International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 /doi:10.14445/22315381/IJETT-V69I4P230

Volume 69 Issue 4, 215-227, April 2021

An Efficient APHT Technique for Requirement-Based Test Case Prioritization

Omdev Dahiya[#1], Kamna Solanki[*2]

Department of Computer Science and Engineering, University Institute of Engineering and Technology, Maharshi Dayanand University Rohtak, Haryana, India

[1]Omdahiya21792@gmail.com, [2]Kamna.mdurohtak@gmail.com

***Abstract* -** *Software testing is carried out to ensure that the developed software product is fault-free and is meeting the expected requirement criteria. If there are defects in the software, then testing will help in uncovering them so that they can be timely fixed. This will improve the quality and reliability of the software, thus providing customer satisfaction as well as time and cost-saving by timely fixing the identified faults. There are various techniques for testing the software. With limited time and budget, exhaustive testing is not possible. One of the efficient techniques is regression testing, which helps in testing the modified part of the software. One of the approaches for regression testing is Test Case Prioritization (TCP) which executes test cases in a priority order instead of executing the whole test suite with a motive to enhance the rate of fault detection as software is developed based on its requirements so it is beneficial to test those requirements first which are complex as they will be the ones where there is maximum possibility of occurrence of faults. This study has worked towards proposing a new Ant colony and Particle swarm optimization Hybrid Technique (APHT) for requirements-based test case prioritization. For this, an industrial case study is taken, and faults are injected into it based on various requirements factors. To show the effectiveness of our proposed technique, a metric known as Average Percentage of Faults Detected (APFD) and average computation are taken, and their values are compared with other existing approaches. The results obtained showed the worthiness of the proposed requirement-based TCP Technique.*

Keywords - *Requirement-based Test Case Prioritization, Ant colony and Particle swarm optimization Hybrid Technique (APHT), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Software Testing.*

I. INTRODUCTION

With the evolution of technology and increasing dependence of humans on software-enabled devices, there is the necessity to develop robust software. The human race is continuously getting involved in smart devices, which is the need of the hour to sustain in competing environments. These devices run on software, and that's why the importance of developing quality and reliable software is increasing at a fast pace. This can be achieved by testing the software which aims to enhance its quality, as this will lead to the satisfaction of the customers. Timely fault fixing is necessary as early detected faults can be fixed in a cost-effective and timely way as compared to the moment when they are detected late in the product [1-3]. The faith of a customer in an organization is also lowered if the delivered software by that organization fails to deliver its specified functionality. The image of a developing software organization is also at stake. Companies test the software so that faults can be identified and fixed before delivering them to the customers. One of the techniques for testing the software thoroughly is exhaustive testing, but it is not practically possible due to limited time and budget [4-5], [46]. Generally, 20 percent of the system is accountable for 80% of errors. Thus, there is a requirement to increase the efficiency of testing resources that recognizes more severe faults early.

Manual testing is also not possible due to continuously increasing test code size and complexities involved. So, automation testing is required in which effective test cases are generated, which are then selected and can be prioritized for execution by encompassing various metaheuristics techniques. Over the years, soft computing has emerged for finding an efficient solution to various optimization problems using various nature-inspired metaheuristics algorithms [6].

This study has worked towards proposing a new Ant colony and Particle swarm optimization Hybrid Technique (APHT) for requirements-based test prioritization. The paper is organized as follows- Section II provides a literature review of the articles in which various requirement-based test prioritization techniques have been addressed. Section III discusses the proposed Ant colony and Particle swarm optimization Hybrid Technique (APHT) and also provides a research methodology flow chart and Pseudocode of the proposed technique. Section IV performs the experimental analysis of the proposed technique. Section V presents the comparative performance analysis of the proposed technique with other prevailing approaches. Section VI presents the conclusion and future scope of the study.

II. RELATED WORK

This section provides an overview of studies performed for test case prioritization along with the studies where metaheuristics algorithms are used for test case prioritization in the software testing domain.

In the study conducted by **Jarzabek *et al.* (2020),** the authors have proposed a technique for test case mapping with the respective test cases. They have shown that due to this, the defects can be addresses with manageable efforts during the regression testing procedure [7].

In work performed by **Nayak *et al.* (2020),** the authors have proposed an enhanced test case prioritization technique. They have compared the results of their technique with other prevailing approaches to show its effectiveness [8].

In the study performed by **Yaseen *et al.* (2019),** the authors proposed a technique for assigning the ranks to the requirements and have demonstrated that how the timely delivery of the efficient product is assured by this technique [9].

In the study performed by **Dhiman & Chopra (2019),** the authors have used the ACO algorithm for proposing an automated technique for regression test case prioritization and have shown increased fault detection in minimum time by their approach [10].

In the study conducted by **Alzaqebah *et al.* (2018),** the authors have used a nature-inspired algorithm to propose a requirement-based test case prioritization technique. They have shown the effectiveness of their approach by comparing it with other approaches [11].

In the study conducted by **Masadeh *et al.* (2018),** the authors have proposed a requirements prioritization technique using nature-inspired technique and compared the results generated by their technique with other existing approaches to show its effectiveness [12].

In work performed by **Khatibsyarbini *et al.* (2017),** the authors have used Particle Swarm Optimization (PSO) for prioritizing the test cases and have shown the supremacy of their technique via higher APFD values [13].

In the study conducted by **Ashraf *et al.* (2017),** the authors have proposed a test case prioritization technique using the PSO algorithm. They have used requirement-based factors in their proposed technique which demonstrates the novelty of their research work. They have shown the efficiency of their approach via higher APFD values [14].

In work performed by **Kumar & Ranjan (2017),** the authors have proposed a modified ACO approach to prioritize the test cases. The experimental results showed that the maximum number of faults are discovered by the proposed technique in the minimum time, and the value of APFD is also high [15].

In the study conducted by **Ansari *et al.* (2016),** the authors proposed a test case prioritization technique using the ACO algorithm. They have demonstrated how their proposed approach has worked towards reducing execution time, cost and discovers the maximum number of faults [16].

In the study conducted by **Srikant *et al.* (2016),** the authors have taken different factors for proposing a requirement-based test case prioritization approach and have shown the effectiveness of their proposed technique [17].

In the study conducted by **Gao & Zhao (2015),** the authors proposed an Ant Colony based test case prioritization technique considering the severity of faults, execution time, and the number of faults detected. They have shown the effectiveness of their technique using the Average Percentage of Faults Detected (APFD) metric [18].

In the work performed by **Tyagi & Malhotra (2014),** the authors have proposed a test case prioritization technique using the PSO algorithm. To show the effectiveness of their approach via higher values of APFD in the least execution time when compared to other existing approaches [19].

In the work performed by **Muthusamy (2014), the** author proposed a test case prioritization technique based on requirement factors and showed the worthiness of their technique via increased fault detection rate [20].

In the work performed by **Suri & Singhal (2011),** the authors have used ACO for analyzing test case selection and prioritization techniques. They have demonstrated that how effectively ACO has delivered efficient results by reducing the number of test cases required for fault detection in a timely manner [21].

In the study conducted by **Krishnamoorthi *et al.* (2008),** the authors proposed a Requirements-based TCP method so that rate of fault identification in novel and regression experiments could be enhanced. Average Test Effort Index (ATEI) was calculated, which showed that test cases carried out to calculate injected faults were minimum. Then sign test was run so that hypothesis can be evaluated, and it proved that this method gave better results [22].

In the study conducted by **Srikant *et al.* (2005),** the authors proposed as "PORT- prioritization of requirements for test" technique to provide criteria for the importance of the requirements [23].

Work of requirements-based test case prioritization is also performed by various researchers owing to the research trends in this field [24-36], [45].

- ***Analysis of requirement-based factors taken into consideration for proposing Requirement based test case prioritization technique.***

In this research work, four factors are taken into consideration which is (1) "Customer assigned priority of requirements" (2) "Developer-perceived code implementation complexity," (3) "Changes in requirements" (4) "Fault impact" [37], [26]. Subsequently, the impacts of

these factors on the requirements are calculated by assigning values on a ten-point scale. The weight of each requirement is computed with these factor values. In the second step, the test cases are mapped towards corresponding requirements.

1) *"Customer-assigned Priority"* (CP): This is a measure by which a customer assigns a value for a requirement, based upon its importance to him. Here, the value of a requirement is assigned from 1 to 10, where 1 denotes the lowest priority requirement and 10 denotes the highest priority requirement according to the customer.

- *Reasoning:* According to this, Customer satisfaction and its perceived value is increased if the requirements are developed focusing on the needs of the customer. So, to improve the satisfaction rate of the customer, the highest important requirement according to the customer should be tested at the earliest and thoroughly.

2) *"Implementation Complexity"* (IC): This is a measure by which it is computed how the development team sees the particular requirement implementation. After analyzing every requirement, a value is assigned to them ranging from 1 to 10, where 1 indicates the lowest complex requirement to be implemented, and 10 indicates the highest complex requirement for implementation.

- *Reasoning*: If a requirement is complex to be implemented, then the chances are that number of faults in it will be the highest.

3) *"Requirement Changes"* (RC): It is a measure to show how many times a particular requirement has been changed starting from its initial stage of origin. Then, the developer assigns values to them from 1 to 10. A 10-point scale is used for quantifying the requirements if certain requirements change more than 10 times. The change for the requirement *p (RCp)* is calculated by dividing it with the figure by which how many times a particular requirement is changed to the highest change in requirements number amongst the requirements of the project. If the pa^{th} requirement is changed *Q* times and the highest number for change in requirements amongst the project requirements is *R*, then the requirement change of *p*, RC*p* is calculated as

$$RCp = (Q / R) \times 10$$

- *Reasoning:* Due to errors committed in the requirements phase, on average, 50% of the faults are introduced. So, the main contribution towards project failure is due to the continuously changing rate of the requirements.

4) *"Fault Impact of requirements"* (FI): It is a measure by which those requirements are identified by the development team in which failure has been reported by the customers. As a product keeps on evolving following its updating from an old version to a new version, the data is collected and used by the developers for identification of the requirements that were most prone to the errors. FI is taken into consideration for the requirements that were already implemented in a product that has been released.

- *Reasoning*: The efficiency of testing will be improved by targeting those areas which are most likely to contain faults in high number.

After assigning values of the corresponding factors for each requirement, factor values are computed for each factor which is computed as –

CPV ("Customer assigned Priority Value") is computed by taking 39% (weight) of the CP value. ICV ("Implementation Complexity Value") is computed by taking 20% (weight) of the IC value. CRV ("Change in Requirements Value") is computed by taking 5% (weight) of the CR value. FIV ("Fault Impact Value") is computed by taking 5% (weight) of the FI value.

After computing these values, the Requirement Factor Value (RFV) for each requirement is computed as-

$$RFV = (CPV+ICV+CRV+FIV)/4;$$

Based upon the RFV value, faults were seeded into the system corresponding to the requirement. A higher RFV value for a requirement states that it is a complex requirement, and there are more chances of faults in that requirement [37].

Accordingly, processed form of data is provided in Table 3.1 by taking an industrial case study of "Cosmosoft Technologies which involves the analysis of student self-service portal written in JAVA language, that comprises 91,733 changed lines of code, built upon approximately 350,000 lines of the code base, 25 modified requirements; and 100 system level test cases" which shows the values of all the factors of the requirements and the test cases according to the requirements.

Table 3.1 Processed Requirement based Dataset- Industrial case study of Cosmosoft Technologies

		(39% of CP value)		(20% of IC value)		(5%of CR value)		(5% of FI value)	(CPV+ICV+ CRV+FIV)/4		
		CPV=CP *Weight (CP)		ICV= IC * Weight (IC)		CRV=CR *Weight (CR)		FIV=FI* Weight (FI)			
	CP	CPV	IC	ICV	CR	CRV	FI	FIV	RFV	Test Cases	No. Of Test Cases
R1	9	3.51	8	1.6	5	0.25	2	0.1	1.37	T1, T2, T3, T4, T5, T6	6
R2	8	3.12	6	1.2	4	0.2	1	0.05	1.14	T5, T6	2
R3	9	3.51	7	1.4	3	0.15	2	0.1	1.29	T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18	12
R4	6	2.34	7	1.4	3	0.15	1	0.05	0.99	T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18	12
R5	9	3.51	7	1.4	5	0.25	2	0.1	1.32	T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18	12
R6	5	1.95	4	0.8	5	0.25	1	0.05	0.76	T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18	12
R7	6	2.34	9	1.8	2	0.1	3	0.15	1.10	T19, T20, T21, T22, T23, T24, T25, T26, T27, T28, T29, T30, T31, T32, T33, T34	16
R8	7	2.73	5	1	3	0.15	2	0.1	1.00	T19, T20, T21, T22, T23, T24, T25, T26, T27, T28, T29, T30, T31, T32, T33, T34	16
R9	6	2.34	9	1.8	4	0.2	1	0.05	1.10	T19, T20, T21, T22, T23, T24, T25, T26, T27, T28, T29, T30, T31, T32, T33, T34	16
R10	8	3.12	8	1.6	5	0.25	2	0.1	1.27	T19, T20, T21, T22, T23, T24, T25, T26, T27, T28, T29, T30, T31, T32, T33, T34	16
R11	4	1.56	3	0.6	5	0.25	2	0.1	0.63	T19, T20, T21, T22, T23, T24, T25, T26, T27, T28, T29, T30, T31, T32, T33, T34	16
R12	8	3.12	7	1.4	2	0.1	1	0.05	1.17	T35, T36, T37, T38, T39, T40, T41	7
R13	6	2.34	7	1.4	2	0.1	1	0.05	0.97	T42, T43, T44	3
R14	5	1.95	5	1	3	0.15	2	0.1	0.80	T42, T43, T44	3
R15	9	3.51	3	0.6	5	0.25	2	0.1	1.12	T45, T46, T47, T48, T49, T50	6
R16	4	1.56	7	1.4	3	0.15	1	0.05	0.79	T45, T46, T47, T48, T49, T50	6
R17	6	2.34	5	1	2	0.1	2	0.1	0.89	T51, T52	2
R18	7	2.73	3	0.6	1	0.05	1	0.05	0.86	T53, T54, T55, T56, T57, T58, T59, T60, T61, T62	9
R19	5	1.95	5	1	5	0.25	2	0.1	0.83	T53, T54, T55, T56, T57, T58, T59, T60, T61, T62	9
R20	3	1.17	7	1.4	4	0.2	3	0.15	0.73	T63, T64, T65, T66, T67, T68, T69	7
R21	6	2.34	5	1	5	0.25	2	0.1	0.92	T70, T71, T72, T73, T74, T75	6

R22	9	3.51	9	1.8	5	0.25	2	0.1	1.42	T76, T77, T78, T79, T80, T81, T82, T83, T84, T85, T86	10
R23	8	3.12	7	1.4	4	0.2	1	0.05	1.19	T87, T88, T89, T90	4
R24	6	2.34	5	1	3	0.15	2	0.1	0.90	T91, T92, T93	3
R25	5	1.95	9	1.8	2	0.1	1	0.05	0.98	T94, T95, T96, T97, T98, T99, T100	7

III. PROPOSED METHODOLOGY FOR REQUIREMENT BASED TEST CASE PRIORITIZATION

For proposing the technique of requirements-based test prioritization, Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) are used in the hybrid form. This section initially provides the overview of ACO and PSO individually, which will be followed by a detailed explanation of the proposed technique.

A. Ant Colony Optimization

It is a population-based search method that simulates actual ants' performance to find the shortest route [38]. This approach was proposed by [39]. The motion of the ants depends on the pheromone that is stored on some routes by other ants. ACO initiates by generating agents situated in various positions of the search space used to construct solutions. When the ants move from one position to another, it leaves the biological pheromone to mark the routes. It helps the succeeding ants to search the way their team members identify pheromones and select the probability routes. This algorithm depends on the alignment of the pheromone on paths of every node. Ants are based on the probability rule to select their solution to the problem. Ants depend on the connection within the colony of simple agents, which is known as agents. Furthermore, it is encouraged by pheromone groups. The pheromone groups work as dispersed and digitalized data in ACO, where ants are used to construct results to the problem to be resolved and adjust at the time of execution [40].

The process stages of the ACO algorithm are:

- **Initialization:** During the algorithm's start, the parameters are fixed, and all the pheromones' variables, heuristic data are initialized.
- **Build the ant solutions:** For every ant j, build a novel solution using the probabilistic rule to select the solution elements. The regulation is a function of the present solution to the sub-problem j, pheromone, and heuristic data.
- **Evaluate the solutions:** Compute the solution of every ant achieved in step 2, locate the non – dominated results, and remove the dominated results.
- **Updating the pheromone matrices:** This phase involves the updating of the pheromone matrix eliminated from newly built results. Hence, a pheromone is related to boundaries in non – dominated increased solutions.
- **Termination standards:** If the specific problem situation is met, like the number of iterations, running time, the algorithm stops, and the result of the non-dominated result is set, or go to step 2.

B. Particle Swarm Optimization

This swarm method is a nature-inspired optimization approach that pretends the communal performance and lively motion of the animals inside the flock. Every bird or animal in the search area regulates the flying behavior in accordance with the individual and other flying knowledge of the birds [41]. In particle swarm optimization, Swarm refers to many homogeneous agents who interact amongst themselves in their environment. This algorithm is based on the behavior of organisms in a group, such as depicted by bird flock, fish, or insects, to achieve an optimal solution [42-43]. In this, group members try to make a shared objective according to feedback from the other group members. Each member of the group tries to find a possible solution at any instant in time. After that candidate's suitability, the solution is communicated to other members of the swarm via signals. Other members, therefore, senses the strength of the transmitted signal, and according to the fitness function, the suitability of the candidate solution is assessed.

The populace is modified randomly with the particles' collection, and every particle demonstrates a result (solution). This approach finds an optimal number of rounds. In every round, particles are executed using fitness_value, and it results from the function known as particle fitness_value. If the fit_value of the output has better performance, then the stored location of the particle is the best value, known as (pbest) personal best. In the last iteration, the particle having the best fit_value is chosen as the global best value (gbest). It is accountable for controlling particles towards a suitable position [44]. Each particle allocates its moving velocity by vigorously consistent with its flying capabilities that depend on the g best value.

Thus, the novel position of the particle is changed in accordance with:

- Its present location

- Its present speed
- The space among its present location and Pbest
- The space among its existing location and Gbest

Hence, the speed (v) and location (l) for every updating particle by utilizing the equation below;

v_j = w.v_{j-1} + C_2 x rand ( ) x ($pbest_j$ – p.l) + C_3 x and ( ) x gbest –pl) …….. (i)

The location equation is;

p.l = p.v.l + v_j ……..(ii)

The inertia weight (w) is computed in accordance with the below equation as;

$$W = \frac{(t_{maxi} - t) - (w_{st} - w_{stop})}{t_{maxi}} + w_{stop} \quad \text{…….(iii)}$$

In equation (iii), w is inertia weight, p.v.l is the particle's location in the previous particle, pl is the present location of the particle, v_j is the present speed for j, v_{j-1} is preceding speed, C_2 and C_3 is static acceleration value, rand( ) is a random amount, pbest is best_particle, gbest is the best_ particle acquired completed whole iterations. And, t_{maxi} is the maximum value of iteration, w_{st} is start weight, inertia, w_{stop} is end inertia weight, and t is the present iteration value.

C. Ant colony and Particle swarm optimization Hybrid Technique (APHT)

In the hybrid technique for requirements-based test case prioritization, the process is having various modules and sub-modules to process the requirements and generate the prioritized lists.

The initial step of the processing is to upload the processed requirement document that has the detail of requirements and the generated test cases. The proposed approach fetches the requirements and pre-processes those entries with the help of attribute selection, priorities calculations, and mapping of the requirements with the processed cases. This step is an iterative process that provides the detailed analysis and selection of all cases gives with the requirements list. The most common formulas that used to process the priorities for mapping are:

$$CPV = CP * Weight\ (CP) \ldots (1)$$

$$ICV = IC * Weight\ (IC) \ldots (2)$$

$$CRV = CR * Weight\ (CR) \ldots (3)$$

$$FIV = FI * Weight\ (FI) \ldots (4)$$

The given priority vectors in eq. 1,2,3,4 helps to generate prioritized and un-prioritized lists in the test case prioritization process. Once the priority vector and mapping process complete, the system releases the un-prioritized list of the test cases with the APFD calculation as in eq. 5.

$$\text{APFD} = 1 - \frac{\text{TF1+TF2+.....TFm}}{mn} + \frac{1}{2n} \ldots (5)$$

The un-prioritized test case lists are processed as the population of the hybrid approach, which is designed to get the better-prioritized list of the test case generation process. Initialized population process with the ant colony optimization modules, which helps to get the selected attributes in processed form and make a list more accurate. It processes with population update the fitness values to get the final population for a hybrid form of PSO optimization process.

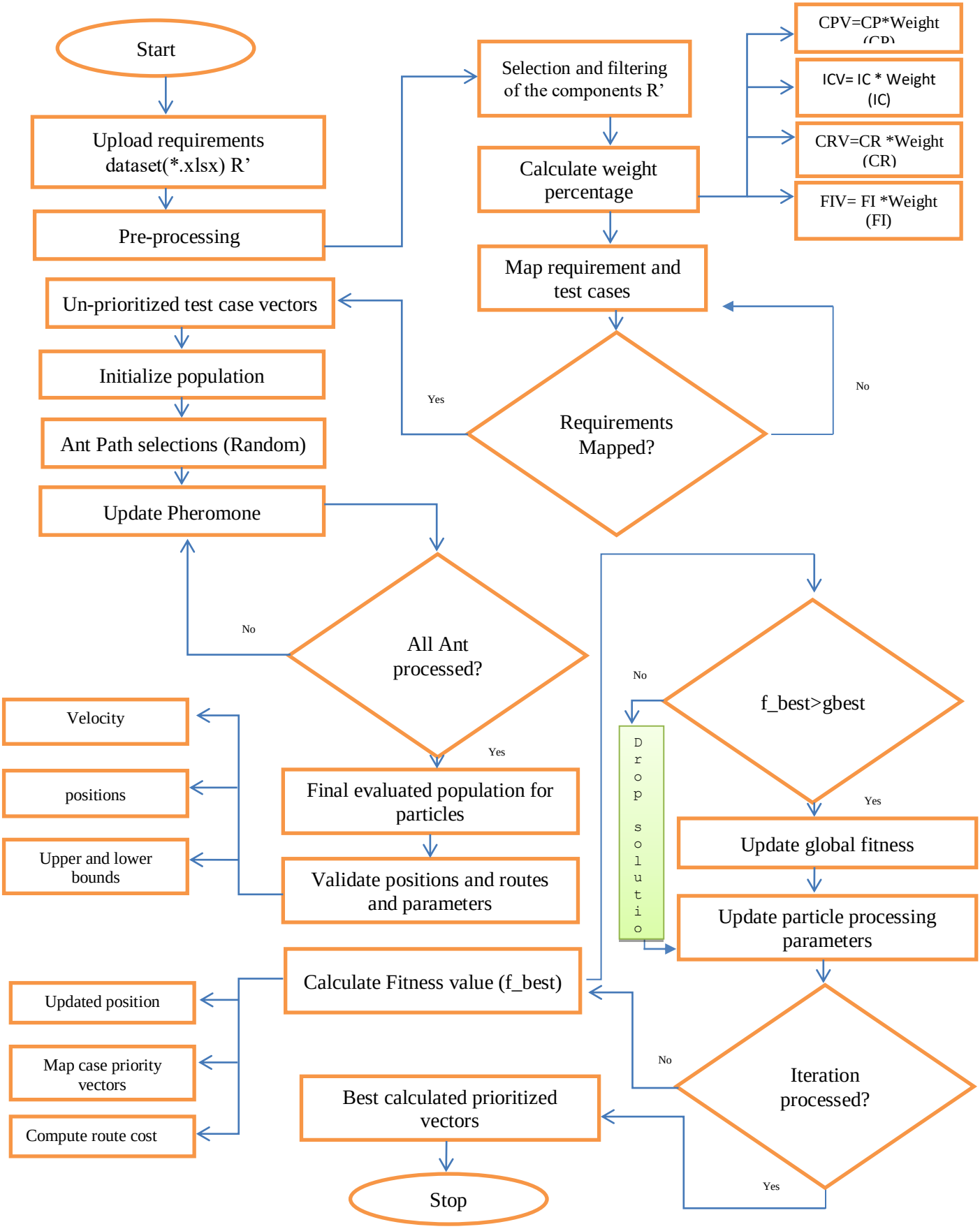
$$\text{Best\_population\_matrixPSO} = \sum_{k=0}^{n} \text{ant\_fitness(k).Cost} > \text{BestSol.Cost.}$$

The additional methods of the hybrid algorithm are used to finalize the population in terms of less error rate and better selection of the population components. The selected component becomes the population for the particle swarm optimization process. The swarm optimization process validates and processes the cases with requirements and finds the best routes toward the priority vectors. It updates the fitness values of calculated vectors and computes the best global fitness value for the priority vectors.

$$\text{Priority\_Fitness\_Vector} = \sum_{p=0}^{n} \text{particle\_route\_fitness(p).Cost} > \text{global\_fitness.Cost.}$$

The iterative process updates random positions and velocity of the particles to find the accurate fitness values as priorities of the test cases. It processes the population till it gets the better priority vector or the end of the iteration give for optimization. The calculated priority vectors visualize on the frontend screen with the mapped requirements with calculated APFD value along with computation time.

The below figure depicts the flowchart of the proposed research work.


Start
Upload requirements dataset(*.xlsx) R'
Pre-processing
Selection and filtering of the components R'
Calculate weight percentage
CPV=CP*Weight (CP)
ICV= IC * Weight (IC)
CRV=CR *Weight (CR)
FIV= FI *Weight (FI)
Map requirement and test cases
Requirements Mapped?
Yes
No
Un-prioritized test case vectors
Initialize population
Ant Path selections (Random)
Update Pheromone
All Ant processed?
No
Yes
Velocity
positions
Upper and lower bounds
Final evaluated population for particles
Validate positions and routes and parameters
Calculate Fitness value (f_best)
f_best>gbest
No
Yes
Drop solutio
Update global fitness
Update particle processing parameters
Iteration processed?
No
Yes
Updated position
Map case priority vectors
Compute route cost
Best calculated prioritized vectors
Stop

- **Pseudocode of the Proposed technique**

Pseudo code of APHT Technique

Input: Requirements, Processed test case matrix, Particles, Pheromone.
Output: Prioritized test cases.

1. Upload test case Matrix, requirements $TC= \{T_1, T_2, T_3 \ldots T_n\}$, $Req= \{R_1, R_2, R_3, \ldots R_n\}$
2. For i=1: length (Req) do, //*weight calculations of pre-processed requirement*
3. CPV=CP*Weight (CP)
4. ICV= IC * Weight (IC)
5. CRV=CR *Weight (CR)
6. FIV= FI *Weight (FI)
7. end For.
8. For map=1: length (TC)
9. MappedR=original_matrix. find (contains (Req,TC)); //*map requirements as 2D fault matrix*
10. End for.
11. Repeat step 8 for mapping all the requirements and test cases.
12. for i=1: size(inp_dat_rearrange,1) do *// Calculate un-prioritized list*
13. [~, col1] = find (MappedR (i, :) == true); // *find for matrix index*
14. Un_prioritized (i,1) = col1(1,1);
15. End for
16. Initialize population and random path for MappedR
17. Selection of paths AP
18. Update tour T for updated AP
19. Get cost of T as popCost= CostFunction(ant_a(k_a). Tour)
20. Update Pheromone.
21. Repeat step 17 till all iterations.
22. Collect best path population PsoPop= popCost.
23. Validate positions and Velocity and initial parameters.
24. Velocity = w*V (i, :) + c1(1, D). *(pbest(i,:)) + c2*rand(1,D).*(gbest - P(i,:));
25. Update position of particles
26. Calculate fitness value // *the cost of a prioritized list*
27. $\text{Priority\_Fitness\_Vector} = \sum_{p=0}^{n} \text{particle\_route\_fitness(p).Cost} > \text{global\_fitness.Cost.}$
28. find_pos= find(posit(i)== gbest.cost); // *simulate prioritized list pattern*
29. Repeat for all the test cases
30. calculate $\text{APFD} = 1 - \frac{\text{TF1+TF2+.....TFm}}{mn} + \frac{1}{2n}$
31. end

w=Inertia, V=random-intial-velocity, c1=Accelaration Coff., D=length(lb), pbest=Fitness, c2=Accelaration Coff, Global best fit=gbest, P= Random position, AP=Ant Positions

IV. EXPERIMENTAL ANALYSIS

In the experimental setup, the proposed APHT technique has been implemented in MATLAB 2018a for computing the results on a computer with 8.00 GB RAM and Intel (R) Core ™ i7-4600 U CPU @ 2.10 GHz 2.70 GHz Intel processor. Initially, in the ACO technique, results were generated upon 300 iterations with 40 ants. Then in the PSO technique, the results were generated up to 100 iterations with 10 birds. For the proposed APHT technique, the results were computed up to 10 iterations and 10 birds as to this point saturation, and an optimum solution has been attained.

- ***Performance Metric***

- Average Percentage of Faults Detected (APFD)-

$$\text{APFD} = 1 - \frac{\text{TF}_1 + \text{TF}_2 + \text{TF}_3 + \cdots \ldots \ldots \ldots \text{TF}_\text{m}}{\text{mn}} + \frac{1}{2\text{n}}$$

"Where '*T*' is the test suite to be evaluated, '*m*' depicts the number of faults in an application under test, *'n'* is total test cases in a test suite, and 'TF_j' describes the location of the first test case in Test Suit *'T'* that reveals fault *j*."

	T1	T2	T3	T4	T5
F1	*		*		
F2	*				*
F3					*
F4	*	*	*		
F5		*		*	*

For example, consider the above fault matrix of 5 test cases and 5 faults. If the APFD is calculated when test cases run in a sequential way from T1 to T5,

$$APFD = 1 - \frac{1+1+5+1+2}{5*5} + \frac{1}{2*5} \qquad APFD = 70\%$$

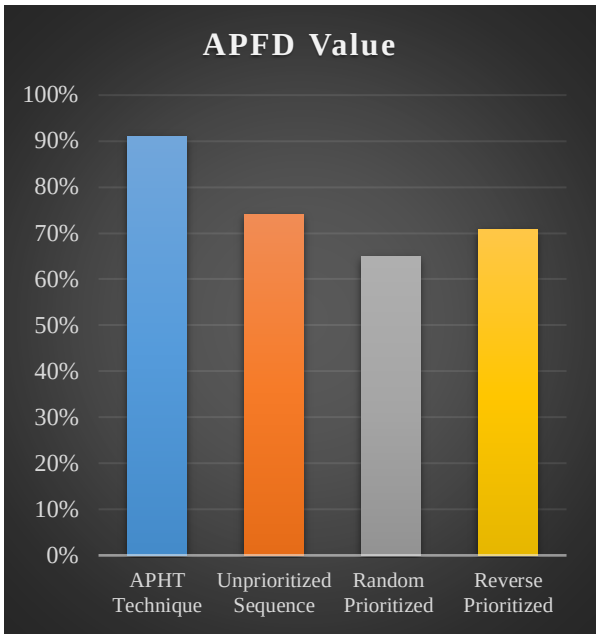
When the list is prioritized by some optimization technique and the prioritization sequence is *T5->T3->T1->T4->T2*, and APFD value is calculated, which is

$$APFD = 1 - \frac{2+1+1+2+1}{5*5} + \frac{1}{2*5} \qquad APFD = 82\%$$

The prioritized sequence of test cases generated by the proposed technique is- *T77->T32->T33->T14->T41->T74->T66->T12->T96->T83->T2->T43->T8->T58->T54->T13->T42->T70->T99>-T11->T7->T73->T26->T93->T19->T35->T78->T61->T57->T6->T51->T98->T39->T52->T76->T30->T17->T65->T34->T4->T85->T82->T68->T79->T53->T25->T15->T89->T37->T94->T92->T100->T95->T48->T45->T16>T91->T9->T81->T59->T24->T20->T40->T27->T62->T55->T60->T3->T5->T38->T28->T47->T97->T29->T56->T71->T31->T36->T67->T88->T46->T1->T44->T23->T87->T90 >T75->T49->T72->T50->T69->T10->T63->T80->T22->T86->T84->T18->T64->T21.*

When the APFD value is computed, it comes out to be 91%.

For the proposed technique, values of APFD are also calculated for unprioritized sequence, random prioritized sequence, and reverse prioritized sequence. For the unprioritized sequence, the value of APFD is 74.04%. When test cases are arranged in a random sequence, the value of APFD is 64.93%. When test cases are arranged in reverse prioritization sequence, i.e., from T100 to T1, the value of APFD comes out to be 70.83%.



Graph 4.1 APFD value comparison of the proposed technique with other prevailing techniques.

The above graph 4.1 depicts the comparison of the APFD value of the proposed technique with other existing techniques. It is evident that the proposed technique has performed well in terms of higher fault detection value.

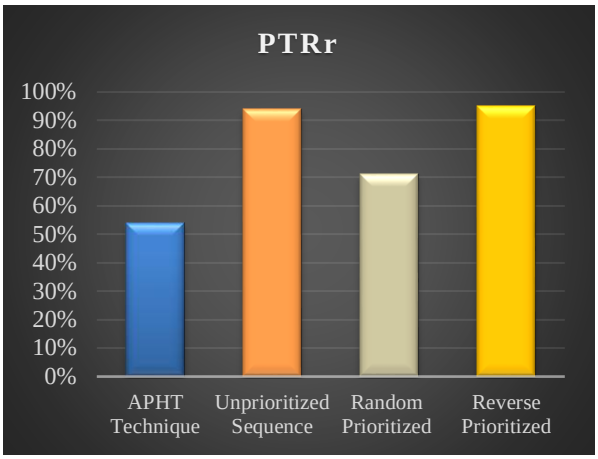
- *Percentage of Test Suite executed for all Requirements coverage (PTR_r)*

$$PTR_r = \frac{\text{No. Of Test Cases Required for Complete Coverage of Requirements}}{\text{Total Number of Test Case}} \times 100$$

PTR is a metric used for measuring the effectiveness of the proposed technique. An efficient technique will schedule the test case execution sequence in such a way that requirements are identified at the starting of the execution sequence of the test cases. So, a relatively low PTR value depicts a better technique as a smaller number of test cases are required for covering all requirements instead of executing a whole test suite.

Consider a test suite of 5 test cases T1 to T5, which captures 5 requirements in a system. For any prioritization order *T1->T2->T3->T4->T5*, if all 5 test cases are required to cover all 5 requirements, then PTR is 100. If only 2 test cases are required for covering all the 5 requirements, then the PTR value will be 40. ($PTR = \frac{2}{5} \times 100$)

When the PTR_r value is computed for the proposed APHT technique, then it comes out to be 54%. When PTR_r values are calculated for the unprioritized sequence, random sequence, and reverse sequence of the execution of test cases, then the corresponding values come out to be 94%, 71%, and 95%, respectively.



Graph 4.2 PTR_r values comparison of the proposed technique with the existing techniques.

From the above graph 4.2, it is evident that the proposed APHT technique has performed well in terms of fewer test cases required to be executed to cover all the requirements.

V. COMPARATIVE PERFORMANCE ANALYSIS

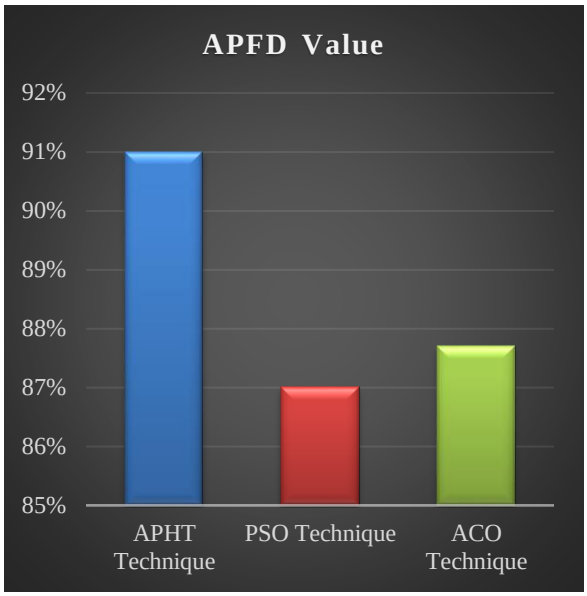
This section presents the comparative performance analysis of the APHT technique with the PSO technique and ACO technique. Initially, the APFD factor is taken for evaluation. Then PTR_r values are taken, which is followed by the comparison of the average computation time. The results obtained showed the effectiveness of the proposed technique.

For the PSO technique, the test case sequence generated is-
T54->T3->T37->T47->T59->T57->T75->T89->T42->T19->T43->T62->T61->T66->T81->T83->T82->T56->T25->T40->T7->T45->T77->T68->T51->T73->T20->T79->T65->T14->T85->T46->T48->T34->T8->T97->T53->T44->T76->T74->T80->T63->T11->T69->T50->T94->T98->T86->T90->T5->T27->T91->T26->T93->T1->T18->T28->T10->T92->T52->T100->T64->T31->T9->T33->T72->T22->T96->T49->T12->T88->T99->T30->T58->T23->T70->T36->T6->T41->T13->T60->T38->T39->T84->T71->T2->T87->T17->T78->T55->T24->T15->T29->T32->T21->T16->T4->T35->T95->T67.

When the APFD value is computed, it comes out to be 87%.

For the ACO technique, the test case sequence generated is-
T45->T50->T91->T83->T58->T4->T19->T73->T55->T13->T75->T17->T48->T18->T5->T72->T51->T16->T7->T97->T28->T92->T68->T98->T47->T41->T9->T78->T57->T31->T53->T22->T10->T87->T21->T94->T32->T69->T96->T27->T100->T46->T37->T62->T64->T93->T40->T1->T3->T99->T49->T66->T82->T34->T70->T44->T63->T2->T59->T95->T26->T52->T84->T12->T6->T30->T60->T90->T61->T54->T86->T38->T71->T80->T39->T56->T15->T85->T14->T43->T76->T42->T77->T35->T81->T65->T89->T8->T11->T29->T23->T74->T24->T33->T36->T25->T67->T20->T79->T88.

When the APFD value is computed, it comes out to be 87.71%.

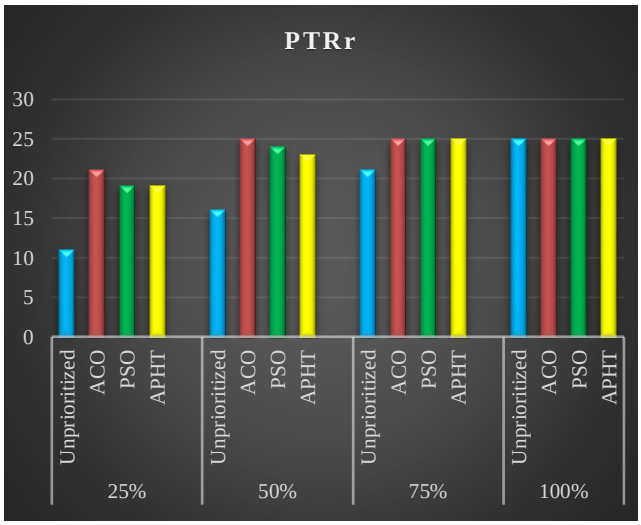


Graph 4.3 APFD value comparison of the proposed technique with prevailing techniques.

It is evident from the above graph 4.3 that the proposed APHT technique has performed well in terms of enhanced rate of fault detection when compared to the ACO and PSO-based techniques.

- *Comparison of percentage of Test Suite required to cover all the Requirements (PTR_r).*

For drawing out the comparison of the percentage of test cases required to be executed for covering all the requirements, the PTR_r value of the proposed APHT technique has been evaluated against the PTR_r values of the unprioritized test execution order, ACO, and PSO based execution orders. For this, the value comparison has been shown by taking 25%, 50%, 75%, and 100% test cases execution values. The values of the APHT technique, ACO, PSO, and unprioritized order are 54%, 43%, 52%, and 94%, respectively. It has been observed that the proposed technique has not performed so well in terms of this parameter. Still, it has shown its efficient performance in terms of enhanced fault detection value and in comparatively low execution time. The results were shown in graph 4.4.

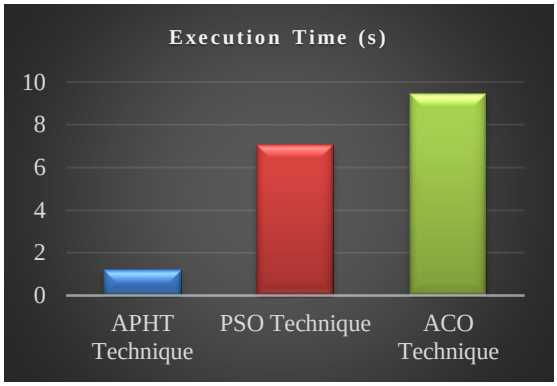


Graph 4.4 Comparison of percentage of Test Suite required to cover all the Requirements (PTR_r).

- *Comparison of the average computation time.*

Software testing is a time-consuming process and costly process, so execution time plays a key role in assessing the budget and quality of the software. If, by a certain optimization technique, the execution time is reduced without compromising the test suite quality along with increased fault detection rate, then the software testing cost and efforts are considerably reduced.

Execution time is the measure of necessary time utilized for carrying out every command for the test case. On the basis of the computation time taken by the proposed APHT technique for generating the results, a comparison is made with the execution time taken by other approaches, namely, ACO and PSO techniques for carrying the desired computation. The results are shown in graph 4.5.



Graph 4.5 Comparison of average computation time of the proposed technique with prevailing techniques.

From the above graph 4.5, it is evident that execution time has been considerably reduced even when the proposed technique is a hybrid of the existing approaches. It is due to the fact that the proposed technique requires a smaller number of iterations to reach an optimum solution. For generalizing the results, the algorithms were run five times, and then their average is taken to compute the average execution time. So, it is deduced that the proposed technique is a time-saving technique, also which has an impact on testing costs and efforts as they will also be reduced.

VI. CONCLUSION

Regression testing is one of the popular techniques of software testing, which involves re-execution of the software program after its modification to check whether no new faults have crept in the product. Test Case Prioritization (TCP) is an approach of regression testing which executes test cases in priority order. Many researchers have worked on making the TCP process effective as software is developed based on its requirements, so it is beneficial to test those requirements first, which are complex as they will be the ones where there is maximum possibility of occurrence of faults. Usage of nature-inspired algorithms is also on the rise in this field to optimize the results, thus saving time and cost involved in the testing procedure. In this research work, an efficient Ant colony and Particle swarm optimization Hybrid Technique (APHT) for requirements-based test prioritization has been proposed. To prove the effectiveness of the proposed technique, an APFD metric and average execution time were taken to measure the performance. When APFD and PTR_r values of the proposed techniques have been compared with other prioritization strategies, it has been observed that the proposed approach is quite efficient. When the APFD and average execution time results obtained were compared with the existing PSO and ACO techniques, then also the worthiness of the proposed technique is proved. In the future, the proposed technique will be compared with various existing approaches by taken other performance metrics also.

ACKNOWLEDGMENT

The first author of the paper is thankful and obliged to the coauthor for the unconditional support in proposing this technique and for giving valuable feedback from time to time to improve the quality of the research work.

REFERENCES

[1] Joshi, S. A., and Tiple, B. S. (2014). Literature Review of Model-Based Test case Prioritization. International Journal of Computer Science & Information Technologies, 5(5).

[2] Arafeen, M. J., and Do, H. (2013, March). Test case prioritization using requirements-based clustering. In 2013 IEEE sixth international conference on software testing, verification and validation (pp. 312-321). IEEE.

[3] Qu, B., Nie, C., Xu, B., & Zhang, X. (2007, July). Test case prioritization for black-box testing. In *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)* (Vol. 1, pp. 465-474). Ieee.

[4] Dahiya, O., & Solanki, K. (2018). A systematic literature study of regression test case prioritization approaches, International Journal of Engineering & Technology, Vol. 7, No. 4, pp.2184-2191.

[5] Dahiya, O., Solanki, K., Dalal, S., and Dhankhar, A. (2020). Regression Testing: Analysis of its Techniques for Test Effectiveness, International Journal of advanced trends in computer science and engineering, Vol. 9, No. 1, pp. 737-744.

[6] Dahiya, O., & Solanki, K. (2021). An Efficient Requirement-based Test Case Prioritization Technique using Optimized TFC-SVM Approach, International Journal of Engineering Trends and Technology, 69 (1).

[7] Jarzabek, S., Liszewski, K., & Boldak, C. (2020). Inferring hints for defect fixing order from requirements-to-test-case mappings. In *Integrating Research and Practice in Software Engineering* (pp. 43-51). Springer, Cham.

[8] Nayak, S., Kumar, C., Tripathi, S., & Majumdar, N. (2020). An improved approach to enhance the test case prioritization efficiency. In *Proceedings of ICETIT 2019* (pp. 1119-1128). Springer, Cham.

[9] Yaseen, M., Ibrahim, N., & Mustapha, A. (2019). Requirements prioritization and using iteration model for successful implementation of requirements. *Int. J. Adv. Comput. Sci. Appl*, *10*(1), 121-127.

[10] Dhiman, R., & Chopra, V. (2019, March). A novel approach for test case prioritization using the ACO algorithm. In *2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT)* (pp. 292-295). IEEE.

[11] Alzaqebah, A., Masadeh, R., & Hudaib, A. (2018, April). Whale optimization algorithm for requirements prioritization. In *2018 9th International Conference on Information and Communication Systems (ICICS)* (pp. 84-89). IEEE.

[12] Masadeh, R., Alzaqebah, A., Hudaib, A., & Rahman, A. A. (2018). Grey Wolf algorithm for requirements prioritization. *Modern Applied Science*, *12*(2), 54.

[13] KHATIBSYARBINI, M., ISA, M. A., & ABANG JAWAWI, D. N. (2017). A HYBRID WEIGHT-BASED AND STRING DISTANCES USING PARTICLE SWARM OPTIMIZATION FOR PRIORITIZING TEST CASES. *Journal of Theoretical & Applied Information Technology*, *95*(12).

[14] Ashraf, E., Mahmood, K., Ahmed, T., & Ahmed, S. (2017). Value-based PSO test case prioritization algorithm. *International Journal of Advanced Computer Science and Applications*, *8*(1), 389-394.

[15] Kumar, S., & Ranjan, P. (2017). ACO-based test case prioritization for fault detection in the maintenance phase. *International Journal of Applied Engineering Research*, *12*(16), 5578-5586.

[16] Ansari, A., Khan, A., Khan, A., & Mukadam, K. (2016). Optimized regression test using test case prioritization. *Procedia Computer Science*, *79*, 152-160.

[17] Srikanth, H., Hettiarachchi, C., & Do, H. (2016). Requirements-based test prioritization using risk factors: An industrial study. *Information and Software Technology*, *69*, 71-83.

[18] Gao, D., Guo, X., & Zhao, L. (2015, September). Test case prioritization for regression testing based on ant colony optimization. In *2015 6th IEEE international conference on software engineering and service science (ICSESS)* (pp. 275-279). IEEE.

[19] Tyagi, M., & Malhotra, S. (2014, July). Test case prioritization using multi-objective particle swarm optimizer. In *2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014)* (pp. 390-395). IEEE.

[20] Muthusamy, T., & Seetharaman, K. (2014). A new effective test case prioritization for regression testing based on the prioritization algorithm. *Int. J. Appl. Inf. Syst. (IJAIS)*, *6*(7), 21-26.

[21] Suri, B., & Singhal, S. (2011). Analyzing test case selection & prioritization using ACO. *ACM SIGSOFT Software Engineering Notes*, *36*(6), 1-5.

[22] Krishnamoorthi, R., & Mary, S. S. A. (2009). Factor-oriented requirement coverage-based system test case prioritization of new and regression test cases. Information and Software Technology, 51(4), 799-808.

[23] Srikanth, H., & Williams, L. (2005). On the economics of requirements-based test case prioritization. *ACM SIGSOFT Software Engineering Notes*, *30*(4), 1-3.

[24] Hujainah, F., Bakar, R. B. A., Abdulgabber, M. A., & Zamli, K. Z. (2018). Software requirements prioritisation: a systematic literature review on significance, stakeholders, techniques, and challenges. *IEEE Access*, *6*, 71497-71523.

[25] Ambreen, T., Ikram, N., Usman, M., & Niazi, M. (2018). Empirical research in requirements engineering: trends and opportunities. *Requirements Engineering*, *23*(1), 63-95.

[26] Srikanth, H., Williams, L., & Osborne, J. (2005, November). System test case prioritization of new and regression test cases. In *2005 International Symposium on Empirical Software Engineering, 2005.* (pp. 10-pp). IEEE.

[27] Srikanth, H., Banerjee, S., Williams, L., & Osborne, J. (2014). Towards the prioritization of system test cases. *Software Testing, Verification, and Reliability*, *24*(4), 320-337.

[28] Ma, T., Zeng, H., & Wang, X. (2016, May). Test case prioritization based on requirement correlations. In *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD)* (pp. 419-424). IEEE.

[29] Arafeen, M. J., & Do, H. (2013, March). Test case prioritization using requirements-based clustering. In *2013 IEEE sixth international conference on software testing, verification and validation* (pp. 312-321). IEEE.

[30] Uusitalo, E. J., Komssi, M., Kauppinen, M., & Davis, A. M. (2008, September). Linking requirements and testing in practice. In *2008 16th IEEE International Requirements Engineering Conference* (pp. 265-270). IEEE.

[31] Zhang, X., Nie, C., Xu, B., & Qu, B. (2007, October). Test case prioritization based on varying testing requirement priorities and test case costs. In *Seventh International Conference on Quality Software (QSIC 2007)* (pp. 15-24). IEEE.

[32] Kalyani, R., Mounika, P. S., Naveen, R., Maridu, G., & Ramya, P. (2018). Test Case Prioritization Using Requirements Clustering. *International Journal of Applied Engineering Research*, *13*(15), 11776-11780.

[33] Salem, Y. I., & Hassan, R. (2010, December). Requirement-based test case generation and prioritization. In *2010 International Computer Engineering Conference (ICENCO)* (pp. 152-157). IEEE.

[34] Kavitha, R. V. R. K., Kavitha, V. R., & Kumar, N. S. (2010, October). Requirement-based test case prioritization. In *2010 International Conference on Communication Control and Computing Technologies* (pp. 826-829). IEEE.

[35] Salehie, M., Li, S., Tahvildari, L., Dara, R., Li, S., & Moore, M. (2011, March). Prioritizing requirements-based regression test cases: A goal-driven practice. In *2011 15th European Conference on Software Maintenance and Reengineering* (pp. 329-332). IEEE.

[36] Vescan, A., Şerban, C., Chisăliţă-Cretu, C., & Dioşan, L. (2017, September). Requirement dependencies-based formal approach for test case prioritization in regression testing. In *2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)* (pp. 181-188). IEEE.

[37] Reddy, D. V., & Reddy, A. R. M. (2016). An approach for fault detection in software testing through optimized test case prioritization. *International Journal of Applied Engineering Research*, *11*(1), 57-63.

[38] Yun, H. Y., Jeong, S. J., & Kim, K. S. (2013, September).

Advanced harmony search with ant colony optimization for solving the traveling salesman problem—Journal of Applied Mathematics, 2013.

[39] Dorigo, M. (1992). Optimization, learning, and natural algorithms. Ph.D. Thesis, Politecnico di Milano.

[40] Akhtar, A. (2019). Evolution of ant colony optimization algorithm–a brief literature review. In arXiv: 1908.08007.

[41] AbuNaser, A., Doush, I. A., Mansour, N., & Alshattnawi, S. (2015, August). Underwater image enhancement using particle swarm optimization. Journal of Intelligent Systems, 24(1), 99-115.

[42] Shi, Y., & Eberhart, R. C. (1998, March). Parameter selection in particle swarm optimization. In International conference on evolutionary programming (pp. 591-600). Springer, Berlin, Heidelberg.

[43] Shi, Y., & Eberhart, R. C. (1999, July). An empirical study of particle swarm optimization. In Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406) (Vol. 3, pp. 1945-1950). IEEE.

[44] Garg, S., Patra, K., & Pal, S. K. (2014, June). Particle swarm optimization of a neural network model in a machining process. Sadhana, 39(3), 533-548.

[45] Dahiya, O., & Solanki, K. (2021). Prevailing Standards in Requirement-Based Test Case Prioritization: An Overview. *ICT Analysis and Applications*, 467-474.

[46] Dahiya, O., & Solanki, K. (2019). Comprehensive cognizance of Regression Test Case Prioritization Techniques. International journal of emerging trends in engineering research, 7(11), 638-646.