mathematics

MDPI

Article

Multi-Objective Fault-Coverage Based Regression Test Selection and Prioritization Using Enhanced ACO_TCSP

Shweta Singhal [1], Nishtha Jatana [2,*], Kavita Sheoran [2], Geetika Dhand [2], Shaily Malik [2], Reena Gupta [3], Bharti Suri [3], Mudligiriyappa Niranjanamurthy [4], Sachi Nandan Mohanty [5] and Nihar Ranjan Pradhan [5,*]

[1] Department of Computer Science and Information Technology, Indira Gandhi Delhi Technical University for Women, New Delhi 110006, India; miss.shweta.singhal@gmail.com
[2] Department of Computer Science and Engineering, Maharaja Surajmal Institute of Technology, New Delhi 110058, India
[3] University School of Information & Communication Technology, Guru Gobind Singh Indraprastha University, New Delhi 110078, India
[4] Department of AI and ML, BMS Institute of Technology and Management, Bengaluru 560064, India
[5] School of Computer Science & Engineering, VIT-AP University, Amaravati 522237, India; sachinandan09@gmail.com
* Correspondence: nishtha.jatana@gmail.com (N.J.); nihar.pradhan@vitap.ac.in (N.R.P.)

Abstract: Regression testing of the software during its maintenance phase, requires test case prioritization and selection due to the dearth of the allotted time. The resources and the time in this phase are very limited, thus testers tend to use regression testing methods such as test case prioritization and selection. The current study evaluates the effectiveness of testing with two major goals: (1) Least running time and (2) Maximum fault coverage possible. Ant Colony Optimization (ACO) is a well-known soft computing technique that draws its inspiration from nature and has been widely researched, implemented, analyzed, and validated for regression test prioritization and selection. Many versions of ACO approaches have been prolifically applied to find solutions to many non-polynomial time-solvable problems. Hence, an attempt has been made to enhance the performance of the existing ACO_TCSP algorithm without affecting its time complexity. There have been efforts to enhance the exploration space of various paths in each iteration and with elite exploitation, reducing the total number of iterations required to converge to an optimal path. Counterbalancing enhanced exploration with intelligent exploitation implies that the run time is not adversely affected, the same has also been empirically validated. The enhanced algorithm has been compared with the existing ACO algorithm and with the traditional approaches. The approach has also been validated on four benchmark programs to empirically evaluate the proposed Enhanced ACO_TCSP algorithm. The experiment revealed the increased cost-effectiveness and correctness of the algorithm. The same has also been validated using the statistical test (independent *t*-test). The results obtained by evaluating the proposed approach against other reference techniques using Average Percentage of Faults Detected (APFD) metrics indicate a near-optimal solution. The multiple objectives of the highest fault coverage and least running time were fruitfully attained using the Enhanced ACO_TCSP approach without compromising the complexity of the algorithm.

Keywords: ant colony optimization; regression testing; test suite prioritization; metaheuristic technique; nature-inspired technique

MSC: 68-04

Citation: Singhal, S.; Jatana, N.; Sheoran, K.; Dhand, G.; Malik, S.; Gupta, R.; Suri, B.; Niranjanamurthy, M.; Mohanty, S.N.; Ranjan Pradhan, N. Multi-Objective Fault-Coverage Based Regression Test Selection and Prioritization Using Enhanced ACO_TCSP. *Mathematics* **2023**, *11*, 2983. https://doi.org/10.3390/math11132983

Academic Editor: Ioannis G. Tsoulos

Received: 8 May 2023
Revised: 26 June 2023
Accepted: 27 June 2023
Published: 4 July 2023

1. Introduction

Software has unprecedentedly altered the lifestyles of people in current and forthcoming times. In order to prevent software from getting superseded, lots of effort is required for its maintenance, thus incurring a hefty amount of money. Errors in software can be

functional or non-functional. Functional errors are related to the operations and actions of software, whereas non-functional errors are related to customer expectations and performance requirements. Function testing includes unit testing, system testing, and acceptance testing, whereas non-function testing deals with evaluating the parameters such as security, reliability, usability, and scalability. Software testing needed to ensure the proper functioning of the software after updates are referred to as Regression Testing [1]. Researchers have been rigorously working toward the development and validation of various regression testing techniques. Regression Test Prioritization and Regression Test Selection are the key regression testing techniques focused on in this paper. Regression test selection and prioritization activities, when carried out in an endeavor to obtain promising results in the minimum possible time, are an NP—complete combinatorial optimization problem [2]. The taxonomy of Regression Test Selection and Prioritization techniques is shown in Figure 1.

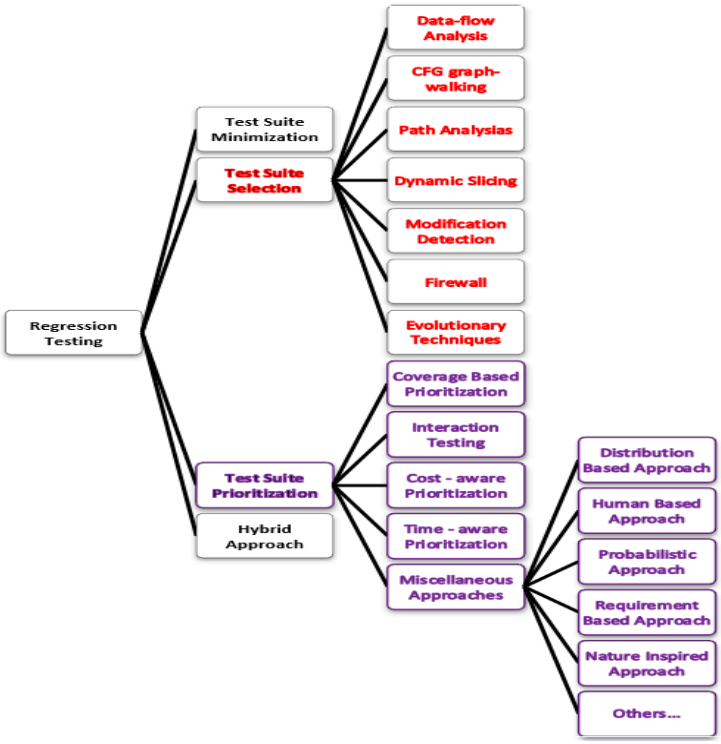

Figure 1. Taxonomy of Regression Test Selection and Prioritization.

Test Case Prioritization (TCP) [2] helps in ordering test cases based on certain criteria such as maximization of fault coverage in order to reduce the testing cost and speed up the delivery of the modified product. Regression test prioritization techniques try to reorder a regression test suite based on decreasing priority. The priority of test cases is established using some predefined testing criterion. The test cases having higher priority are preferred over lower priority ones in the process of regression testing [3]. Prioritization can be achieved on the basis of one or more objectives such as faults, code coverage, cost of execution, etc. Code coverage is a metric that measures the degree of coverage of the program code by a particular test suite [3]. The improved performance of the prioritized test suite in terms of reduced run time is a forever goal in this ever-time-constrained world. Furthermore, intelligent prioritization would mean that more and more faults are exposed as early as possible. Thus, the multi-objective goal is to deliver a regression test suite with the capacity to reveal maximum faults in the least possible time. The prioritization problem can thereby be stated as follows:

"Given: A test suite, *T*, the set of permutations of *T*, *PT*, and a function from *PT* to real numbers, f: PT->R. To find T' € PT such that (¥T")(T" € PT)(T '' !=T')[f (T') $\geq$ f (T")]."

Test suite minimization or reduction seeks to minimize the number of test cases in it by removing the redundant ones. These concepts of reduction and minimization are interchangeable as all the reduction techniques may be utilized in producing a provisional subset of the test suite, whereas minimization techniques permanently remove the test cases. More formally, the test suite minimization problem is stated as follows [4]:

"To find a representative set of test cases from the original test suite that satisfies all the requirements considered 'adequate' for testing the program".

Test Case Selection (TCS), resembles test suite minimization; as both of them intend to select few and effective test cases from the original test suite. The significant difference is whether the selection criteria focus on modifications in the SUT or not. Test suite minimization is generally built on the basis of metrics such as coverage measured from a sole version of the PUT [5]. On the other hand, in TCS, test cases are picked up based on the relevance of their execution to the variations between the original and the modified versions of the SUT. While TCS techniques also aim to reduce the number of test cases, the majority of these techniques focus on modified parts of the software under test. This means that the emphasis is to identify the modified sections of the software under test. This requires statistical analysis of the functional view of the software under test.

ACO_TCSP algorithm maps the real-world ants to digital ants while solving the regression test selection and prioritization problem. This approach has been proposed and validated and found to be very promising in earlier studies. Although, the major drawback, as found in other ACO applications, was ACO_TCSP falling into the local optima due to over-exploitation and less exploration of the solution space. The current study tries to solve this problem by enhancing the search space (increasing the number of ants) and making the exploitation more elite or intelligent. This work undertakes the following multi-objectives to be achieved:

(1) Highest fault coverage achieved i.e., maximum faults should be caught by the test suite, and
(2) The least possible test case execution time.

Hence, Enhanced ACO_TCSP has been proposed to achieve these two objectives, and the technique has been validated on four benchmark programs.

ACO is an established and well-tested optimization approach. There are a huge number of more recent optimization algorithms such as butterfly optimization algorithm (BOA), water optimization algorithms (WOA), Trees social relations optimization algorithm (TSR), red deer optimization algorithm (RDO), and many more. These have not been chosen for the current research as it tries to improve an already existing, well-established test case selection and prioritization algorithm that has already provided highly motivating results on prioritization.

The remaining paper is structured as follows: Section 2 presents the related work. Section 3 details the concepts of ACO and the limitations of the existing technique. Section 4 elucidates the proposed enhancements. Section 5 presents the implementation details of the enhanced technique by giving the algorithmic steps. Section 6 presents the experimental design. Section 7 presents the analysis of the results obtained. Sections 8 and 9 present the discussion and conclusion respectively.

2. Related Work

The way nature and its living beings co-exist in harmony is astonishing. The last two decades have witnessed extensive research in the area of developing and applying nature-inspired techniques for solving various combinatorial optimization problems [6]. Nature-inspired techniques such as ACO (Ant Colony Optimization) [7–9], GA (Genetic Algorithms), BCO (Bee Colony Optimization) [10,11], PSO (Particle Swarm Optimization) [12], NSGA -II (Nondominated Sorting Genetic Algorithm II) [13], Bat-inspired algorithm [14], flower pollination algorithm [15], cuckoo search algorithm [16,17], Cuscuta search [18], CSA [19], and hybrid approaches [20,21] (combining two or more different approaches) have already been applied to solve the regression test selection and prioritization problem.

There are other well-known regression testing techniques that deploy techniques such as fuzzy expert systems [22] and online feedback information [23].

All the ants have eaten your sweets? Powerful nature inspires us to build some artificial ants and achieve optimization goals. Inspired by the intelligent behavior of ants in food foraging, Dorigo gave a new soft computing approach and named it Ant Colony Optimization (ACO) [24]. This metaheuristic has already been applied to solve several non-polynomial time-solvable problems [25]. As given by Singh et al. [7], ACO can be applied to time-constrained test suite selection and prioritization. ACO variants have been applied rigorously by various researchers working in the area. Some of them include: epistasis based ACO [26], using ACO for prioritizing test cases with secure features [27] time-constraint-based ACO [28], history-based prioritization using ACO [29], prioritizing test cases based on test factors using ACO [30]. The widespread usage of ACO for test case selection and prioritization highlights the fact that ACO is a very prolific technique in the area. The improvements suggested in ACO [7,31,32] further ascertain that there are limitations associated with the technique. A survey of ACO in software testing [33] showed various limitations of ACO as identified by various researchers. This provided the key motivation for the authors for the present work. Thus, an improved version of the ACO technique for Test Case Selection and Prioritization has been proposed and implemented. The enhancement proposed has been validated in terms of: (1) no complexity overhead as compared to the previous approach, (2) improvement in correctness over the previous approach tested on four benchmark programs, and (3) APFD analysis against the traditional regression testing approaches [8]. The three enhancements proposed in the current manuscript have not been proposed for modification together in any of the above-mentioned related texts. Henceforth, the presented approach provides a new ACO approach having combined enhancements.

3. Ant Colony Optimization

This section gives a conceptual view of ACO and explains the limitations of the existing technique, thereby leading to the proposed enhancements in the subsequent section.

3.1. Concept

ACO is a popular metaheuristic method that provides a solution to many combinatorial optimization problems [34–36]. It is inspired by the concept of stigmergy, which is an indirect means of communicating with fellow members using the surrounding environment. This complex yet effective mode of communication is utilized by ants for food search. This working of ACO is shown in Figure 2. There are three possible paths (Path 1, Path 2, Path 3) from the ant nest to the food source. Ants cannot see, yet they magnificently coordinate within their colony for food search, with the use of a chemical substance known as pheromone. While foraging for food and fetching it back to their nests, ants keep laying the pheromone trail on the traversed path. The other ants thereby smell the maximum amount of pheromone and start following that path. The interesting fact to notice here is, that the ant on the minimum length path returns fastest, thus laying an additional amount of pheromone on their forward and return journeys. This enhances the probability of fellow ants taking this path. The ants taking this corresponding path will further drop more pheromones on that path. Therefore, attracting more fellow ants to take up this path. This process continues, and eventually, the whole colony of ants would converge to the minimum length path (here Path 1 in Figure 2).

The optimization algorithm based on the artificial behavior of ants works iteratively by the generation of an initial population. It then repeatedly endeavors to construct candidate solutions by exploration and exploitation of the search space. The exploration is guided by heuristic information and the experience gathered by the ants in the preceding iterations (well-known as 'pheromone trails') via a shared pool of memory. Using the concepts of using ACO, effective solutions have been achieved to many hard combinatorial problems [8]. The heuristic function (that marks the quality of the candidate solution

during the construction phase), and the pheromone values (signifying the information collected over different iterations) are the two major components of the working of ACO as a solution to combinatorial problems.

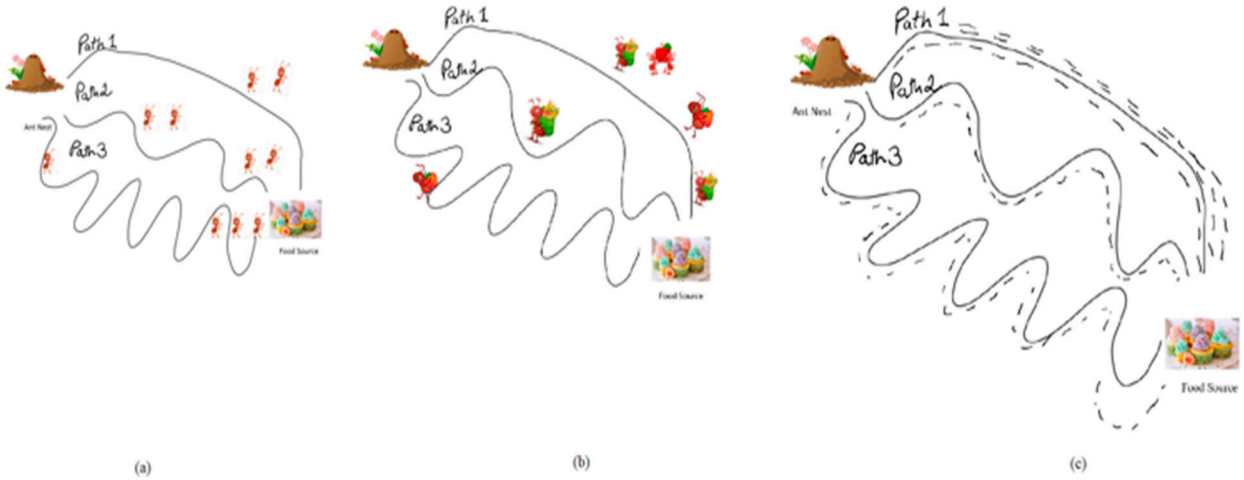

Figure 2. Pheromone on various paths. (**a**) Before ants begin searching, (**b**) while ants are searching for food, (**c**) finally, when all the ants follow the shortest path to the food source—the shortest path from the nest to the food source.

3.2. Limitations of ACO

ACO has been magnificently used to yield effective solutions to various optimization problems, yet its convergence has not been proven. This is also justified by the ACO approaches being approximation approaches. The general limitations include: slow convergence speed [33] and the problem of subsiding in the local optimal solution [33]. The reason behind it may be that the ACO updates the pheromone on the basis of the present promising path, and subsequently, after certain iterations, the amount of pheromone on this path rises substantially, while the pheromone for the probable worth path is frail. All the ants are inclined to this promising path and it becomes extremely tough to skip this path. Hence, the chances of marking the optimal solution being the locally optimal one, are increased.

Practically it is not viable to re-run the whole original test suite for building confidence in the modified software during regression testing. Thus, it becomes crucial to use selection and prioritization for the tests. Current research undertakes the highest fault coverage achieved in the least possible test case execution time as the objectives. ACO for test selection and prioritization was proposed in [7], implemented as ACO_TCSP in [8], and analyzed in [31,37]. From the analysis conducted in [31,37], some limitations in the ACO_TCSP technique were discovered:

(1) Lower Termination condition (*TC*) values lead the ACO_TSCP to fall into the local optima problem by restricting exploration of newer paths.
(2) Around 5% of the time, the ACO_TCSP could explore the optimum path, but due to the lack of pheromone that could be deposited, it is not the best path, i.e., the algorithm ended by meeting set *TC* criteria.
(3) Lower *TC* value directly implies a lower number of iterations. Hence, changes are made in the algorithm to reach the set *TC* value as delayed as possible.

4. Proposed Enhancements

The major problem was falling into local optima due to a lack of explored search space and over-exploitation at an early stage. To solve this problem, it has been tried to expand the explored search space in each iteration while maintaining elite or intelligent exploitation to confirm as early convergence as possible to the optimal solution. Balancing increased

exploration with the intelligence introduced into the manner of exploitation should ensure that the run times are not adversely affected, while the solutions found are improved to avoid local optima problems. The proposed enhancements have been elaborated as follows:

4.1. Expand the Searched Space

In order to enhance the search space, we propose an increment in the number of exploring ants, which was earlier equal to the number of test cases in previous implementations. The enhanced algorithm asks the user to enter the enhancement factor from which the number of digital ants to be sent exploring paths per test case would be computed. Consequently, the overall number of ants grows multiple times the input entered by the user. If '*EF*' is the Enhancement Factor input by the user, and '$|TS|$' represents the size of the original test suite, the total amount of ants (after enhancement) per program run can be considered to be:

$$\text{No. of Digital Ants} = (\text{EF}*|\text{TS}|) \tag{1}$$

The algorithm should now be able to overcome the problematic premature convergence of ants to a local optimal result. As now the ants will travel through additional paths in the initial iterations also, the number of paths to select the most promising one from for the current iteration correspondingly is multiplied by EF. Given as:

$$\text{Number of paths discovered in 1 iteration} = \text{EF}*|\text{TS}| \tag{2}$$

Enhancing the number of digital ants could directly affect the run time of our algorithm, but since the increase is by a constant enhancement factor (*EF*), it does not increase the complexity. Moreover, expanding the search space should lead to earlier convergence toward finding the optimal path. The same was also confirmed by the experimental results achieved (as shown in the next sections). So, the small constant time increase caused due to *EF* is counterbalanced by convergence at earlier iterations of the algorithm.

4.2. Elitism

In total, 5% of the earlier sample runs could find the optimum path if the overall best path computation neglects the maximum pheromone over the optimum best path found in any iteration. The calculation of the Global Best Path (*GBP*) was updated as:

$$GBP = \min\{\text{max pheromone path, Best\_Path in any iteration}\} \tag{3}$$

This modification alone led to improved accuracy of ACO_TCSP by 4.1%. The elitist strategy for choosing the *GBP* over following the normal ACO approach is thus formulated. This step introduces eliteness in the process of exploiting already explored paths. This intelligent exploitation would lead to early convergence to the optimal solution. while at the same time, an increase in exploration would ensure the algorithm from falling into the local optima problems.

4.3. Modifying Total Time Calculation

The earlier algorithm took the MAX execution time (*MET*) of the local paths discovered in an iteration. The stopping criteria (*TC*) were compared with the *MET* added in every iteration. The modification to resolve this issue is recalculating *MET* using the execution time corresponding to the Local Best Path (*LBP*) discovered in every iteration.

$$\text{MET} = \text{MET} + \text{ExecTime of LBP} \tag{4}$$

The modification leads to an increased number of iterations for which the ACO_TCSP would run before reaching the set *TC* value. It leads to increased exploration and increased intelligent exploitation of paths and hence more chances of discovering the optimal path. It is of utmost importance to make sure that modifying MET or enhancing the search space does not adversely affect the execution time of the improved algorithm. Hence, the same

was taken care of by ensuring that the algorithm is time bounded by the user-entered *TC* value and cannot run after MET has reached *TC*. Hence, the Enhanced ACO_TCSP algorithm ensures better results without an increase in the run time of the process.

5. Implementing the Enhancements

Given '*TS*' (Test Suite) with $|TS|$ test cases in it, the selection and prioritization problem is specified as follows:

Find the subset '*S*' of test suite '*TS*', so as to have 'm' test cases ($m < |TS|$, *S TS*). The subset '*S*' is selected with the aim of maximum fault coverage and prioritized on the basis of the minimum execution time taken to run the entire subset.

5.1. Problem Representation and Execution Steps

A mapping for the selection/prioritization problem as an undirected graph *G* (*N*, *E*) having *N* and *E* as the set of nodes and edges correspondingly was performed. Test cases were mapped as the nodes of the graph. Here, 'w_i' represents the cost of 'i^{th} 'edge in the graph. This cost of edges maps to the trail of pheromone deposited on the edges $e_i \in E$. Pheromone deposition was correspondingly mapped to the multiple objectives of (1) fault coverage 'f_i' on the current path achieved within (2) time constraint, 'Z_i'. Originally, the pheromone or cost on all the edges is nil.

- MAX represents the overall Time Constraint.
- Z stands for an intermediary variable used for TC calculations.
- {a1 × EF, a2 × EF, . . . , an × EF} represents a set of artificial ants, where 'EF' is the enhancement factor.
- S1 to S (n × EF) be the sets in which (n × EF) ants maintain the record of the covered test cases.

The execution steps of Enhanced ACO_TCSP are detailed below:

Step 1: Initialization—Generating N × EF ants to be sent for exploring the solution space. All other parameters are initialized accordingly.

Step 2: Exploration—Once generated, the ants begin searching in all random directions initially. As they move from one test case to another, their paths and killed faults are constantly updated on the way.

Step 3: Pheromone deposition—Once a probable solution has been found by each ant in the current iteration, pheromones are deposited on the most effective path (causing minimum time to run the test cases).

Step 4: Iterating—Steps 1 to 2 are repeated over and over till the stopping *TC* is achieved. Now there are many iterations and the most effective path from every iteration has been found.

Step 5: Finding *GBP*—Out of all the paths discovered during all the repetitions, *GBP* is the one with the least running time. This represents the selected and prioritized set of test cases as the final answer.

These steps are depicted in Figure 3.

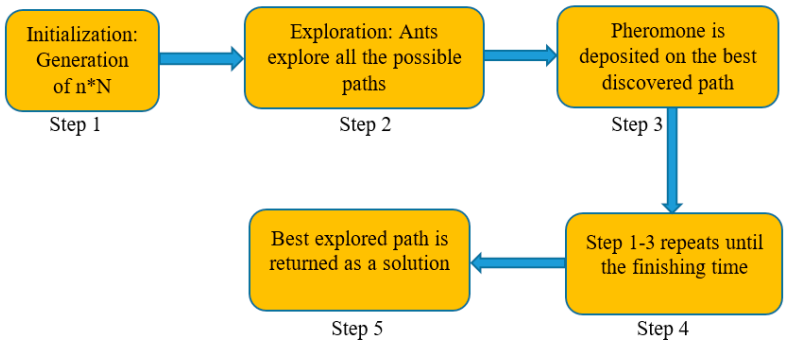

Figure 3. Working of Enhanced ACO_TCSP for Test Case Selection and Prioritization.

5.2. Modified Algorithm and its Complexity

Aiming to ascertain that the cost of applying the technique is no more than performing the complete regression testing, complexity has to be calculated. The complexities of all the sequential stages are added to compute the total complexity of the new proposed Enhanced ACO_TCSP (Algorithm 1) (Enhancements from [7,8]).

Algorithm 1: Enhanced ACO_TCSP

Stage-1	
1.Initialization	**Statement-wise complexity**
Set Wi = 0	 1
Set TC = MAX (User Defined)	
. 1	
Set EF = Enhancement Factor (User defined)	 1
Set Zx = 0 for all x = 1 to (N × EF)	. . . . . N × EF
Create N × EF artificial ants {a1, a2, ... a(N*EF)}	. . . . . N × EF
... S1, S2, ... S(N × EF) = NULL	. . . . . N × EF
Stage-2	
2. Do // LOOP 3—outermost	
For x = 1 to (N × EF) // LOOP 2	. . . . . Runs N × EF times
Sx = Sx + { t_{xEF} } // Initial test case for ant a_x is	 1
TmpT = t_{xEF} // the starting vertex for ant a_x on the graph	 1
Do // Loop 1—Innermost loop	
tt = Call_select_test_case (a_x, tmpT)	 6
tUpdate Sx = Sx + {t} .	 1
$tZ_x = Z_x + t_{exec\_time}$	 1
tmpT = t	 1
While (total faults are covered) // Max N times (no of all Test Cases)	
// Loop 1 (Innermost loop)—ENDS	
EndFor // Loop 2 ENDS	
minTime = min {Zx}	. . . . . N * EF
currTime = CuurTime + minTime	 1
Zx = 0, for all x =0 to N × EF	. . . . . N × EF
Update pheromone on all egdes of bestPath with minTime	. . . . . n − 1
Evaporate k% pheromonee from each edge	. . . . . n − 1
Last_Path = Sx for k of the last iteration of Loop2	 1
Sx = null for all x	. . . . . EF
While (TC ≥ currTime) // End of Loop 3	
// MODULE select_test_case is presented below (As taken from [7,8])	
select_test_case (x, next_node)	
{	
If(max_pheromene edge out of all the edges from next_node to a node 'k' (W[nxtnode][k]) does not exist in S[i][1 . . . N])	 2
Then	
Return 'k'	 1
Else	
Select a random edge [next_node,k], from next_node to node 'k' not existing in P[i][1 . . . N], from all edges having next max (W[nxtnode][h])	 3
Return 'h	 1
}	

Stage 1 represents the initialization stage. The cost of the edges is initialized to be nil. *EF* is read from the user and corresponding digital ants are created. A set of paths and other variables are also initialized as above. The total complexity of Stage 1, say S1, can be

computed by adding the individual complexity of all the sequential statements from above as follows:

$$S1 = 1 + 1 + 1 + (n \times EF) + (n \times EF) \quad (5)$$

$$S1 = 3 + 3 \times (n \times EF) \quad (6)$$

The exploration of various paths keeping in mind the multiple objectives of fault coverage and budget time happens in this stage. Over the various repetitions of the do-while loop, pheromone updates occur, causing the next digital ants to exploit the already discovered potential paths. The complexity of Stage 2, say S2, can be calculated stepwise from the above individual statement complexities as shown below:

$$\text{Complexity of Loop 1} = n \times (6 + 1 + 1 + 1) = 10 \times n \quad (7)$$

$$\text{Complexity of Loop 2} = n \times EF \times (2 + (10 \times n)) = 2 \times EF \times n + 10 \times EF \times n2 \quad (8)$$

$$\text{Complexity of Loop 3} = \text{Const} \times (3 \times EF + 2 \times (n - 1) + 2 + (2 \times EF \times n + 10 \times EF \times n2)) \quad (9)$$

(Taking 'C' to be some positive constant)

$$S2 = C \times (10 \times EF \times n2 + 2 \times EF \times n + 2 \times n + 3 \times EF) \quad (10)$$

Stage-3

Stage 3 takes constant time steps to find the best out of the potential best paths according to the pre-decided multi-objectives. This stage takes (9—constant time instructions) as the worst-case complexity

S3 = 9

Thus, the **Total Complexity** of the improved algorithm can be calculated as:

$$S = \text{Complexity of (Stage 1 + Stage 2 + Stage 3)} \quad (11)$$

$$S = S1 + S2 + S3 \quad (12)$$

$$S = 3 + 3 \times (n \times EF) + C \times (10 \times EF \times n2 + 2 \times EF \times n + 2 \times n + 3 \times EF) + 9 \quad (13)$$

$$S = C \times 10 \times EF \times n2 + C \times 2 \text{ x } EF \times n + 3 \times EF \times n + C \times 2 \times n + C \times 3 \times EF + 12 \quad (14)$$

$$S \leq C \times 10 \times EF \times n2 + C \times 2 \times EF \times n + 3 \times EF \times n + C \times 2 \times n + C \times 3 \times EF + 12 \times EF \quad (15)$$

(taking C1 = C x 3 +12, to be some other positive constant)

$$S \leq C \times 10 \times EF \times n2 + C \times 2 \times EF \times n + 3 \times EF \times n + C \times 2 \times n + C1 \times EF \quad (16)$$

$$S \leq C \times 10 \times EF \times n2 + C \times 2 \times EF \times n + 3 \times EF \times n + C \times 2 \times n \times EF + C1 \times EF \times n \quad (17)$$

(taking C2 = 3 + C × 2 + C1 + C × 2)

$$S \leq C \times 10 \times EF \times n2 + C2 \times EF \times n \quad (18)$$

$$S \leq C \times 10 \times EF \times n2 + C2 \times EF \times n \times n$$

(taking C3 = C × 10 + C2)

$$S \leq C3 \times EF \times n^2 \quad (19)$$

(Na is also a positive constant, thus C4 = C3 × EF, be another positive constant)

$$S \leq C4 \times n^2 \quad (20)$$

Therefore, the overall complexity of the proposed can be approximated as n2, or in terms of Big-O notation, it is $O(n^2)$ or $O(|TS|^2)$. This is the same as the complexity of the

old ACO_TCSP algorithm. Thus, the proposed approach is found to be as time efficient as the earlier approach (ACO_TCSP).

6. Experimental Design

The proposed approach (Enhanced ACO_TCSP) has been developed in C++. The organization of the code included ten modules and five global functions. The time for executing the code was computed with the help of a clock() function (available in the file "time.h"). The code was implemented on Macbook Pro 2011 model hardware, macOS 10.12 Sierra operating system and the C++ runtime environment used to run the code was Code::Blocks.

6.1. Benchmark Programs

We used 4 programs as benchmark programs. P1, P2, P3, and P4 were chosen so that the cases for minimum (P8), maximum (P2), and medium (P1, P3) correctness found by the previous algorithm could be compared with the proposed algorithm. Concise depiction of the programs, the lines of code in them, the mutations induced, and the test suite execution times are listed in Table 1.

Table 1. Depiction of Benchmark programs.

Program	Program Title	Lines of Code (kloc)	No. of Mutations	Test Suite Size (TS)	Execution Time (ms)
P1	Coll_Admison	0.281	5	9	1.0532
P2	Triangle_sides	0.037	6	19	3.82
P3	Basic_calculator	0.101	9	25	0.825
P4	Railways Booking	0.129	10	26	1.77

6.2. Design

Each benchmark program was executed on 4 varying values of the *EF* (Enhancement Factor), along with 5 varying *TC* values. ACO being an approximation technique, does not yield the same result for the same set of inputs every time. Henceforth, 10 runs for each *EF* value for each program and each *TC* value were recorded (10 * 4 * 5 = 200 runs for each program). Every execution run returns the path found (may be optimal or non-optimal). The time constraints chosen as the termination criteria were chosen to be 200, 300, 400, 500, and 600 with forty runs of each program. Reduction in total execution time needed, resultant optimality of solution, and the correctness of the technique were obtained from the output data. We also found the percentage improvement in correctness using the new algorithm.

6.3. ACO Parameter Settings for Enhanced ACO_TCSP

To maintain the levels of exploration (finding new paths) and exploitation (using existing pheromone knowledge) five ACO parameters need to be set in order to produce the near optimum results. Various parameter settings have been suggested and used over the years [8,38,39]. On the basis of experimentation and results from [8] and the provided enhancements, the parameter settings used for Enhanced ACO_TCSP are described in Table 2.

Alpha α, and Beta β are the control parameters to keep in check the exploration versus exploitation balance of ACO algorithm. The values chosen were kept the same as used by many researchers and in [7,8]. 10% Evaporation rate was used to make sure that digital ants map to real ants, and that the pheromone does not keep on depositing, it evaporates at the end of each iteration. Additionally, +1 was tuned to be the amount of pheromone to be deposited on the edges of the best path. Q0 is a constant used for calculating the amount of pheromone to be evaporated joint with the evaporation rate. The value of Na was updated

as per the proposed enhancements. To counter-balance the enhanced exploration, the value of τ has been updated to ensure elitism and q0 ensures convergence. The rest of the parameter settings are the same as those used in the earlier version [8].

Table 2. Parameter Values used for Enhanced ACO_TCSP.

Na	N * EF
A	1.0 (Parameter to control exploration level)
B	0.2 (Parameter to keep exploitation in check)
P	0.1 (Evaporation Rate of pheromone—10% as used by majority researchers)
T	1.0 (Amount of Pheromone to be deposited on best path edges)
q0	1.0 (chosen constant used on calculation of pheromone to be evaporated)

7. Result Analysis

We now depict and explain the results of the implementation of the proposed enhanced technique here.

7.1. Execution Time of Paths Discovered

The details about the path and their corresponding execution time for four open-source programs have been given here. The combined results are shown in the next section.

The Enhanced ACO_TCSP was repeated for 10 Runs with *EF* values taken as 1, 2, 3, and 4 for program P1 using five values of *TC*. The averaged output of 10 runs for four values of the *EF* is summarized in Table 3. The details of the rest of the programs have been omitted due to redundancy and space constraints. However, the graphs depicting the results of the execution of all four programs have been depicted in the next section.

Possible best paths discovered in each of the 40 (5 *TC*, 4 *EF*) runs of Enhanced ACO_TCSP were compared for their execution times for all 4 test programs and are depicted graphically in Figure 4. Unlike previous results, a very slight decreasing trend is observed for increasing values of *EF*. It can be thus derived from here that an increased number of ants (proportional directly to the *EF*) leads to decreased execution time for the possible best paths. The exception for P2 is that the possible best path was found every time due to multiple possible best paths possible in the problem.

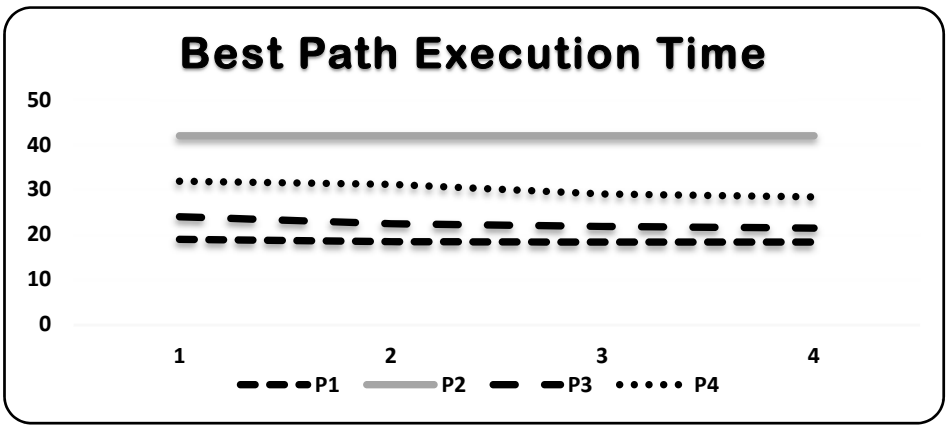

Figure 4. Best Path Execution Time of Enhanced ACO_TCSP versus EF.

Table 3. Enhanced ACO_TCSP on P1 with varying *TC & EF*.

RUN	Na	Running Time (sec)	Best Path	Total Exec. Time	Best P Exec. Time	No. of Test Cases Covered	No. of Iterations	Best P Found at Iteration	Optimal P Found at Iteration
TC = 200									
ants	1	0.054945	8	82.10127	18.796	2	10.7	2.4	2.25
ants	2	0.054945	9	82.3225	18.563	2	10.8	3.8	4
ants	3	0.049451	10	82.54895	18.33	2	10.9	3.2	3.2
ants	4	0.054945	10	82.54373	18.33	2	11	2.3	2.3
avg		**0.053571**		**82.37911**	**18.50475**	**2**	**10.85**	**2.925**	**2.945946**
TC = 300									
ants	1	0.043956	6	80.87737	20.096	2.1	15.3	3.2	3.666667
ants	2	0.06044	9	82.23752	18.647	2	16.2	4.5	4.555556
ants	3	0.054945	10	82.54373	18.33	2	16.9	1.8	1.8
ants	4	0.065934	10	82.53329	18.33	2	17	1.7	1.7
avg		**0.056319**		**82.04798**	**18.85075**	**2.025**	**16.35**	**2.8**	**2.8**
TC = 400									
ants	1	0.049451	8	81.96406	18.946	2	20.8	7.2	7.5
ants	2	0.054945	10	82.54373	18.33	2	22	3.4	3.4
ants	3	0.06044	10	82.53851	18.33	2	22	1.3	1.3
ants	4	0.071429	10	82.52808	18.33	2	22	1.8	1.8
avg		**0.059066**		**82.39359**	**18.484**	**2**	**21.7**	**3.425**	**3.289474**
TC = 500									
ants	1	0.054945	9	82.3225	18.563	2	26.5	6.2	6.555556
ants	2	0.054945	10	82.54373	18.33	2	27.4	3.7	3.7
ants	3	0.06044	10	82.53851	18.33	2	27.8	2.2	2.2
ants	4	0.071429	10	82.52808	18.33	2	27.4	1.3	1.3
avg		**0.06044**		**82.4832**	**18.38825**	**2**	**27.275**	**3.35**	**3.358974**
TC = 600									
ants	1	0.054945	10	82.54373	18.33	2	32.2	5.6	5.6
ants	2	0.065934	10	82.53329	18.33	2	32.5	5.7	5.7
ants	3	0.06044	10	82.53851	18.33	2	32.8	4.2	4.2
ants	4	0.076923	10	82.52286	18.33	2	33	1.8	1.8
avg		**0.06456**		**82.5346**	**18.33**	**2**	**32.625**	**4.325**	**4.325**

The average number of iterations over 40 runs each, observed in the case of *TC* and *EF* are presented in Figure 5. A steady surge in the number of iterations needed by Enhanced ACO_TCSP with an increase in input *EF* values can be observed. This is promising and encouraging because this implies that there is a reduction in the execution time of possible best paths earlier in the iterations. This causes more iterations to take place before reaching the stopping *TC*. Thereby, higher values of *EF*, indicate a rise in the number of iterations taken by Enhanced ACO_TCSP for the execution of the regression test suite under consideration, for the generation of the selected and prioritized test suite as resultant.

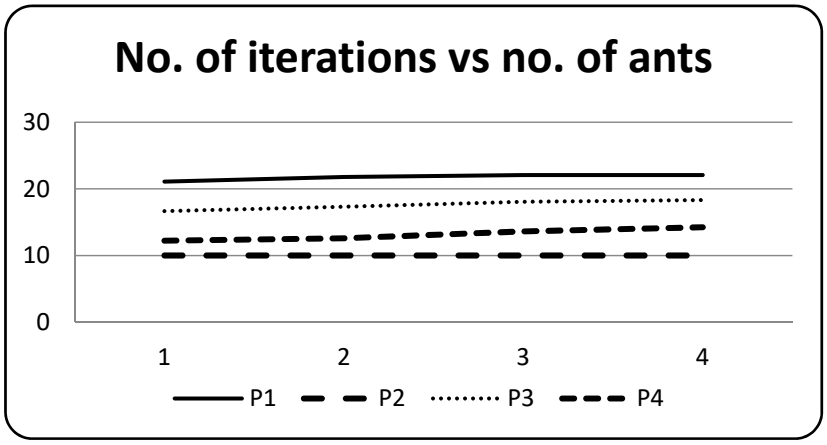

Figure 5. Average No. of Iterations v/s No. of ants sent.

The average iteration number at which the optimum path is found by the Enhanced ACO_TCSP execution out of 40 runs and its variation with different *EF* (no. of ants) is shown in Figure 6 (tabulated in the last column of Table 2 also). A steady fall in the iteration number capable of revealing the optimum path with the rising *EF* is clearly observable. P2 portrays an exception by revealing the optimum path in first iteration for all 200 test runs each. Consequently, by enhancing *EF*, the likelihood of locating the optimum path arises at even earlier stages of the improved technique. This fall in the 'accurate iteration number' (iteration that reveals the optimal path for the first time is called the 'accurate iteration'), with the increase in *EF* is the couple effect caused by the improved exploration search space (that now avoids falling into local optima that could lead to inaccurate iterations), and elite exploitation of the earlier explored paths.

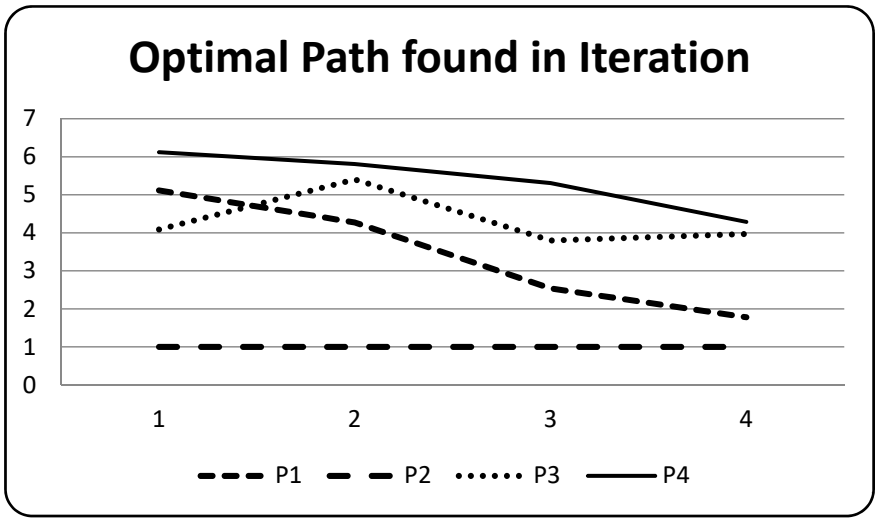

Figure 6. Averaged iteration number finding the Optimal path v/s EF.

The fact that ACO provides an optimum reduction of execution time has been witnessed in the previous section. The results have been averaged over 40 runs with four varying values of *EF* on the benchmark programs. The graphical analysis is presented in Figure 7. There is a slight rise in the reduction of execution time with the rise in *EF*. The observations are a result of the enhancements carried out in the technique. Better results have been yielded even at lesser values of *TC*, and are well-adjusted by the enhanced number of ants.

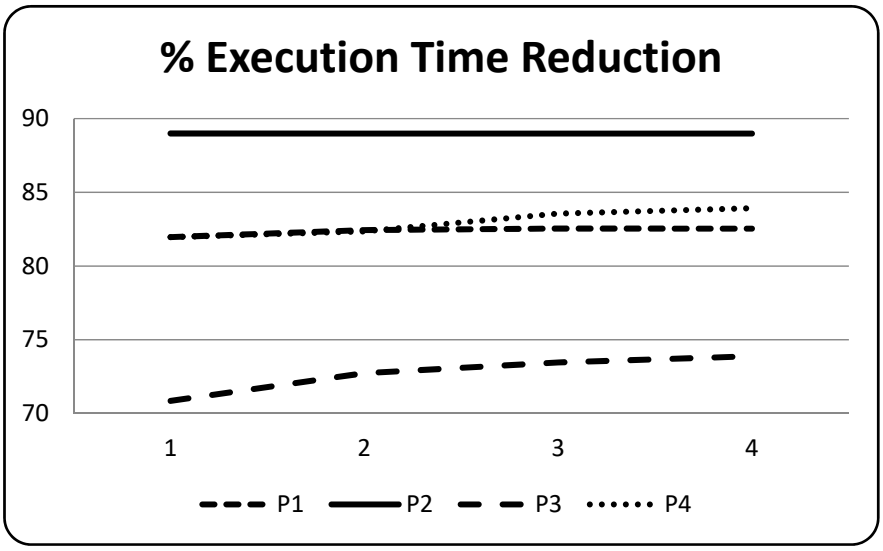

Figure 7. % execution time reduction for Enhanced ACO_TCSP selected test cases v/s EF.

7.2. Correctness of the Technique

This sub-section tries to prove the correctness of the technique, both theoretically as well as experimentally using the results of our empirical evaluation.

Theoretically, the improved algorithm ensures that a path is completed if all faults are found or all test cases have been visited. This ensures that the algorithm would stop within the computed complexity. In addition, the pheromone is then deposited on the best path from each iteration, ensuring exploitation of already found paths. ACO is a randomized approximation approach. Hence, the authors do not claim that the final test suite has minimum execution time. However, it is definitely found that the best APFD or fault coverage would be achieved by the final test suite. Experimentation will prove how many times ACO results in minimum execution time as well.

In order to prove the correctness of the proposed technique, Figure 8, Figure 9, and Table 4 shall be used to depict the correctness of the proposed work.

The percentage correctness of the Enhanced ACO_TCSP versus the *EF* has been picturized graphically in Figure 8. A very motivating and clear observation is the rise in the correctness achieved with the rise in *EF* (no. of ants). This validates the improvement and enhancements made in the prior ACO technique, which now is not falling to the local optima problems.

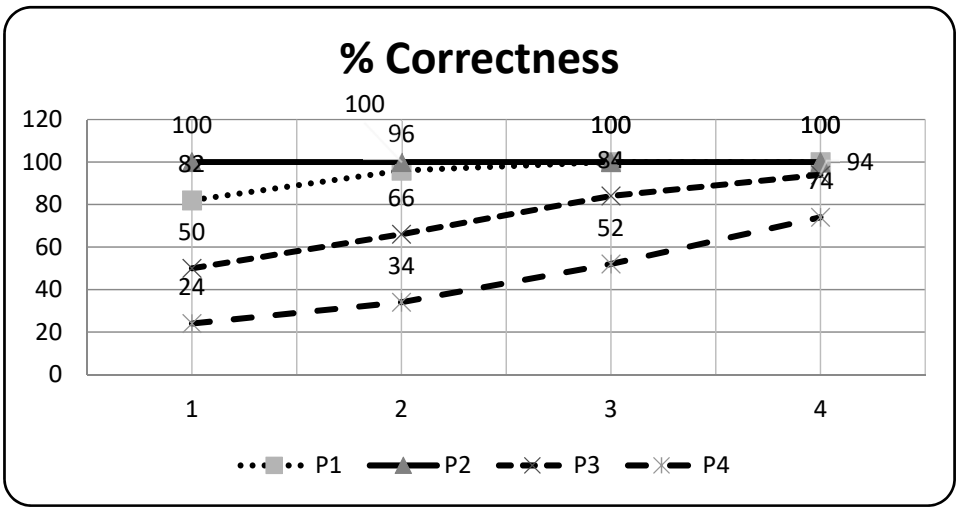

Figure 8. Percentage Correctness Achieved by Enhanced ACO v/s EF.

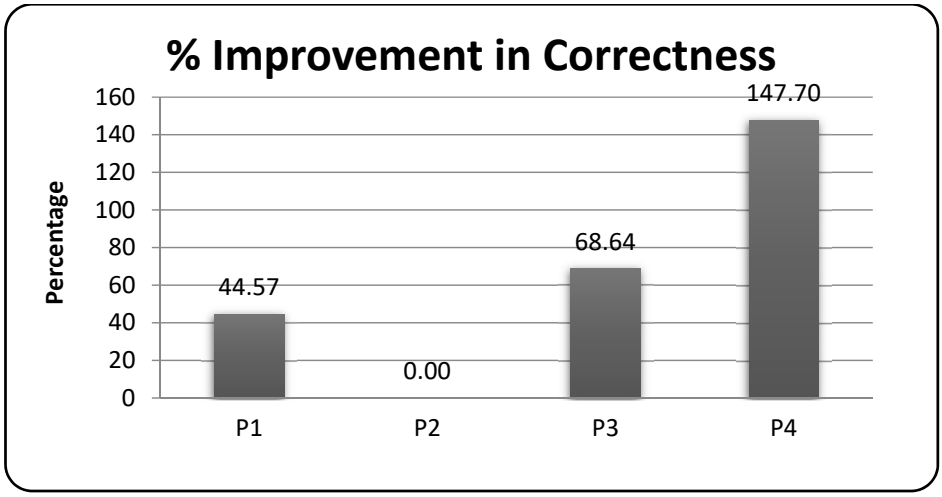

Figure 9. Percentage Improvement in Correctness Achieved by Enhanced ACO.

Table 4. Percentage improvement in Correctness Achieved by Enhanced ACO_TCSP.

Prog. No.	OLD % Correctness	NEW % Correctness	% Improvement
P1	65.714	95	44.565846
P3	100	100	0
P6	47.143	79.5	68.635853
P8	18.571	46	147.69802

Figure 9 depicts the improvement of percentage correctness achieved by the Enhanced ACO_TCSP approach. The achieved percentage improvement is nearly 50% in general, while for the earlier worst case of P4 using ACO_TCSP, an improvement of 147.7% is achieved using Enhanced ACO_TCSP. This clearly proves that the proposed and improved algorithm is better than the old ACO_TCSP algorithm.

7.3. APFD and Statistical Analysis

In order to obtain APFD (Average Percentage of Faults Detected), we calculated the area below the plotted line using a graph plot drawn between % of faults detected with the number of test cases needed. The notations used for calculation of APFD are:

'TS': the test suite containing the set of '|T|' test cases,

'F': the set of '|F|' faults revealed using 'TS'.

For prioritization of test suite 'TS', let TF_i denote the priority order of the initial test case revealing the ith fault. The APFD for 'TS' can be obtained from the following equation:

$$\text{APFD} = 1 - \frac{TF1 + \ldots + TFm}{|T|*|F|} + \frac{1}{2*|T|} \tag{21}$$

Although, APFD is the popularly used criteria for the evaluation of the techniques used in the prioritization of test cases. Maximization of the APFD is not the objective of test case prioritization techniques. Maximization of APFD is a possibility when it is known in advance which faults are killed by a given test suite, thereby implying that the execution of entire test cases is already completed. Then, there would be absolutely no need for prioritization of test cases. APFD is thus needed after the task of prioritization for the evaluation of the prioritization technique.

The Enhanced ACO_TCSP orderings achieved for the four sample programs have been empirically evaluated (with respect to: No order, Random order, Reverse order, and optimum order of the test cases). These approaches are evaluated using APFD. Figures 9–12

depict the results obtained. It is evident that ACO attains results similar to that of optimum ordering, and has been shown to outweigh the reference techniques in terms of % of fault coverage achieved.

From Figure 10, it can be inferred that the same APFD value of 72.22% is achieved for the optimum and the ACO ordering for P1. Similar results are yielded for other programs also, as depicted in Figures 11–13.

The best APFD results achieved using the improved ACO algorithm, as depicted above, ensure that not only the maximum faults are covered in the entire test suite, but also maximum faults are revealed at earlier stages of running the prioritized test suite. These further motivate us to use the Enhanced ACO_TCSP algorithm.

In order to further ascertain the efficiency of our proposed work, we statistically analyzed the performance of existing ACO and the proposed Enhanced ACO_TCSP approach. We use the Independent Two Sample *t*-Test for statistical analysis. The objective here is to inspect if the proposed technique is more efficient than its earlier version. For this, we apply a *t*-test to compare % correctness and % time reduction achieved in the case of the four benchmark programs.

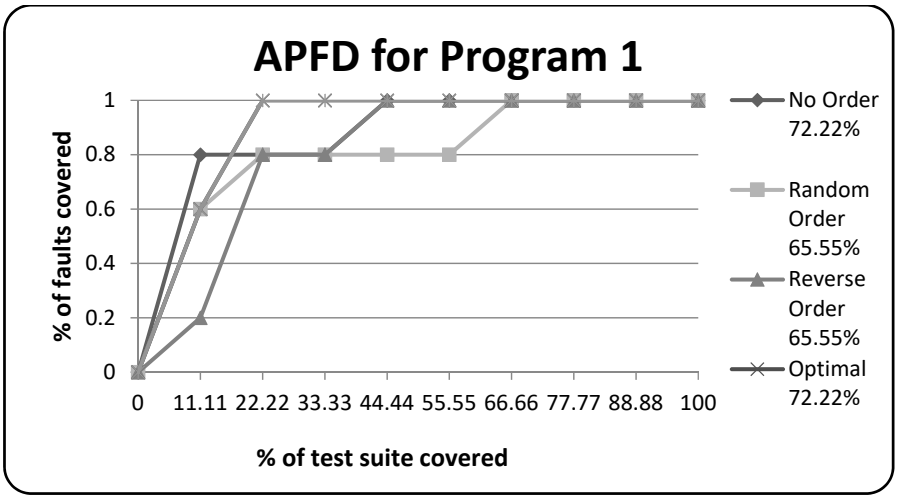

Figure 10. APFD for P1.

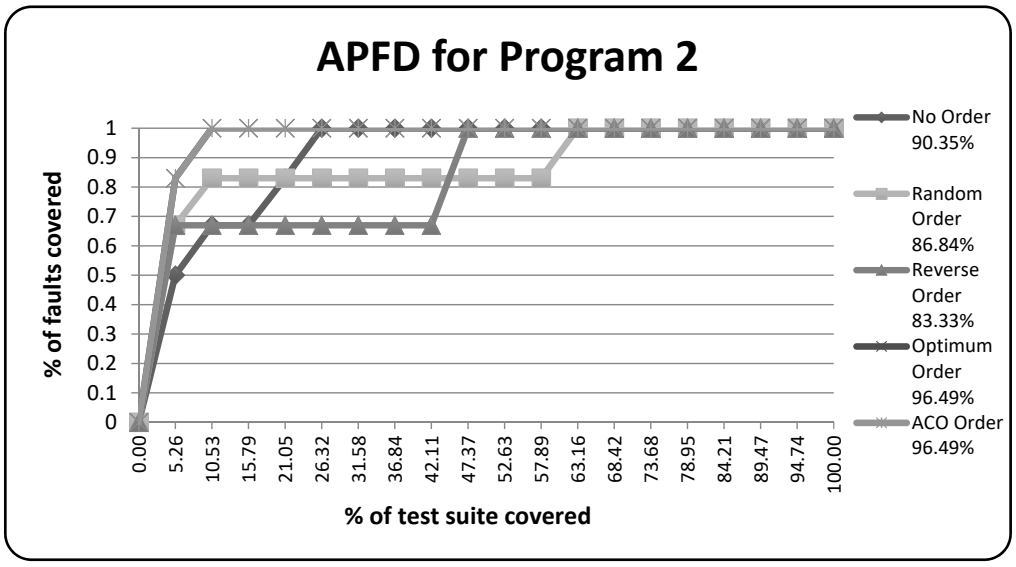

Figure 11. APFD for P2.

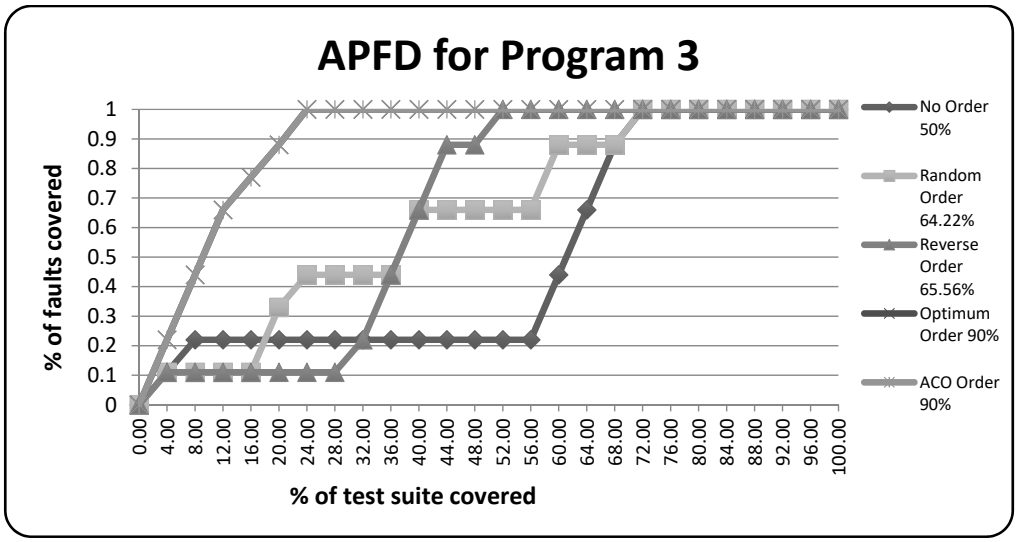

Figure 12. APFD for P3.

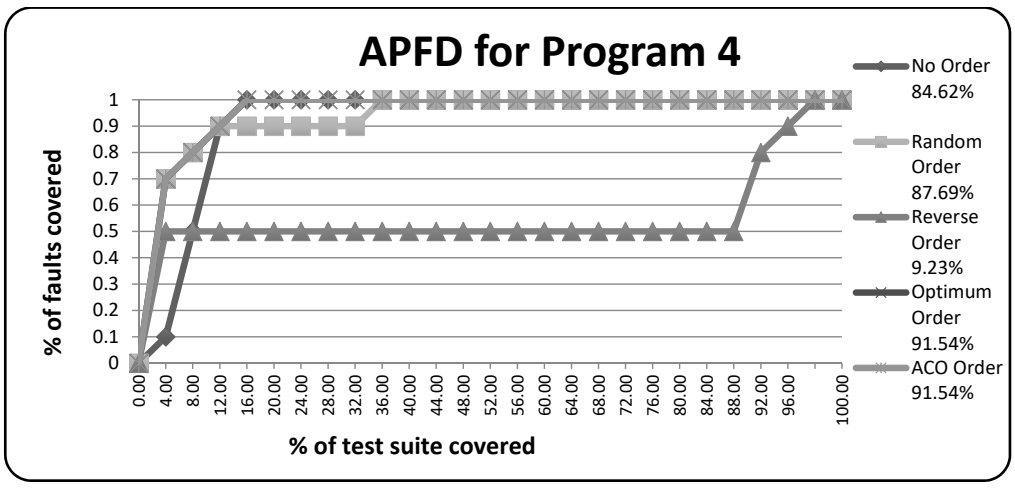

Figure 13. APFD for P4.

The Null Hypothesis (H0) for the *t*-test is taken as:

(H0.) There is no significant improvement in the performance of Enhanced ACO_TCSP (in terms of % correctness and % time reduction) as compared to its earlier version (ACO_TCSP) for test data selection and prioritization.

To ascertain our claim, proving our research hypothesis and thereby rejecting the H0, the outcome of statistical analysis was conducted using Python language. The outcomes have been tabulated in Table 5 and depicted in Figures 14 and 15 below.

Table 5 can be observed to find that the probability of H0 being true is significantly less (*p*-value < 0.5). Thereby we can reject the H0 (Null Hypothesis) and state that our research hypothesis is true, that is, our proposed technique exhibits significantly improved performance over its earlier version.

Table 5. *p*-value obtained after applying an independent two-sample *t*-test.

p-value obtained for % correctness	0.05896016215814072
p-value obtained for % time reduction	0.11581073519437922

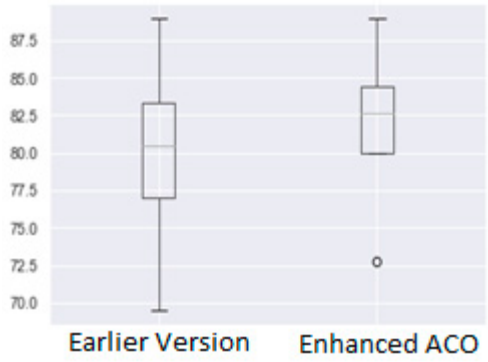

Figure 14. Comparison of % Correctness achieved.

Figure 15. Comparison of % time reduction achieved.

The results of the *t*-test can be graphically analyzed using the following box plots:

Encouragingly, Figures 13 and 14 help us to unambiguously show the significant improvement achieved in the case of Enhanced ACO_TCSP in terms of % correctness and % time reduction for test case selection and prioritization. This provides a concrete validation to the proposed enhancements and motivates us to further use the enhanced approach in the field.

8. Discussion

In this paper, an enhancement of ACO for improving the test case selection and prioritization technique proposed by Singh et al. [7] has been developed and validated on four benchmark programs. Moreover, a comparison with five traditional prioritization techniques has been accomplished using APFD. The results achieved are encouraging owing to the following reasons:

(1) The proposed technique results in the minimization of the test suite as the EF increases,
(2) The running time is substantially reduced, and with the rise in EF, it is further reduced,
(3) The precision of results achieved is encouraging for most of the test programs,
(4) The percentage improvement in correctness is very high compared to the previous technique,
(5) A comparison of the Enhanced ACO_TCSP prioritized test suite with No Order, Reverse Order, Random Order and Optimal Order prioritized test suites using APFD has been carried out. The results yielded APFD values for ACO that are equivalent to the optimum values (values that have maximum possible fault exposure in minimum possible time). The effect of the enhancement factor for different values of *EF* depicted motivating observations validates the Enhanced ACO_TCSP against the old approach [31]. The time reduction for the chosen resultant test suite by ACO was found to be almost the same for varying values of *EF*. This is due to the balancing provided by the increase in the number of ants for the new algorithm. The resultant test suite thus obtained potentially provides fast fault coverage.

The investigation of the usage of enhancement factor for different values of EF steers one to the following substantial leads:

1. A higher number of possible best paths are found at increased EF.
2. The selected best path is about the same for low and high values of EF.
3. The iteration number for convergence of Enhanced ACO_TCSP reduces with the increase in EF, this validates more exploration at the initial stages of the algorithm also.
4. Enhanced ACO_TCSP tends to yield optimum results at higher values of EF.

In addition to the above, statistical validation of the Enhanced ACO_TCSP has also been conducted. An Independent Two Sample *t*-Test has been performed to examine the correctness and execution time improvement achieved. Box Plots have also been used to represent the same. To affirm the validation of Enhanced ACO_TCSP, the *t*-test produced excellent results. All the aforementioned observations indicate that the proposed Enhanced ACO_TCSP technique yields promising solutions and exhibits better results than the existing technique in terms of solution correctness. The paper contributes to the literature by presenting the Enhanced ACO_TCSP approach and providing detailed enhancements and their validation on four benchmark programs without compromising the complexity of the algorithm. This can be easily re-implemented and fruitfully used by researchers for selecting and prioritizing test cases in the future.

9. Conclusions and Future Scope

The enhanced ACO_TCSP proposed in this work enhances the search space and makes the exploitation elite. The experimental results of the proposed approach on four benchmark programs were found to validate the enhancements. The average accuracy of the Enhanced ACO_TCSP was found to improve by over 30% over the original ACO_TCSP approach. Furthermore, the optimal paths converged at earlier iterations. All this could be achieved without an increase in execution time. This was ensured by the stopping criteria of *TC* (time-constraint) entered by the user. In addition to this, APFD analysis also proved the earlier exposure of faults achieved in comparison with the traditional prioritization approaches. Hence, as in the real world, increasing the size of the ant colony ensures more exploration, and elitism ensures intelligent exploitation of the already discovered paths. As in real ants, these ensure finding the optimal path with earlier convergence. Henceforth, this paper presents and validates the Enhanced ACO_TCSP approach for solving regression test selection and prioritization. As a part of future work, we can implement newer techniques [40] and empirically evaluate them for test case selection and prioritization. The proposed enhancements in ACO_TCSP proposed in this work can be applied and tested on many more metaheuristics that have been applied in the area of test case selection and prioritization for better efficiency in terms of results, without increasing the time complexity of these techniques. The limitations of the proposed work are as follows. The results have been experimentally evaluated on small codes. They can be experimentally evaluated in future work. The limitations associated with ACO as a methodology are also applicable to our proposed technique; however, the most prominent limitation of ACO getting stuck in local minimum has been averted using our technique due to the proposed enhancements. We have tried to imitate the behavior of real ants in our work; however, the intuitive behavior of the real ants cannot be incorporated even in the Enhanced ACO_TCSP technique.

Author Contributions: Conceptualization, S.S. and N.J.; methodology, S.S. and N.J.; software, S.S., N.J., K.S., G.D. and S.M.; validation, R.G. and B.S.; formal analysis, M.N., S.N.M. and N.R.P.; investigation, S.S. and N.J.; resources, S.S. and N.J.; data curation, S.S. and N.J.; writing—original draft preparation, S.S. and N.J.; writing—review and editing, S.S., N.J., K.S., G.D. and S.M.; visualization, S.S. and N.J; supervision, B.S.; project administration, R.G. and B.S.; funding acquisition, M.N., S.N.M. and N.R.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No data was needed in the work related to this research. The online sources from which the programs under test were downloaded re mentioned in the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bajaj, A.; Abraham, A.; Ratnoo, S.; Gabralla, L.A. Test Case Prioritization, Selection, and Reduction Using Improved Quantum-Behaved Particle Swarm Optimization. *Sensors* **2022**, *22*, 4374. [CrossRef] [PubMed]
2. Ansari, A.; Khan, A.; Khan, A.; Mukadam, K. Optimized regression test using test case prioritization. *Procedia Comput. Sci.* **2016**, *79*, 152–160. [CrossRef]
3. Kumar, S.; Ranjan, P. ACO based test case prioritization for fault detection in maintenance phase. *Intern. J. Appl. Eng. Res.* **2017**, *12*, 5578–5586.
4. Mohapatra, S.K.; Prasad, S. Finding Representative Test Case for Test Case Reduction in Regression Testing. *Int. J. Intell. Syst. Appl.* **2015**, *7*, 60. [CrossRef]
5. Noemmer, R.; Haas, R. An evaluation of test suite minimization techniques. In *Software Quality: Quality Intelligence in Software and Systems Engineering*; Springer: Vienna, Austria, 2019.
6. Sharma, B.; Hashmi, A.; Gupta, C.; Jain, A. Collaborative Recommender System based on Improved Firefly Algorithm. *Comput. Y Sist.* **2022**, *26*, 2. [CrossRef]
7. Singh, Y.; Kaur, A.; Suri, B. Test case prioritization using ant colony optimization. *ACM SIGSOFT Softw. Eng. Notes* **2010**, *35*, 1–7. [CrossRef]
8. Suri, B.; Singhal, S. Implementing Ant Colony Optimization for Test Case Selection and Prioritization. *Int. J. Comput. Sci. Eng.* **2011**, *3*, 1924–1932.
9. Wu, L.; Huang, X.; Cui, J.; Liu, C.; Xiao, W. Modified adaptive ant colony optimization algorithm and its application for solving path planning of mobile robot. *Expert Syst. App.* **2023**, *215*, 119410. [CrossRef]
10. Bajaj, A.; Sangwan, O.P. A systematic literature review of test case prioritization using genetic algorithms. *IEEE Access* **2019**, *7*, 126355–126375. [CrossRef]
11. Li, F.; Zhou, J.; Li, Y.; Hao, D.; Zhang, L. Aga: An accelerated greedy additional algorithm for test case prioritization. *IEEE Trans. Softw. Eng.* **2021**, *48*, 5102–5119. [CrossRef]
12. Jatana, N.; Suri, B. Particle swarm and genetic algorithm applied to mutation testing for test data generation: A comparative evaluation. *J. King Saud Univ.-Comput. Inf. Sci.* **2020**, *32*, 514–521. [CrossRef]
13. Chaudhary, N.; Sangwan, O. Multi Objective Test Suite Reduction for GUI Based Software Using NSGA-II. *Int. J. Inf. Technol. Comput. Sci.* **2016**, *8*, 59–65. [CrossRef]
14. Öztürk, M.M. A bat-inspired algorithm for prioritizing test cases. *Vietnam. J. Comput. Sci.* **2018**, *5*, 45–57. [CrossRef]
15. Dhareula, P.; Ganpati, A. Flower Pollination Algorithm for Test Case Prioritization in Regression Testing. In *ICT Analysis and Applications: Proceedings of ICT4SD 2019*; Springer: Singapore, 2020; Volume 93, pp. 155–167. [CrossRef]
16. Srivastava, P.; Reddy, D.P.K.; Reddy, M.S.; Ramaraju, C.V.; Nath, I.C.M. Test Case Prioritization Using Cuckoo Search. In *Advanced Automated Software Testing: Frameworks for Refined Practice*; IGI Global: Hershey, PA, USA, 2020; pp. 113–128.
17. Kaushik, A.; Verma, S.; Singh, H.J.; Chhabra, G. Software cost optimization integrating fuzzy system and COA-Cuckoo optimization algorithm. *Int. J. Syst. Assur. Eng. Manag.* **2017**, *8*, 1461–1471. [CrossRef]
18. Mann, M.; Sangwan, O.P. Test case prioritization using Cuscuta search. *Netw. Biol.* **2014**, *4*, 179–192.
19. Jatana, N.; Suri, B. An Improved Crow Search Algorithm for Test Data Generation Using Search-Based Mutation Testing. *Neural Process. Lett.* **2020**, *52*, 767–784. [CrossRef]
20. Panwar, D.; Tomar, P.; Singh, V. Hybridization of Cuckoo-ACO algorithm for test case prioritization. *J. Stat. Manag. Syst.* **2018**, *21*, 539–546. [CrossRef]
21. Yoo, S.; Harman, M. Using hybrid algorithm for Pareto efficient multi-objective test suite minimisation. *J. Syst. Softw.* **2010**, *83*, 689–701. [CrossRef]
22. Xu, Z.; Gao, K.; Khoshgoftaar, T.M.; Seliya, N. System regression test planning with a fuzzy expert system. *Inf. Sci.* **2014**, *259*, 532–543. [CrossRef]
23. Zhou, Z.Q.; Sinaga, A.; Susilo, W.; Zhao, L.; Cai, K.-Y. A cost-effective software testing strategy employing online feedback information. *Inf. Sci.* **2018**, *422*, 318–335. [CrossRef]
24. Dorigo, M.; Maniezzo, V.; Colorni, A. The Ant System: An Autocatalytic Optimizing Process. In *Technical Report TR91-016*; Politecnico di Milano: Milano, Italy, 1991.
25. Chaudhary, R.; Agrawal, A.P. Regression Test Case Selection for Multi-Objective Optimization Using Metaheu-ristics. *Int. J. Inf. Technol. Comput. Sci.* **2015**, *7*, 50–56.
26. Bian, Y.; Li, Z.; Zhao, R.; Gong, D. Epistasis based aco for regression test case prioritization. *IEEE Trans. Emerg. Top. Comput. Intell.* **2017**, *1*, 213–223. [CrossRef]

27. Vescan, A.; Pintea, C.M.; Pop, P.C. Solving the test case prioritization problem with secure features using ant colony system. In Proceedings of the International Joint Conference: 12th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2019) and 10th International Conference on European Transnational Education (ICEUTE 2019), Seville, Spain, 13–15 May 2019; Springer: Cham, Switzerland, 2019.
28. Suri, B.; Singhal, S. Understanding the effect of time-constraint bounded novel technique for regression test selection and prioritization. *Int. J. Syst. Assur. Eng. Manag.* **2015**, *6*, 71–77. [CrossRef]
29. Noguchi, T.; Washizaki, H.; Fukazawa, Y.; Sato, A.; Ota, K. History-Based Test Case Prioritization for Black Box Testing Using Ant Colony Optimization. In Proceedings of the IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), Graz, Austria, 13–17 April 2015. [CrossRef]
30. Ahmad, S.F.; Singh, D.K.; Suman, P. Prioritization for Regression Testing Using Ant Colony Optimization Based on Test Factors. In *Intelligent Communication, Control and Devices*; Springer: Berlin, Germany, 2018; pp. 1353–1360. [CrossRef]
31. Singhal, S.; Jatana, N.; Suri, B.; Misra, S.; Fernandez-Sanz, L. Systematic literature review on test case selection and prioritization: A tertiary study. *Appl. Sci.* **2021**, *11*, 12121. [CrossRef]
32. Suri, B.; Singhal, S. Evolved regression test suite selection using BCO and GA and empirical comparison with ACO. *CSI Trans. ICT* **2016**, *3*, 143–154. [CrossRef]
33. Suri, B.; Singhal, S. Literature survey of Ant Colony Optimization in software testing. In Proceedings of the CONSEG, CSI Sixth International Conference On Software Engineering, Indore, India, 5–7 September 2012. [CrossRef]
34. Kavitha, R.; Jothi, D.K.; Saravanan, K.; Swain, M.P.; Gonzáles, J.L.A.; Bhardwaj, R.J.; Adomako, E. Ant colony optimization-enabled CNN deep learning technique for accurate detection of cervical cancer. *BioMed Res. Int.* **2023**, *2023*, 1742891. [CrossRef]
35. Singhal, S.; Suri, B. Multi objective test case selection and prioritization using African buffalo optimization. *J. Inf. Optim. Sci.* **2020**, *41*, 1705–1713. [CrossRef]
36. Singhal, S.; Jatana, N.; Dhand, G.; Malik, S.; Sheoran, K. Empirical Evaluation of Tetrad Optimization Methods for Test Case Selection and Prioritization. *Indian J. Sci. Technol.* **2023**, *16*, 1038–1044. [CrossRef]
37. Suri, B.; Singhal, S. Analyzing test case selection & prioritization using ACO. *ACM SIGSOFT Softw. Eng. Notes* **2011**, *36*, 1–5.
38. Dorigo, M.; Di Caro, G.; Gambardella, L.M. Ant Algorithms for Discrete Optimization. *Artif. Life* **1999**, *5*, 137–172. [CrossRef]
39. Chen, X.; Gu, Q.; Zhang, X.; Chen, D. Building Prioritized Pairwise Interaction Test Suites with Ant Colony Optimization. In Proceedings of the 2009 Ninth International Conference on Quality Software, Jeju, Korea, 24–25 August 2009; pp. 347–352. [CrossRef]
40. Zhu, A.; Xu, C.; Li, Z.; Wu, J.; Liu, Z. Hybridizing grey wolf optimization with differential evolution for global optimization and test scheduling for 3D stacked SoC. *J. Syst. Eng. Electron.* **2015**, *26*, 317–328. [CrossRef]