Contents lists available at Science-Gate

International Journal of Advanced and Applied Sciences

Journal homepage: http://www.science-gate.com/IJAAS.html

Test case prioritization techniques in software regression testing: An overview

Muhammad Qasim [1], Asifa Bibi [2], Syed Jawad Hussain [3], N. Z. Jhanjhi [4], Mamoona Humayun [5], Najm Us Sama [6, *]

[1]*Department of Computer Science, Shaheed Zulfikar Ali Bhutto Institute of Science and Technology, Karachi, Pakistan*
[2]*Department of Computer Science, University of Northern Virginia, Annandale, USA*
[3]*Department of Computer Science, Barani Institute of Information Technology, Pir Mehr Ali Shah Arid Agriculture University, Rawalpindi, Pakistan*
[4]*School of Computer Science and Engineering, Taylor's University, Subang Jaya, Malaysia*
[5]*Department of Information Systems, College of Computer and Information Sciences, Jouf University, Sakakah, Saudi Arabia*
[6]*Department of Computer Science, Deanship of Common First Year, Jouf University, Sakaka, Saudi Arabia*

A R T I C L E I N F O

Article history:
Received 20 October 2020
Received in revised form
24 January 2021
Accepted 5 February 2021

Keywords:
Software development life cycle
Regression testing
Software testing
Test case prioritization

A B S T R A C T

The importance of Software Testing (ST) in the Software Development Life Cycle (SDLC) can never be ignored. Smarter ST can give us more relievable and defect-free products which are as per our stakeholder demand. That is the reason it takes more time and resources. But unfortunately, the limited time has left when the product reached the testing stage, especially in Regression Testing (RT). That is the reason proper planning is required in each SDLC phase, especially in the testing phase. Regression testing is an essential part of the testing phase. It guarantees defect-free software after any changes made to the requirement or software product. Because of the limited time available, it is impossible to execute all the test cases every time any changes made in the code there comes the role of the test case prioritization. TCP chooses only test cases that are most important to execute. The priority of test cases could be based on code, requirement, defects, execution time, cost, etc. After TCP the developer and tester have a minimum test case suite with better coverage in all respects. TCP definitely improves the quality of software and brings the best product within a limited time and cost. In this paper, we have provided a detailed survey of the TCP methodologies. This survey includes 2010 to the most recent studies. In this study, we have select carefully the most recent and relevant study of our topic.

1. Introduction

Software Testing (ST) plays an important role in the SDLC as it guarantees defect-free software and customer satisfaction. Despite the complexity and importance of this phase when it comes to testing a small amount of time has been left behind which makes this phase costlier. ST techniques involve the execution of software to find if all the components are as per customer need. If any issue is found, it will be considered a defect or bug.

There are lots of stages of ST and lots of ST techniques are used. Among all the ST techniques the most important one is Regression Testing (RT). The importance of RT could never be ignored as this verifies that newly updated code impacting previously build functionality or not? It doesn't matter how many times we make changes in the code we have to run RT to make sure software stability against the code change.

Because of the limited time given for RT, it becomes the difficult phase in the sense of how and which test case to be selected for RT suits. Different methodologies have been used to make the best test suite for regression testing which is called Test Case Prioritization (TCP) or Test Case Selection (TCS). These methodologies include search, coverage, fault, requirement, history, risk and cost-effective based, etc. (Shah et al., 2019). Each of the methodologies has its own algorithms. Best TCP/TCS is dependent upon different criteria like the selection of dataset, incomplete data extraction (Shah et al., 2016). In this study, we have selected the most recent algorithms

* Corresponding Author.
Email Address: najmussama@gmail.com (N. U. Sama)
https://doi.org/10.21833/ijaas.2021.05.012
Corresponding author's ORCID profile:
https://orcid.org/0000-0002-0093-6461
2313-626X/© 2021 The Authors. Published by IASE.

used for software TCP from the most popular repositories like IEEE Access, Elsevier, and Springer.

In this survey, we have studied 40 articles from 2010 to 2020 on TCP. This study organized as follows. In Section 2 which is a literature review, we will describe all the chosen algorithms in Table 1. In Section 3 we will compare the results generated by using each algorithm. In Section 4 we will describe research methodologies and try to answer the research questions. Section 5 contains state-of-the-art challenges and Section 6 contains conclusions and future research topics.

Table 1: TCP algorithms

S.No.	Name of Algorithm	
1	Firefly Algorithm	Firefly Algorithm Optimal FA Hybrid FA
2	Genetic Algorithm (GA)	Hypervolume GA
3	Ant Colony Optimization (ACO) Algorithm	ACO Modified ACO ETS ACO Hybrid ACO
4	Local Beam Search (LBS)	LBS
5	Particle Swarm Optimization (PSO)	PSO Weight Hybrid PSO Greedy PSO
6	Greedy Algorithm (GA)	Additional GA Enhanced GA Graphite GA

2. Literature review

Test Case Prioritization (TCP) not only saves time and resources but also stakeholder confidence in us if we used it properly. Various methods and techniques have been used in this regard to obtain the best optimal test suite for Regression Testing. In this study, we will cover and summarize all those algorithms as shown in Fig. 1.

3. Firefly algorithm

Firefly Algorithm (FA) after getting inspired by the flashy behavior of the fireflies (Shah et al., 2019).

- Fireflies are attracted to each other because all of them are unisexual.
- The attraction of fireflies is depending upon their brightness. Both are directly proportional to each other
- Fireflies only move randomly if the same/no brightness is found nearby.

Quite a few algorithms were proposed using the Firefly technique.

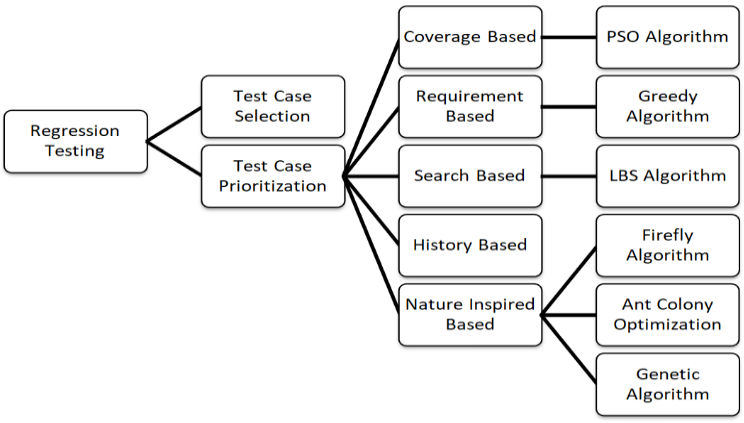

Fig. 1: Test case prioritization techniques

3.1. Optimal firefly algorithm

This algorithm uses a metaheuristic firefly algorithm to generate an optimal path. In this modified firefly algorithm, he used a new objective function/brightness function and guidance matrix to traverse the graph (Shah et al., 2016).

The process starts with creating Control flow graphs (CFG) and Data flow diagrams (DFD) and generating XML files from the CFG diagram. This XML file goes as input into Optimal Firefly Test Sequence Generator (OFTSG). OFTSG perform the following steps (Shah et al., 2016):

- Calculate nodes and edges
- Calculate Cyclomatic Complexity
- Create Adjacency matrix which is the edge between the nodes is considered as 1 everything else is 0.
- Create Guidance matrix using Eq. 1
- Path traversal: Algorithm generates best path sequences.
- Path Prioritization
- Calculation of brightness value at every node using the Eq. 2
- Calculation of mean value of brightness.

Higher the mean value of brightness will have higher priority.

$$GF = 10 * \left[CC_i * \left((N - i) - 0.1\right)\right] \quad (1)$$

$$f(x) = \left[\frac{1000}{(CC_i * rand())}\right] \quad (2)$$

The author tested this algorithm using 4 different programs as shown in Table 2.

This shows that as the number of states increases the cyclomatic complexity increases which causes an increase in the percentage of redundancy in path coverage. Results of this study were compared with the state of the art algorithm Ant Colony Optimization Algorithm (ACO). This shows FA generates better paths with reducing or no redundancy as compared to ACO as shown in Table 3.

Following limitations were found during this study, first, the proposed algorithm was not tested on any benchmarks program with a big test case pool. Secondly, it's not clear that which testing criteria are used for example statement coverage, fault coverage, etc.

Panthi and Mohapatra (2015) used the same methodology to minimized test cases by exploring the state diagrams. Nor Hashim and Dawood (2018) proposed a similar approach to minimized test cases and the process has been analyzed using a UML statechart diagram. This study also shows prominent results in the form of minimizing the number of test cases.

Both used ATM case studies to test their approaches which provide satisfactory results.

3.2. Firefly algorithm

Khatibsyarbini et al. (2019) used Firefly Algorithm with string metrics as a fitness function to find the best TCP arrangement.

Steps of Firefly Algorithm:

- Extraction of test cases from dataset
- Calculating distance and weight of test cases
- Perform movement update based on highest values of weight/distance
- Selection of test cases based on shortest distance as an optimal sequence.

Two variables Term Distance Inverse Document Frequency (TDIDF) and Term Frequency Inverse Document Frequency (TFIDF) were used to store distance among the test case and frequency (weight) of the test case respectively. The weight of test cases could be calculated as follows given in Eq. 3 (Khatibsyarbini et al., 2019).

$$TFIDF = \frac{t}{T} \times \log N = nt \tag{3}$$

The next move decided by the highest value of weight over distance and the shortest distance among test cases was selected as the optimum test case sequence. FA uses average percentage fault detection rate (APFD) to calculate the rate of fault detection which could be calculated using the formula given in Eq. 4 (Khatibsyarbini et al., 2019).

$$APFD = 1 - \frac{TF_1 + TF_2 + \ldots + TF_n}{n \times m} + \frac{1}{2n} \tag{4}$$

Six Software-artifact Infrastructure Repository (SIR) programs were used for testing. Three of them are UNIX-based programs (flex, grep, gzip) and three are SIEMENs programs (tcas, cs-tcas, j-cas) as shown in Table 2. Comparisons were made in this approach with state-of-the-art algorithms PSO, LBS, Greedy, and GA and among all of them, FA shows prominent results although LBS has quite better results at some places as shown in Table 3. Following limitations were observed during this study. The first one is a selection of datasets. The dataset used for testing this approach is very small SIR benchmark programs. This approach may give different results when applied to a bigger size of test case spools.

The second is less data utilization. For example, only distance and weight of test cases were used as input while we have a lot of other data available like system coding, etc. that could give totally different results. The third and final one is that the only TCP techniques were considered on the other hand we have a lot of other similar techniques available which was not included in validation.

3.3. Hybrid firefly algorithm (HFA)

Su et al. (2020) proposed this hybrid model of the firefly algorithm, main purpose is to prioritize test cases and reduce the test cost. HFA adopted a hybrid model in which first it correlates all the available test data and test cases and secondly applies enhanced FA on this hybrid model. The new fitness function of this HFA is shown in Eq. 5.

$$f(x_{i,j}) = max_k \left\{ \frac{W_{i-1}}{Random(.)} * Brigntness_{i,j} \,|\, j-1 \in Order, i\ unOrder \right\} \tag{5}$$

where, Wi-1 shows the test cases weight i-1, a label "Order" is used for the test cases where the priority is known, "un-order" is for test cases where the priority is unknown, Random (·)={N-2<[[Ni-i]-0.1]<N}, where N is the total number of test cases.

The steps of this algorithm are as follows (Su et al., 2020):

- Define the new fitness function as shown in Eq. 5.
- Calculate brightness of firefly using Eq. 6.
- Calculate the degree of attraction by creating a distance matrix as shown in Eq. 7.
- The movements of the fireflies are based on brightness only. Once all nodes have been visited, the firefly stop moving and their path is recorded.
- In the end, the smallest path will be an optimal sequence.

$$Brightness_{i,j} = W_i / \left(\frac{n_{i,j}}{\Sigma_k n_{k,j}} * \log \frac{|D|}{|\{d : d \ni t_i\}|} \right) \tag{6}$$

$$X_i^{t+1} = X_i^t + \beta e^{-\gamma S_{i,j}^2} (X_i^t - X_j^t) + \alpha \varepsilon^t \tag{7}$$

where, β and α are constants, β is the light absorption rate, Є is a random factor of a uniform distribution, ɣ is attraction coefficient, HFA used three Software-artifact Infrastructure Repository (SIR) programs for testing. Which are flex, grep and gzip as shown in Table 2. Comparisons were made in this approach with state-of-the-art algorithms FA, PSO, and Greedy and among all of them, HFA gave the better execution time and best performance as shown in Table 3.

The following limitation was observed during this study is the selection of the dataset. The dataset used in this approach is three very small SIR benchmark programs. This approach may give different results when applied to a bigger size of test case spools.

3.4. Hypervolume genetic algorithm (HGA)

The method uses GA with Hypervolume Indicator to prioritized test cases with more than three testing criteria. He proposed an enhanced genetic algorithm which is called HGA (Hypervolume-based Genetic Algorithm). Earlier GAs was used to solve only single-objective optimization problems. But hypervolume allows us to combine multiple-objects to teat as a single object. HGA consist of two parts (Di Nucci et al., 2018; 2015).

Part-I

1. Initialization of cumulative coverage scores.
2. Computation of cumulative coverage scores (cost, branch, and statements) for each testing case.
3. Computation of actual I_H using equation8.
4. Normalized I_{HP} using Eq. 9.

$$I_H(\tau) = \sum_{i=1}^{n-1}[cost(p_{i+1}) - cost(p_i) \times |Cov_i(p_i)| \times \ldots \times |Cov_m(p_i)|]$$

$$I_{HP}(\tau) = \frac{\sum_{i=1}^{n-1}[cost(p_{i+1}) - cost(p_i) \times |Cov_i(p_i)| \times \ldots \times |Cov_m(p_i)|]}{cost(p_n) \times |Cov_1^{max}| \times \ldots \times |Cov_m^{max}|}$$

Eq. 8, 9

Part-2

1. Test cases order generation.
2. Permutations evaluation by IHP (t) metric.
3. GA evolution derivation.

HGA used five Software-artifact Infrastructure Repository (SIR) programs for testing. Which are bash, flex, grep, gzip and sed as shown in Table 2.

Comparisons were made in HGA with state-of-the-art algorithms Additional Greedy Algorithm, GA (AUC metrics), NSGA-II, GDE3, MOEA/D-DE, and among all of them HGA shows the prominent results as shown in Table 3.

Following limitations were observed during the study of this algorithm. The first one is the type of construction used to validate the algorithm depends upon the results obtained. In order to validate the results APFDc matrix was used and only three testing criteria were used which include statement coverage, fault coverage, and execution cost. On the other hand, if we include the time parameter the results could be different. The second one is internal validity. a) Only different types of GA were used to compare the results of this algorithm. It would be highly appropriate to choose some other algorithm as well. b) Tuning of the parameter could really affect the validity of results Study shows that the value of parameters was kept the same as used in basic GA. Results should be tested by changing the values of parameters. The third which could affect the results is the dataset used in this approach which are six trademark programs from SIR only. The algorithm should be validated against real-time industry programs to test results.

3.5. Ant colony optimization (ACO)

ACO is another prominent approach used for test case prioritization. Following different types of the algorithm has been proposed based on ACO.

Ant colony optimization is a metaheuristic optimization methodology that was first introduced by Márquez et al. (2016). Zhang et al. (2019) used requirement-based criteria in his proposed ACO technique to solve the prioritization problem. He divided it into two ways TCP-ACA-1 in which he calculated the distance between two test cases and TCP-ACA-2 which is used to update the value of pheromone continuously (Zhang et al., 2019).

This method was successfully used in solving various optimization problems and the Test case prioritization problem is one of the combinatorial optimization problems where ACO showed impressive results. As per the technique used by Gao et al. (2015) during the test case prioritization problem, there are two rules that must be followed while using ACO methodology.

ACO algorithm works as follows as at the start ants are placed on the initial Test case list from where they randomly select the next node until all the faults are identified. It updates the pheromone value once each iteration is finished. The two most important steps in this algorithm are given below (Gao et al., 2015):

- Pheromone updating rule: Pheromone is a kind of variable used in ACO which is representing the Ants pheromone. Which they deposit on their path during food search when the next ant comes in contact with pheromone they know that they are on right track and add more pheromone for the next group and so on (Márquez et al., 2016).

This Algorithm updates the pheromone values after each iteration by using this Eq. 10 (Gao et al., 2015).

$$\tau_{id}(t+1) = (1-\rho).\tau_{id}(t) + \Delta\tau_{id} \tag{10}$$

where, ρ is evaporation rate; $\tau_{id}(t+1)$ is updated value; $\tau_{id}(t)$ is last value of pheromone; $\Delta\tau_{id}$ is increse in pheromone.

- Test case selection rule: As per Ants' behaviors, the movement of ants is irrelevant to all other ants. Similarly, an ACO algorithm selecting the next node is the most important step which uses pheromone quantity and then decided which node to move. So, the next test case selection is calculated using Eq. 11 (Gao et al., 2015).

$$p_{id}^k(t) = \frac{[\tau_{id}(t)]^\alpha.[\eta_{id}(t)]^\beta}{\sum s\epsilon L_{j+1}^k[\tau_{id}(t)]^\alpha.[\eta_{id}(t)]^\beta} \tag{11}$$

where, $\eta_{id}(t)$ is visibility function; L_{j+1}^{k} is a list of alternative test cases. Where α and β are the parameters to regulates the influence of pheromone and visibility respectively. Both ACOs techniques used really small programs with a set of 8-10 test cases having 10 faults only as shown in Table 2.

In TCP-ACA comparisons were made with particle swarm optimization (PSO) algorithm, genetic algorithm (GA), and random test methods which showed better results. On the other hand, Gao-ACO was only compared with random test case selection none of the other comparison algorithms were mentioned as shown in Table 3.

Following limitations were observed during the study of this algorithm. The first one is the type of construction used to validate the algorithm depends upon the results obtained. In order to validate the results, the APFDc matrix was used. Secondly, a new technique was not compared with any of the state-of-the-art methodologies and the third but most important one is that the technique is tested with a very small program with eight test cases having 10 faults only.

3.6. Modified ant colony optimization (m-ACO)

Solanki et al. (2015; 2016) proposed a modified ant colony optimization technique (m-ACO). In this algorithm when an ant finds food, it brings food back home and goes again to collect and this process repeats until all food sources are finished. Moreover, if the path is short all ants follow the same path until food finishes. This technique used the following test criteria code coverage, fault detected, and execution time. In order to achieve the maximum fault coverage, the APFD metric was used proposed by Elbaum et al. (2002). Algorithm proceeds as follows:

- Initialization of Ant variable
- Randomly search food source
- Calculate food fitness
- Calculate the probability of selection
- Calculate the reduction in pheromone

In order to validate m-ACO author used 3 small case studies were used. 1) 10 test cases with 10 faults 2) 9 test cases with 5 faults 3) 5 test cases with 5 faults. All the test case was executed on a Perl program as shown in Table 2. In m-ACO none of the comparisons were made with any state-of-the-art methodologies to verify the correctness of this algorithm. Instead of that only APFD was calculated with prioritized and un-prioritized test cases with showed comparatively better results than un-prioritized test cases as shown in Table 3.

This technique has the following limitations. First is only 3 small case studies with a limited number of test cases were used and it is highly recommended that this methodology should be validated with real-time and big data sets. By doing this we can get a real picture of results. The second is external validity. This approach is not compared with any of the state-of-the-art methodologies. It would be good to perform a comparison with some state-of-the-art prioritization methodologies. The third is the type of construction used to validate the algorithm depends upon the results obtained. In order to validate the results only the APFDc matrix was used, in which the APFD value of prioritized and un-prioritized results was compared.

3.7. ETS-based ACO algorithm for TCP

Epistasis-based Ant Colony Optimization technique is Multiobjective search-based regression Test Case Prioritization (MoTCP). This technique is a Metaheuristics that is inspired by nature (movement of ants to find food) (Gao et al., 2015). Study shows that epistasis theory also improved the effectiveness and efficiency in a single-objective GA (Deb et al., 2000). The use of epistasis theory in the multiobjective problem also showed promising results. ETS-Based ACO Algorithm for TCP is given below (Yuan et al., 2015).

Algorithm

Steps of ETS-Based ACO Algorithm for TCP.

- Setting of parameters of ACO
- Initialization of all the new sequences.
- Initialization of pheromone (weights of transitions).
- New test case sequence creation based on ant method equation 12.
- Test sequence evaluation using a fitness function.
- Ranking of test sequences as per NSGA-II method (Deb et al., 2000).
- Comparisons and Updation of the test sequence.
- Updating pheromone.

$$P_{ij}^{k}=\begin{cases}\dfrac{[\tau_{ij}]^{\alpha}\prod_{d=1}^{2}[\eta_{ij}^{d}]^{\lambda d\beta}}{\sum l\epsilon N_{t}^{k}[\tau_{tl}]^{\alpha}\prod_{d-1}^{2}[\eta_{tl}^{d}]^{\lambda d\beta}}\\ 0\end{cases} \tag{12}$$

where, τ_{ij}=pheromone (weight of the transitions); d=number of objectives; η_{ij}=heuristic information of an object; α=heuristic factor; β=relative weight of heuristic value; N_i^k=is the set of candidate test cases; k=ant.

In order to validate E-ACO author used the SIR program for unit test and the V8 program for the smoke test. The size of the V8 program is quite large which is good but it's still not that large enough for big experiments as shown in Table 2.

E-ACO showed very impressive results as compared with the above-explained ACO algorithms. This new methodology was also compared with the state-of-the-art algorithms NSGA-II for MoTCP and ACO which shows the prominent results as shown in Table 3. E-ACO showed very prominent results still it has some limitations that must need to address to make it a standard algorithm. The first one is instrumentation accuracy, which could yield different results. It would be better if we test this using different software instead of using only gcov

(A GCC tool). Secondly, since the real-time program was not used in testing, in compensation to this test was run 10 times. It would be much better to use the real-time big program to get accurate analysis results. Third swarm sizes of different scales should be used. So we could easily compare the efficiency of ACO and E-ACO. The fourth and last one is the parameters configuration used in E-ACO which is the same used by NSGA-II. It would be rather good to change the parameters to verify the conclusion.

3.8. Hybrid ACO

This approach is proposed by Ahmad et al. (2018) which is the combination of two different approaches.

Step-I: In the first section by using test factors priorities are assigned to the test cases.

Step-II: Once we have priorities assigned to test cases we apply ACO to calculate the optimal sequence which has the lowest execution time but the highest fault rate.

The approach seems interesting but the study doesn't show anything about testing or if any comparisons were made with any state of art techniques.

3.9. Local beam search algorithm (LBS) for TCP

Jiang and Chan (2015) proposed a novel Local Beam Search Algorithm (LBS) which is an input-based search technique. LBS technique uses lightweight input as compared to other techniques like code-coverage. In each iteration of this algorithm width of the beam is kept constant that's the reason this approach is much more efficient than the other. For example, the greedy algorithm doesn't select the new test cases from uncovered test cases only directly increases cost and time (Elbaum et al., 2002). This input-based LBS algorithm takes randomly selected test case set as input, and processes it, and generates the set prioritized set of test cases.

This approach uses distance matric f to calculate the distance between prioritized and un-prioritized test cases (Gusfield, 1997). The steps of the algorithm are given below (Jiang and Chan, 2015).

Algorithm

- Selection of candidate set which is not yet selected
- Calculation of distance matrix between the candidate and already prioritized ones
- Test cases are sorted in descending order.
- Appending the top test case in the successor pool.
- Repeating the process until all the test case has been prioritized.

LBS used 4 Software-artifact Infrastructure Repository (SIR) programs for testing. Three of them are UNIX-based programs (flex, grep, gzip, and sed) as shown in Table 2. Comparisons were made in LBS with state-of-the-art algorithms ART, Greedy, and Genetic Algorithm (GA). LBS showed more impressive results than GA and Greedy but less than the Adaptive Random Test case (ART) as shown in Table 3.

Results shown by LBS are very impressive but this algorithm has some limitations. 1) Platform used, during the testing of this approach only a C program was used. In order to cross-validate the finding other programming languages should be considered. 2) Testing Suites, during the analysis of this approach only branch-adequate test suites were used. The use of different test case suites could improve the results. 3) Use of distance metric, there are a lot of other metrics available like Hamming distance matric could also be tried to analyze results. 4) Since the LBS follows a randomized procedure so, the author repeats the testing 50 to get better results. It would be much more appropriate to use a big population of the test case to avoid such issues. 50 study shows that search-based TCP techniques don't use fault rate detections. While this research used APFD which make it difficult to access that whether the results produced by this research are consistent with previous studies or not.

3.10. Particle swarm optimization (PSO)

Particle Swarm Optimization (PSO) is a multi-object TCP technique which prioritizes test case based on changes in the code. PSO is an optimization methodology that was first introduced by Eberhart et al. (2001). Souza used the Binary Constrained PSO approach to prioritize test cases using the following three steps (Tyagi and Malhotra 2014; de Souza et al., 2010):

- Removal of extra test cases.
- Selection of test cases with the highest coverage but lowest execution time.
- Prioritizing test cases.

Hla et al. (2008) proposed an algorithm that performed test case prioritization based on changes in software sections by the PSO algorithm. This algorithm has been used a lot in various functional optimization problems because of its fast convergence property. This algorithm uses multi-objects to calculate fitness values using statement, branch, function, and code coverage as shown in Eq. 13. Also, it uses the velocity and position of the test cases for the next fitness evaluation which can be calculated using Eqs. 14 and 15. PSO algorithm has two parts (Hla et al., 2008).

Part-I: Tokenizer

- Take code as input
- Converts into token line by line
- Save into database

Part-II: Prioritizer

- Initialize priorities and fitness values
- Fitness value calculation using Eq. 13

- Position value calculation using Eq. 14, 15

$$F(f) = E(f_s \wedge f_b \wedge f_p \wedge f_u) \tag{13}$$

where, f_s maximum statement coverage out of set of statements $\{s_1, s_2, \dots, s_n\} = \arg max\, f_s(ui, T, S, m)$; f_b total branch coverage out of set of compound conditions $\{c_1, c_2, \dots, c_n\} = \arg max\, f_b(ui, T, C, m)$; f_p total function coverage out of set of procedures $\{p_1, p_2, \dots, p_n\} = \arg max\, f_p\,(ui, T, P, m)$; f_u total code coverage out of set of software units $\{u_1, u_2, \dots, u_n\} = \arg max\, f_u(ui, T, U, m)$.

$$V_{ik+1} = c_0.V_{ik} + c_1.rand.(f_{ik} - x_{ik}) + c_2.rand.(f_{gk} - x_{ik}) \tag{14}$$
$$X_{ik+1} = X_{ik} + V_{ik+1} \tag{15}$$

where x_i is the position vector and v_i is rate of change of position vector. c_0, c_1, and c_2 are weight factors.

PSO used 20 test cases from JUnit as shown in Table 2. A JUnit test suite provides automatic test case creation and execution as shown in Table 2.

Comparisons were made in PSO with the state-of-the-art Greedy algorithm. PSO showed less run time complexity as shown in Table 3. Following limitations were observed during this study. 1) In order to keep internal validity in mind. This approach was tested using 20 test cases from JUnit which is really less. 2) Second is external validity, this new approach is only compared with the Greedy algorithm. It is highly recommended to validate the procedure using some real-time application and make a comparison with some other state-of-the-art techniques to get a better picture of results.

3.11. Weight hybrid TCP using PSO (WH-PSO)

Khatibsyarbini et al. (2017) proposed this weight-hybrid string distance technique. The main purpose of this technique was to get a higher APFD rate and efficiency in execution. A hybrid string distance was created using weight and distance between test cases. String distance is of two types; character-based and term-based. Two character-based string metrics used in this study are Manhattan Distance by Ledru et al. (2012) Eq. 16 and Levenshtein Distance by Jiang and Chan (2015) Eq. 17.

$$\sum_{i=1}^{n} |x_i - y_i| \tag{16}$$

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) \\ min \begin{cases} lev_{a,b}(i-1,j)+1) \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} \end{cases} \tag{17}$$

Two term-based string metrics used are Cosine Similarity distance (Jiang and Chan, 2015) Eq. 18 and Jaccard Coefficient distance (Cha, 2007) Eq. 19.

$$S_{Cos} = \frac{\sum_{i=1}^{d} P_i Q_i}{\sqrt{\sum_{i=1}^{d} P_i^2}\sqrt{\sum_{i=1}^{d} Q_i^2}} \tag{18}$$

$$S_{Jac} = \frac{\sum_{i=1}^{d} P_i Q_i}{\sum_{i=1}^{d} P_i^2 + \sum_{i=1}^{d} Q_i^2 - \sum_{i=1}^{d} P_i Q_i} \tag{19}$$

Hybrid string distance is calculated by multiplying the distance with the next test case weight.

"Term frequency-inverse document frequency" (TFIDF) shows the importance of a word in the whole document.

$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} tf_{ik}}.\log(f_{ik}) \tag{20}$$
$$idf_k = \log\frac{N}{n_k} \tag{21}$$

The whole process is divided into three phases:

A. Information Extraction phase: In this phase test cases and their inputs are extracted. The extracted inputs are saved into a separate document which will be used so, that they could be utilized in the next phase. After that, the test cases from both programs (original and version) are executed and a fault matrix sheet is prepared to compare both results.
B. String Distance Calculation: The extracted in the previous phase is used to calculate the string distances using Eqs. 16 and 17. The weight of test cases is also calculated using Eqs. 18 and 19 and the results of both calculations are populated into the test case distance matrix. Then hybrid string distance is calculated. After the completion of calculations, PSO is used to prioritize the distance matrix.
C. Evaluation phase: In this phase, a matrix sheet is created using results obtained in an earlier phase. WH-PSO used 3 Software-artifact Infrastructure Repository (SIR) programs for testing. Three of them are SIEMENs benchmark programs (tcas, jtcas, and cs-tcas) as shown in Table 2. This new approach was not compared with any state-of-the-art algorithms only comparison was made among character-based, term-based, weight-hybrid character-based, and weight-hybrid term-based results as shown in Table 3. Results shown by weight hybrid PSO are very impressive but it would be really good to test this approach with other states of the art methodologies.

3.12. Greedy particle swarm optimization (G-PSO)

Allawi et al. (2020) proposed this novel hybrid greedy and PSO algorithm (GPSO) which is the combination of both greedy and PSO algorithms.

In this approach, the author creates the fitness function using Particle Swarm Optimization (PSO) and then applies the Greedy approach to find the partial best and makes them as global best and select the optimal solution from there. After this step new algorithm moves back to the PSO approach where it finds the velocity of each particle is calculated using Eq. 22 and the location of the next partial using Eq. 23. This process repeats until the termination condition occurs.

$$v_i(t) = wv_i(t) + c_1r_1[pb_i(t) - x_i(t)] + c_2r_2[gb_i(t) - x_i(t)] \quad (22)$$
$$x_i(t) = x_i(t) + v_i(t) \quad (23)$$

Algorithm proceeds as follows (Allawi et al., 2020):

- Random Initialization of particles.
- Calculation of fitness function using PSO.
- Greedy part:

  - Determines and selects the particle with the *partial Best* position value to be placed as the *global Best* solution.
  - Selects the optimal solution from the available candidate solutions at a particular point.

- Each particle velocity can be calculated using Eq. 22.
- According to Eq. 23 move to the next position.
- The algorithm will be stopped if the termination criteria are satisfied; otherwise, return to Step 2.

Six different Java card applications (JCS Applets) were used as data set to test the algorithm which includes a Network connection tracker, HelloWorld, PKI, RSACrypto, Calculator, OATH, Passport, CoolKey as shown in Table 2.

The comparison was done between the GPSO algorithm and standard genetic algorithm (GA), which showed that GPSO exceeds the GA in various ways. GPSO takes a fewer number of iterations and time but gives us high coverage percentage as shown in Table 3.

Following are the limitations found during this study. 1) Real-time application was used to test this approach but that includes small programs with few hundreds of branched was used. In a real-time program, there are thousands of branches of code that could give a different point of view about this approach. 2) Discussion about fault matrix is missing in the study, whether it's created or not. 3) Only one comparison was done with GA, it would be better to test this new approach with other state-of-the-art methodologies.

3.13. Greedy algorithm (GA)

3.13.1. Additional greedy algorithm (AGA)

Yoo and Harman (2007) proposed the Additional Greedy Algorithm. This technique uses NSGA-II and its variant which is called vNSGA-II algorithms.

The vNSGA-II has two main modifications as compared to NSGA-II which are as follows:

- vNSGA-II uses a separate group of sub-population which is good because different objectives can be assigned to each sub-group.
- vNSGA-II keeps the record of the best sub-population.

The additional greedy algorithm only supports up to two objects. In order to work with more than two objects, we have to combine the objects to make one by using the weight sum approach. That is the reason AGA doesn't show good results for multi-object TCP.

AGA used 4 Software-artifact Infrastructure Repository (SIR) programs SIEMENs based programs (printtokens, printtokens2, schedule, schedule2), and 1 European Space Agency (ESA) program (space) as shown in Table 2. Comparisons were made in this study among two genetic algorithms, NSGA-II, and its variation. It has been seen that the greedy algorithm doesn't work well for multi-objective TCP it is only good for single objects as shown in Table 3.

Following limitations were found during this study.1) Accuracy of tools used to find the coverage information. The author used professional software in order to minimize the risk but it would be highly recommended to try to use some other software to just see the variety of results. 2) Most of the testing data used in this study is from the SIR repository. Only one program was used from ESA. 3) Results of this approach also depend upon the type of algorithm used for multi-object prioritization. The author tried to use so far best algorithm NSGA-II but it is highly recommended to test the approach with some other multi-object test case prioritization techniques like firefly etc. to test the variation in results.

3.13.2. Enhanced additional greedy algorithm (EAGA)

This technique is proposed by Hsu et al. (2014), EAGA is a modified form of AGA. In this technique control flow is used over the data flow criteria. Experimental analysis shows that the EAGA technique performed well in "fault severity/unit cost". EGA used 8 Software-artifact Infrastructure Repository (SIR) programs SIEMENs based programs (Tcas, Totinfo, Replace, Schedule, Schedule2, Printtokens, Printtokens2, and Space) as shown in Table 2. Comparisons were made in this study with AGA which shows that EAGA with GA is far much better than AGA for large programs only but when we use small programs there is not a difference between AGA and EAGA as shown in Table 3.

Following limitations were found during this study:

1) Same as AGA, EAGA is not fit for multi-object TCP.
2) Results are not compared and checked with any other state-of-the-art techniques to get clear pictures of results.

Graphite TCP: One of the most popular prioritization techniques is greedy. The greedy technique used code coverage criteria. In this technique test cases with higher code coverage are considered better than the lower coverage (Do et al., 2008; 2010). Azizi and Do (2018) proposed a novel graph-based greedy technique that utilizes a graph traversal algorithm to prioritized test cases. This algorithm consists of two parts, graph generator, and graph traversal.

$$Node\ value = \frac{code\ coverage}{execution\ time} \tag{24}$$

Proposed two algorithms:

Part-I: Graph Generation:

1) Nodes creation.
2) Node value calculation using Eq. 24.
3) Edge value calculation using Jaccard Distance Eq. 19.

Part-II: Traversal Procedure:

1) Select node with maximum value as a start point.
2) Calculate the highest gain using the traversal algorithm.
3) Repeat this process until all connection edges will eliminate.

In order to test this approach, the author used 4 open-source programs, among them 2 are obtained from SIR (jmeter, jtopas) and two of them are none SIR (nopCommerce, Umbraco-CMS) as shown in Table 2. Experiment results showed indicated approach is effective and efficient as compared with the unprioritized approach only as shown in Table 3.

There are few findings of this new methodology. 1) This new technique is not compared with any state-of-the-art TCP methodology to get a clear picture. 2) The choice of the distance function. The most popular method was used for it but it could give different results with a different function. It is highly recommended to test the new approach using other distance functions as well so, we could get a clear picture of this algorithm.

Table 2: Datasets used by each algorithm

S.No	Algorithms	Repository	Datasets
1	Firefly Algorithm	SIR	UNIX benchmarks Programs (flex, grep, gzip) SIEMENs benchmarks Programs (tcas, cs-tcas, j-cas)
2	Optimal FA	Non-SIR	4 small programs with set of 16-48 test cases.
3	Hybrid FA	SIR	UNIX Benchmarks Programs (flex, grep, gzip)
4	Hypervolume Genetic Algorithm (HGA)	SIR	UNIX Benchmarks Programs (bash, flex, grep, gzip, sed)
5	ACO	Not SIR	A small program with set of 8-10 test cases with 10 faults only.
6	m-ACO	Not SIR	Perl Program based 3 case study used to test
7	ETS based ACO	SIR	3 UNIX Benchmarks Programs (flex, space, bash) Open-source JavaScript engine V8
8	Hybrid ACO	None	
9	LBS	SIR	4 UNIX Benchmarks Programs (flex, grep, gzip, sed)
10	PSO	Non-SIR	JUnit 20-test case
11	WH-PSO	SIR	3 SIEMENs Benchmarks Programs (tcas, jtcas, and cstcas)
12	Greedy PSO	Non-SIR	Six Java Card Application Programs (Network connection tracker, HelloWorld, PKI, RSACrypto, Calculator, OATH, Passport, CoolKey)
13	Additional Greedy	SIR	4 SIEMENs Benchmarks Programs (printtokens, printtokens2, schedule, schedule2) 1 European Space Agency (space)
14	Enhanced Greedy	SIR	8 SIEMENs Benchmarks Programs (Tcas, Totinfo, Replace, Schedule, Schedule2, Printtokens, Printtokens2, and Space)
15	graphite greedy	SIR, NON-SIR	4 UNIX Benchmarks Programs (jmeter, jtopas) nopCommerce, Umbraco-CMS

4. Research methodology

This study was divided into the following stages:

1. Research question.
2. Repositories Selection and Searching Strategies Search methodology.
3. Inclusion-Exclusion Criteria for Selection.

4.1. Research questions

Following are the research questions of this paper:

Q1: How TCP is important in Software Testing?
Q2: What are the different classifications of test case prioritization techniques?
Q3: What are single object and multi-object TCP techniques?
Q4: What are the latest studies in TCP?
Q5: Which metrics are commonly used in TCP?
Q6: Which prioritization methods are used often and what are their proportions?

4.2. Repositories selection and searching strategies

In this study, we sued keywords like "Test Case Prioritization", "Test Case Selection", "Test Case Optimization" to search the well-known repositories like Google Scholar, IEEE Xplore, Elsevier, ACM Digital Library, and Springer as shown in Figs. 2, 3, 4, 5, 6, 7, and 8.

Table 3: Algorithms comparisons

S.No	Algorithms	Compared With	Results
1	Firefly Algorithm	PSO, LBS, Greedy, GA	FF Showed better Results but LBS was almost close to FF.
2	Optimal FA	ACO	FF Algorithm Showed better results.
3	Hybrid FA	FA, PSO, and Greedy	HGA showed prominent results over FA and PSO
4	Hypervolume GA	Additional Greedy Algo, GA(AUC metric), NSGA-II, GDE3, MOEA/D-DE	HGA is more cost-effective and efficient in dealing with more criteria
5	ACO	PSO and Random Test Case Selection	In TCP-ACA comparisons were made with particle swarm optimization algorithm, genetic algorithm, and random test methods which showed better results. On the other hand, Gao-ACO was only compared with random test case selection non-of the other comparison algorithms were mentioned.
6	m-ACO	ACO	Prioritized with m-ACO and un-prioritized showed better results.
7	ETS-ACO	Original ACO for MoTCP NSGA-II algorithm for MoTCP	E-ACO algorithm has significant improvement effectiveness and the efficiency to the state-of-the-art NSGA-II algorithm.
8	Hybrid ACO	ACO	better results as compared to ACO
9	LBS	Genetic, Greedy(Total and Additional), Optimal, Hill Climbing and ART	LBS showed more efficient results than GA and Greedy but less than ART
10	PSO	Greedy Algorithm	PSO has time complexity less than Greedy Algorithm
11	WH-PSO	Comparison was made among character-based, term-based, weight-hybrid character-based, and weight-hybrid term-based	Hybrid PSO showed impressive results
12	Greedy PSO	Genetic Algorithm	GPSO outperform the GA in terms of the average number of iterations, execution time, and coverage percentage
13	Additional Greedy	NSGA-II	Greedy algorithm is not good for multi objective TCP
14	Enhanced Greedy	AGA	EAGA showed better results as compared to AGA but only for a single objective.
15	Graphite Greedy	None	High Average fault detection results

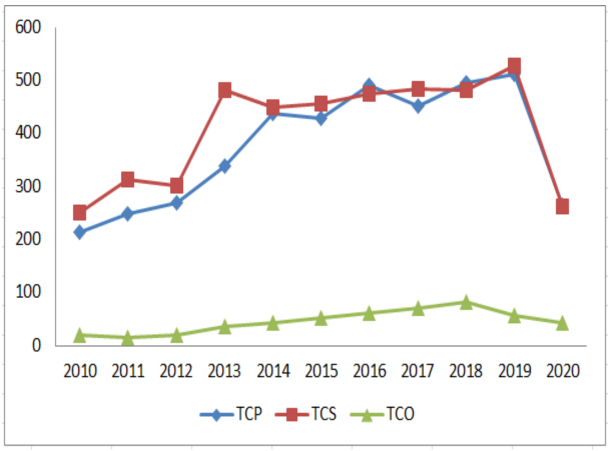

Fig. 2: Search results of google scholar (TCP, TCS, TCO)

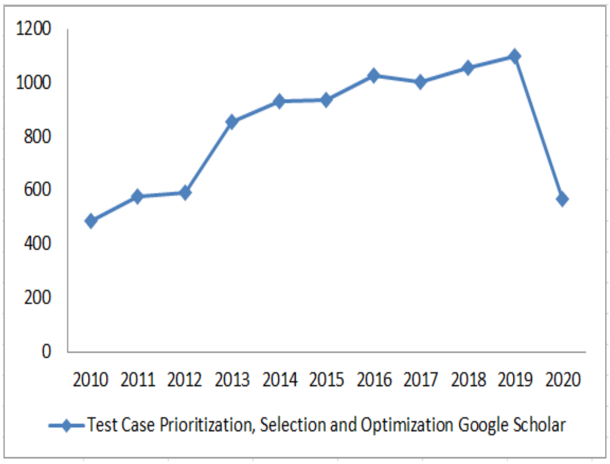

Fig. 3: Total search results of google scholar

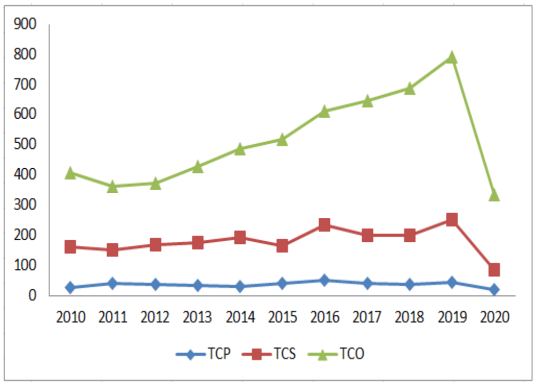

Fig. 4: Search results of IEEE (TCP, TCS, TCO)

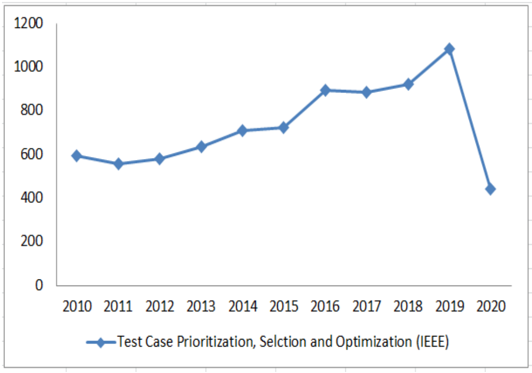

Fig. 5: Total search results of IEEE

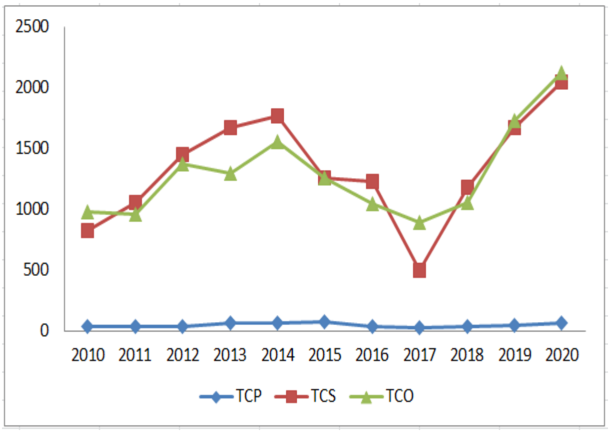

Fig. 6: Search results of Springer (TCP, TCS, TCO)

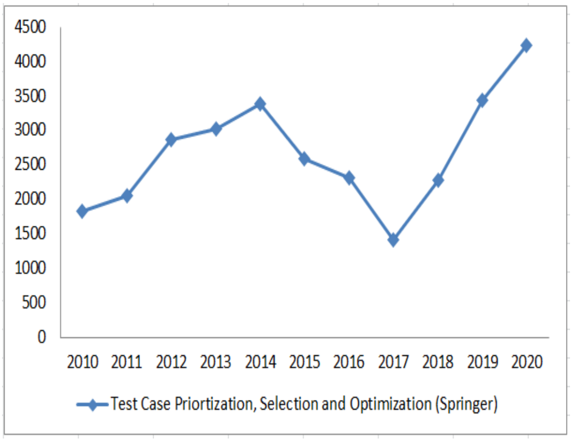

Fig. 7: Total search results of Springer

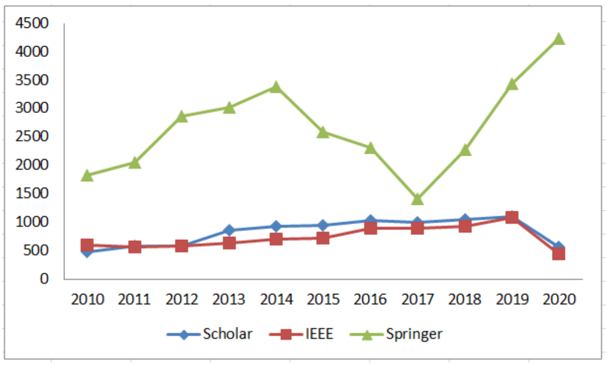

Fig. 8: Total search results of Scholar, IEEE, Springer

5. Results

In this section each research questions were answered:

Q1: How TCP is important in Software Testing?
In Regression Testing (RT), the TCP is the most difficult phase of the RT that allows testers to risk management, test planning, cost-value analysis of the testing phase. Different approaches are used to prioritized test cases for example time, cost, effort, code coverage, and fault coverage which increase RT performance. By using these techniques test cases with the highest priority are selected for RT suites that give us the highest rate of fault detection.
Q2. What are different classifications of test case prioritization techniques?
Fig. 2, shows the detailed classification of the test case prioritization techniques.

Q3. What are single object and multi-object TCP techniques?
Table 4 shows which techniques are multi-object TCP and single-object TCP. This table also shows which metrics were used to validate the algorithm.
Q4. What are the latest studies in TCP?
Table 5 shows the latest studies in Test Case Prioritization techniques.
Q5. Which metrics are commonly used for TCP?
About 70% of the multi-object test case prioritization techniques used the Average Percentage Fault Detection (APFD) metric as shown in Table 4.
Q6. Which prioritization methods are used often and what are their proportions?

Table 6, shows the percentage of each algorithm from different repositories.

Table 4: Metrics used in different algorithms

Techniques	Algorithms	Metrics Used
Multi-Object	Optimal Firefly Algorithm	
	Firefly Algorithm	APFD
	Hybrid FA	APFD
	Hypervolume GA	APFD
	ACO	APFD
	m-ACO	APFD
	ETS based ACO	APSC, EET
	Hybrid ACO	
	LBS	APFD
	PSO	
	WH-PSO	APFD
	Greedy PSO	
Single Object	Additional Greedy	
	Enhanced Greedy	
	Graphite greedy	

Table 5: Latest test case prioritization techniques

S.No	Ref ID	Algorithm	Year	Repository
1	Shah et al.	Optimal Firefly Algorithm	2016	Elsevier
2	Khatibsyarbini et al.	Firefly Algorithm	2019	IEEE
3	Su et al.	Hybrid Firefly Algorithm	2020	ICCEA
4	Di Nucci et al.	Hypervolume GA	2018	IEEE
5	Zhang et al.	Ant Colony Optimization	2019	IEEE
6	Solanki et al.	Modified ACO	2016	Springer
7	Bian et al.	ETS Based ACO	2017	IEEE
8	Ahmad et al.	Hybrid ACO	2018	Elsevier
9	Jiang and Chan	LBS	2015	
10	Tyagi and Malhotra	PSO	2014	ICSPCT
11	Khatibsyarbini et al.	Hybrid PSO	2017	JATIT
12	Allawi et al.	G-PSO	2020	Springer
13	Yoo and Harman	Additional Greedy Algorithm	2007	ACM
14	Hsu et al.	Severity-weighted Greedy Algorithm	2014	IEEE
15	Azizi and Do	Graphite Greedy Algorithm	2018	IEEE

Table 6: Algorithm wise TCP percentage

S.No.	Algorithm	Google Scholar	IEEE	Springer	Total	%
1	Firefly	15800	30	867	16697	4.5
2	GA	238000	1105	21076	260181	70.5
3	ACO	16900	111	1998	19009	5.1
4	LBS	19100	20	635	19755	5.3
5	PSO	16700	556	6396	23652	6.4
6	Greedy	16800	193	12468	29461	7.9
Total		323300	2015	43440	368755	

Table 7 shows the inclusions and exclusions criteria of this study.

Table 7: Inclusions and exclusions criteria of this study

Inclusion criteria	Exclusion Criteria
(Regression Testing and/or Test case) and (Prioritization or Selection) and "Name of Algorithm" and (Journals or Conference Paper) and (IEEE or Springer or Elsevier)	Non-Software Testing and/or Non-English paper

5.1. State of the art challenges

5.1.1. Measurement

- Effectiveness: Most of the TCP techniques are validated in terms of their effectiveness by using APFD metrics. However, APFD is not quite suitable and has quite constraints. For example, in APFD metrics all test cases and faults have the same cost (Askarunisa et al., 2010). On the other hand, APFDc works well because it allows variable test cases and fault costs. But it only allows two parameters. It is recommended to uses such metrics which allow multiple parameters. There is another metric that could be utilized like Average Percentage Statement Coverage (APSC), Average Percentage Branch Coverage (APBC), Average Percentage Loop Coverage (APLC), and Average Percentage Condition Coverage (APCC) (Li et al., 2007).
- Efficiency: Efficiency is also an important issue and there is no such proper matrix use to measure the efficiency of a technique. So far only things considered are actual prioritization time and test-case execution time. It is not enough to consider these two parameters because most techniques require more information e.g., code coverage, etc. which may add more cost. Matrix-like ANOVA and ANCOVA (Nayak et al., 2019) and Average Percentage of Combinatorial Coverage (APCC) which considers combination weights and test costs (Wang et al., 2011). It is highly recommended to keep these things in mind before start prioritization so we could get accurate cost and computation time.

5.1.2. Data utilization

Study shows that effectiveness and efficiency of a logarithm depend upon various factors for example programming language and its framework, coverage, types of faults, etc. (Hao et al., 2016). Other than these factors the most important thing that could affect the effectiveness and efficiency of an algorithm is as follows:

- Size of data set: Most of the above describe studies are only using a small set of SIR programs like bash, flex, grep, gzip, sed, etc. and some of them even used very small programs having 8-10 test cases (Khatibsyarbini et al., 2019; 2017; Su et al., 2020; Di Nucci et al., 2018; Jiang and Chan, 2015; Yoo and Harman, 2007).
- Incomplete data: Second most important thing is input data used in the algorithm. For example, the results of the APFD matrix depend upon the distance and weight of test cases which depends upon input given to the system (Khatibsyarbini et al., 2019). Test cases have a lot of other data available in them. But it is very difficult, time-consuming, and expensive to utilize all the available data. The best example is system coding. The extraction and utilization of complete data could be a big challenge in the test case prioritization techniques which could give us different results.
- Parameter Tuning: The tuning of parameters is could also affect the effectiveness and efficiency of an approach. For example, in few studies, parameter values were not changed and kept as it

is as in the previous work (Li et al., 2007). Tuning to that parameter could influence the overall results. Lu studies the influence of change in code or its parameter. Study shows that change in code and parameter could influence the effectiveness an algorithm (Lu et al., 2016).

5.1.3. Execution time

The study shows that most of the techniques are tested using the SIR program which is a small set of programs. These programs have test cases whose execution time is very low. It would be worth testing the above-given prioritization techniques on a test case where execution time is high. But testing of high execution time test cases could increase the cost. It would be better to consider the individual cost of test cases instead of the total cost.

6. Conclusion and future work

In this paper, an empirical survey has been conducted on Test case prioritization techniques. In this study, we review 40 TCP papers published in journals and conferences with a systematic process. The main objective of this study is to explore and determine which approaches have been studied in detail and which of them are left behind and where is more chance of improvement. We divided this study into the following stages. In Stage-I we created six research questions. Stage-II we selected repositories from where we are going to get our literature. Stage-III we created a search methodology in which we search the most recent papers on state-of-the-art techniques. Stage-IV we setup inclusion and exclusion criteria which we utilized to search our data as shown above. The following suggestion was provided in this study:

- **Latest public data set utilization:** Almost 60% of the papers selected in this study are using a public data set which is good but data is being updated regularly which is not satisfactory. By the development of different programming languages and programming techniques updated code must be utilized for testing. It looks like becoming a trend of using SIR data instead of actual data or updated public datasets. Although there are challenges of getting actual data set and re-testing by other researchers.
- **Consideration of multi-object TCP approaches:** Study shows that 35% of the approaches are well-defined multi-object and 20% of them are purely single object TCP techniques and the rest of them are not properly stated as single or multi objects. It would be highly recommended to create a multi-object model-based approach that could set standards.
- **Consideration of cost criteria:** Survey showed that that most of the studies focus on code base techniques and less focus is given to cost criteria. Some studies used cost criteria as a total cost of execution of test cases. It would be better to consider the individual cost of test cases instead of the total cost.
- **State of the Art Techniques Comparison:** Study showed that 46% of the techniques have only performed a comparison with the well-defined state-of-the-art methodologies. 26% of the approaches have not compared any state-of-the-art technique. It would be highly recommended to set up such models which accept the studies having compared with the state-of-the-art techniques. Also, it is good to conduct more studies for comparison of TCP techniques.
- **Survey study for Multi-Object TCP:** Study showed that none of the review/survey studies were conducted on multi-object TCP techniques. It would be highly recommended to separate publishing the review study of multi-object TCP techniques.
- **Utilization of real-time dataset:** Study showed that none of the approaches has been tested using any industrial or real-time dataset. It would be highly recommended to validate the test case prioritization techniques using some big industrial or real-time data set to get a clear picture of the new approach. However, only 10% of the studies used non-SIR data set but not industrial data set.

Compliance with ethical standards

Conflict of interest

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

Ahmad SF, Singh DK, and Suman P (2018). Prioritization for regression testing using ant colony optimization based on test factors. In: Singh R, Choudhury S, and Gehlot A (Eds.), Intelligent communication, control and devices: 1353-1360. Springer, Singapore, Singapore. https://doi.org/10.1007/978-981-10-5903-2_142

Allawi HM, Al Manaseer W, and Al Shraideh M (2020). A greedy particle swarm optimization (GPSO) algorithm for testing real-world smart card applications. International Journal on Software Tools for Technology Transfer, 22(2): 183-194. https://doi.org/10.1007/s10009-018-00506-y

Askarunisa MA, Shanmugapriya ML, and Ramaraj DN (2010). Cost and coverage metrics for measuring the effectiveness of test case prioritization techniques. INFOCOMP Journal of Computer Science, 9(1): 43-52.

Azizi M and Do H (2018). Graphite: A greedy graph-based technique for regression test case prioritization. In the IEEE International Symposium on Software Reliability Engineering Workshops, IEEE, Memphis, USA: 245-251. https://doi.org/10.1109/ISSREW.2018.00014 **PMid:30045707 PMCid:PMC6060527**

Bian Y, Li Z, Zhao R, and Gong D (2017). Epistasis based ACO for regression test case prioritization. IEEE Transactions on Emerging Topics in Computational Intelligence, 1(3): 213-223. https://doi.org/10.1109/TETCI.2017.2699228

Cha SH (2007). Comprehensive survey on distance/similarity measures between probability density functions. International Journal of Mathematical Models and Methods in Applied Sciences, 1(4): 300-307.

de Souza LS, Prudêncio RBC, and de Almeida Barros F (2010). A constrained particle swarm optimization approach for test case selection. In the 22nd International Conference on Software Engineering and Knowledge Engineering, Redwood City, USA: 259-264.

Deb K, Agrawal S, Pratap A, and Meyarivan T (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In the International Conference on Parallel Problem Solving from Nature, Springer, Paris, France: 849-858. https://doi.org/10.1007/3-540-45356-3_83

Di Nucci D, Panichella A, Zaidman A, and De Lucia A (2015). Hypervolume-based search for test case prioritization. In: Barros M and Labiche Y (Eds.), International symposium on search based software engineering: 157-172. Springer, Cham, Switzerland. https://doi.org/10.1007/978-3-319-22183-0_11

Di Nucci D, Panichella A, Zaidman A, and De Lucia A (2018). A test case prioritization genetic algorithm guided by the hypervolume indicator. IEEE Transactions on Software Engineering, 46(6): 674-696. https://doi.org/10.1109/TSE.2018.2868082

Do H, Mirarab S, Tahvildari L, and Rothermel G (2008). An empirical study of the effect of time constraints on the cost-benefits of regression testing. In the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Association for Computing Machinery, Atlanta, Georgia: 71-82. https://doi.org/10.1145/1453101.1453113 **PMid:17983459**

Do H, Mirarab S, Tahvildari L, and Rothermel G (2010). The effects of time constraints on test case prioritization: A series of controlled experiments. IEEE Transactions on Software Engineering, 36(5): 593-617. https://doi.org/10.1109/TSE.2010.58

Eberhart R, Shi Y, and Kennedy J (2001). Swarm Intelligence. Morgan Kaufmann Publisher, Burlington, USA.

Elbaum S, Malishevsky AG, and Rothermel G (2002). Test case prioritization: A family of empirical studies. IEEE Transactions on Software Engineering, 28(2): 159-182. https://doi.org/10.1109/32.988497

Gao D, Guo X, and Zhao L (2015). Test case prioritization for regression testing based on ant colony optimization. In the 6th IEEE International Conference on Software Engineering and Service Science, IEEE, Beijing, China: 275-279. https://doi.org/10.1109/ICSESS.2015.7339054 **PMCid:PMC4411200**

Gusfield D (1997). Algorithms on strings, trees, and sequences: Computer science and computational biology. Cambridge University Press, Cambridge, UK. https://doi.org/10.1017/CBO9780511574931

Hao D, Zhang L, and Mei H (2016). Test-case prioritization: Achievements and challenges. Frontiers of Computer Science, 10(5): 769-777. https://doi.org/10.1007/s11704-016-6112-3

Hashim NL and Dawood YS (2018). Test case minimization applying firefly algorithm. International Journal on Advanced Science, Engineering and Information Technology, 8(4-2): 1777-1783. https://doi.org/10.18517/ijaseit.8.4-2.6820

Hla KHS, Choi Y, and Park JS (2008). Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting. In the IEEE 8th International Conference on Computer and Information Technology Workshops, IEEE, Sydney, QLD, Australia: 527-532. https://doi.org/10.1109/CIT.2008.Workshops.104

Hsu YC, Peng KL, and Huang CY (2014). A study of applying severity-weighted greedy algorithm to software test case prioritization during testing. In the IEEE International Conference on Industrial Engineering and Engineering Management, IEEE, Bandar Sunway, Malaysia: 1086-1090. https://doi.org/10.1109/IEEM.2014.7058806

Jiang B and Chan WK (2015). Input-based adaptive randomized test case prioritization: A local beam search approach. Journal of Systems and Software, 105: 91-106. https://doi.org/10.1016/j.jss.2015.03.066

Khatibsyarbini M, Isa MA, and Abang Jawawi DN (2017). A hybrid weight-based and string distances using particle swarm optimization for prioritizing test cases. Journal of Theoretical and Applied Information Technology, 95: 2723-2732.

Khatibsyarbini M, Isa MA, Jawawi DN, Hamed HNA, and Suffian MDM (2019). Test case prioritization using firefly algorithm for software testing. IEEE Access, 7: 132360-132373. https://doi.org/10.1109/ACCESS.2019.2940620

Ledru Y, Petrenko A, Boroday S, and Mandran N (2012). Prioritizing test cases with string distances. Automated Software Engineering, 19(1): 65-95. https://doi.org/10.1007/s10515-011-0093-0

Li Z, Harman M, and Hierons RM (2007). Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering, 33(4): 225-237. https://doi.org/10.1109/TSE.2007.38

Lu Y, Lou Y, Cheng S, Zhang L, Hao D, Zhou Y, and Zhang L (2016). How does regression test prioritization perform in real-world software evolution? In the 38th International Conference on Software Engineering, Association for Computing Machinery, Austin, Texas: 535-546. https://doi.org/10.1145/2884781.2884874

Márquez FPG, Papaelias M, and Zaman N (2016). Non-destructive testing. BoD–Books on Demand, Norderstedt, Germany.

Nayak S, Kumar C, Tripathi S, and Jena L (2019). Efficiency enhancement in regression test case prioritization technique. International Journal of Innovative Technology and Exploring Engineering, 8(12): 5445-5451. https://doi.org/10.35940/ijitee.K1595.1081219

Panthi V and Mohapatra DP (2015). Generating prioritized test sequences using firefly optimization technique. In: Jain L, Behera H, Mandal J, and Mohapatra D (Eds.), Computational intelligence in data mining: 627-635. Volume 2, Springer, New Delhi, India. https://doi.org/10.1007/978-81-322-2208-8_57

Shah SAA, Bukhari SSA, Humayun M, Jhanjhi NZ, and Abbas SF (2019). Test case generation using unified modeling language. In the International Conference on Computer and Information Sciences, IEEE, Sakaka, Saudi Arabia: 1-6. https://doi.org/10.1109/ICCISci.2019.8716480

Shah SAA, Shahzad RK, Bukhari SSA, and Humayun M (2016). Automated test case generation using UML class and sequence diagram. Current Journal of Applied Science and Technology, 15(3): 1-12. https://doi.org/10.9734/BJAST/2016/24860

Solanki K, Singh Y, and Dalal S (2015). Test case prioritization: an approach based on modified ant colony optimization (m-ACO). In the International Conference on Computer, Communication and Control, IEEE, Indore, India: 1-6. https://doi.org/10.1109/IC4.2015.7375627 **PMid:27512541 PMCid:PMC4959403**

Solanki K, Singh Y, Dalal S, and Srivastava PR (2016). Test case prioritization: An approach based on modified ant colony optimization. In: Shetty N, Prasad N, and Nalini N (Eds.), Emerging research in computing, information, communication and applications: 213-223. Springer, Singapore, Singapore. https://doi.org/10.1007/978-981-10-0287-8_19

Su W, Li Z, Wang Z, and Yang D (2020). A meta-heuristic test case prioritization method based on hybrid model. In the International Conference on Computer Engineering and Application, IEEE, Guangzhou, China: 430-435. https://doi.org/10.1109/ICCEA50009.2020.00099

Tyagi M and Malhotra S (2014). Test case prioritization using multi objective particle swarm optimizer. In the International Conference on Signal Propagation and Computer Technology, IEEE, Ajmer, India: 390-395. https://doi.org/10.1109/ICSPCT.2014.6884931

Wang Z, Chen L, Xu B, and Huang Y (2011). Cost-cognizant combinatorial test case prioritization. International Journal of

Software Engineering and Knowledge Engineering, 21(06): 829-854. https://doi.org/10.1142/S0218194011005499

Yoo S and Harman M (2007). Pareto efficient multi-objective test case selection. In the International Symposium on Software Testing and Analysis, Association for Computing Machinery, London, UK: 140-150. https://doi.org/10.1145/1273463.1273483 **PMid:20799183**

Yuan F, Bian Y, Li Z, and Zhao R (2015). Epistatic genetic algorithm for test case prioritization. In: Barros M and Labiche Y (Eds.), International symposium on search based software engineering: 109-124. Springer, Cham, Switzerland. https://doi.org/10.1007/978-3-319-22183-0_8

Zhang W, Qi Y, Zhang X, Wei B, Zhang M, and Dou Z (2019). On test case prioritization using ant colony optimization algorithm. In the IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems, IEEE, Zhangjiajie, China: 2767-2773. https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00388 **PMCid:PMC6469475**