-551.

[9] Suri, B., Mangal, I., & Srivastava, V. (2011). Regression test suite reduction using an hybrid technique based on BCO and genetic algorithm. *Special Issue of International Journal of Computer Science & Informatics (IJCSI), ISSN (PRINT)*, 2231-5292.

[10] Zhang, C., Chen, Z., Zhao, Z., Yan, S., Zhang, J., & Xu, B. (2010, July). An improved regression test selection technique by clustering execution profiles. In *2010 10th International Conference on Quality Software* (pp. 171-179). IEEE.

[11] Wong, W. E., Horgan, J. R., London, S., & Agrawal, H. (1997, November). A study of effective regression testing in practice. In *PROCEEDINGS The Eighth International Symposium on Software Reliability Engineering* (pp. 264-274). IEEE.

[12] Singh, Y., Kaur, A., & Suri, B. (2009). Empirical validation of variable based test case prioritization/selection technique. In*International Journal of Digital Content Technology and its Applications*, (pp. 116-123).

[13] Siddik, M. S., & Sakib, K. (2014, December). RDCC: An effective test case prioritization framework using software requirements, design and source code collaboration. In *2014 17th International Conference on Computer and Information Technology (ICCIT)* (pp. 75-80). IEEE.

[14] Mittal, S., & Sangwan, O. P. (2018). Prioritizing test cases for regression techniques using metaheuristic techniques. *Journal of Information and Optimization Sciences*, *39*(1), 39-51.

[15] Singh, Y., Kaur, A., & Suri, B. (2010). Test case prioritization using ant colony optimization. *ACM SIGSOFT Software Engineering Notes*, *35*(4), 1-7.

[16] Ahmad, S. F., Singh, D. K., & Suman, P. (2018). Prioritization for regression testing using ant colony optimization based on test factors. In *Intelligent communication, control and devices* (pp. 1353-1360). Springer, Singapore.

[17] Mohapatra, S. K., & Prasad, S. (2013, December). Evolutionary search algorithms for test case prioritization. In *2013 International Conference on Machine Intelligence and Research Advancement* (pp. 115-119). IEEE.

[18] Carlson, R., Do, H., & Denton, A. (2011, September). A clustering approach to improving test case prioritization: An industrial case study. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)* (pp. 382-391). IEEE.

[19] Xu, G., & Rountev, A. (2007, May). Regression test selection for AspectJ software. In *29th International Conference on Software Engineering (ICSE'07)* (pp. 65-74). IEEE.
[20] http://travis.ci.com
[21] http://spring.io.projects
[22] http://github.com
[23] http://sir.csc.ncsu.edu
[24] Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability*, *22*(2), 67-120.
[25] Yang, X. S., & Press, L. (2010). Nature-inspired metaheuristic algorithms second edition.
[26] Rothermel, G., & Harrold, M. J. (1996). Analyzing regression test selection techniques. *IEEE Transactions on software engineering*, *22*(8), 529-551.
[27] Said, G. A. E. N. A., Mahmoud, A. M., & El-Horbaty, E. S. M. (2014). A comparative study of meta-heuristic algorithms for solving quadratic assignment problem. *arXiv preprint arXiv:1407.4863*.
[28] Myers, G. J. (1979). The Art of Software Testing John Wiley and Sons Ltd.
[29] Rajabi Moshtaghi, H., Toloie Eshlaghy, A., & Motadel, M. R. (2021). A comprehensive review on meta-heuristic algorithms and their classification with novel approach. *Journal of Applied Research on Industrial Engineering*, *8*(1), 63-89.
[30] Stegherr, H., Heider, M., & Hähner, J. (2020). Classifying Metaheuristics: Towards a unified multi-level classification system. *Natural Computing*, 1-17.
[31] Rajabi Moshtaghi, H., Toloie Eshlaghy, A., & Motadel, M. R. (2021). A comprehensive review on meta-heuristic algorithms and their classification with novel approach. *Journal of Applied Research on Industrial Engineering*, *8*(1), 63-89.
[32] Sommerville, I. (2011). *Software Engineering, 9/E*. Pearson Education India.
[33] Fister Jr, I., Yang, X. S., Fister, I., Brest, J., & Fister, D. (2013). A brief review of nature-inspired algorithms for optimization. *arXiv preprint arXiv:1307.4186*.
[34] Hao, J. K., & Solnon, C. (2020). Meta-heuristics and artificial intelligence. In *A Guided Tour of Artificial Intelligence Research* (pp. 27-52). Springer, Cham.
[35] Kazmi, R., Jawawi, D. N., Mohamad, R., Ghani, I., & Younas, M. (2017). A Test Case Selection Framework and Technique: Weighted Average Scoring Method. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, *9*(3-4), 15-22.
[36] Mahapatra, R. P., Ranjith A. and Kulothungan. (2018). A Framework for improving Test Case Selection and Randomized Prioritization. International Journal of Pure and Applied Mathematics, 18(22), (pp. 1927-36).
[37] Mirjalili, S., & Lewis, A. (2016). The whale optimization algorithm. *Advances in engineering software*, *95*, 51-67.
[38] Pham, Q. V., Mirjalili, S., Kumar, N., Alazab, M., & Hwang, W. J. (2020). Whale optimization algorithm with applications to resource allocation in wireless networks. *IEEE Transactions on Vehicular Technology*, *69*(4), 4285-4297.
[39] Gharehchopogh, F. S., & Gholizadeh, H. (2019). A comprehensive survey: Whale Optimization Algorithm and its applications. *Swarm and Evolutionary Computation*, *48*, 1-24.
[40] Rana, N., Latiff, M. S. A., Abdulhamid, S. I. M., & Chiroma, H. (2020). Whale optimization algorithm: a systematic review of contemporary applications, modifications and developments. *Neural Computing and Applications*, *32*(20), 16245-16277.
[41] Salgotra, R., Singh, U., & Saha, S. (2019). On some improved versions of whale optimization algorithm. *Arabian Journal for Science and Engineering*, *44*(11), 9653-9691.
[42] Indumathi C. P. and Madhumathi S. (2017). Cost Aware Test case suite Reduction Algorithm for Regression Testing. *International Conference on Trends in Electronics and Informatics*, ICEI.
[43] Harrold, M. J., Gupta R. and Soffa M. L. (1993).A Methodology for Controlling the size of a

test case suite," *ACM transactions on Software Engineering and Methodology*, 2 (3).

[44] Chen,T. Y. and Lau, M. F. (1998). A new heuristic for test case suite reduction. *Information and software Technology*, *ELSEVIER*, 48, pp. 347-354.

[45] Tallam S., and Gupta N. (2005). A Concept Analysis Inspired Greedy Algorithm for Test case suite Minimization.*ACM publications*.

[46] Lin, J.W. and Huang,C.Y. (2009). Analysis of Test –Suite reduction with enhanced tie breaking techniques. *Information and software technology,* ELSEVIER.

[47] Black, Melachrinoudis, E. and D. Kaeli (2004). Bi-Criteria Models for all-uses test case suite reduction. *Proceedings of the 26th IEEE International Conference on Software Engineering (ICSE'04)*.

[48] Mohapatra S. K.&Pradhan, S. (2015). Finding Representative Test case suite for the Test Case Reduction in Regression Testing. *IEEE International Conference on Computer, Communication and Control*.

[49] Mirarab,S. and Tahvildari, L.(2007). A prioritization approach for software test cases based on Bayesian networks. *International Conference on Fundamental Approaches to Software Engineering*, Springer, Berlin, Heidelberg, pp. 276-290.

[50] Hla, K.H., Choi Y. and Park, J. S.(2008). Applying Particle Swarm Optimization to prioritizing test cases for embedded real time software retesting," in *proceedings of International Conference on Computer and Information Technology workshops*, IEEE,pp. 527-532.

[51] Bhawna, Kumar, G.& Bhatia, P.K. (2016). Software Test Case reduction using Genetic Algorithm: A Modified Approach.*International Journal of Innovative Science, Engineering and Technology,*3(5), pp. 349-354.

[52] Chowdhury, Aditya, & Vishnu G. Nair. (2017) "Optimization of PID controller gains of an aircraft pitch control system using particle swarm optimization algorithm" International Journal of Mechanical and Production Engineering Research and Development 7.6: 223-229.