International Journal of Advanced Technology and Engineering Exploration, Vol 9(88)
ISSN (Print): 2394-5443 ISSN (Online): 2394-7454
http://dx.doi.org/10.19101/IJATEE.2021.874727

Research Article

Test case prioritization using modified genetic algorithm and ant colony optimization for regression testing

T.K. Akila[1*] and A. Malathi[2]
Research Scholar, Department of Computer Science, Government Arts College, Coimbatore, India[1]
Assistant Professor, Department of Computer Science, Government Arts College, Coimbatore, India[2]

Received: 10-October-2021; Revised: 12-March-2022; Accepted: 15-March-2022

Abstract
Regression testing (RT) plays an essential role in software maintenance. The occurrence of any new fault during the re-testing or modification process needs to be analyzed effectually. RT needs enormous effort to produce a higher fault detection rate. Test case prioritization (TCP) is an efficient way to predict the fault detection rate. Various researchers modelled the TCP method to provide a single objective solution; however, this work concentrates on providing a multi-objective solution using a meta-heuristic optimization approach. Here, two different approaches known as ant colony optimization (ACO) and genetic algorithm (GA) are adopted to offer a multi-objective solution with a better fault detection rate. The characteristics of the ACO and GA are analyzed to prioritize the test case by combining the multi-dimensional characteristics under the test environment to enhance the fault detection rate. Here, some experimentation is made to compute the performance of the proposed model by evaluating the number of test cases, number of iterations, and ant traversal path. The proposed model shows better trade-off in contrast to the prevailing approaches where the fault detection rate of multi-objective (ACO and GA) model provides outcomes of 94.5%, 94.8%, 93.8%, 92.5%, 95.8%, 97.8%, 99.2%, and 93.5% respectively.

Keywords
Regression test, Test case prioritization, Genetic algorithm, Ant colony optimization, Meta-heuristic optimization.

1.Introduction
Regression testing (RT) plays an essential role in the software development life cycle (SDLC), and it assists in validating the faults which are not identified in the existing software products [1]. These faults are identified when there are some variations in prior software products. The test case execution is a complex task where the evaluation of a subset (test case) over the testing process has to be concentrated by the upcoming researchers [2]. Various approaches prioritize, minimize, optimize, and select test cases; whereas other methods evaluate the group of test cases with the assistance of RT [3]. These approaches are listed as graph-based relations, greedy algorithms, fuzzy entropy, and capacity-based fault detection for selection, minimization, and optimization and so on [4]. The computation cost of executing the RT model is associated with the product size that needs validation, and more appropriate test cases have to be analyzed [5].

The incremental and iterative model is considered during software development and short cycles.

The complete test case execution is determined as an unnecessary trial over the provided amount of test cases (agility processing target is lost) [6]. It leads to the rise of soft computing techniques in coincidence with agile approaches to predict the test cases and fulfil the acceptable verification level over the software products. The outcomes of various studies specify the efficiency of alternative RT optimization for clustering the test cases based on the available characteristics, patterns, and criteria like test cases that predicts a similar set of faults from the open cluster [7]. For instance, specific strategies composed of various test profile execution, sampling, test run functionality, database access, function calls, filtering patterns, program partitioning, etc., are appropriate test cases for all clusters [8]. Moreover, the problem in handling these approaches is determined based on the cluster size and quality determination to balance the efficiency and cost of the model [9].

*Author for correspondence

With the advancements in machine learning approach, cluster analysis is a method used for organizing and training the data based on the mathematical model of the application domain autonomously [10]. However, it is necessary to add information for data like cluster size and total clusters are formed, which causes the processing to be expensive and sometimes constraints [11]. However, a supervised learning-based cluster needs particular experimentation to group the information manually. The alternative process is to adopt an unsupervised clustering model where the probabilistic model determines the number of clusters.

Traditionally, RT approaches are applied from extensive code perspectives, i.e., determination of software product with finite state machine (FSM), maintaining the database (DB) constant with alternative evolution. Moreover, with the software products and the DB access, two different database states are sensed concurrently (data and code state). Therefore, RT approaches need to be determined in two other aspects [12]. In the first case, the code state is defined as the labelled set with the memory location of individual data stored (code state). Moreover, in the alternative state, the database is organized based on various data models like the relational model [13]. Thus, the enormous data volume is influenced by single code instruction that exploits the characteristics of instruction nature over the software product components evaluated on DB respectively [14, 15]. Based on the review, it is known that test case prioritization (TCP) is not achieved efficiently and lacks a global solution, which is a significant research constraint. To handle this issue, this research aims to provide a novel meta-heuristic optimization approach to enhance the prioritization process and attain both local and global solutions. Therefore, this work integrates both genetic algorithm (GA) and ant colony optimization (ACO) to enhance the prioritization process. In this research, an approach for analyzing the TCP with RT using ACO and an improved GA is proposed. The features of these two models are identifying the fault with the number of iterations, number of test cases, and the determination of path traversal efficiently. The empirical evaluation is conducted in the manual process to attain suitable outcomes to establish the trade-off of the proposed model with prevailing approaches. The TCP approach works effectually with the adoption of ACO and improved GA, respectively. The objective of this research with TCP and RT involves scheduling test cases for RT to enhance its effectiveness to fulfil specific performance. However, it is inefficient to re-execute all the available tests cases in RT based on the software modification. This challenge needs to be addressed using meta-heuristic optimization approaches like ACO and GA. Using ACO for TCP aims to increase the fault detection rate, and execution time is reduced with automated TCP compared to manual TCP in RT. Similarly, GA helps prioritize the test case with the statement coverage technique. Both the methods work effectively in the case of TCP in RT and intend to address the challenge with more significant effort.

This document is further partitioned as section 2 provides the detailed review analysis of RT and TCP approach. Section 3 provides the extensive analysis of TCP for RT using ACO and improved GA, respectively. Section 4 is numerical results and discussion to analyze faults, followed by the conclusion and future research direction in Section 5.

2.Related work

Recently, most of the TCP approaches have been used for code-based analysis. Mei et al. [15] executed various functional coverage and statement coverage models for TCP. Yoo et al. [16] adopted a cluster analysis model to specify multiple test cases. Marchetto et al. [17] anticipated a novel multi-heuristic approach for TCP. This method establishes the link between the software artefacts and predicts the fault-prone region using the software artefacts. Di et al. [18] proposed a novel hyper-volume-based GA to deal with the test cases adopting various coverage criteria. The empirical study determines the functionality of hyper-volume GA as an efficient model when evaluated with five different approaches. Guo et al. [19] evaluated the statistical TCP model with a dynamical TCP approach. The experimental outcomes demonstrate 60 different java programs to project the statically designed model with better performance in an unexpected way for certain specific criteria. The provided measurements do not drastically influence the mutant's type and the number of mutants for effectual TCP. Laali et al. [20] adopted an online fault detection approach for test case sorting. Khatibsyarbini et al. [21] adopted the firefly approach for test case sorting using the similarity distance measure. The experimentation demonstrates that the anticipated firefly model performs superior functionality than the prevailing techniques.

Jahan et al. [22] explained a semi-automatic risk-based TCP method using functional invocation

relationship and software modification information. Solanki et al. [23] anticipate a model with the adoption of ACO for sorting the test cases with the nature of the ant colony-based food selection process. Similarly, a prioritization method attains the investigator's attraction for the past few decades. Kundu et al. [24] anticipates three diverse prioritization approaches with performance metrics that embrace edge and message weights. Kundu et al. [24] adopt information from various unified modeling language (UML) sequence diagrams to test cases. Kundu et al. [24] transformed the UML activity graph to control flow graph (CFG) for test case ranking. Kaur et al. [25] used the most effective industrial system as an experimental subject to study general approaches. Sharma and Singh [26] anticipated various TCP approaches using a powerful finite state machine. The empirical study shows that the proposed model can diminish the testing time contrary to the prevailing code-based model. Niu et al. [27] provided an extensive review of the ACO approach for sorting the model-based test cases. Conversely, Ye et al. [28] merged the selective approach and spread the count-based technique to employ test cases by scrutinizing ordering criteria. The integrated model performs superior functionality with a single system based on a finite decision model.

Re-testing or RT model is essential to verify the evolution phase and software maintenance over the SDLC. It is performed to fulfil certain modifications that are not hampered by prevailing software versions, and a newer version of this model is known as backward compatibility. The testing process consumes half the cost used for the entire software development [28]. It is a highly time-consuming process and considered an expensive process. RT process is optimized in three different ways: 1) minimization (test cases), which reduces the test suites by avoiding the obsolete test cases and patterns; 2) selection (test cases) which chooses the test cases that are associated with specific criteria. For example, handles the updated region only; 3) prioritization (test cases) orders the available test case with specific properties which are highly ranked with test cases that are executed over the high prioritization model [29].

Both selection and minimization are determined as the truncated version of original test suites. At the same time, prioritization is considered re-ordered with various test case suites devoid of eliminating the test cases. However, in some cases, the test cases are not essential in specific versions; but it is helpful in the recent version of the provided software. Subsequently, the prioritization process is safer than the permanent removal process; TCP is a more reliable, secure, and cost-effective approach for the RT. Hence, the investigators need to concentrate on TCP rather than test-suite reduction or selection.

The investigator extensively categorizes the TCP model into searching-based, fault-based, history-based, requirements-based, code-coverage based models. When there is some source code availability, then the process of prioritization is based on test case ranking, which relies on the coverage/block/statement coverage model known as the code-coverage based prioritization method [29].

The requirement based prioritization uses customer requirements for test case analysis. The fault coverage-based prioritization method uses the test case based on faulty rate coverage. Other approaches like history-based prioritization utilize historical information regarding the software model. The search-based prioritization assists in determining the optimal ordering of diverse test cases by predicting the global space for multiple and single objective processes [30].

The investigators adopt various search-based models for TCP, for instance, hill climbing and the greedy algorithm. This research concentrates on performing RT using the meta-heuristic ACO and GA to adopt TCP. *Table 1* depicts the comparison of various existing methods with their pros and cons.

Based on all these analyses, it is observed that TCP helps the testers to predict the faults over the applications with high prioritization. However, the primary research challenge and constraints rely on the execution time. The execution time is higher while testing all the available cases for a specific application. The significant research gap is the lack of computational complexity and execution time which needs to be addressed. This research adopts ACO and GA to handle the issue and pretends to give better and feasible solutions to the existing research challenges.

Table 1 Comparison of various existing approaches

S. No.	Authors	Methods	Advantages	Disadvantages
1	Yoo et al. [16]	Cluster analysis	Validates various test cases	It does not concentrate on prioritization
2	Marchetto et al. [17]	Models meta-heuristic optimization approach	Concentrates on TCP	Higher computational complexity and execution time
3	Di et al. [18]	Hyper-volume based GA	Adopts various test coverage criteria	Higher complexity
4	Guo et al. [19]	Statistical TCP model for dynamical analysis	It uses 60 programs for analysis and gives unexpected results	Complex computation and
5	Laali et al. [20]	Firefly approach for TCP	Measures similarity distance	RT is not performed and consumes more time for execution
6	Khatibsyarbini et al. [21]	Semi-automated risk-based TCP	Considers functional invocation and software modification information	Execution time is higher while handling the software modification information
7	Solanki et al. [23]	Adaptive ACO	The higher fault detection rate	But execution time is higher
8	Kundu et al. [24]	UML sequence diagram based TCP	Measures edges and weights Reduced computational complexity	No proper prioritization is done
9	Sharma and Singh [26]	The extreme finite state machine	Reduce testing time	Lack in the prioritization process
10	Ye et al. [28]	Integrated selective approach and count-based technique	Designs finite decision model	--

3.Methodology

This section discusses the analysis of RT with TCP using meta-heuristic approaches. They are GA and ACO. The empirical research is done with manual computation, where the fault detection rate is considered the essential evaluation metric. Here, a test case in prioritized order using ACO and GA gives better results. ACO will find the best test cases with the maximum fault in minimum time. The proposed approach has been validated with a benchmark example. The result shows the effective test case has been selected with a higher score. Similarly, TCP is becoming a hot topic in software testing research. Combining GA with test-points coverage attains superior results in TCP, especially for functional testing. The hybrid model works well by handling the shortcomings of the other.

Genetic algorithm (GA)

The problem related to software testing is more complex and challenging to resolve efficiently using deterministic algorithms. This work adopts GA with an underlying principle: the individual over the population needs to fight for resources. The success of the individuals relies on the production of offspring and genes propagated to successive generations. The parent chromosomes mate together to alter the gene to produces the offspring, giving better fitness (parents). This process preserves the generation to form species, individuals and adapt to the provided environment [31−35].

Test case prioritization (TCP) with GA

The TCP is mapped to GA to specify the genetic information; chromosomes are considered the sequence of test cases. Therefore, chromosomes are identified using encoding permutation in which sequence numbers are allocated for all test cases. The process is initiated by forming random individuals (initial population). The population is given in fitness testing, which is evaluated with specific coverage criteria for prioritizing the test cases. The selection mechanism needs to choose the test cases, where the fitness is better than another model to generate successive generations. The best test suites are merged with a pair of mutation and cross-over operators to develop the most delicate offspring for consecutive generations. Similarly, the cross-over operator is the re-combination of two chromosomes that inherits the parent characteristics. It adopts cross-over operators, as matched and ordered cross-over. In the former model, the crossover process is performed based on the position exchange indeed of sliding motion. In contrast, in the latter model, two positions

are randomly chosen with pre-defined locations. They are analyzed of the test cases left over of another parent to generate newer offspring. Mutation operators are utilized to preserve individual diversity in a successive generation. In the case of swap mutation, the positions are randomly chosen based on two positions, which are continued to a certain termination point. GA assists in predicting the optimal test case ordering [36−40]. It utilizes chromosomes' genetic information to help search the location of the most satisfactory solution over search space. Various investigators adopt GA for TCP which tries to identify the potential of the GA. This model is extensively partitioned into single objective and multi-objective GA, respectively. Various researchers analyze the single objective constraints as it is a simple model. The multi-objective GA model holds two or more purposes during the test prioritization during a specific time. It is shown in *Figure 1*.

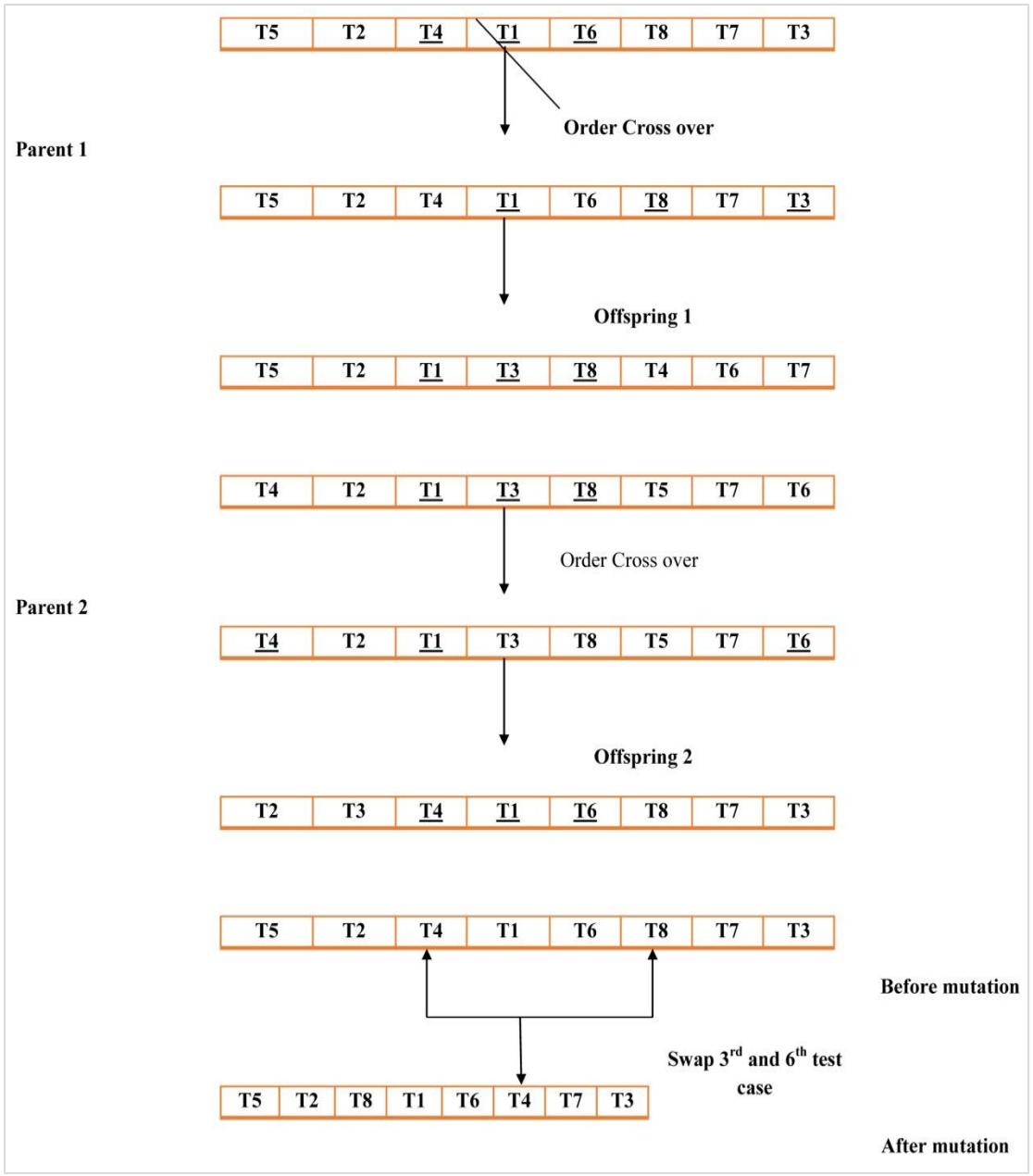

Figure 1 GA process

Ant colony optimization (ACO)

ACO is a swarm intelligence algorithm is constructed based on the characteristics of ant seeking food and individual population. Multiple investigators have designed various ACO models; however, all those models concentrate on the single-objective problem but lack multi-objective constraint concentration. This work intends to give a multi-objective ACO where the test cases are in sequence with the ant's path travelling from one place to another. It is known as a transition path. There are six different stages in this multi-objective algorithm.

Initialization: The individuals are initialized with a random sequence with the available parameters of ACO, which is constructed with the best-so-far method by choosing the individuals (non-dominate information) where the pheromone needs to be built.

Construction (individual): The ants need to select the appropriate test case using visiting point probability, and execution is redundant until all the test cases are visited. The sequences of test cases are modelled individually. The possibilities of selecting successive cases are represented as P_{ij}^{k}, determined as $'k'$ ant (probability) moves from test case $(i \to j)$. It is expressed as in Equation 1:

$$P_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha} \sum_{d=1}^{2} \left[\eta_{ij}^{d}\right]^{\lambda_d^{\beta}}}{\sum_{l \epsilon N_i^k} [\tau_{il}]^{\alpha} \sum_{d=1}^{2} \left[\eta_{ij}^{d}\right]^{\lambda_d^{\beta}}} & l, j \epsilon N_i^k \\ 0 & else \end{cases} \quad (1)$$

Here, τ_{ij} is transition weight (pheromone) for test cases $(i \to j)$, $'d'$ is the number of objectives $(d = 2)$, η_{ij}^{d} is heuristic information, $' \propto '$ is the heuristic factor with pheromone information $\tau_{ij}, '\beta'$ handles the relative weight η_{ij}^{d}, and N_i^k is set of test cases from ant 'k' for all test cases $'i'$.

Evaluation (individual):

The fitness value is evaluated after constructing individual evaluation.

Multi-objective (ranking):

Here, Pareto non-dominant method is used where the individuals are ranked, and individuals with rank (high) are chosen as the best iteration.

Individual set updation:

Iteration (best) is utilized to evaluate individual sets, where the individual (best so far) is replaced while dominated by other individuals. At last, the best individuals are set as optimal individuals.

Update pheromone:

Pheromones are adopted to assist in searching the most proper direction in the front set and diminish the value on relative weaker individuals with fewer visited ants. Here, the pheromone values are reduced due to the evaporation rate. The pheromone values of individuals are increased to assistants for selecting the transitions, and the diminished transition values are to be visited. Pheromone updation leads the ant to choose the superior change, which is highly likely than superior paths in a successive generation. It is expressed in Equations 2 and 3:

$$\tau_{ij} = (1 - \rho) * \tau_{ij} + \Delta\tau_{ij} \quad (2)$$

$$\Delta\tau_{ij} = \sum_{k=1}^{h} \Delta\tau_{ij}^{k} \quad (3)$$

Here, $'\rho'$ is the evaporation factor, τ_{ij} is the test case pheromone $(i \to j)$ belongs to the individual $(iteration - best\ set)$, $\Delta\tau_{ij} = 0\ or\ 1$ respectively. The calculation is based on transition pheromone increment $(i \to j)$, and $'k'$ is number $'k'$ non-dominated individual, $'h$is total amount of iteration (best). Therefore, $\Delta\tau_{ij}$ is equal to the total pheromone increment (iteration-best). In alliteration, τ_{ij} (various test cases) is computed to assistants in successive generations to determine superior transitions, and ants are facilitated to realize pheromone to fulfil essential information needs to be collected. It is explained in Algorithm 1.

Algorithm 1: ACO algorithm

1. Initialize the test criteria
2. Fulfil the optimal set
3. Initialize the ant individuals and set parameters
4. Multi-objective ranking
5. Model best set
6. Set pheromone trials
7. While ≠ terminal conditions do
8. Individual evaluation and construction
9. Multi-objective ranking
10. The model best iterative set
11. Update pheromone and best set
12. End while

From *Figure 2*, the pheromone update is shown using ACO. Here, five test cases are considered, where four individual test cases are in the front set. The steps are updated, and the results are presented separately. In step 1, the pheromone is updated where A-B-C-D-E is evaluated with test cases,$A \to B, BC, C \to D, and D \to E$, respectively. The updated pheromone is given in *Figure 3.* The pheromones from transition test cases are provided, and the last key test case is updated. The updated pheromone is supplied as:$A \to E, B \to E, C \to E, and\ D \to E$, respectively.

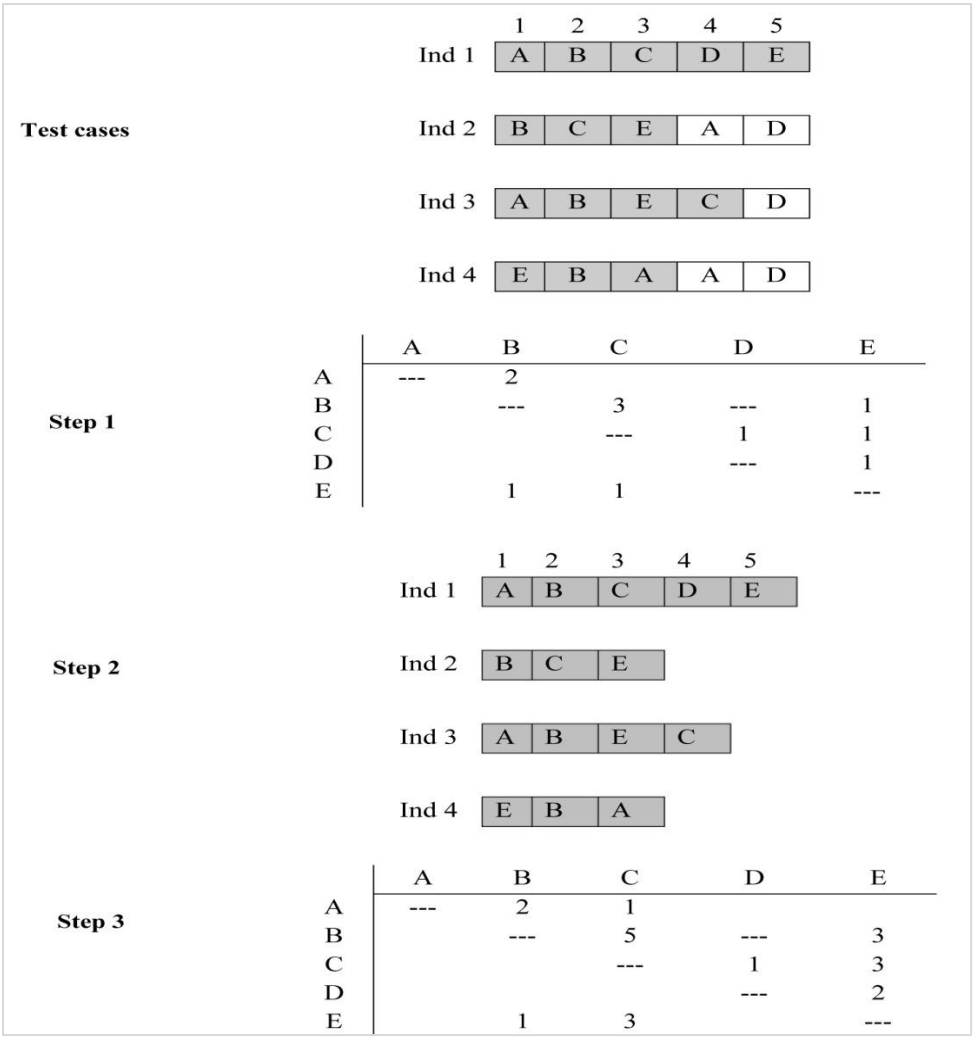

Figure 2 ACO based test case prioritization

Figure 3 depicts the flow diagram of the anticipated model, which includes three phases: initialization phase, pheromone update and solution construction phase. The fine-tuning processes of the ants are provided with the computation of heuristics of all candidates with ant edge computation. The solution phase provided a better fault rate prediction outcome than other approaches.

Case study

Let n be the number of a test suite, F be the number of faults identified, and execution time (ET) specifies the execution time of each test suite. With the initial random population, $N/4$ ants are considered for initial foraging. The generated populated looked as upon as a matrix, and foremost, the matrix is sorted in ascending order based on the execution time of each test suite. *Table 2* depicts the parental matrix of test cases and fault coverage. Here, F1, F2, F3, F4, F5, F6, F. F8, F9, F10 are the number of faults considered with the execution time. The test cases are ordered randomly over the table based on the execution time. The highlighted contents provide the fault over the test case. *Table 3* is the Sorted matrix based on execution time. The primary research objective is the concentration towards the execution time, and the model consumes lesser execution time than other approaches.

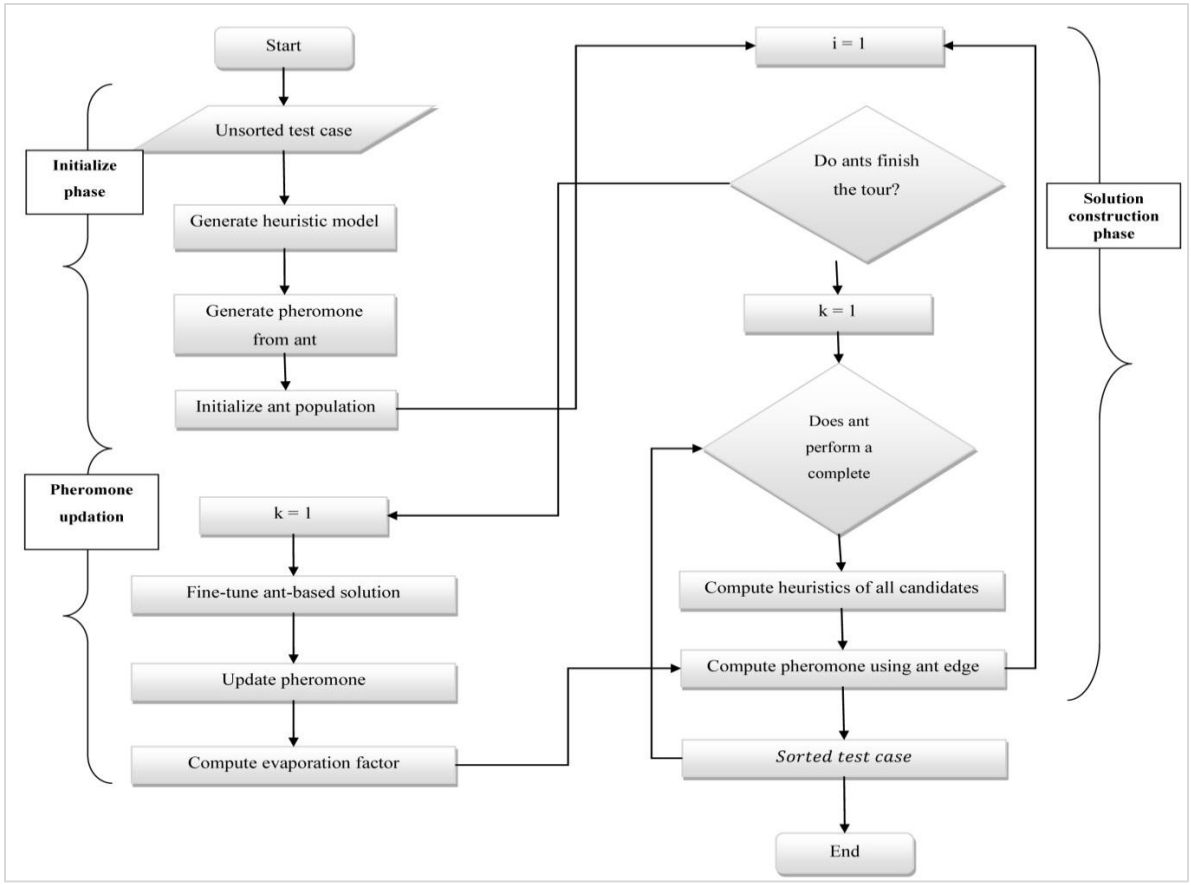

Figure 3 Flow diagram of ACO model

Table 2 Parental matrix of test cases and fault coverage

		Faults										
	Test cases	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**Execution time**
	T7	*			*							**1**
	T8	*	*					*		*	*	**2**
	T5	*			*						*	**21**
	T12	*			*	*					*	**23**
	T9	*										**4**
	T10	*					*					**7**
	T17				*						*	**27**
	T18			*	*							**29**
	T11	*									*	**9**
Test	T6	*							*	*		**10**
cases	T19				*				*		*	**31**
	T20				*						*	**33**
	T1	*									*	**13**
	T2	*			*							**14**
	T13				*							**35**
	T14				*						*	**38**
	T3	*		*							*	**16**
	T4	*			*							**18**
	T15				*							**40**
	T16	*			*						*	**41**

* Specified fault

Table 3 Sorted matrix based of Table 2 on execution time [34]

		Faults										
	Test cases	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**Execution time**
Test cases	T7	*										**1**
	T8	*	*					*		*	*	**2**
	T9	*										**4**
	T10	*					*					**7**
	T11	*									*	**9**
	T6	*							*	*		**10**
	T1	*									*	**13**
	T2	*			*							**14**
	T3	*		*							*	**16**
	T4	*			*							**18**
	T5	*			*						*	**21**
	T12	*			*	*					*	**23**
	T17				*						*	**27**
	T18			*	*							**29**
	T19				*				*		*	**31**
	T20				*						*	**33**
	T13				*							**35**
	T14				*						*	**38**
	T15				*							**40**
	T16				*					*	*	**41**

* Specified fault

First round of iteration

The matrix is converted into a binary form with 1's as the faults covered by the test cases and 0's being the uncovered faults. Next follows the test case assignment and coverage test.

Assigning test cases to ants - In this stage, the test cases are assigned to each ant. Every ant is given two test cases, each for N/4 number of ants in the sequential order. Now, half of the test cases are covered, and the remaining test cases are assigned to each and again from ant one in the same sequential order taking two test cases each. It will cover all the test cases. All ants are now ready for foraging with 4 test cases each (See *Figure 4*).

Fault coverage is checked to see if any of the 5 ants gives complete fault coverage. If achieved, then that ant provides the optimal solution. Else, the next round of iteration is started.

Table 4 depicts the round 1 assignment of ants and test cases with the least execution time. The total execution time is measured in seconds. The least time is 47 seconds and 67 seconds, respectively.

Table 4 Round I assigning ants and test cases with the least execution time

		Faults												
	Test cases	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**Execution time**	**Assigned Ant**	**Total ET**
Test cases	T7	1	0	0	1	0	0	0	0	0	0	1	A1	47
	T8	1	1	0	0	0	0	1	0	1	1	2	A1	
	T5	1	0	0	1	0	0	0	0	0	1	21	A1	
	T12	1	0	0	1	1	0	0	0	0	1	23	A1	
	T9	1	0	0	0	0	0	0	0	0	0	4	A2	67
	T10	1	0	0	0	0	1	0	0	0	0	7	A2	
	T17	0	0	0	1	0	0	0	0	0	1	27	A2	
	T18	0	0	1	1	0	0	0	0	0	0	29	A2	
	T11	1	0	0	0	0	0	0	0	0	1	9	A3	83
	T6	1	0	0	0	0	0	0	1	1	0	10	A3	
	T19	0	0	0	1	0	0	0	1	0	1	31	A3	
	T20	0	0	0	1	0	0	0	0	0	1	33	A3	
	T1	1	0	0	0	0	0	0	0	0	1	13	A4	100

							Faults						
T2	1	0	0	1	0	0	0	0	0	0	14	A4	
T13	0	0	0	1	0	0	0	0	0	0	35	A4	
T14	0	0	0	1	0	0	0	0	0	1	38	A4	
T3	1	0	1	0	0	0	0	0	0	1	16	A5	115
T4	1	0	0	1	0	0	0	0	0	0	18	A5	
T15	0	0	0	1	0	0	0	0	0	0	40	A5	
T16	1	0	0	1	0	0	0	0	1	1	41	A5	

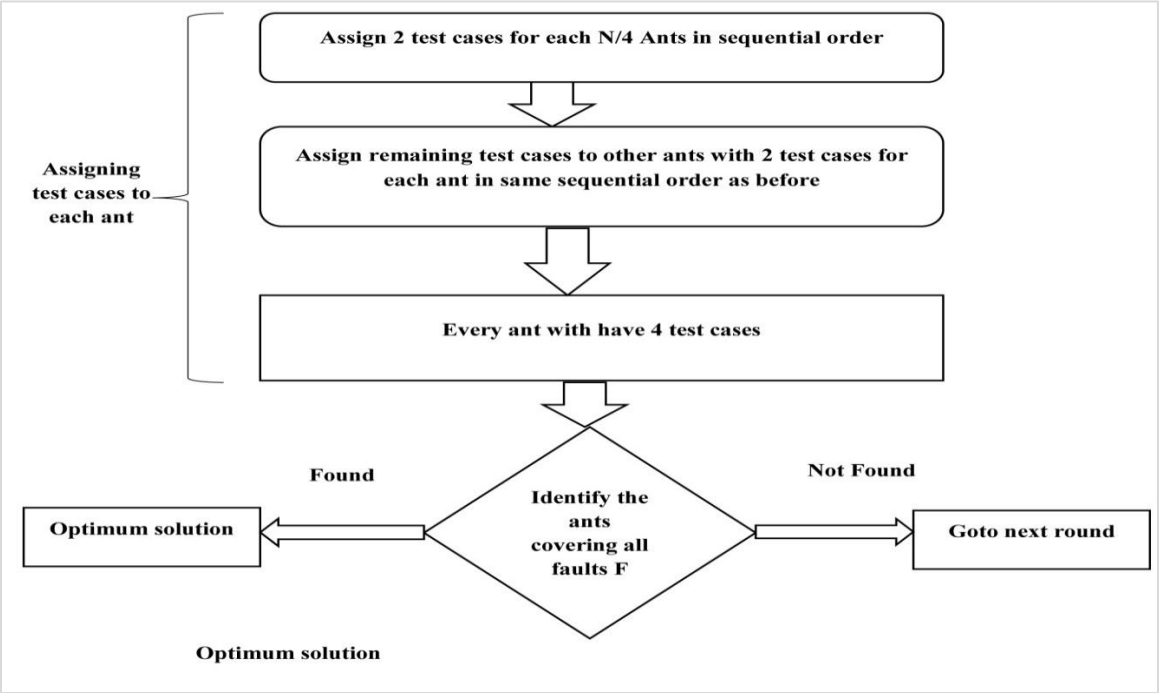

Figure 4 First round of iteration

Second round of iteration - procedure for selection and cross over

The resultant test cases are Ta Tb TC TD and Tc Td TA TB. Assign the two consequential test cases to new ants A6 and A7. In this problem, ants A1 and A2 have the least execution time. These ants are selected for the cross of the GA process (See *Figure 5*).

Cross Over A1, A2

$$A1 \times A2 = \begin{matrix} T7 & T8 \\ T5 & T12 \end{matrix} \times \begin{matrix} T9 & T10 \\ T17 & T18 \end{matrix}$$

The ants A1 and A2 are the ones with the least execution time. These ants are selected for the cross over GA process. These new combinations of test cases are assigned to new ants and in the existing data set.

T7 T8 T17 T18 - A6

T5 T12 T9 T10- A7

Fault coverage -Full fault coverage is not achieved. Thus, moving to the next round.

The third round of iteration

Adding new test cases- Two more specific test cases are randomly added to the four existing test cases. Now each ant will have 6 test cases. The test cases are Ta Tb Tc TD TE TF and Td TeTf TA TB TC. The ants A1 and A2 are the ones with the least execution time. These ants are selected for the cross over the process of the GA (*Figure 6*).

C*ross Over A1, A2*

$$A1 \text{ x } A2 = \begin{matrix} T7 & T8 & T1 \\ T2 & T5 & T12 \end{matrix} \times \begin{matrix} T7 & T8 & T9 \\ T10 & T17 & T18 \end{matrix}$$

$$A8 = \begin{matrix} T7 & T8 & T1 \\ T10 & T17 & T18 \end{matrix}$$

$$A9 = \begin{matrix} T7 & T8 & T1 \\ T2 & T5 & T12 \end{matrix}$$

Fault coverage -Full coverage of faults not achieved. Moving to next round.

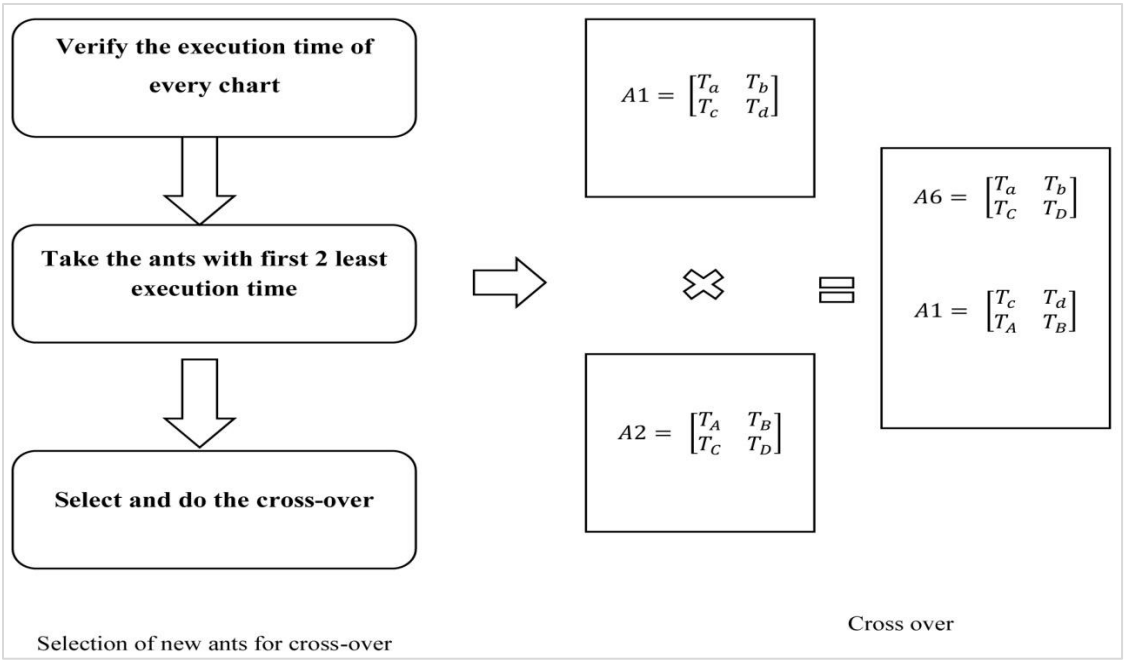

Figure 5 Second round of iteration of selection and cross over

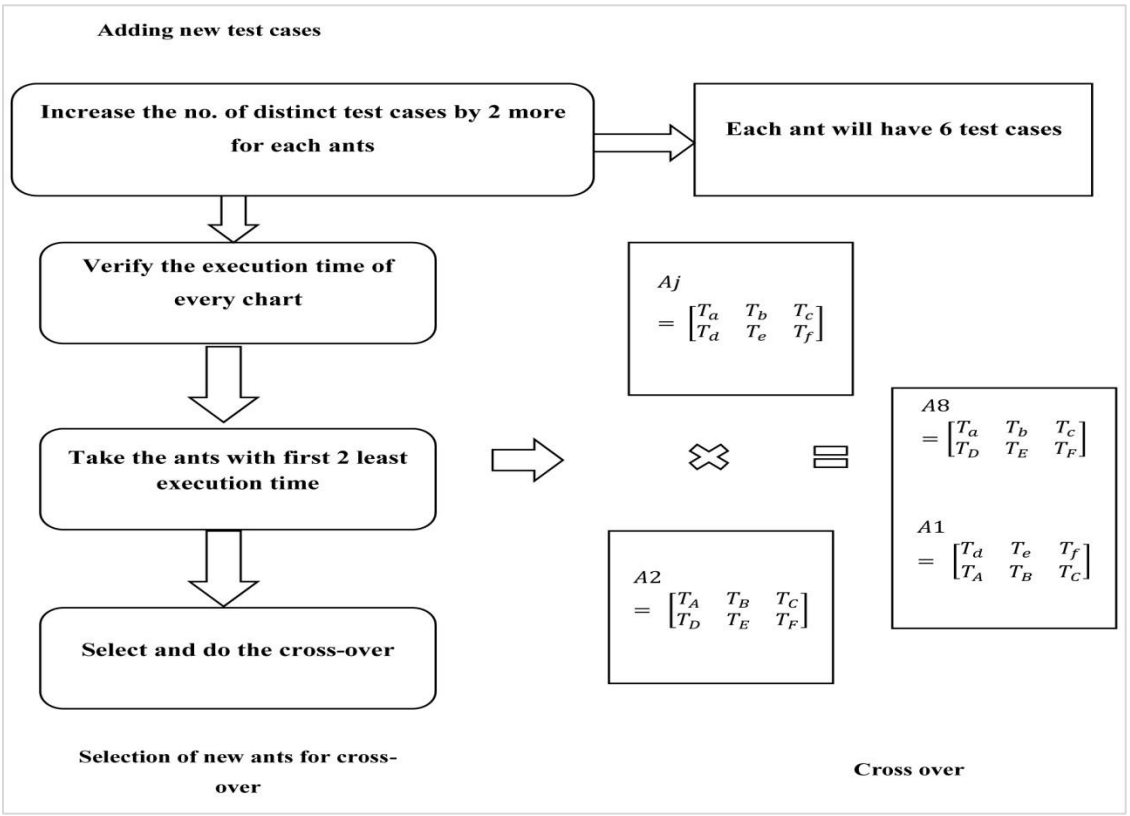

Figure 6 Third round of iteration

The fourth round of iteration

Adding new test cases- Two more specific test cases are randomly added to the 6 existing ones. Now each ant will have 8 test cases. The fault coverage is checked for every ant in the existing combination. The ant that satisfies full coverage of faults is taken as the optimum solution. Complete fault coverage is achieved at this point.

4.Results and discussion

The ant A8 has a full coverage of faults from round 4. It is identified as the solution for the problem. The iteration stops here. The initial count of 20 test cases is minimized to 10 test cases. The optimum solution is T2 T6 T7 T8 T10 T12 T17 T18 T19. *Figure 7* depicts the ant's traversal path. Based on the representation, the fault is plotted from $F1 \rightarrow F2 \rightarrow F3 \rightarrow F4 \rightarrow F5 - F6 \rightarrow F7 \rightarrow F8 \rightarrow F9 \rightarrow F10$. The plan is provided in contrast to the ant's traversal path and the total number of faults. *Figure 8* shows the TCP process of the non-prioritization approach, GA+ACO and the adaptive model, and the proposed model consumes less time for prioritization and offers superior performance. The numbers of iterations are shown in *Figure 9*. The number of iterations is evaluated here, and the comparison is made. Based on all these analyses, the anticipated model works more effectually than the other.

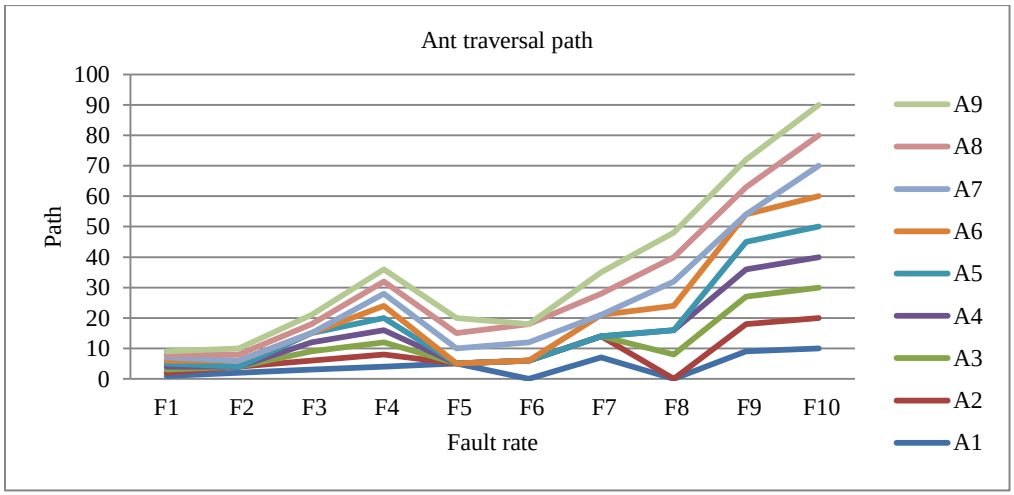

Figure 7 Ant traversal path

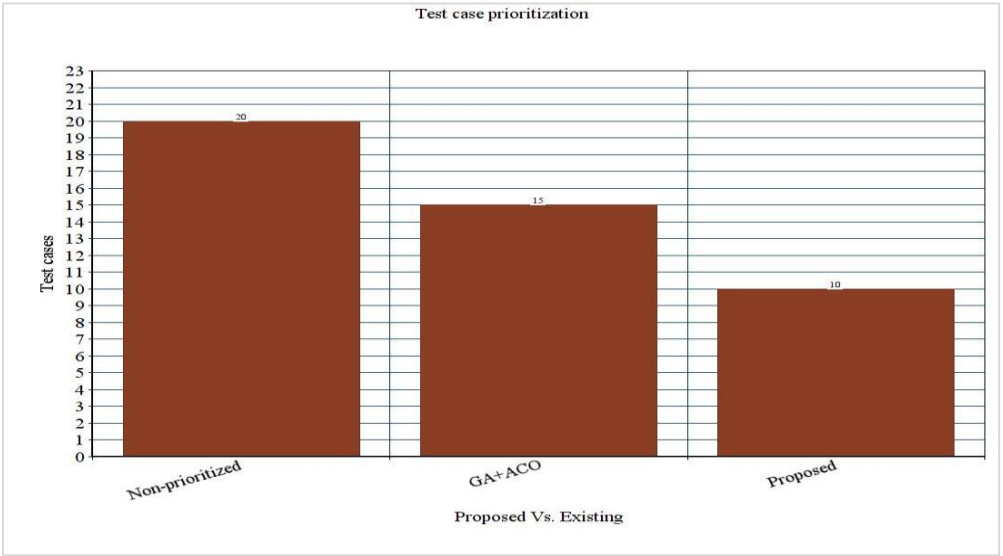

Figure 8 Test case prioritization

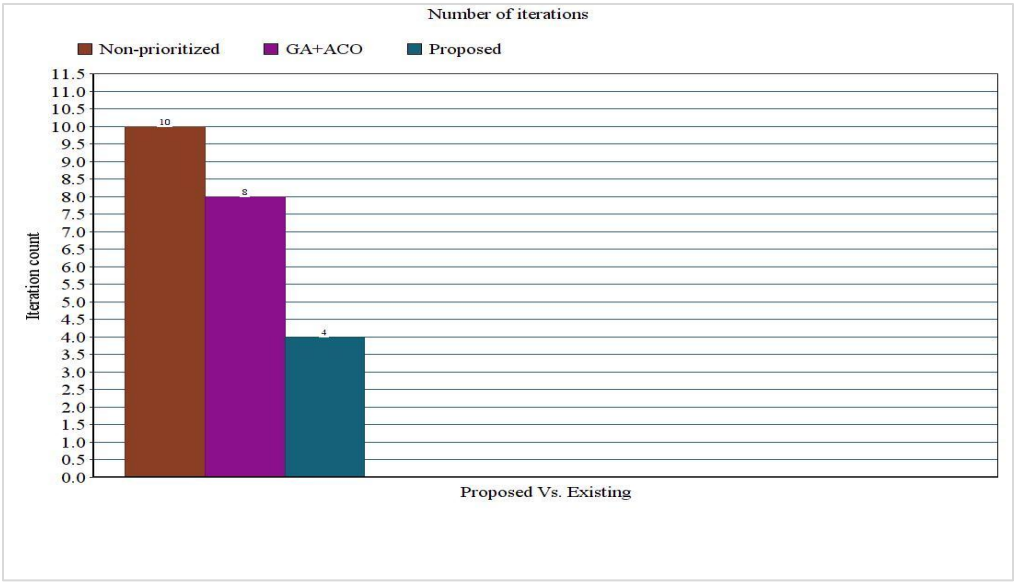

Figure 9 No. of iterations carried out

The functionality of the proposed model is evaluated to diminish the test case suites and competency to identify faults during RT [32−36]. The following are metrics considered for evaluation.

Reduction percentage: Assume $'T'$ is the total number of test cases accessed over the database and $'T'$ is the number of test cases chosen for regression. It is expressed as in Equation 4:

$$TR = \frac{T-T'}{T} \times 100 \quad (4)$$

Precision: Assume, $T'f$ be set of test cases chosen from DB to reveal the faults where $'P'$ is expressed using Equation 5:

$$Precision = \frac{T'f}{T'} \quad (5)$$

Recall: Assume, $T'f$ is a set of test cases with faults where the metrics are expressed as in Equation 6:

$$recall = \frac{|T'f|}{|T_f|} \quad (6)$$

Fault detection: It is directly related to recall. Equation 6 is utilized to compute the fault detection ability of chosen test cases.

F-measure: It integrates recall (R) and precision (P). It is expressed as in Equation 7:

$$F - measure = 2 * \frac{precision*recall}{precision+recall} \quad (7)$$

Figure 10 depicts the comparison of performance metrics like TR, recall, precision, F-measure with 10 iterations. The recall value of proposed model is 100%, precision value ranges from 8% to 29%, TR value ranges from 6.5%-80.8%, and the f-measure value ranges from 0.13 to 0.45 respectively. Here, total numbers of iterations are 10 with 100% recall for all the iterations; TR, precision and F1-measure changes for the provided iteration. The TR for all the 10 iterations are 9.3%, 9.9%, 26.5%, 80.8%, 6.5%, 26.8%, 18.8%, 6.5%, 9.9% and 9.3%. The precision values are 80%, 80%, 90%, 29%, 8%, 9%, 9%, 8%, 8% and 8%. The F1-measure is 0.13, 0.14, 0.16, 0.45, 0.13, 0.16, 0.15, 0.13, 0.14, and 0.13 respectively.

Table 5 depicts the execution time in minutes. The test case ranges from T_1 *to* T_{20} where the execution time is measured in milliseconds where the execution time for all these test cases are 13 ms, 14 ms, 16 ms, 18 ms, 21 ms, 10 ms, 1 ms, 2 ms, 4 ms, 7 ms, 9 ms, 23 ms, 35 ms, 38 ms, 40 ms, 41 ms, 27 ms, 29 ms, 31 ms, and 33 ms respectively. *Table 5* depicts the comparison of proposed and existing methods for mutant 1 to mutant 8, respectively. The proposed multi-objective GA with ACO model gives better outcomes with 94.5%, 94.8%, 93.8%, 92.5%, 95.8%, 97.8%, 99.2%, and 93.5% respectively. *Table 6* compares proposed vs. existing approaches where 8 mutants are considered and compared with existing approaches like convolutional GA, simulated annealing (SA), ACO and GA. The proposed model gives 94.5%, 94.8%, 93.8%, 92.5%, 95.8%, 97.8%, 99.2% and 93.5% fault prediction rate for 1 to 8 mutants which is comparatively higher than conventional GA, SA, ACO and multi-objective GA.

The mutant achieved with the proposed multi-objective GA and the ACO is 94.5, 94.8, 93.8, 92.5,

95.8, 97.8, 99.2 and 93.5, respectively. Based on all these performance metrics, the anticipated model shows a better trade-off than other TCP approaches [37−40]. But, the primary research limitation is the lack of real-time datasets from various organizations like Facebook and Twitter. This research significantly concentrates on modelling an efficient TCP using ACO with GA to enhance the prediction rate of the fault during the process of RT. The idea of fault prediction diversity is done by altering the selection of food source criteria using the nature-inspired ant algorithm. This model modifies the flow of ACO uses the food source (faults), food source evaluation (faults handled by the provided test cases), and natural ants selection criteria (test cases) before the selection of the food source (test case selection). Some real ants in ACO do not measure the uniqueness or food fitness before selecting a food source with the adaptive ACO+GA and select the food that deals with the fitness criteria and improves the food source diversity. The proposed multi-objective GA with ACO model gives better outcomes with 94.5%, 94.8%, 93.8%, 92.5%, 95.8%, 97.8%, 99.2%, and 93.5% respectively. Also, the anticipated model is 27%, 18%, 40% and 5% higher than non-prioritized test cases, reverse order prioritization, random order prioritization and hybrid artificial bee colony with GA. Any test case prioritization is considered as a better model than the other contemporary approaches. Thus, the anticipated ACO+GA are used to prove the significance of the model with superior fault detection percentage and improve the fault diversity even in smaller test suites. A complete list of abbreviations is shown in *Appendix I.*

Table 5 Execution time (minutes)

Test case	Execution time (minutes)
T_1	13
T_2	14
T_3	16
T_4	18
T_5	21
T_6	10
T_7	1
T_8	2
T_9	4
T_{10}	7
T_{11}	9
T_{12}	23
T_{13}	35
T_{14}	38
T_{15}	40
T_{16}	41
T_{17}	27
T_{18}	29
T_{19}	31
T_{20}	33

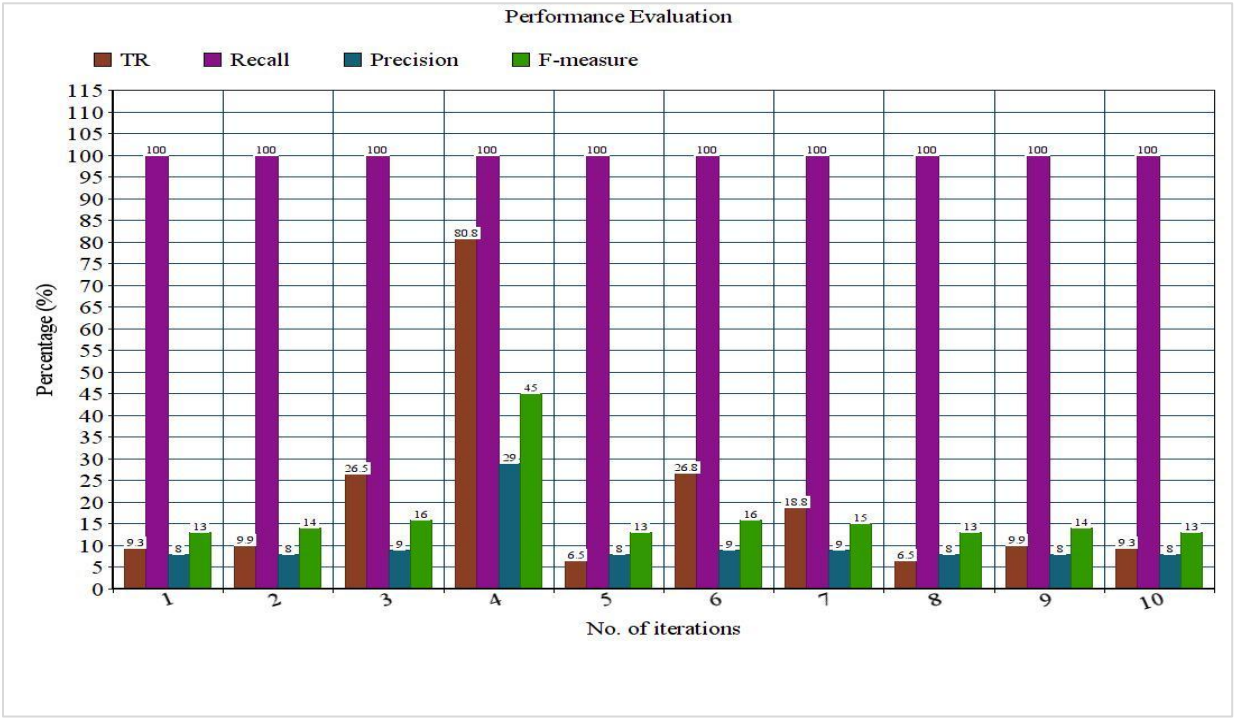

Figure 10 Performance metrics evaluation

Table 6 Comparison of proposed versus existing approaches

Mutant	Conventional GA	Conventional SA (Simulated annealing)	Conventional ACO	Multi-objective GA	Multi-objective GA with ACO
Mutant 1	81	92.2	88.6	91.2	94.5
Mutant 2	83	92.7	87.5	90.8	94.8
Mutant 3	88	91.8	90.1	88.6	93.8
Mutant 4	81	90.7	91.8	86.5	92.5
Mutant 5	81.5	93.5	89.7	93.4	95.8
Mutant 6	81	95.3	89.3	96.5	97.8
Mutant 7	80	99.1	82.3	94.2	99.2
Mutant 8	82.2	89.2	90.7	88.3	93.5

5.Conclusion and future work

RT is essential for establishing software maintenance and quality assurance. TCP has attained the researcher's interest for re-combining the test cases indeed of permanent removal. Here, TCP is performed with meta-heuristic optimization approaches like ACO and GA. Young researchers extensively explore these approaches. The GA and ACO approach efficiently achieves the TCP based on the extensive analysis.

Therefore, this research extensively analyses RT uses TCP with the adoption of GA and ACO. Therefore, there is a rigorous impact over the coverage criteria, test case and tools used for analysis. The critical factors of GA are population size and representation, operator type and generation size, fitness function, and operator's probabilistic rate. Similarly, with ACO, the factors like population initialization, evaluation, construction, ranking, and pheromone updation are performed efficiently. The fault detection rate of multi-objective (ACO and GA) model provides outcome of 94.5%, 94.8%, 93.8%, 92.5%, 95.8%, 97.8%, 99.2%, and 93.5% respectively. Similarly, the execution times for all the provided test cases are 3, 3, 4, 2, 5, 4, 3, and 5 minutes respectively. Similarly, other performance metrics like recall, F-measure, TR, and precision are also evaluated to analyze the efficiency of the proposed model. In future, this work will be extended with the adoption of a real-time TCP dataset, and evaluation will be made with these above-mentioned statistical measures.

Acknowledgment

None.

Conflicts of interest

The authors have no conflicts of interest to declare.

Author's contribution statement

T.K. Akila: Conceptualization, investigation, data curation, writing – original draft, writing – review and editing. **A. Malathi:** Study conception, design, data collection, supervision, investigation on challenges and draft manuscript preparation.

References

[1] Yoo S, Harman M. Regression testing minimization, selection and prioritization: a survey. Software Testing, Verification and Reliability. 2012; 22(2):67-120.

[2] Yuan F, Bian Y, Li Z, Zhao R. Epistatic genetic algorithm for test case prioritization. In international symposium on search based software engineering 2015 (pp. 109-24). Springer, Cham.

[3] Bajaj A, Sangwan OP. A survey on regression testing using nature-inspired approaches. In 4th international conference on computing communication and automation 2018 (pp. 1-5). IEEE.

[4] Wang S, Buchmann D, Ali S, Gotlieb A, Pradhan D, Liaaen M. Multi-objective test prioritization in software product line testing: an industrial case study. In proceedings of the international software product line conference 2014 (pp. 32-41).

[5] Ansari A, Khan A, Khan A, Mukadam K. Optimized regression test using test case prioritization. Procedia Computer Science. 2016; 79:152-60.

[6] Huang YC, Peng KL, Huang CY. A history-based cost-cognizant test case prioritization technique in regression testing. Journal of Systems and Software. 2012; 85(3):626-37.

[7] Briand L, Labiche Y, Chen K. A multi-objective genetic algorithm to rank state-based test cases. In international symposium on search based software engineering 2013 (pp. 66-80). Springer, Berlin, Heidelberg.

[8] Pradhan D, Wang S, Ali S, Yue T, Liaaen M. CBGA-ES: a cluster-based genetic algorithm with elitist selection for supporting multi-objective test optimization. In international conference on software testing, verification and validation 2017 (pp. 367-78). IEEE.

[9] Srikanth H, Hettiarachchi C, Do H. Requirements based test prioritization using risk factors: an

industrial study. Information and Software Technology. 2016; 69:71-83.
[10] Khanna M, Chauhan N, Sharma D, Toofani A, Chaudhary A. Search for prioritized test cases in multi-objective environment during web application testing. Arabian Journal for Science and Engineering. 2018; 43(8):4179-201.
[11] Suri B, Singhal S. Implementing ant colony optimization for test case selection and prioritization. International Journal on Computer Science and Engineering. 2011; 3(5):1924-32.
[12] Geng J, Li Z, Zhao R, Guo J. Search based test suite minimization for fault detection and localization: a co-driven method. In international symposium on search based software engineering 2016 (pp. 34-48). Springer, Cham.
[13] Li Z, Harman M, Hierons RM. Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering. 2007; 33(4):225-37.
[14] Yoo S, Harman M, Ur S. Highly scalable multi objective test suite minimisation using graphics cards. In international symposium on search based software engineering 2011 (pp. 219-36). Springer, Berlin, Heidelberg.
[15] Mei H, Hao D, Zhang L, Zhang L, Zhou J, Rothermel G. A static approach to prioritizing junit test cases. IEEE Transactions on Software Engineering. 2012; 38(6):1258-75.
[16] Yoo S, Harman M, Tonella P, Susi A. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. In proceedings of the eighteenth international symposium on software testing and analysis 2009 (pp. 201-12).
[17] Marchetto A, Islam MM, Asghar W, Susi A, Scanniello G. A multi-objective technique to prioritize test cases. IEEE Transactions on Software Engineering. 2015; 42(10):918-40.
[18] Di ND, Panichella A, Zaidman A, De LA. A test case prioritization genetic algorithm guided by the hypervolume indicator. IEEE Transactions on Software Engineering. 2018; 46(6):674-96.
[19] Guo P, Wang X, Han Y. The enhanced genetic algorithms for the optimization design. In international conference on biomedical engineering and informatics 2010 (pp. 2990-4). IEEE.
[20] Laali M, Liu H, Hamilton M, Spichkova M, Schmidt HW. Test case prioritization using online fault detection information. In Ada-Europe international conference on reliable software technologies 2016 (pp. 78-93). Springer, Cham.
[21] Khatibsyarbini M, Isa MA, Jawawi DN, Hamed HN, Suffian MD. Test case prioritization using firefly algorithm for software testing. IEEE Access. 2019; 7:132360-73.
[22] Jahan H, Feng Z, Mahmud SM. Risk-based test case prioritization by correlating system methods and their associated risks. Arabian Journal for Science and Engineering. 2020; 45(8):6125-38.
[23] Solanki K, Singh Y, Dalal S, Srivastava PR. Test case prioritization: an approach based on modified ant colony optimization. In emerging research in computing, information, communication and applications 2016 (pp. 213-23). Springer, Singapore.
[24] Kundu D, Sarma M, Samanta D, Mall R. System testing for object-oriented systems with test case prioritization. Software Testing, Verification and Reliability. 2009; 19(4):297-333.
[25] Kaur P, Bansal P, Sibal R. Prioritization of test scenarios derived from UML activity diagram using path complexity. In proceedings of the CUBE international information technology conference 2012 (pp. 355-9).
[26] Sharma S, Singh A. Model-based test case prioritization using ACO: a review. In international conference on parallel, distributed and grid computing 2016 (pp. 177-81). IEEE.
[27] Niu H, Keivanloo I, Zou Y. Learning to rank code examples for code search engines. Empirical Software Engineering. 2017; 22(1):259-91.
[28] Ye X, Bunescu R, Liu C. Learning to rank relevant files for bug reports using domain knowledge. In proceedings of the ACM SIGSOFT international symposium on foundations of software engineering 2014 (pp. 689-99).
[29] Yu YT, Lau MF. Fault-based test suite prioritization for specification-based testing. Information and Software Technology. 2012; 54(2):179-202.
[30] Paiva AC, Faria JC, Tillmann N, Vidal RA. A model-to-implementation mapping tool for automated model-based GUI testing. In international conference on formal engineering methods 2005 (pp. 450-64). Springer, Berlin, Heidelberg.
[31] Singh G, Gupta D. An integrated approach to test suite selection using ACO and genetic algorithm. International Journal of Advanced Research in Computer Science and Software Engineering. 2013; 3(6):1770-8.
[32] Tahat L, Korel B, Koutsogiannakis G, Almasri N. State-based models in regression test suite prioritization. Software Quality Journal. 2017; 25(3):703-42.
[33] Nayak S, Kumar C, Tripathi S. Enhancing efficiency of the test case prioritization technique by improving the rate of fault detection. Arabian Journal for Science and Engineering. 2017; 42(8):3307-23.
[34] Wang Y, Zhu Z, Yang B, Guo F, Yu H. Using reliability risk analysis to prioritize test cases. Journal of Systems and Software. 2018; 139:14-31.
[35] Jahan H, Feng Z, Mahmud SM, Dong P. Version specific test case prioritization approach based on artificial neural network. Journal of Intelligent & Fuzzy Systems. 2019; 36(6):6181-94.
[36] Khatibsyarbini M, Isa MA, Jawawi DN, Tumeng R. Test case prioritization approaches in regression testing: a systematic literature review. Information and Software Technology. 2018; 93:74-93.
[37] Fu W, Yu H, Fan G, Ji X, Pei X. A regression test case prioritization algorithm based on program changes and method invocation relationship. In Asia-pacific

software engineering conference 2017 (pp. 169-78). IEEE.
[38] Dash U, Acharya AA. A systematic review of test case prioritization approaches. In proceedings of international conference on advanced computing applications 2022 (pp. 653-66). Springer, Singapore.
[39] Gupta P. Test case prioritization based on requirement. In communication and intelligent systems 2021 (pp. 309-14). Springer, Singapore.
[40] Hettiarachchi C, Do H, Choi B. Effective regression testing using requirements and risks. In eighth international conference on software security and reliability 2014 (pp. 157-66). IEEE.

T.K. Akila is a Ph.D student in the field of Computer Science in Government Arts College in the Department of Computer Science, Coimbatore, Tamil Nadu. She has a Bachelor's Degree in Mathematics from Nirmala College for Women and Master's Degree in Computer Application from Bharathiar University, Coimbatore. Her research interests include Software Testing Techniques and Software Management.
Email: akila.tk123@gmail.com

Dr. A. Malathi is currently working as an Assistant Professor in Government Arts College in the Department of Computer Science, Coimbatore, Tamil Nadu. She has worked at various universities like Bharathiar University, Bharathidasan University, Periyar University and Bangalore University. She has over 26 years of experience in the field of teaching. Her wide area of interest is Networks.
Email: malathi.arunachalam@yahoo.com

Appendix I

S. No.	Abbreviation	Description
1	ACO	Ant Colony Optimization
2	CFG	Control Flow Graph
3	DB	Database
4	ET	Execution Time
5	FSM	Finite State Machine
6	GA	Genetic Algorithm
7	RT	Regression Testing
8	SA	Simulated Annealing
9	SDLC	Software Development Life Cycle
10	TCP	Test Case Prioritization
11	UML	Unified Modelling Language