Software Test Case Prioritization Using Genetically Modified Flower Pollination Algorithm (Gm-Fpa)

Priyanka Dhareula and Anita Ganpati

Abstract: The Nature Inspired Metaheuristic Optimization Algorithms (NIMOA) institutes towards explanations to optimization glitches in countless realms. Numerous NIMOA have been employed in the dominion of Test Case Prioritization (TCP) for Regression Testing (RT). There are some NIMOA that have not been thoroughly investigated for TCP in RT. One of the recent additions made to NIMOA is Flower Pollination Algorithm (FPA) that has not been used for TCP in RT as per the literature survey. This study has made a theoretical attempt to find the most efficacious metaheuristic technique for TCP. The theoretical results have reflected that the Genetic Algorithm (GA) is the most prevalent NIMOA used for TCP. Although a particular NIMOA cannot be considered as efficacious for TCP. This study tries to bridge the gap between the most popular metaheuristic techniques used for TCP and recent addition made to NIMOA. The main objective of this study is to propose an enhanced metaheuristic TCP technique. Henceforth, the study proposes Genetically Modified-Flower Pollination Algorithm (GM-FPA) for TCP in RT. Also, GA is used in the present study to perform TCP. GM- FPA is empirically evaluated by comparing the value of Average Percentage of Faults Detected (APFD) metrics with the APFD results of FPA, GA, and traditional approaches for TCP in RT. The empirical results have reflected that GM-FPA outperformed the FPA, GA, and traditional ways of TCP.

Index Terms: APFD, Flower Pollination Algorithm, Genetic Algorithm, Test Case Prioritization

1. INTRODUCTION

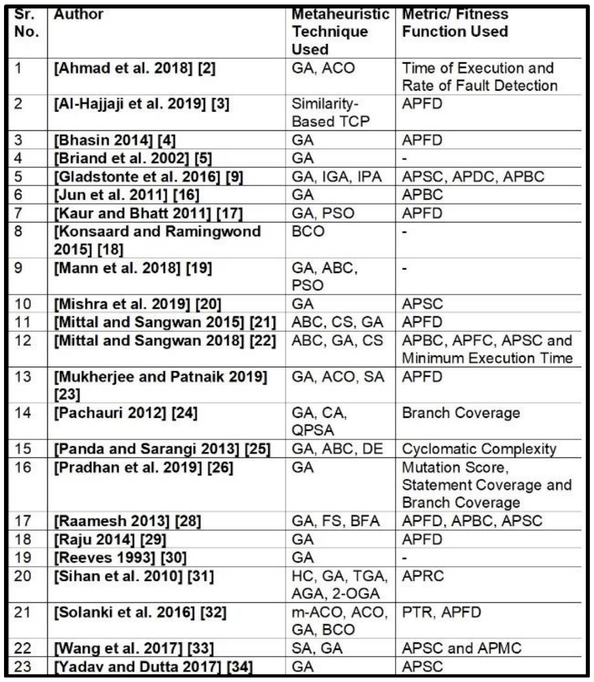
In the globe of technology, we are surrounded by software ubiquitously. With the paradigm shift, it becomes imperative to update the software as and when required. Whenever the software undertakes maintenance, it may lead to unwanted behavior that was not manifested in the earlier working version. Unintended behavior of the software can be analyzed by performing Regression Testing (RT). RT is a part of the maintenance phase; it is performed whenever software changes. RT is performed on the software to assess if the earlier working functionalities have been altered with the changes made to the software or not. To perform RT a pool of test cases is exercised upon the application under consideration. Whenever the changes are incorporated in the software, the size of the test pool becomes bulky. It is rationally not possible to perform RT with such huge size of test pool due to limited time and cost constraints. Therefore, there is a huge cry in the area of testing to deduce an effective mechanism, to reduce the size of a test pool, so that effective RT can be performed in given constraints. There are few traditional ways of optimizing test cases namely; minimization, selection, prioritization, and re-test all approach [Pressman 2005, Aggarwal & Yogesh 2007]. In this study Test Case prioritization (TCP) is used to order the test cases; so that higher priority test cases can be executed earlier to achieve maximum fault coverage in minimum possible time. A number of traditional approaches are known to exist [Dhareula and Ganpati 2015, Dhareula and Ganpati 2016] in various studies that perform test case optimization for RT. This study tries to bridge a gap between traditional ways of TCP and recent approaches, known as Nature Inspired Metaheuristic Optimization Algorithms (NIMOA), for optimizing problems in the dominion of software engineering [Yang 2014]. This study tries to find the most efficacious metaheuristic technique for TCP as per the literature available. Genetic Algorithm (GA) is used in this study to perform TCP. The main objective of this study is to propose an enhanced metaheuristic technique for TCP. Henceforth, the study proposes Genetically Modified-Flower Pollination Algorithm (GM-FPA) for TCP in RT. The proposed technique is compared with Flower Pollination Algorithm FPA [Dhareula and Ganpati 2019], GA and Traditional approaches for TCP in RT.

2. RELATED WORK

[Mittal and Sangwan 2018] used Artificial Bee Colony (ABC) optimization algorithm to propose TCP technique. Their technique is compared with GA and Cuscutta Search (CS). As per their study, GA and ABC outperformed Cuscutta Search for test suite with small size. Average Percentage of Statement Covered (APSC), Average Percentage of Blocks Covered (APBC), Average Percentage of Faults Covered (APFC) metrics, and time of execution is used to perform the comparison. [konsaard and Ramingwong 2015] used code coverage criteria to perform TCP by proposing modified GA. Their technique is compared with Bee Colony Optimization (BCO) algorithm. They inferred that BCO is highly complex as compared to GA. Therefore, it is stated in their study that BCO is not suitable for TCP. [Jun et al. 2011] used improved GA for TCP. To determine the effectiveness of their approach, APBC metrics is used. Their technique resulted in better rates of error finding. [Yadav and Dutta 2017] used statement coverage detail as input to GA for TCP. APSC is used to determine the effectiveness of the proposed technique. Comparison of their technique is performed with random and reverse prioritization. Their study produced optimum results. [Wang et al. 2017] performed TCP by hybridizing Simulated Annealing (SA) and GA algorithm. Applications written in Java and C were used to analyze the efficiency of proposed Annealing- Genetic Algorithm. [Ahmad et al. 2018] proposed a hybrid approach for TCP by combining Ant Colony Optimization (ACO) and GA. In their technique initially, test cases are generated and later prioritized based on fault detection rate and time of execution. [Mann et al. 2018] came up with Test Case Generation (TCG) and TCP techniques. Particle Swarm Optimization (PSO), ABC, and GA are used for TCG and PSO is used for TCP. In their study, ABC outperformed PSO and GA for TCG. [Pachauri 2012] proposed enhanced Clonal Algorithm (CLONALG). They used a benchmark program for extensive experiments and compared CLONALG with Quantum Particle Swarm

Optimization Algorithm (QPSO) and GA. Their technique outperformed the other techniques. [Raamesh 2013] performed TCP by using Feature Selection (FS), Bird Flocking Algorithm (BFA), and GA. For the evaluation of the proposed system APBC, APSC and APFD metrics were used. Their study inferred that NIMOA are highly effective in TCP. [Raju 2014] performed TCP by using GA, clustering techniques and test case weight. APFD metrics is used to compare the effectiveness of different techniques. Their results stated that GA is best suited for TCP with application of varying sizes. [Reeves 1993] performed TCP in minimum cost by employing GA. They steered the search course by means of "survival of the fittest" theory and discovered the effectiveness of GA for TCP. [Panda and Sarangi 2013] projected a technique for TCG by using path coverage as the criteria. ABC, GA, and Differential Evolution (DE) algorithms are used in their study. Cyclomatic complexity and control flow graph are used for the comparison of the definite number of paths existing and number of paths traversed by the test cases. ABC outperformed other techniques in their study. [Briand et al. 2002] proposed a technique by merging GA and inter-class coupling for TCP. GA performs the complexity analysis of the system under study and coupling measurement is used to reduce the complex cost functions. Their method is effective in deliberating systematic and optimal results. [Sihan 2010] used five algorithms, namely GA, Hill Climbing (HC), Total Greedy Algorithm (TGA), 2- Optimal Greedy Algorithm (OGA), and Additional Greedy Algorithm (AGA) for TCP. Average Percentage of Requirement Coverage (APRC) metrics is used to compare the techniques. From the results, it is stated that AGA and OGA outperformed the rest of the techniques. [Bhasin 2014] used GA for TCP and gave a fitness function that accepts cost and order of test cases for the application under consideration. APFD metrics analyses the effectiveness of their approach. [Solanki et al. 2016] performed TCP by proposing a modified (m-ACO) technique. Quadratic equation and triangle classification problem were taken into consideration for TCP. Percentage of Test Suite Required and APFD metrics were used to determine the effectiveness of their technique. Their technique outperformed the other techniques. [Mittal and Sangwan 2015] performed TCP by using ABC optimization algorithm. Their technique was compared with CS and GA. APFD metric is used to compare the results of all three algorithms. ABC optimization algorithms outperformed the other techniques. CS gave the lowest performance of all. [Mishra et al. 2019] presented a TCP and optimization technique using GA by considering execution time, statement coverage, risk exposure, and requirements factors of test cases. The proposed method is implemented on the Income Tax Calculator case study. [Al-Hajjaji et al. 2019] proposed similarity-based TCP. Prioritization is done to enhance the probability of finding faults as early as possible. APFD metrics is used to compare their approach with random order, default order, and an interaction-based approach. Their approach outperformed the other approaches. [Mukherjee and Patnaik 2019] explored multi-objective GA. They also performed TCP by using SA and ACO. APFD metrics was used for the comparison of results. Their results reflected the superiority of NIMOA over other techniques. They stated that multi-objective GA outperformed single- objective GA for TCP. [Pradhan et al. 2019] proposed a search-based test case implantation approach (SBI) by using variants of GA. It comprises of two main components 1) Test case analyzer 2) Test case implanter. They used three metrics namely, mutation score, branch coverage, and statement coverage, to analyze the performance of their technique. Their results stated that SBI variants outperform the original test suite for two case studies. [Kaur and Bhatt 2011] performed TCP by hybridizing GA and PSO technique. Global solutions are constructed by traversing local solution with the aid of PSO. GA further produces the fittest population by analyzing the current population. [Gladstonte et al. 2016] performed TCP by using GA. Their technique used decision coverage, statement coverage, and block coverage. They proposed Immune Prioritization Algorithm (IPA). IPA outperformed GA for their study. Metrics for evaluation used in their study are APBC, APSC, and APDC. Various NIMOA technique used in the dominion of TCP are summarized in Table 1.

Table 1: Most Prevalent Metaheuristic Techniques used for TCP



Sr. No.	Author	Metaheuristic Technique Used	Metric/ Fitness Function Used
1	**[Ahmad et al. 2018] [2]**	GA, ACO	Time of Execution and Rate of Fault Detection
2	**[Al-Hajjaji et al. 2019] [3]**	Similarity-Based TCP	APFD
3	**[Bhasin 2014] [4]**	GA	APFD
4	**[Briand et al. 2002] [5]**	GA	-
5	**[Gladstonte et al. 2016] [9]**	GA, IGA, IPA	APSC, APDC, APBC
6	**[Jun et al. 2011] [16]**	GA	APBC
7	**[Kaur and Bhatt 2011] [17]**	GA, PSO	APFD
8	**[Konsaard and Ramingwond 2015] [18]**	BCO	-
9	**[Mann et al. 2018] [19]**	GA, ABC, PSO	-
10	**[Mishra et al. 2019] [20]**	GA	APSC
11	**[Mittal and Sangwan 2015] [21]**	ABC, CS, GA	APFD
12	**[Mittal and Sangwan 2018] [22]**	ABC, GA, CS	APBC, APFC, APSC and Minimum Execution Time
13	**[Mukherjee and Patnaik 2019] [23]**	GA, ACO, SA	APFD
14	**[Pachauri 2012] [24]**	GA, CA, QPSA	Branch Coverage
15	**[Panda and Sarangi 2013] [25]**	GA, ABC, DE	Cyclomatic Complexity
16	**[Pradhan et al. 2019] [26]**	GA	Mutation Score, Statement Coverage and Branch Coverage
17	**[Raamesh 2013] [28]**	GA, FS, BFA	APFD, APBC, APSC
18	**[Raju 2014] [29]**	GA	APFD
19	**[Reeves 1993] [30]**	GA	-
20	**[Sihan et al. 2010] [31]**	HC, GA, TGA, AGA, 2-OGA	APRC
21	**[Solanki et al. 2016] [32]**	m-ACO, ACO, GA, BCO	PTR, APFD
22	**[Wang et al. 2017] [33]**	SA, GA	APSC and APMC
23	**[Yadav and Dutta 2017] [34]**	GA	APSC

As per the literature review, metaheuristic techniques used for TCP, as shown in Table1 varies on the basis of the platforms used, different applications used with varying sizes, different metrics used to measure the effectiveness of proposed techniques, different criterions used to perform TCP. Therefore, as per the theoretical analysis, it can be stated that there is no metaheuristic technique that is efficacious for TCP in RT since no standards have been followed to measure them on a single platform. As per the literature review, it can be stated that GA is the most popular technique used by most researchers in TCP. In this study, twenty-three papers were analyzed, out of which twenty-one papers used the GA algorithm either in hybridized form or for the comparison of study. As reflected in Figure 1, 91% of the studies used GA, making it one of the most prevalent NIMOA for TCP

.

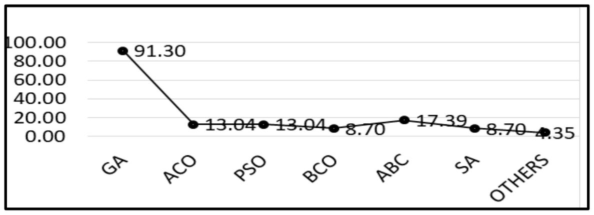


Figure 1: *% of Usage of Metaheuristic Techniques for TCP*

There are various NIMOA that have yet not been fully explored for TCP for e.g., FPA, Harmony Search Algorithm, Fish Schooling Algorithm, Firefly Algorithm, Bat Algorithm, etc. Therefore, it is stated that there is a need to propose an enhanced metaheuristic technique for TCP, keeping in mind that various NIMOA have not yet been explored for TCP as per our study. One of the recent additions to NIMOA namely, the Flower Pollination Algorithm (FPA), has not been used for TCP in RT in recent literature. An attempt has been made by [Dhareula and Ganpati 2019] to use FPA for TCP. In this study, FPA outperformed the traditional ways of TCP. The results have motivated the present study to propose an enhanced metaheuristic technique for TCP using FPA and GA. This study tries to bridge a gap between the most popular metaheuristic techniques used for TCP and recent addition made to NIMOA. Therefore, here, FPA and GA have been augmented to perform TCP in RT.

3. GENETICALLY MODIFIED FLOWER POLLINATION ALGORITHM (GM-FPA)

3.1 Genetic Algorithm for TCP

In the 1960s and 1970s, John Holland with his collaborators developed the GA. GA is based on the natural selection theory given by Charles Darwin's. GA constitutes genetic operators as an integral strategy for problem-solving. GA has two major advantages over traditional optimization techniques, firstly, it can handle complex problems and secondly, parallelism. GA can handle different types of optimization, being the fitness function with random noise, linear or nonlinear, stationary or nonstationary (changing with time), continuous or discontinuous parameters. Since various offspring in a population behaves as independent agents, therefore, the population can be explored in various directions leading to parallelism. Due to these advantages, GA is one of the most widely used algorithms for optimization. Crossover, mutation, and Elitism are the three primary operators in GA. Their roles vary differently. Crossover implies swapping a part of the solution from a chromosome with a part of a solution from another chromosome. This results in mixing of solutions and convergences of search space. Mutation on the other side randomly changes one solution from a given part of solutions, hence leading to the diversity of solutions and also gives a way of escaping from getting stuck at a local optimum. And lastly, Elitism refers to the mechanism where a solution having high fitness is passed on to the next higher generations, which is carried out by selecting the best solution. In this study, GA is used for TCP. Figure 2 below gives the pseudocode for GA as proposed by John Holland [Yang 2017].

```
Objective function f(x), x = (x_1, …, x_n)^T
Encode the solutions into chromosomes (strings)
Define fitness F (e.g., F ∝ f(x) for maximization)
Generate the initial population
Initialize the probabilities of crossover (p_c) and mutation (p_m)
    While (t< Max number of generations)
        Generate new solution by crossover and mutation
        Crossover with a crossover probability p_c
        Mutate with a mutation probability p_m
        Accept the new solution if their fitness increase
        Select the current best for the next generation
        Update t= t+1
    End while
Decode the results and visualization
```

Figure 2: Genetic Algorithm

GA for TCP is implemented in Java on the eclipse platform. Here 'n' represents the number of test cases. The constants used for GM- FPA, FPA, and GA are displayed in Table 2.

Table 2: *Values of Constants used for FPA and GA*

Algorithm	Values of Constants Used
FPA	n= no. of test cases, rand= value ranging between 0 and 1, g*= index of test case with highest code coverage, L=1, Ƴ =1, p (switch probability) = 0.8, ε = 0 or 1 representing either the test case is selected or not
GA	n= no of test cases, RCP= 0.9, RMP= 0.1
GM_FPA	n= no. of test cases, rand= value ranging between 0 and 1, g*= index of test case with highest code coverage, L=1, Ƴ =1, p (switch probability) = 0.8, ε = 0 or 1 representing either the test case is selected or not, RCP= 0.9, RMP= 0.1

GA takes the value of 'n' (no of test cases) as the input to produce the number of chromosomes (m) as the initial population. Each chromosome is created randomly from the original test suite. The uniqueness of all the created chromosome is maintained. All the chromosomes created by GA are displayed as output. Fitness value f(x) of the original test suite with n test cases (i.e., the total code coverage) is determined by Eclemma code coverage tool [13].

Code coverage of all the created chromosomes is also determined with the help of Eclemma tool. Calculate Crossover Probabilities (p_c) with the help of Eq. (1) for all the 'm' number of chromosomes.

$$p_c = \frac{Code\ coverage\ of\ i^{th}\ chromosome}{Code\ Coverage\ of\ original\ tets\ suite} \quad \dots \text{Eq. (1)}$$

If the value of p_c for a given chromosome is greater than RCP, the chromosome is saved, which means that if the chromosome gave code coverage more than 90% it is saved else crossover is performed with i^{th} and i^{th} +1 chromosome for e.g., Puzzle Game application [9] is taken into consideration with thirty-three test cases. In Figure 3, two chromosomes namely, Chromosome1 and Chromosome2, are created as the initial population for crossover. Single midpoint crossover is performed on both the chromosomes. Each chromosome is partitioned into two parts at the midpoint as shown in Figure 3.

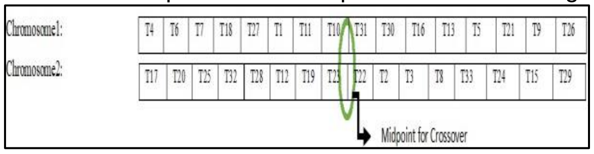

Chromosome1:
T4 T6 T7 T18 T27 T1 T11 T10 T31 T30 T16 T13 T5 T21 T9 T26
Chromosome2:
T17 T20 T25 T32 T28 T12 T19 T23 T22 T2 T3 T8 T33 T24 T15 T29
Midpoint for Crossover

Figure 3: *Chromosome 1 and Chromosome 2 with Midpoint for Crossover*

After crossover second half of chromosome1 is copied to the second half of chromosome2 and vice versa. After crossover, two offspring are produced named Child1 and Child2, as shown in Figure 4.

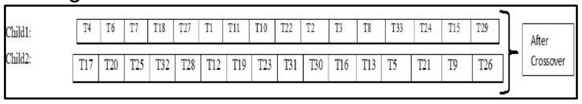
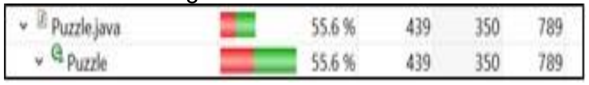

Child1:
T4 T6 T7 T18 T27 T1 T11 T10 T22 T2 T3 T8 T33 T24 T15 T29
Child2:
T17 T20 T25 T32 T28 T12 T19 T23 T31 T30 T16 T13 T5 T21 T9 T26
After Crossover

Figure 4: *Child 1 and Child 2 After Crossover*

All the duplicates are eliminated from both the offspring. For both the offspring, if coverage of any test case is less than RMP, then perform mutation else do nothing. While performing mutation, the test case with less than 0.1 % coverage will be swapped with the test case from the original test pool not present in child1 or child2, respectively. For mutation test case with highest coverage is picked from the original test pool. If this test case is already present, the next higher code coverage test case is selected by maintaining the uniqueness of test cases in both the offspring. After the mutation process again pc value of both the offspring is calculated by using Eq (1). If pc of child is more than RCP it is saved else if pc is less than RCP, it is again crossover with next i, till all 'm' have been covered. GA was performed on the thirty-three test cases for Puzzle Game. Since no test case in both the offspring had code coverage less than RMP; therefore, no mutation was performed. Both the offspring were exercised on the application under study to determine its code coverage. Both the offspring gave equal code coverage with 55.6% as shown in Figure 5.



Puzzle.java		55.6 %	439	350	789
Puzzle		55.6 %	439	350	789

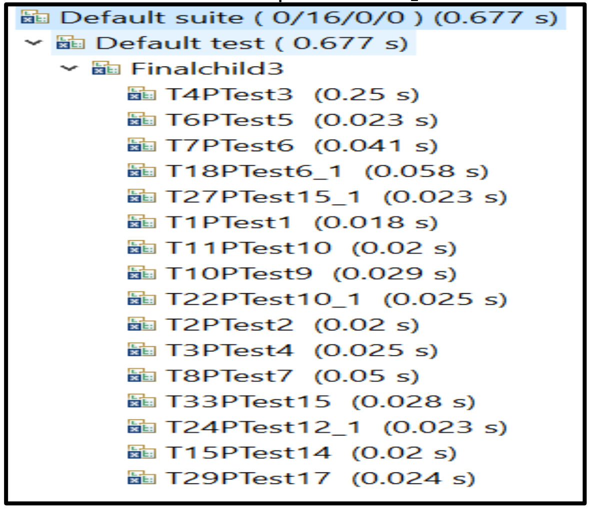
Figure 5: *Code Coverage for Child1*

Therefore, Child1 was randomly picked as prioritized chromosome (TS1) to be exercised on all the versions of the application. Child1 identified all the errors in five versions of Puzzle Game, as shown in Table 3 below.

Table 3: *Faults Determined by Child1 (TS_1) for Puzzle Game*

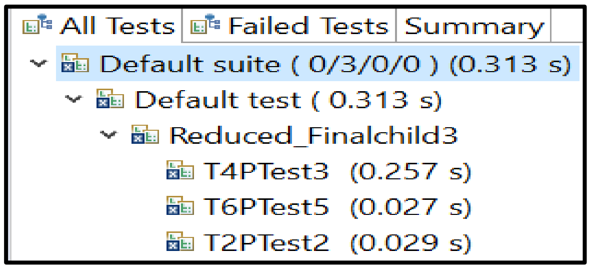
Versions/ Faults ↓ / Test Cases →	T4	T6	T7	T18	T27	T1	T11	T10	T22	T2	T3	T8	T33	T24	T15	T29
F1 (Version 1)	①		1	1		1	1		1			1				1
F2 (Version 2)										①				1	1	
F3 (Version 3)		①			1			1			1		1			
F4 (Version 4)		①			1			1		1	1		1	1	1	
F5 (Version 5)	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

As it is evident from Table 3 that test case T4 identifies faults F1 and F5, Test case T6 identifies faults F3 and F4, and test case T2 identifies fault F2. Therefore, it can be stated that the final prioritized set of test cases TS_p = {T4, T6, T2} respectively are complete to cover all the faults in the Puzzle Game application. In Figure 6 and Figure 7 it can be shown that TS_1 takes 0.677 seconds to cover all the faults and TS_p takes 0.313 seconds to cover the same set of faults which is almost 50% less as compared to TS_1.



```
Default suite ( 0/16/0/0 ) (0.677 s)
  Default test ( 0.677 s)
    Finalchild3
      T4PTest3 (0.25 s)
      T6PTest5 (0.023 s)
      T7PTest6 (0.041 s)
      T18PTest6_1 (0.058 s)
      T27PTest15_1 (0.023 s)
      T1PTest1 (0.018 s)
      T11PTest10 (0.02 s)
      T10PTest9 (0.029 s)
      T22PTest10_1 (0.025 s)
      T2PTest2 (0.02 s)
      T3PTest4 (0.025 s)
      T8PTest7 (0.05 s)
      T33PTest15 (0.028 s)
      T24PTest12_1 (0.023 s)
      T15PTest14 (0.02 s)
      T29PTest17 (0.024 s)
```

Figure 6*: Order and Time of Execution with TS_1 for Puzzle Game*



```
Default suite ( 0/3/0/0 ) (0.313 s)
  Default test ( 0.313 s)
    Reduced_Finalchild3
      T4PTest3 (0.257 s)
      T6PTest5 (0.027 s)
      T2PTest2 (0.029 s)
```

Figure 7*: Order and Time of Execution with TS_p for Puzzle Game*

APFD metrics, as shown in Eq (2) is used in this study to measure the effectiveness of GA for TCP. APFD results using GA of TS_p for Puzzle Game is computed below:

$$APFD = 1 - \frac{F_1+F_2+F_3+\ldots\ldots F_m}{nm} + \frac{1}{2n} \qquad \ldots\text{Eq. (2)}$$

$$APFD = 1 - \frac{4+2+6+6+4}{5*33} + \frac{1}{2*33}$$

$$APFD = 1 - \frac{22}{165} + \frac{1}{66}$$

$$APFD = 1 - 0.13333 + 0.01515$$

$$APFD = 1 - 0.14848 = 0.85152 \; in \; 0.313 \; seconds$$

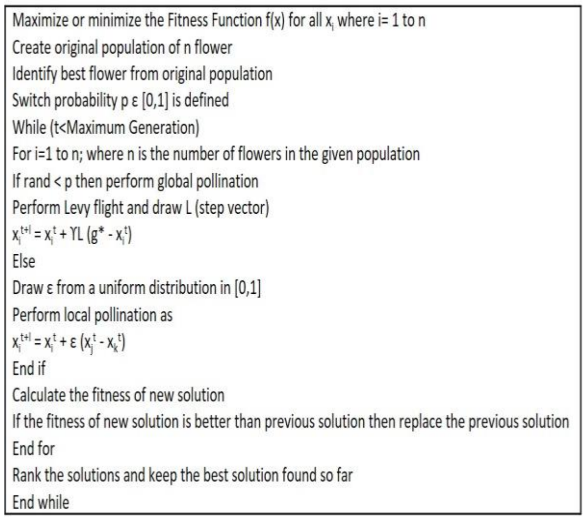
3.2 FLOWER POLLINATION ALGORITHM

FPA was contributed by Xin- She Yang in 2012, as shown in Figure 8. The four primary rules of flower pollination process [Yang 2014] are as follows.

1. Biotic and cross-pollination process is used to achieve global pollination; Levy flight is used by the vectors to transfer pollens. (Rule 1)
2. Abiotic and self- pollination is used to perform local pollination. (Rule 2).
3. Flower Constancy (FC) is the affinity between the flower and the vector, which acts as the reproduction probability. (Rule 3).
4. Local and global pollination is achieved through switch variable p ε [0,1].

All NIMOA performs local search by exploitation of the search space that may lead to local optimum and may get stuck at a point in search space. Global search is performed by exploration of the search space leading to the diversity of

solutions. FPA belongs to the newest addition made to NIMOA [Yang 2014]. Pollination in flowers is mainly achieved by the transfer of pollens. Pollinators perform the transfer of pollens. Various pollinators are known to perform pollination viz., insects, birds, bats, etc.
Pollination is achieved through two means firstly, abiotic, where transfer of pollens takes place through water and wind which is only 10% in nature. Secondly, biotic, in which vectors transfer the pollens, nearly 90% transfer takes place biotically. FC is seen in various vectors and particular species of flower. FC results in the reproduction of specific species of flower, which can be looked at as the survival of the fittest. As per literature, there is no study that performs TCP using FPA. [Dhareula and Ganpati 2019] used FPA for TCP, this study is further extended here to propose an enhanced metaheuristic approach for TCP.



```
Maximize or minimize the Fitness Function f(x) for all x_i where i= 1 to n
Create original population of n flower
Identify best flower from original population
Switch probability p ε [0,1] is defined
While (t<Maximum Generation)
For i=1 to n; where n is the number of flowers in the given population
If rand < p then perform global pollination
Perform Levy flight and draw L (step vector)
$x_i^{t+1} = x_i^t + YL (g^* - x_i^t)$
Else
Draw ε from a uniform distribution in [0,1]
Perform local pollination as
$x_i^{t+1} = x_i^t + ε (x_j^t - x_k^t)$
End if
Calculate the fitness of new solution
If the fitness of new solution is better than previous solution then replace the previous solution
End for
Rank the solutions and keep the best solution found so far
End while
```

***Figure 8**: Flower Pollination Algorithm (FPA)*

3.3 GENETICALLY MODIFIED FLOWER POLLINATION ALGORITHM (GM-FPA)

The main objective of this study is to propose an enhanced metaheuristic approach for TCP in RT. Figure 9 shows the enhanced Genetically Modified- Flower Pollination Algorithm (GM-FPA) pseudocode for TCP. In this section, GM-FPA is proposed for TCP in RT. FPA and GA belong to the NIMOA. In the proposed technique, the output of FPA is supplied as input to the GA. The output of FPA serves as chromosome1 and is given as input to the GA. GA further produces Chromosome2. GA creates chromosome2 from the original test suite with an equal number of test cases as in chromosome1. Chromosome 2 will contain unique test cases not present in chromosome1. Crossover is performed between chromosome1 and chromosome2. After crossover, two offspring are produced namely, child1 and child2. For both the children, if coverage of any test case is less than 0.1%, then perform mutation.

```
**//Flower Pollination Algorithm (FPA) begins**
Set the Fitness Function f(x) for all $x_i$, where i= 1 to n
Set the original population of n test cases as random solutions
Set g* as test case with maximum coverage
Set a switch variable p= 0.8
        **While** (t< stopping criteria)
            **For** i= 1: n (all test cases)
                **If** rand<p then perform global pollination
                Perform Levy flight and draw L (step vector)
                $x_i^{t+1} = x_i^t + \gamma L(g^* - x_i^t)$
                **Else**
                Draw ε from a uniform distribution in [0,1]
                Perform local pollination as
                $x_i^{t+1} = x_i^t + \varepsilon(x_j^t - x_k^t)$
                **End if**
            Calculate the fitness of new solution
                **If** the fitness of new solution is better than previous solution
                Then replace the solution
            **End for**
        Rank the solution and keep the best solution found so far
        **End while**
// **FPA end**. Output of FPA will be considered as Chromosome1 and it will be given as input to Genetic Algorithm (GA)
**//Genetic Algorithm GA begins**
Objective function f(x), x = $(x_1, …, x_d)^T$
Encode the solutions into chromosomes (strings)
Define fitness F (e.g., F ∝ f(x) for maximization)
Generate the initial population
Initialize the probabilities of crossover ($p_c$) and mutation ($p_m$)
    **While** (t< Max number of generations)
    Generate new solution by crossover and mutation
    Crossover with a crossover probability $p_c$
    Mutate with a mutation probability $p_m$
    Accept the new solution if their fitness increase
    Select the current best for the next generation (elitism)
    Update t= t+1
    **End while**
Decode the results and visualization
```

***Figure 9:** Genetically Modified-Flower Pollination Algorithm*

For mutation, pick a test case with maximum coverage in the original test suite and swap it with the test case in the respective offspring. If a mutated test case is already present in the offspring, pick another highest code coverage test case from the original test suite and maintain uniqueness by removing redundant test cases. Out of the two offspring, pick the one with the highest code coverage for TCP for respective applications. At last, elitism is performed. From the original test suite, test case with maximum coverage is added to all the offspring. Eliminate redundant test cases from all the saved offspring. From the saved set of offspring, the chromosome with the maximum coverage is selected for further TCP. FPA was augmented with the feature of GA, since it is believed that "survival of the fittest" offers a decent mechanism to choose the finest solution; elitism promises that the finest solution will persist in the population, which will boost the convergence rate of the algorithm.

4. IMPLEMENTATION

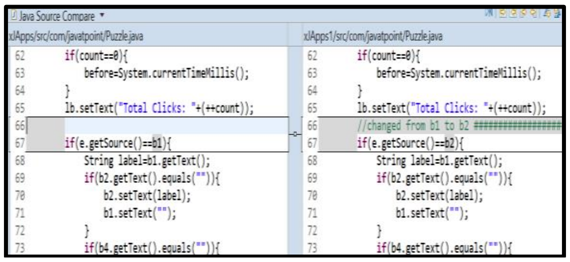
GM-FPA for TCP in RT is implemented in java using eclipse IDE. Three applications are written in java namely, Tritype, Puzzle Game, and AreaandPerimeter are used for the validation of results. The test scripts for all the applications are designed using TestNG tool. TestNG tool facilitates the prioritization of test cases and the execution in the order of their priority. Code coverage of the test cases is acquired through EclEmma tool. Details of the applications with their version count, number of test cases, Line of Code (LOC) is tabulated in Table 4 below.

***Table 4**: Details of Applications Used*

Sr. No.	Name of the Applications	Size in LOC	Number of Versions/ Unique Faults	No. of Test Cases Used
1.	Puzzle Game	246	5	33
2.	AreaandPerimeter	916	8	113
3.	Tritype	106	6	45

5. RESULTS

Three applications are considered for this study, but for the explanation, only one application Puzzle Game will be used. For the Puzzle Game, there are thirty-three test cases in the original test suite (TS_o), namely (T1, T2, …, T33). Five versions of the application have been created. Each version of Puzzle Game is induced with a unique fault. The faults induced are typographical or logical types. Figure 10 shows the typographical fault induced at line number 67 for version 1 of the Puzzle Game.



xlApps/src/com/javatpoint/Puzzle.java

```
62      if(count==0){
63          before=System.currentTimeMillis();
64      }
65      lb.setText("Total Clicks: "+(++count));
66
67      if(e.getSource()==b1){
68          String label=b1.getText();
69          if(b2.getText().equals("")){
70              b2.setText(label);
71              b1.setText("");
72          }
73          if(b4.getText().equals("")){
```

xlApps1/src/com/javatpoint/Puzzle.java

```
62      if(count==0){
63          before=System.currentTimeMillis();
64      }
65      lb.setText("Total Clicks: "+(++count));
66      //changed from b1 to b2 #################
67      if(e.getSource()==b2){
68          String label=b1.getText();
69          if(b2.getText().equals("")){
70              b2.setText(label);
71              b1.setText("");
72          }
73          if(b4.getText().equals("")){
```

***Figure 10:** Induced Fault at Line No. 67 if Version 1 for Puzzle Game*

For the five versions, the faults have been termed as F1, F2, …, F5. For the Puzzle Game application, thirty-three test cases are given as input to the proposed GM-FPA. Two chromosomes are produced by the approach. One chromosome namely, Chromosome1 is produced as the output given by FPA and the second chromosome, Chromosome2, is generated by the GA. Since FPA produced Chromosome1 with eight test cases, so Chromosome2 generated by GA will also contain eight test case selected randomly and unique in comparison to Chromosome1, as shown in Figure 11.

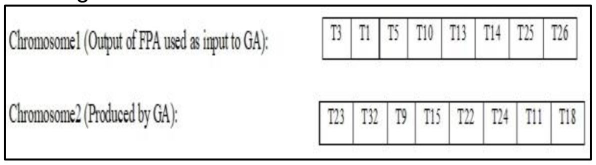

Chromosome1 (Output of FPA used as input to GA):
T3 T1 T5 T10 T13 T14 T25 T26
Chromosome2 (Produced by GA):
T23 T32 T9 T15 T22 T24 T11 T18

***Figure 11:** Chromosome 1 and Chromosome 2 produced by GM-FPA*

Chromosome1 and Chromosome2 are crossed over at a single midpoint as shown in Figure 12.

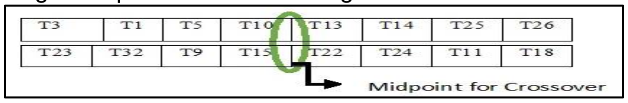


***Figure 12:** Midpoint of Crossover for Chromosome 1 and Chromosome 2*

During crossover second half of chromosome1 is copied to the second half of chromosome2 and vice versa. After crossover two offspring are produced named Child1 and Child2 as shown in Figure 13.

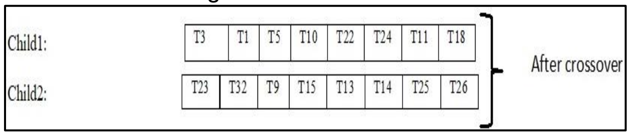


***Figure 13:** Child1 and Child2 produced by GM-FPA after Crossover*

Both the offspring are exercised upon the application under consideration to get the code coverage detail. Child1 resulted in 55.6% and child2 resulted in 52.6% code coverage for Puzzle Game. On the basis of the code coverage, child1 was selected for further prioritization of the test cases. Since all the test cases in child1 have code coverage of more than 0.1% therefore, no mutation was performed for child1. As per elitism best traits of the ancestors are passed on to the higher generation. Elitism is performed on child1 since, as per history test case, T2 identified maximum errors. Therefore, test case T2 is added to child1, so that now it contains nine test cases as shown in Figure 14. Child1 (TS_1) is exercised on all the versions of the Puzzle Game to analyze the number of faults detected by TS1.

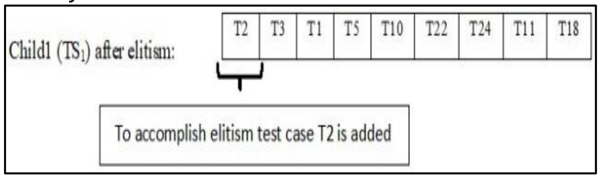
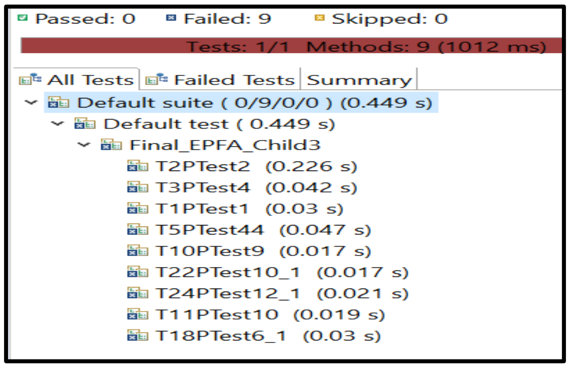

Child1 (TS1) after elitism:
T2 T3 T1 T5 T10 T22 T24 T11 T18
To accomplish elitism test case T2 is added

***Figure 14:** Child1 after Elitism*

Figure 15 reflected the order of execution of TS_1 in 0.449 seconds.



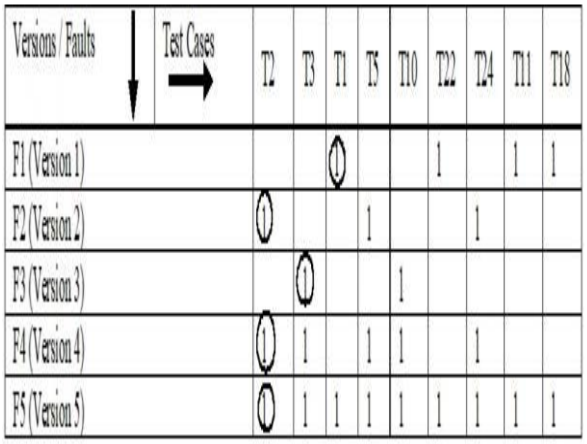
```
Passed: 0    Failed: 9    Skipped: 0
Tests: 1/1   Methods: 9 (1012 ms)

Default suite ( 0/9/0/0 ) (0.449 s)
  Default test ( 0.449 s)
    Final_EPFA_Child3
      T2PTest2  (0.226 s)
      T3PTest4  (0.042 s)
      T1PTest1  (0.03 s)
      T5PTest44  (0.047 s)
      T10PTest9  (0.017 s)
      T22PTest10_1  (0.017 s)
      T24PTest12_1  (0.021 s)
      T11PTest10  (0.019 s)
      T18PTest6_1  (0.03 s)
```

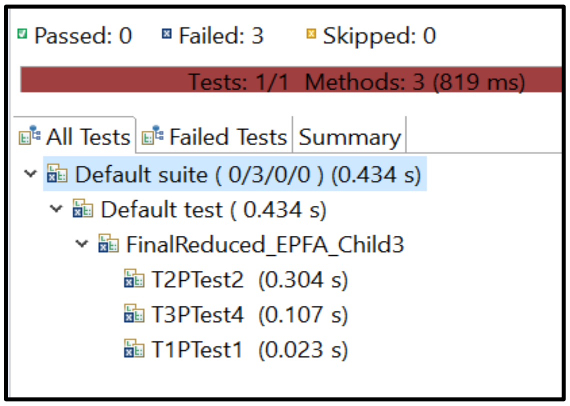
***Figure 15:** Order of Execution of Child1 (TS_1) for Puzzle Game*

As it is evident from Table 5 that, test case T2 identifies fault F2, F4, and F5, test case T3 identifies faults F3 and test case T1 identifies faults F1. Only higher-order three test cases are required to cover all the faults. Therefore, TS_1 is reduced to prioritized test suite (TS_p) = {T2, T3 and T1}. Figure 16 shows the prioritized order (TS_p) of test cases for Puzzle Game in 0.434 seconds, which is comparatively lower than TS_1.

***Table 5:** Faults Identified by TS_1 for Puzzle Game by GM-FPA*



Versions / Faults ↓	Test Cases →	T2	T3	T1	T5	T10	T22	T24	T11	T18
F1 (Version 1)				①			1		1	1
F2 (Version 2)		①			1			1		
F3 (Version 3)			①			1				
F4 (Version 4)		①	1		1	1		1		
F5 (Version 5)		①	1	1	1	1	1	1	1	1



Passed: 0 Failed: 3 Skipped: 0

Tests: 1/1 Methods: 3 (819 ms)

All Tests | Failed Tests | Summary

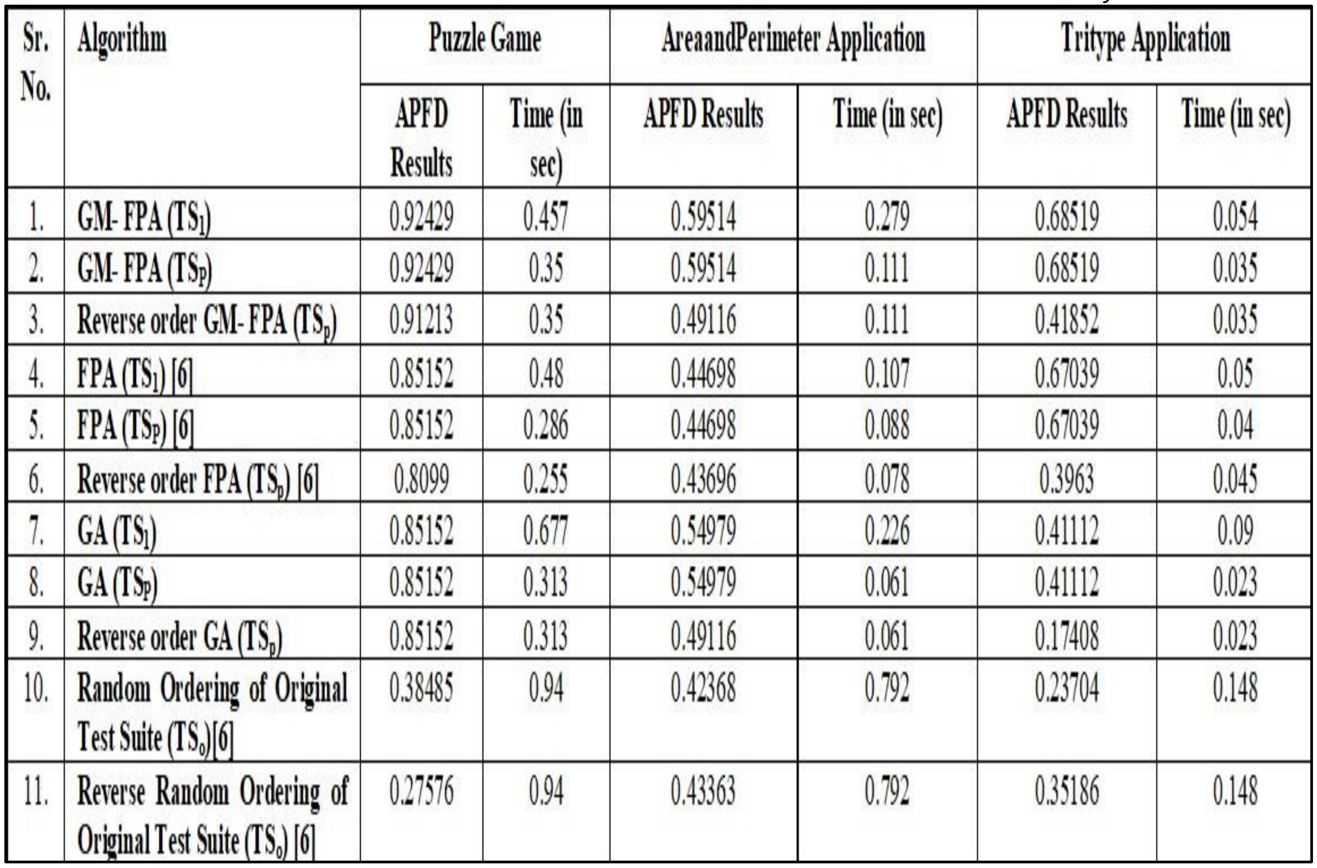
```
Default suite ( 0/3/0/0 ) (0.434 s)
  Default test ( 0.434 s)
    FinalReduced_EPFA_Child3
      T2PTest2  (0.304 s)
      T3PTest4  (0.107 s)
      T1PTest1  (0.023 s)
```

***Figure 16:** Order of Execution for Child1 (TS_p) for Puzzle Game*

Using the information from Table 5, APFD value was computed for TS_p of Puzzle Game.

$$APFD = 1 - \frac{F_1 + F_2 + F_3 + \ldots..F_m}{nm} + \frac{1}{2n}$$
$$APFD = 1 - \frac{1+2+3+2+2}{5*33} + \frac{1}{2*33}$$
$$APFD = 1 - \frac{10}{165} + \frac{1}{66}$$
$$APFD = 1 - 0.06060 + 0.01515$$
$$APFD = 1 - 0.075751 = 0.92429 \text{ in 0.434 seconds}$$

***Table 6:** APFD Results and Time of Execution for NIMOA and Traditional way of TCP*



Sr. No.	Algorithm	Puzzle Game		AreaandPerimeter Application		Tritype Application	
		APFD Results	Time (in sec)	APFD Results	Time (in sec)	APFD Results	Time (in sec)
1.	GM- FPA (TS_1)	0.92429	0.457	0.59514	0.279	0.68519	0.054
2.	GM- FPA (TS_P)	0.92429	0.35	0.59514	0.111	0.68519	0.035
3.	Reverse order GM- FPA (TS_p)	0.91213	0.35	0.49116	0.111	0.41852	0.035
4.	FPA (TS_1) [6]	0.85152	0.48	0.44698	0.107	0.67039	0.05
5.	FPA (TS_P) [6]	0.85152	0.286	0.44698	0.088	0.67039	0.04
6.	Reverse order FPA (TS_p) [6]	0.8099	0.255	0.43696	0.078	0.3963	0.045
7.	GA (TS_1)	0.85152	0.677	0.54979	0.226	0.41112	0.09
8.	GA (TS_P)	0.85152	0.313	0.54979	0.061	0.41112	0.023
9.	Reverse order GA (TS_p)	0.85152	0.313	0.49116	0.061	0.17408	0.023
10.	Random Ordering of Original Test Suite (TS_o)[6]	0.38485	0.94	0.42368	0.792	0.23704	0.148
11.	Reverse Random Ordering of Original Test Suite (TS_o) [6]	0.27576	0.94	0.43363	0.792	0.35186	0.148

As is evident from Table 6, the APFD values for GM- FPA, FPA, and GA are higher than the Random Ordering of TS_o and Reverse Random Ordering of TS_o. Therefore, it can be stated that NIMOA performs better for TCP as compared to the traditional approach of TCP. From GM- FPA's prioritized test suite TS_1 only higher-order test cases were selected to represent TS_p. TS_p exercised complete fault coverage with the same value of APFD in reduced time of execution as shown in Table 6. It is also evident from the results in Table 6 that GM_ FPA outperformed FPA and GA for TCP. Therefore, it can be stated that augmentation of FPA and GA elevates the APFD values for all the three applications used in this study as compared to the APFD results of GA, FPA [Dhareula and Ganpati 2019] and traditional approaches.

6. CONCLUSION AND FUTURE SCOPE

The study has made a theoretical attempt to find the most efficacious technique for TCP. The theoretical results have reflected that GA is the most prevalent NIMOA used for TCP, although a particular NIMOA cannot be considered as efficacious for TCP. However, the main objective of this study is to propose an enhanced metaheuristic technique for TCP in RT by the amalgamation of FPA and GA. The study presented Genetically Modified-Flower Pollination Algorithm (GM-FPA) technique for TCP in RT. GM-FPA was implemented in Java on the eclipse platform. Three applications with different versions written in Java were used to empirically evaluate the performance of the proposed technique. Various versions of all the applications were used for the validation of the results. GM- FPA initially generated prioritized Test Suite (TS_1) from which higher-order test cases were executed to covered maximum faults and the test suite is termed as (TS_p). The APFD results reflected that TS_p gave same APFD results as TS_1 but in reduced time of execution. The present study also used GA for TCP. GA and FPA also produced TS_1 and TS_p for all the three applications. It is stated in this study that TS_p gave better results for all the three metaheuristic algorithms as compared to their respective TS_1. APFD metrics and time of execution of the test suite is used to compare the performance of GM- FPA with FPA, GA, Random Ordering of Original Test Suite (TS_o) and Reverse Random Ordering of TS_o. From the empirical results it is evident that NIMOA performed much better than the traditional ways of TCP and also GM- FPA outperformed the FPA and GA. There is a further need to evaluate the results of the proposed technique by comparing it with other NIMOA and on large size applications.

REFERENCES

[1] AGGARWAL, K. K. and SINGH, Y. 2007. Software Engineering (3rd ed.), © *New Age International Publishers*.

[2] AHMAD, S.F., SINGH, D.K. and SUMAN, P. 2018. Prioritization for Regression Testing Using Ant Colony Optimization Based on Test Factors. *In Intelligent Communication, Control and Devices*, Springer, Singapore, 1353-1360.

[3] AL-HAJJAJI, M., THÜM, T., LOCHAU, M., MEINICKE, J. and SAAKE, G. 2019. Effective Product-Line Testing Using Similarity-Based Product Prioritization. *Software & Systems Modeling*, 18(1), 499-521.

[4] BHASIN, H. 2014. Cost-priority Cognizant Regression Testing. *ACM SIGSOFT Software Engineering Notes*, 39(3), 1-7.

[5] BRIAND, L.C., FENG, J. and LABICHE, Y. 2002. Using Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders. *In Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, ACM, 43-50.

[6] DHAREULA, P. AND GANPATI, A. (In press). Flower Pollination Algorithm for Test Case Prioritization in Regression Testing. *In 4th International Conference on Information and Communication Technology for Sustainable Development* (ICT4SD 2019) on July 5th -6th, 2019, Springer, Goa, India.

[7] DHAREULA, P. AND GANPATI, A. 2015. Prevalent Criteria's in Regression Test case Selection Techniques: An exploratory study. *In International Conference on Green Computing and Internet of* Things, IEEE, 871-876.

[8] DHAREULA, P. AND GANPATI, A. 2016. Identification of Attributes for Test Case Reusability in Regression Test Selection Techniques. *In 3rd International Conference on Computing for Sustainable Global Development, IEEE,* 1144-1147.

[9] GLADSTON, A., NEHEMIAH, K., NARAYANASAMY, P. and KANNAN, A. 2016. Test Case Prioritization for Regression Testing Using Immune Operator. *The International Arab Journal of Information Technology*, 13(6), 686-692.

[10] https://github.com/jkriti/JApps/blob/master/JApps.java

[11] https://sir.csc.ncsu.edu/portal/bios/trityp.php

[12] https://stackoverflow.com/questions/4479624/area-and-perimeter-calculation-of-various-shapes?rq=1

[13] https://testng.org/doc/download.html

[14] https://www.eclemma.org/download.html

[15] https://www.eclipse.org/downloads/packages/release/oxygen/3a

[16] JUN, W., YAN, Z. and CHEN, J. 2011. Test Case Prioritization Technique Based on Genetic Algorithm. *In Internet Computing & Information Services, International Conference*, IEEE, 173-175.

[17] KAUR, A. and BHATT, D. 2011. Hybrid Particle Swarm Optimization for Regression Testing. *International Journal on Computer Science and Engineering*, 3(5), 1815-1824.

[18] KONSAARD, P. and RAMINGWONG, L. 2015. Total Coverage Based Regression Test Case Prioritization Using Genetic Algorithm. *In Electrical engineering/electronics, computer, telecommunications and information technology*, IEEE, 1-6.

[19] MANN, M., TOMAR, P. and SANGWAN, O.P. 2018. Bio-Inspired Metaheuristics: Evolving and Prioritizing Software Test Data. *Applied Intelligence*, 48(3), 687-702.

[20] MISHRA, D.B., MISHRA, R., ACHARYA, A.A. and DAS, K.N. 2019. Test Case Optimization and Prioritization Based on Multi-objective Genetic Algorithm. *In Harmony Search and Nature Inspired Optimization Algorithms*, Springer, Singapore, 371-381.

[21] MITTAL, S. and SANGWAN, O.P. 2015. Metaheuristic Based Approach to Regression Testing. *International Journal of Computer Science and Information Technologies*, 6 (3), 2597-2605.

[22] MITTAL, S. and SANGWAN, O.P. 2018. Prioritizing Test Cases for Regression Techniques Using Metaheuristic Techniques. *Journal of Information and Optimization Sciences*, 39(1), 39-51.

[23] MUKHERJEE, R. and PATNAIK, K.S. 2019. Prioritizing JUnit Test Cases Without Coverage Information: An Optimization Heuristics Based Approach. *IEEE Access*, 7, 78092-107.

[24] PACHAURI, A. 2012. Software Test Data Generation Using Metaheuristic Search Based Techniques. *http://shodhganga.inflibnet.ac.in/handle/10603/22181*.

[25] PANDA, M. and SARANGI, P.P. 2013. Performance Analysis of Test Data Generation for Path Coverage Based Testing Using Three Meta-Heuristic Algorithms. *Int. J. Comput. Sci. Inf*, 3(2), 34-41.
[26] PRADHAN, D., WANG, S., YUE, T., ALI, S. and LIAAEN, M. 2019. Search-Based Test Case Implantation for Testing Untested Configurations. *Information and Software Technology*, 111, 22-36.
[27] PRESSMAN, R.S. 2005. Software Engineering: A Practitioner's Approach. *Palgrave Macmillan*.
[28] RAAMESH, L. 2013. Integrated Multi Objective Test Case Prioritization System. *http://ir.inflibnet.ac.in:8080/jspui/handle/10603/38627*.
[29] RAJU, S. 2014. Multi Factor Approach for Effective Regression Testing Using Test Case Optimization. *http://ir.inflibnet.ac.in:8080/jspui/handle/10603/38627.*
[30] REEVES, C.R. 1993. Modern Heuristic Techniques for Combinatorial Problems. *John Wiley & Sons, Inc*.
[31] SIHAN, L.I., BIAN, N., CHEN, Z., YOU, D. and HE, Y. 2010. A Simulation Study on Some Search Algorithms for Regression Test Case Prioritization. *In Quality Software International Conference*, IEEE, 72-81.
[32] SOLANKI, K., SINGH, Y. and DALAL, S. 2016. Experimental Analysis of m-ACO Technique for Regression Testing. *Indian Journal of Science and Technology*, 9(30), 1-7.
[33] WANG, Z., ZHAO, X., ZOU, Y., YU, X. and WANG, Z. 2017. Improved Annealing-Genetic Algorithm for Test Case Prioritization. *Computing and Informatics*. 36(3), 705-32.
[34] YADAV, D.K. and DUTTA, S. 2017. Regression Test Case Prioritization Technique Using Genetic Algorithm. *In Advances in Computational Intelligence,* Springer, Singapore, 133-140.
[35] YANG, X.S. 2014. Nature-Inspired Optimization Algorithms. *Elsevier*.