Journal of Autonomous Intelligence (2024) Volume 7 Issue 5
doi: 10.32629/jai.v7i5.1045

ORIGINAL RESEARCH ARTICLE

Requirement Aware Optimisation of Test Case Selection (R-OTCS) approach

Vivek Kulkarni[1,*], R. Madhanmohan[1], H. Venkateswara Reddy[2]

[1] *Department of CSE, Annamalai University, Chidambaram, Tamilnadu 608002, India*

[2] *Department of CSE, Vardhaman College of Engineering, Shamshabad 501218, Hyderabad, India*

*** Corresponding author:** Vivek Kulkarni, vivek@vardhaman.org

ABSTRACT

TCP (test case prioritization) is a difficult problem to solve. Its complexity grows in direct proportion to the addition or update of project modules. Prioritizing an enormous amount of test cases is a complicated task. To optimize test case prioritization, a variety of strategies and techniques are available. It has been noted that eradicating all flaws from a software project is very hard, and even after using several testing methodologies, some flaws remain in the project. The work that is suggested priorities not only the fault detection abilities of test cases across the whole suite, but also the defect degree and commercial relevance of the test case execution. The purpose of this study is to locate all undiscovered faults using a Requirement Aware Optimization of Test Case Selection (R-OTCS) approach. We applied the genetic algorithm (GA) to optimize this proposed approach. We calibrated GA for performance before applying it to a dataset by fine-tuning the algorithm. The proposed technique improves software dependability by finding errors early and detecting serious problems first. Prioritizing test cases that address business critical/major requirements also improves reliability. The average percentage of fault detection (APFD) metric is used to assess all generated sequences. The business criticality value is used to determine the test case score. The suggested ROTCS approach yielded encouraging results in terms of APFD score and fault detection rate.

Keywords: Optimisation; Genetic Algorithm; Testing; Prioritization; Test Cases

ARTICLE INFO

Received: 26 July 2023
Accepted: 12 April 2024
Available online: 12 November 2024

1. Introduction

Software engineering is a method of developing software in a methodical manner. Software testing is an essential part of the software development life cycle (SDLC) that ensures the completed product meets the client's expectations. Testing guarantees that the expanded version of a programme functions properly, in addition to maintaining product uniformity. A computer programme must change in order to remain competitive in today's highly competitive marketplace. Because intuition testing is dependent on individual outcomes, it is frequently advised to carefully examine a specialised team of testing specialists. Inadequate testing causes missed problems with a substantial quantity of rework that requires more expensive repair. Testing is the practise of detecting problems in software and fixing them as soon as feasible in order to create software of excellent quality by fulfilling user requirements[1]. To assure the general efficacy and stability of the programme, testing is an expensive operation in terms of time, resources, and money[2]. The most important aspect of software testing is software quality assurance.

The process of designing and running test cases is critical to the success of the software's testing phase. A test case is a basic, well-designed, and easily comprehensible unit of testing that uses test data as input and is created for a particular situation to ensure that the feature works as intended. Testers create test scenarios based on the goal of the testing. There are several sorts of test cases, including functional, user interface (UI), unit, integration, usability, database, performance, security, and user acceptability. A test suite is a group of these test cases. Additional tests are produced and added to the current test suite as the app develops and improves. Additional tests are produced and added to the current test suite as the app develops and improves. Econometric analysis is a technique for ensuring that any modifications do not disrupt previously operating software. Changes are planned and an inevitable part the software development life cycle. The software test suite grows in complexity and scope, increasing the cost of running it. Regression testing, according to studies, is a costly operation that can take over thirty percent of the overall cost of the product[3]. Regression testing is a laborious and expensive testing process that confirms that code updates have not affected existing functioning[4]. Due to time restrictions, running the whole test suite after any code modification is just impossible.

Test case prioritisation (TCP) is a tried-and-true regression testing strategy that accelerates fault detection by prioritising test case (TC) execution. Scheduling the carrying out of all TCs is a difficult and time-consuming process that necessitates the application of optimisation methods. The TCP approach supports choosing on the test scenario execution order to fulfil the early fault identification aim. Ordering the entire test set is an NP-hard optimisation issue. It is difficult to address the TCP problem in the time limit due to the rapid expansion of the test suite. Even after a successful TCP implementation utilising any standard optimisation, there are many test cases that are likely to be crucial for that business application. Because testing software is a continuous process and no one can guarantee that a software product is completely bug-free, this research paper addresses the problem of TCP by employing a multi-model multi-objective-based optimisation strategy for the left-over test cases. To tackle the TCP problem, we applied the genetic algorithm (GA). GA is initially applied to the TCP issue. The calibrated GA results are then compared in order to fine-tune the GA settings. GA is used in this case by the test case weighted fitness function. We used the genetic algorithm (GA) to solve the TCP problem. This suggested technique has been proven on three projects, and the APFD is computed using the R-OTCS approach.

2. Related work

This section provides a literature overview on the use of algorithms inspired by nature for regression testing. For Example, Li et al.[5] weighed search-based approaches against classic algorithm. GA has been shown to have been more effective in investigating search space. It stimulated subsequent research into the application of nature-inspired algorithms. For example, the findings reported in Zhang et al.[6] beat GA, PSO, and RS when a distance-based and index-based version of ACO was utilised to prioritise test cases. To solve the TCP problem, a novel fixup procedure for permutation encoding was created using the cuckoo search algorithm (CSA)[7]. The configuration-aware software testing test suite was also reduced using CSA[8].

Ant Colonies optimisation was utilised by Mohapatra and Prasad [9] on Java applications, and the efficiency for reduced suites and complexity was compared with normal procedures. The quantum-inspired ACO approach for test suite reduction was developed by Zhang et al.[10]. In terms of size reduction, the proposed strategy beat earlier ACOs. NSGA-II was employed by Mondal et al.[11]. TCS may be enhanced by employing test suite diversity and code coverage as fitness measures. With a 20% time limitation, diversity increased the fault identification rate by up to 16.

PSO has been used by several researchers, including Khatibsyarbini et al.[12], who arranged the test examples using string distances and verified it on the real-world TSL dataset. The binary restriction PSO and its hybrid variants with local search strategies were developed to locate test cases that use redundancy[13]. PSO was developed with the purpose of increasing branch coverage and lowering expenses by using local search to identify test cases[14]. Because of the favorable results, PSO was merged with harmonic search, which outperformed NSGA-II[15]. Correia[16] developed a test suite variety diagnosability measure, and the results were improved by integrating local search algorithms with PSO to boost requirement coverage while limiting associated costs.

By integrating the PSO and the dragonfly algorithm, TCP was also employed to minimise test cases. The results show that hybrid algorithms beat other search methods[17]. To prioritise, select, and reduce test cases, tri-level regression testing was designed based on statement coverage. Tri-level regression testing was created to establish priorities, select risks, and minimise them. In terms of statement coverage, the combination of PSO and gravity search algorithm (PSOGSA) beat GA, PSO, and GSA[18]. Using a hybrid PSO and the dragonfly algorithm, the test suite was reduced while taking fault coverage into consideration[19]. To prioritise the test cases in PSO, modified precondition decision saturation criteria were employed as fitness measures[20]. The PSO Algorithm was also presented by Deneke et al.[21] for lowering the test suite determined by requirement coverage and cost. Samad et al.[22] employed PSO algorithm with several objectives for code, fault coverage, and cost optimisation was proposed. Using PSO, Agrawal and Kaur selected the test cases according to the defect information[23].

PSO algorithms have moved to cutting-edge level, with promising outcomes in a variety of fields, including CO_2 emission reduction in air luggage systems[24]. On the other hand, the original PSO experienced drawbacks like as being caught in local optima and attaining premature convergence[25]. PSO enhanced and hybrid versions outperformed PSO for complex systems, according to the research[13]. QPSO (Quantum-behaved PSO) is one such algorithm. It is based upon quantum physics, which states that particles can go across a huge search space in quest of global convergence[26]. This approach has produced positive results in a range of applications, including cancer categorization[27], extraction of characteristics[28–30] and constrained engineering problems[31], and others[32].

Following an in-depth review of the most recent advances in TCP techniques, current business practices, and research findings, it is discovered that there is sufficient potential to improve the test case prioritizing strategy in regression by taking test case significant elements into account. As a result, in this work, we propose a Requirement Aware Optimisation of Test Case Selection (R-OTCS) technique based on a Genetic Algorithm that would specifically tailor the test case prioritization (TCP) for the most essential test cases based on the project needs. The APFD matrix is used as a performance evaluation metric to demonstrate the suggested methodology's results.

3. Proposed methodologies

To tackle the TCP problem, we employed GA and evaluated chromosome population, where every chromosome indicates a potential collection of test cases chosen at random from the test suite. This TCP challenge is about TC ordering and is discontinuous in nature. The process starts with population generation. We used permutation encoding to encode the chromosome. The Average Percentage of Fault Identification (APFD) fitness operation is used to assess the fitness of each chromosome in the population. The fitness value, abbreviated as APFD, is the number of errors found by chromosomal when a minimum test suite is executed. The total amount of generations is the termination requirement for terminating iterations. When the termination or converge conditions are not met, chromosomes are subjected to tournament selection. Any pair of chromosomes are picked at random, and they are then subjected to single point crossover, where the site of crossing is also chosen at random. The mutation stage will randomly change chromosomal parts in order to produce better children for the following generation. We conducted the experiment to fine-tune the crossover rate and mutations probability rates. The detailed results are given in the next section. Individuals are chosen based on fitness scores that are more than or equal to 0.4. The generated offspring repeat indefinitely until the convergence requirements are not fulfilled. Algorithm 1 depicts a GA implementation algorithm based on a single goal: Capability to identify flaws. **Figure 1** depicts working flow of GA for TCP. And **Table 1** shows Mapping of TCP problem with GA operators.

Algorithm 1 Fault-based GA implementation for TCP

```
1: Input: T- Original Test Suite, Test cases t1, t2, ..., tn, Seeded Faults f1, f2, …, fm, Fault Matrix
2: Output: Ordered test suite based fitness score, TS'
3: START
4:      Set the chromosomal length to l.
5:      Initialize population size p
6:      Initialize input file as BigFaultMatrix data file or projects data
7:      Permutation Encoding of population
8:          Function Run(n): input number of run
9:      For i up to n number_of_runs
10:         Generate random population from BigFaultMatrix data file
11:         generate_population(population_size p, chromosome_size l)
12:     End For
13:     While (termination criteria is not reached)
14:     Do
15:         Calculate fitness score of population (APFD), ft
16:         Filter individuals having FS > 0.4
17:         Selection of best Parents through tournament selection
18:         S_population:Tournament_selection(population)
19:         Crossover with probability pc
20:         Newpop: Crossover(S_population)
21:             Mutation with probability pm
22:             M_population: Mutation(Newpop)
23:             Decode and fitness calculation ft
24:             Survivor selection
25:             Find best
26:             Return best
27:     End Do
28: End
```

Table 1. Mapping of TCP problem with GA operators.

GA operators	TCP mapping to operators
Individual	Possible solution as test case sequence
Population	Test case sequences (individuals) maintained in a search space
Encoding	Permutation encoding scheme
Selection	Tournament selection
Fitness function	APFD score
Crossover	Single point ordered crossover
Mutation	Swap approach
Convergence criteria	Number of generations

Following **Figure 1** shows flow of GA working for addressing TCP.

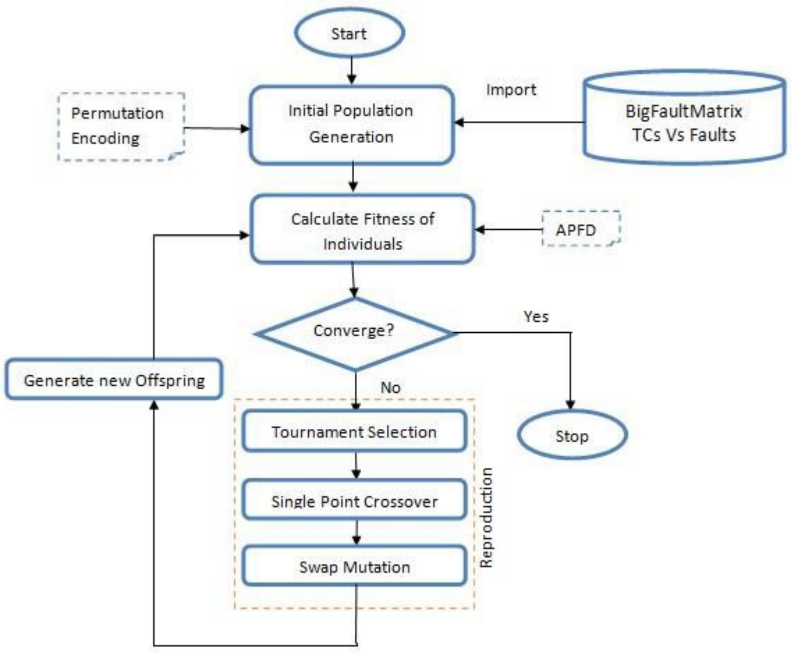

Figure 1. GA working flow for TCP.

GA operators

GA is a population-based random search that is used to solve optimisation problems by looking for the optimal answer from a particular population to another. Natural biology and natural evolution are the foundations of the philosophy.

GA encodes a search issue's decision variables in a finite-length strings of alphabets with a specified cardinality. Chromosomes are strings that indicate possible remedies for the search issue. Unlike traditional search methods, genetic algorithms rely on a set of potential solutions. Once the problem has been stored in a genetic form and a fitness metric for distinguishing excellent solutions from bad ones has been developed, we may begin to develop solutions to the search issue using the methods described below. The genetic algorithm through which events flow is then discussed in detail:

1) Encoding and generate population:

For a particular input parameter or collection of parameters, this represents the solution or chromosome. GA uses metaphor consisting of two distinct elements: i) individual, ii) population. A population is a group of prospective solutions, whereas an initial is just one prospective solution. Initially, the population is made up of randomised test case sequences. Encoding of test case sequences are presented in **Table 2**. We estimated the population to be 200 people. A chromosome defines an individual. A chromosome preserves an individual's genetic makeup (known as phenotypic). The chromosomal length is 50. We utilised ordering GA or permutations GA encoding method since the TCP problem is the ordering of the test instances. Our objective in structuring the review cases is to achieve the greatest fault coverage while executing the fewest number of test cases.

Algorithm 2 Pseudo code to generate initial population

```
1: Generate Population
2: generate_population(population_size p, chromosome_size l)
3: For I upto population_size:
4:     For j upto in chromosome_size:
5:         chromosome: randometest_case from bigfaultmatrix
6:     check for duplicity
7:     End for
8: End for
```

Table 2. Encoding of test case sequence.

I =	[tc0, tc1, tc2, …, tc($n-1$)]							
I =	T1	T3	T4	T7	T5	T2	T6	T8

2) Fitness evaluation:

The purpose of fitness assessment is to determine each individual's survival in the existing population. Individual will be sent into this function, which will bring back the individual's fitness. The defect detection rate is used to compute the fitness value $f(x)$ of each chromosome in the overall solution space. The evaluation approach entails selecting persons for kid reproduction based on their suitability for and value as parents.

Each chromosome represents a series of test cases. To assess each individual's fitness, we employed the APFD function, which computes the sequence's defect detection capabilities.

$$\text{APFD}(T) = 1 - \frac{\sum_{i=1}^{m} TF_i}{nm} + \frac{1}{2n} \tag{1}$$

where,

'T' is test suite under evaluation containing 'n' number of test cases.

'F' is set of faults containing 'm' number of faults detected by T.

'TF_i' is position of TC in prioritization sequence detecting i-th number of fault.

We have considered thresh hold value as 0.4 which acts as an acceptance or rejection of individual. If evaluated APFD is above 0.4 then individual is processed for further selection process else rejected.

3) Selection:

The major goal of the selection step in GA is that "the better an individual is, the more likely it is to be a parent." Before moving on to the converge stage, we must analyse the best option or fittest solution. Our goal with selection is to select individuals from the present population at any given time and then pass on the best answer to the next generation in order to generate superior offspring if the convergence requirements are not fulfilled.

Essentially, the selection strategy adheres to Darwin's premise of preservation of the fittest. A selection strategy in GA is a technique that encourages the choice of better persons in the community for the breeding pool, so that superior genes are passed on to new offspring and thus the search progresses to the global optima.

We adopted the tournament selection approach since it is computationally quicker than both the roulette wheel and the rank-based selection schemes. This selection approach works well when the population has a wide range of fitness values and the pace of convergence is moderate. A excellent individual may be replicated into the population that mates more than once. **Figure 2** depicts example of tournament selection.

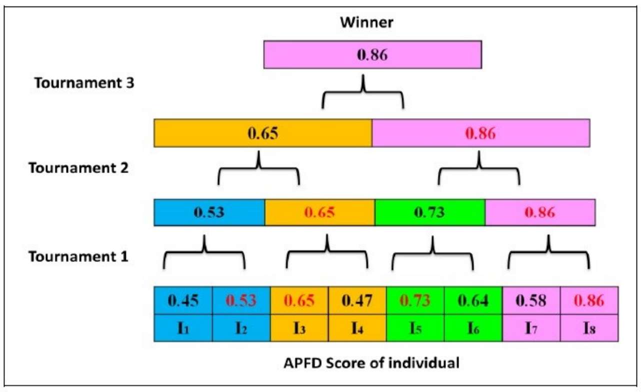

Figure 2. Tournament selection.

Algorithm 3 Pseudo code for selection operator

```
1: Tournament Selection
2: tournament_selection(population):
3:     For i up to population_size():
4:         For j upto tournament_size:
5:             participant: select_random_participant(population)
6:             fitness(Individual)
7:             if fitness > winning_pobability
8:                 winner = Individual
9:             else
10:                remove(Individual)
11:            End if
12:        End for
13: End for
```

4) Crossover rate:

It is a major GA operator that works on people in the mating pool to reproduce new progeny. Crossover is the mutual swapping or swapping of bits that results in the creation of new progeny. This function will be given two solutions as input, which will then swap some genes to generate a new set of chromosomes. The crossover approach is chosen based on the encoding scheme. Mutation and crossover are reproduction operators that work in the mating pool function, where the input is a population and the population meets to form or reproduce a new population.

Mutations and crossover are reproduction operators that operate in the pool of mates function, where the input is a population that meets to establish or reproduce a new population. Individual encoding utilising order or permutations encoding does not function in the same way that binary encoding does. As seen in **Figure 3**, imagine P1 and P2 are two parents picked from the mating pool for the crossover function, which results in two new children C1 and C2 after a single point crossover operation. L is the length of parental P1 and P2 chromosomes.

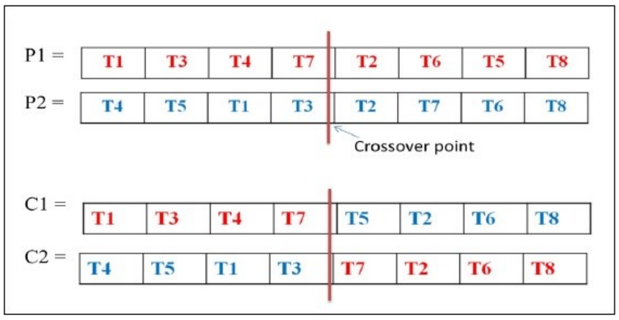

Figure 3. Single point order crossover.

Steps for Single point order crossover:

(1) Create a crossover point K at random such that (1 K L).

(2) Copy P1's left schema into C1 and P2's left schema into C2, where all C1 and C2 are originally empty.

(3) Copy the gene's value from P2 in the identical order as they appear but do not currently exist in the left schema for the schema on the right side of C1.

(4) Repeat the same procedure complete C2 from P1.

5) Mutation rate:

Mutation is a form of genetic operator that is employed in algorithmic genetics to preserve genetic variation from one generation to the next. It is analogous to biological mutation. In GA, the concept of biological mutation is modeled artificially to bring a local change over the current solutions. This stage allows Change in gene within same individual. We induce unpredictability in people after completing single point ordered crossover. **Figure 4** shows swap mutation working. In this scenario, we choose two test cases at random from chromosome 4, say 4th and 7th in the picture, and swap their places.

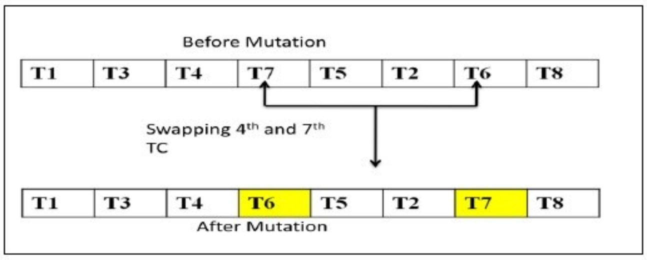

Figure 4. Mutation operator.

6) Convergence:

If we supply the population or generation, it is examined to see if we have reached the end. It serves as a stumbling block for generation reproduction. The number of iterations is used as the convergence criteria for this implemented GA.

4. Results and discussion

GA is adaptable in that it changes depending on the environment/setup operator. GA parameters are critical to the success of the GA outcome. This section conducts tests and provides simulation results created by a mandatory the importance value-based calibration GA solution in order to improve the rate of fault detection as measured by the APFD score.

The performance of the implemented GA is estimated APFD metric.

i) Experimental set up:

The established GA with fault detection aim, as well as fine tuning of its significant operators, is running in the PYTHON platform, and the PC used for simulation has 8 GB RAM, Windows 11 OS, and an Intel I5 CPU.

ii) Database description:

We ran an experiment on three different projects. **Table 3** contains information on all three initiatives.

Table 3. Input project details.

S.No.	System	No of requirements	No of test cases	No of faults
Project 1	Attendance management system	18	123	45
Project 2	Online exam system	21	186	42
Project 3	Real estate system	27	237	52

iii) Evaluation metric:

The APFD metric is a fitness function that is used to determine the fault detection rate score. The overall number of errors found by the prioritised test case order is measured by APFD. APFD is the mean of the two APFD values obtained from OTCS and OTCP.

$$APFD = APFD(OTCS) + APFD(OTCP) \tag{2}$$

Performance analysis on projects:

Three projects are used to assess the performance of the proposed GA-based R-OTCS. The findings for APFD ratings and a unique set of TCs produced after execution are compared.

Performance analysis of project 1:

Project 1 is an attendance management system with a total of 18 criteria built in C++ programming language and over 6 KLOC. The following sequence was obtained after submitting its fault matrix, fault severity, and requirement criticality values in order to obtain an optimised test case execution sequence:

Requirement aware Optimized Test Case Sequence (R-OTCS) = {T3, T11, T110, T76, T100, T9, T114, T121, T104, T98, T70, T105, T10, T115, T79, T77, T17, T107, T117, T49, T93, T12, T108, T91, T83, T88, T122, T21, T22, T50, T68, T38, T26, T27, T120, T34, T35, T99, T36, T181, T5, T190, T142, T102, T95, T48, T71, T72, T69, T53}

Performance analysis of project 2

Project 2 is an online examination system with a total of 21 criteria written in the Python Django programming language and spanning over 9 KLOC. The following sequence was obtained after submitting its fault matrix, fault severity, and requirement criticality values in order to obtain an optimised test case execution sequenc:

Requirement aware Optimized Test Case Sequence (R-OTCS) = {T118, T138, T4, T142, T68, T22, T50, T2, T27, T70, T153, T194, T182, T104, T171, T117, T9, T151, T135, T107, T38, T75, T129, T147, T123, T25, T122, T87, T120, T158, T115, T47, T150, T3, T103, T89, T78, T30, T88, T69, T144, T112, T139, T134, T73, T169, T125, T186, T161, T184, T110, T181, T148, T45, T180, T162, T159, T36, T15, T24, T56, T190, T86, T102, T188, T58, T165, T167, T170, T44, T126, T168, T19, T96, T20, T163, T130, T63, T33, T106}

Performance analysis of project 3:

The third project is a Real Estate System with a total of 27 requirements written in Java and including over 13 KLOC. The following sequence was obtained after submitting its fault matrix, fault severity, and requirements importance values in order to obtain an optimised test case execution sequence:

Requirement aware Optimized Test Case Sequence (R-OTCS) ={T16, T95, T223, T147, T59, T96, T32, T98, T128, T165, T40, T168, T47, T229, T18 , T38, T212, T33, T202, T222, T163, T45, T233, T8, T182, T184, T44, T50, T5, T208, T97, T110, T191, T119, T138, T189, T42, T83, T56, T226, T148, T155, T36, T181, T12, T117, T90, T4, T156, T139, T172, T125, T137, T115, T131, T162, T116, T218, T124, T80, T60, T113, T203, T157, T76, T41, T167, T192, T166, T63, T199, T160, T173, T28, T85, T49, T69, T72, T91, T51, T104, T105, T78, T161, T61, T123, T108, T14, T154, T183, T193, T237, T86, T174, T75, T159, T9, T142, T149, T2, T152, T140, T21, T126, T35}

The performance analysis for R-OTCS is summarized and documented numerically in **Table 4**.

Table 4. Summarized performance analysis for R-OTCS.

S/No.	No of TC by faults	OTCS: No of unique TC's	OTCP: No of unique TC's	Total TC	% of TC Executed	% APFD (OTCS)	% APFD (OTCP)	% Avg APFD
Project 1	123x45	28	51	79	64.23	98.52	72.26	85.39
Project 2	186x42	22	80	102	54.84	97.71	82.26	89.995
Project 3	237x52	34	105	139	58.65	91.03	71.50	81.265

The Average APFD resulted for project 1, project 2 and project 3 are 85.39, 89.985 and 81.265.

When we plot the graph to confirm the findings of the analysis using the amount of test cases run (as shown in **Figure 5**), we discover that 64.22% of the test instances in project 1 were optimised using our recommended R-OTCS approach. In projects 2 and 3, the optimised test case percentages are 54.33% and 58.64%, respectively. So, looking at the percentage of test case optimisation, we obtained roughly 55% to 65% in all three projects by combining our proposed R-OTCS technique with GA optimisation, which is a decent ratio. **Figures 6** and **7** also demonstrate the performance evaluation of test cases optimisation for all three projects, with results ranging from 55% to 65% for each. We may deduce from this that our suggested strategy achieves a reasonable optimisation ratio for test case prioritisation.

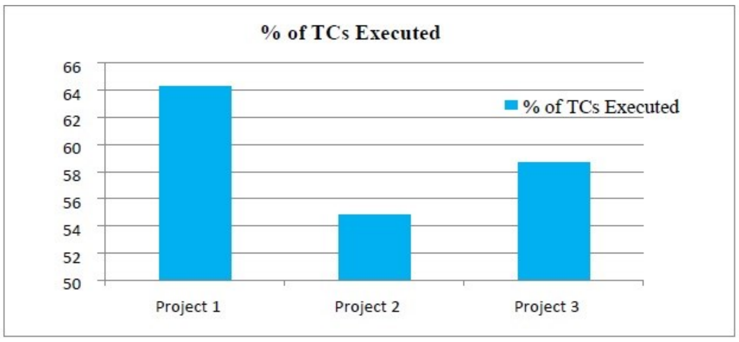

Figure 5. Performance analysis using No. of test cases executed.

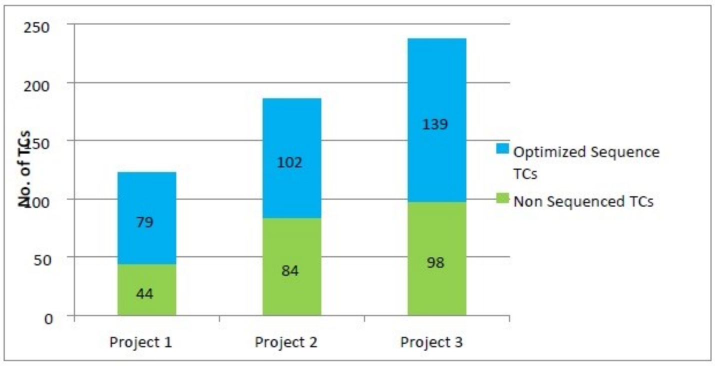

Figure 6. Performance analysis using % of TC execution.

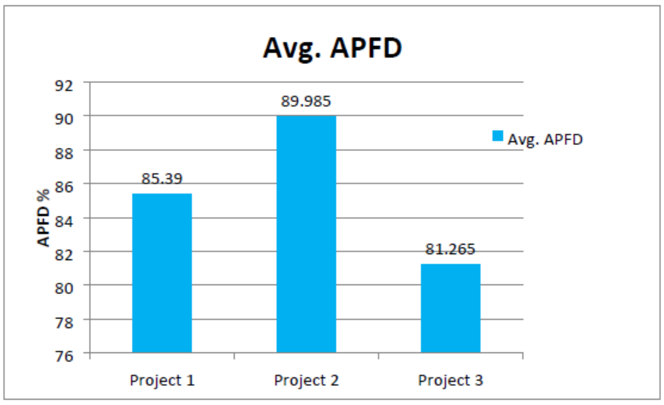

Figure 7. Performance analysis using average APFD score.

5. Conclusion

This module is primarily a test case selection strategy, with the goal of providing TCs that cover important business vital capabilities. Business important functions can be either core or secured business functions. Because they are core defects, TCs implementing these capabilities are more likely to encounter them, and the cost of repairing them is higher than for other errors. Thus, by carrying out such TCs, AUT dependability will improve. We used the GA optimise strategy because there are a lot of TCs here. This module operates in two stages. First, we calibrated the GA to solve the TCP problem by fine-tuning its settings. TCs are rated in phase two based on the required critical factor. For the best results, APFD is assessed fitness function. The APFD values provided by this module range from 80% to 90%.

Author contributions

Conceptualization, VK and RM; methodology, VK; software, VK; validation, VK, RM and HVR; formal analysis, VK; investigation, VK and HVR; resources, VK; data curation, VK and HVR; writing—original draft preparation, VK and RM; writing—review and editing, VK and, RM and HVR; visualization, VK and RM; supervision, RM and HVR; project administration, VK; funding acquisition, RM. All authors have read and agreed to the published version of the manuscript.

Conflict of interest

The authors declare no conflict of interest.

References

1. McDermid J. Book review: Software Engineering: A Practitioner's Approach. Software Engineering Journal. 1995; 10(6): 266. doi: 10.1049/sej.1995.0031
2. Patton R. Software testing. 2001. p. 389.
3. Chittimalli PK, Harrold MJ. Recomputing Coverage Information to Assist Regression Testing. IEEE Transactions on Software Engineering. 2009; 35(4): 452-469. doi: 10.1109/tse.2009.4
4. Elbaum S, Kallakuri P, Malishevsky A, et al. Understanding the effects of changes on the cost-effectiveness of regression testing techniques. Software Testing, Verification and Reliability. 2003; 13(2): 65-83. doi: 10.1002/stvr.263
5. Li Z, Harman M, Hierons RM. Search Algorithms for Regression Test Case Prioritization. IEEE Transactions on Software Engineering. 2007; 33(4): 225-237. doi: 10.1109/tse.2007.38
6. Zhang W, Qi Y, Zhang X, et al. On Test Case Prioritization Using Ant Colony Optimization Algorithm. 2019 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 (HPCC/SmartCity/DSS). Published online August 2019. doi: 10.1109/hpcc/smartcity/dss.2019.00388
7. Bajaj A, Sangwan OP. Discrete cuckoo search algorithms for test case prioritization. Applied Soft Computing. 2021; 110: 107584. doi: 10.1016/j.asoc.2021.107584
8. Ahmed BS. Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing. Engineering Science and Technology, an International Journal. 2016; 19(2): 737-753. doi: 10.1016/j.jestch.2015.11.006
9. Mohapatra SK, Prasad S. Test Case Reduction Using Ant Colony Optimization for Object Oriented Program. International Journal of Electrical and Computer Engineering (IJECE). 2015; 5(6): 1424. doi: 10.11591/ijece.v5i6.pp1424-1432
10. Zhang YN, Yang H, Lin ZK, et al. A Test Suite Reduction Method Based on Novel Quantum Ant Colony Algorithm. 2017 4th International Conference on Information Science and Control Engineering (ICISCE). Published online July 2017. doi: 10.1109/icisce.2017.176
11. Mondal D, Hemmati H, Durocher S. Exploring Test Suite Diversification and Code Coverage in Multi-Objective Test Case Selection. In: Proceedings of the 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST). doi: 10.1109/icst.2015.7102588
12. Khatibsyarbini M, Isa MA, Jawawi DNA. Particle Swarm Optimization for Test Case Prioritization Using String Distance. Advanced Science Letters. 2018; 24(10): 7221-7226. doi: 10.1166/asl.2018.12918
13. de Souza LS, Prudêncio RBC, Barros F de A, et al. Search based constrained test case selection using execution effort. Expert Systems with Applications. 2013; 40(12): 4887-4896. doi: 10.1016/j.eswa.2013.02.018
14. de Souza LS, Prudencio RBC, Barros F de A. A Hybrid Binary Multi-objective Particle Swarm Optimization with Local Search for Test Case Selection. In: Proceedings of the 2014 Brazilian Conference on Intelligent Systems. doi: 10.1109/bracis.2014.80
15. de Souza LS, Cavalcante Prudêncio RB, de Barros FA. A hybrid particle swarm optimization and harmony search algorithm approach for multi-objective test case selection," J. Brazilian Comput. Soc. 2015; 21(1): 1-20. doi: 10.1186/S13173-015-0038-8/TABLES/4
16. Correia D. An industrial application of test selection using test suite diagnosability. In: Proceedings of the 2019

27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. doi: 10.1145/3338906.3342493

17. Bajaj A, Abraham A. Prioritizing and Minimizing the Test Cases using the Dragonfly Algorithms. 2021; 13: 62-71.
18. Bajaj A, Sangwan OP. Tri-level regression testing using nature-inspired algorithms. Innovations in Systems and Software Engineering. 2021; 17(1): 1-16. doi: 10.1007/s11334-021-00384-9
19. Bharathi M. Hybrid Particle Swarm and Ranked Firefly Metaheuristic Optimization-Based Software Test Case Minimization. International Journal of Applied Metaheuristic Computing. 2021; 13(1): 1-20. doi: 10.4018/ijamc.290534
20. Nayak G, Ray M. Modified condition decision coverage criteria for test suite prioritization using particle swarm optimization. Int. J. Intell. Comput. Cybern. 2019; 12(4): 425-443. doi: 10.1108/IJICC-04-2019-0038/FULL/XML
21. Deneke A, Gizachew Assefa B, Kumar Mohapatra S. Test suite minimization using particle swarm optimization. Materials Today: Proceedings. 2022; 60: 229-233. doi: 10.1016/j.matpr.2021.12.472
22. Samad A, Mahdin HB, Kazmi R, et al. Multiobjective Test Case Prioritization Using Test Case Effectiveness: Multicriteria Scoring Method. Ali S, ed. Scientific Programming. 2021; 2021: 1-13. doi: 10.1155/2021/9988987
23. Agrawal AP, Kaur A. A comprehensive comparison of ant colony and hybrid particle swarm optimization algorithms through test case selection. Adv. Intell. Syst. Comput. 2018; 542: 397-405. doi: 10.1007/978-981-10-3223-3_38/COVER
24. Lodewijks G, Cao Y, Zhao N, et al. Reducing CO_2 Emissions of an Airport Baggage Handling Transport System Using a Particle Swarm Optimization Algorithm. IEEE Access. 2021; 9: 121894-121905. doi: 10.1109/access.2021.3109286
25. Sun J, Xu W, Feng B. A global search strategy of quantum-behaved particle swarm optimization. IEEE Conference on Cybernetics and Intelligent Systems, 2004. doi: 10.1109/iccis.2004.1460396
26. Lukemire J, Mandal A, Wong WK. d-QPSO: A Quantum-Behaved Particle Swarm Technique for Finding D-Optimal Designs with Discrete and Continuous Factors and a Binary Response. Technometrics. 2018; 61(1): 77-87. doi: 10.1080/00401706.2018.1439405
27. Iliyasu A, Fatichah C. A Quantum Hybrid PSO Combined with Fuzzy k-NN Approach to Feature Selection and Cell Classification in Cervical Cancer Detection. Sensors. 2017; 17(12): 2935. doi: 10.3390/s17122935
28. Peng C, Yan J, Duan S, et al. Enhancing Electronic Nose Performance Based on a Novel QPSO-KELM Model. Sensors. 2016; 16(4): 520. doi: 10.3390/s16040520
29. Guo X, Peng C, Zhang S, et al. A Novel Feature Extraction Approach Using Window Function Capturing and QPSO-SVM for Enhancing Electronic Nose Performance. Sensors. 2015; 15(7): 15198-15217. doi: 10.3390/s150715198
30. Wen T, Yan J, Huang D, et al. Feature Extraction of Electronic Nose Signals Using QPSO-Based Multiple KFDA Signal Processing. Sensors. 2018; 18(2): 388. doi: 10.3390/s18020388
31. dos Coelho LS. Gaussian quantum-behaved particle swarm optimization approaches for constrained engineering design problems. Expert Systems with Applications. 2010; 37(2): 1676-1683. doi: 10.1016/j.eswa.2009.06.044
32. Omkar SN, Khandelwal R, Ananth TVS, et al. Quantum behaved Particle Swarm Optimization (QPSO) for multi-objective design optimization of composite structures. Expert Systems with Applications. 2009; 36(8): 11312-11322. doi: 10.1016/j.eswa.2009.03.006