Test Case Optimization and Prioritization Based on Multi-objective Genetic Algorithm

Deepti Bala Mishra, Rajashree Mishra, Arup Abhinna Acharya and Kedar Nath Das

Abstract The validation of modified software depends on the success of Regression testing. For this, test cases are selected in such a way that can detect a maximum number of faults at the earliest stage of software development. The selection process in which the most beneficial test case are executed first is known as test case prioritization which improves the performance of execution of test cases in a specific or appropriate order. Many optimizing techniques like greedy algorithm, genetic algorithm, and metaheuristic search techniques have been used by many researchers for test case prioritization and optimization. This research paper presents a test case prioritization and optimization method using genetic algorithm by taking different factors of test cases like statement coverage data, requirements factors, risk exposure, and execution time.

Keywords Regression testing · Test case prioritization · Risk exposure Requirement factor · Genetic algorithm

D. B. Mishra · A. A. Acharya
School of Computer Engineering, KIIT University, Bhubaneswar 751024, India
e-mail: dbm2980@gmail.com

A. A. Acharya
e-mail: aacharyafcs@kiit.ac.in

R. Mishra (✉)
School of Applied Sciences, KIIT University, Bhubaneswar 751024, India
e-mail: rajashreemishra011@gmail.com

K. N. Das
Department of Mathematics, NIT Silchar, Silchar, Assam, India
e-mail: kedar.iitr@gmail.com

N. Yadav et al. (eds.), *Harmony Search and Nature Inspired Optimization Algorithms*,
Advances in Intelligent Systems and Computing 741,
https://doi.org/10.1007/978-981-13-0761-4_36

1 Introduction

Nowadays, we are surrounded by many automated software. Qualitative, robust, and trustworthy software are maintained by successful testing as it is the most important phase of Software Development Life Cycle (SDLC) [1]. In the maintenance phase when Software Under Test (SUT) is modified, we must ensure that the software remains defect free and for this regression testing is required which is a process of retesting of whole software and it is frequently performed on the altered version of software for checking the validity. The same process needed more time, cost and recourses during regression testing, so it is not always possible to run the whole test cases again and again [2]. To avoid this problem, we need to prioritize the test cases in such a manner that the most priority test cases are executed first than the lower one and sometimes low priority test cases are need not be executed. The priority criteria depend on the different factors of test cases [3].

The rate of risk can be identified during regression testing so that the debugging process begins as soon as possible and high-risk faults are detected in testing life cycle [4, 5]. Test case prioritization improves the cost-effectiveness of regression testing by reordering the most important test cases to be executed very first. It also increases the probability of running of the most beneficial test cases if the testing process complete before the stipulated time [6, 7]. Several test case prioritization and optimization techniques are developed by applying metaheuristic search techniques. Genetic algorithm (GA) is one of the optimizing techniques used to solve different optimization problem in software engineering field as it gives an exact fitness value for each and every test case of a specific SUT. Based on the fitness value, the test cases are prioritized [8, 9].

In this research paper, a test case prioritization method is developed based on requirement priority, risk exposures, statement coverage, and execution time associated with test cases for a specific SUT within a given time constraint. The proposed technique uses a multi criteria based GA for prioritization and optimization of test cases by considering a stipulated time to execute test cases. The rest of the paper is organized as: Sect. 2 describes test case prioritization for regression testing, Sect. 3 discusses related work on test case prioritization and minimization using GA. A brief outline of GA is drawn in Sect. 4 and in Sect. 5 the proposed algorithm for prioritization is described. In Sect. 6, the proposed algorithm is implemented on a case study and Sect. 7 discusses various factors taken for the proposed algorithm. Section 8 describes the experimental setup and result analysis for the case study taken. Finally, the conclusion of the paper is drawn in Sect. 9 followed by some future works.

2 Test Case Prioritization for Regression Testing

Regression testing is done to make sure that any type of enhancement made to existing software does not impact the previous functionality. It also ensures that the existing

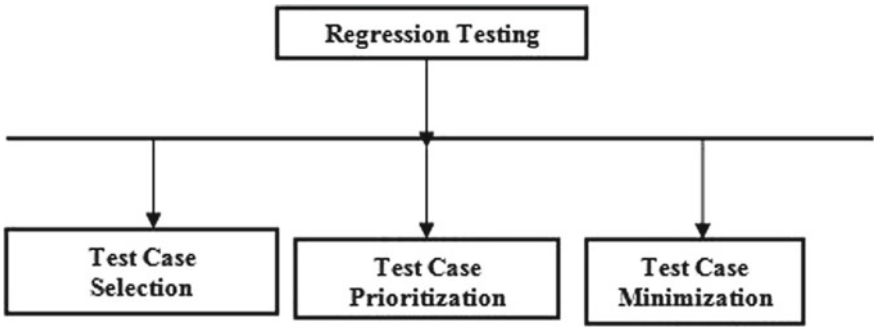

Fig. 1 Various activities in regression testing

bug does not result in new bugs at the time of software modification [10]. During regression testing all test cases are needed to run again with the new test cases created for modified version. So it is very time-consuming and an expensive task to reexecute all test cases again. To avoid this nonviable situation various activities are carried out during Regression testing shown in Fig. 1 [11]. Test case selection and prioritization provides the facility to execute a less numbers of test cases for maximum coverage of the software, in such a manner that the most important test cases are executed first than the others [8]. Through minimization, the redundant or obsolete test cases are eliminated and hence the minimization techniques lower the cost of regression testing by reducing a test suite to a minimal subset [5].

2.1 Problem Identification for Test Case Minimization and Prioritization

Test Case Minimization Problem [12]:

Given: A test suite ***Ts*** and a set of test case requirements r_1, r_2, r_3, …, r_n that must be satisfied to provide the testing coverage of a software. The subsets of ***Ts***, T_1, T_2, T_3, …, T_n are associated with Traceability matrix, in such a way that each test case T_j belongs to T_i can be used to test r_i.

Problem: We have to find a representative set of test cases from ***Ts*** that satisfies all the r_i's, where the r_i's can represent either all the requirements of the program or the requirements related to the modified program.

Test case Prioritization problem [12]:

Given: A test suite T and $T_{\mathbf{order}}$ refers to a number of ways that test cases are chosen. Fitness f of T is calculated depending on some criteria to a real value.

Problem: We have to find T' in such a way that $T' \in T_{\mathbf{order}}$ for all T, where $T \mathrel{!}= T'$ and $f\ (T') >= f\ (T)$.

3 Related Work

Kaur and Goyal [13] proposed a new GA based method to prioritize the test cases by taking the complete code coverage. They have used APCC metric to represent the effectiveness of their result and analyzed different prioritized approaches. Konsaard et al. [9] proposed total coverage based regression test case prioritization using GA. Modified GA is used to simplify and it has the ability to change the population, that supply a number of test cases used for prioritization process. Amr et al. [14], presented an approach for automatic test case generation and prioritization using GA. They have used multicriteria fitness function which evaluates the multiple control flow data. They have taken different factors for prioritization such as coverage based data, faults detected by test cases and their severity value. Their comparative result showed superior to other similar work done previously. Prakash and Gomathi [1] developed a multiple criteria coverage method for test case prioritization to improve test efficiency by taking average information to prioritize test cases and found their proposed method improves the performance of regression testing and the rate of fault detection capacity for various SUT. Sharma and Sujata [15] defined an effective model-based approach to generate and prioritize the effective test cases. They have used GA to generate effective test paths based on the requirement and user view analysis. They have taken cost factor for a specific model and estimate the overall cost to test the functional behavior of the model. Kumar et al. [16] proposed a prioritization technique based on requirement analysis such as requirement priority and requirement factor with varying nature. Their proposed system improves the testing process by ensuring the quality, cost, effort of the software and the user's satisfaction. Rhmann et al. [5] presented an approach for test came minimization and prioritization by taking several factors of software projects under test. The selection of test cases is based on given time constraints. They have used a novel approach of 0–1 integer programming to model the fitness.

4 Genetic Algorithm (GA)

GA is an evolutionary search technique used to solve many optimization problems. It is inspired by the biological concept of evolution and based on "surviving the fittest" [17]. The algorithm starts with the process of random generation of populations depending on the specified problem domain. Then the basic operators such as selection, crossover, mutation, and elitism are applied on the initial population after evaluation of fitness. The same process is repeated until reach an optimal solution. In software engineering field, it is used to solve many complex and real-life problems by producing high-quality test data automatically during testing phase [18].

In this research paper, permutation encoding [19] is used to create chromosomes for the proposed method shown in Fig. 2. Average crossover [14] and insertion mutation operators [20] are used to find the new offspring chromosome.

Fig. 2 Permutation encoding

Chromosome 1: 1, 2, 3, 4
Chromosome 2: 2, 3, 4, 1

4.1 Average Crossover

Average crossover takes two parents to perform crossover and creates only one offspring by finding the average of two parents [14]. New offspring can be found by averaging the genes of chromosomes of Fig. 2 and the resultant offspring is shown in Fig. 3.

4.2 Insertion Mutation

In the proposed method, mutation is performed on a particular chromosome by changing the gene position in that chromosome. The position of the gene is arbitrarily changed by reinserting in a new position with a small mutation probability factor [20]. An example of insertion mutation is shown in Fig. 4.

4.3 Fitness Function

The fitness function evaluates individual's performance. Based on the fitness value, the individuals with higher are selected to the next generation for better optimum solution [21]. The fitness is defined in Eq. (1), which is based on total statement coverage, total risk exposure, requirement priority values, and the time of execution of test cases.

Fig. 3 New offspring after crossover

Fig. 4 Insertion mutation

Initial Chromosome: 1, 2, 3, 4
Mutated Chromosome: 1, 2, 3, 4
New Chromosome after Mutation: 1, 3, 2, 4

5 Proposed Algorithm for Prioritization

This section describes the proposed algorithm to prioritize test cases and the factors required for prioritization. To implement the proposed method, a case study as Accept Saving Details for an Income Tax Calculator [11] is taken and GA is used to optimize test cases and further, we prioritized those test cases based on their fitness. We have taken some test case-oriented factors for prioritization such as requirement value, statement coverage, and test case execution time during regression testing.

Prioritization and Minimization Algorithm:

Step 1. Create Initial Population i.e. Chromosomes ($C1, C2, C3, \ldots, Cn$)
Step 2. Initialize the Population as Test Suites
Test Suite = No. of Chromosomes
Step 3. Calculate Fitness
Step 4. Select Best Two Populations Based on Fitness
(Best 2 Chromosomes)
Step 5. Apply Cross Over Operator
Step 6. Apply Insertion Mutation on the new Chromosome
Step 7. Remove the Duplicates
Step 8. Check for the Multi objective fitness

```
If (Solution = Feasible)
    Print the Optimized Solution
Else
    Go to Step 1
End
```

6 Case Study: Accept Saving Details for an Income Tax Calculator

The proposed approach is implemented on a small java program as Accept Saving for an Income Tax Calculator [11] shown in Fig. 5. The program takes 3 arguments as Account number, Account type and the Amount to deposit in his/her saving account. The correct values for different variables are given below. Test cases are generated and listed in Table 1.

1. Account No—12345
2. Account type—"Saving"
3. Amount—Positive integer with 2 decimal points.

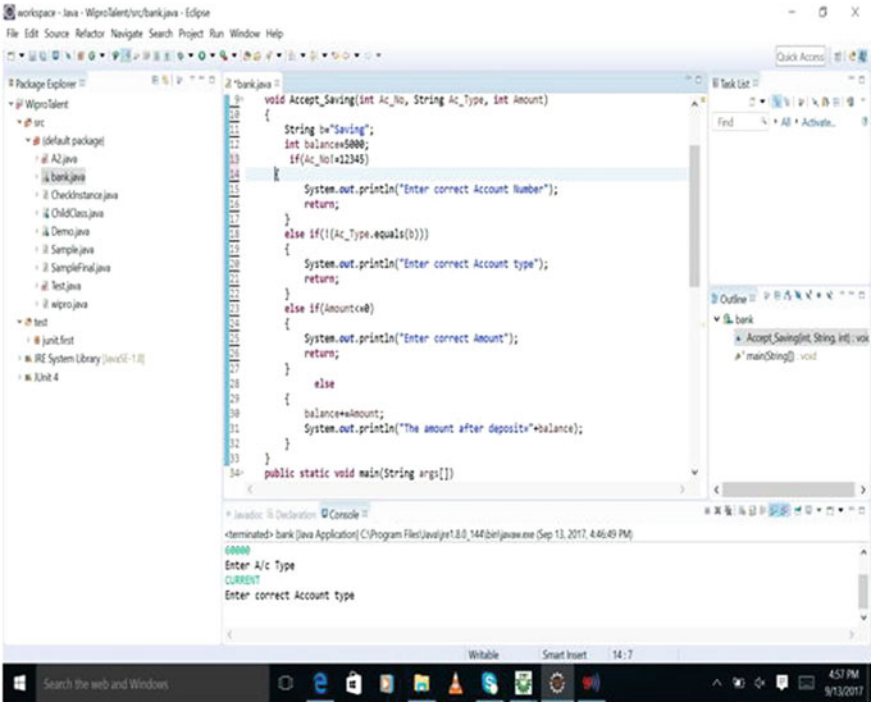

Fig. 5 Java program for Accept_saving

Table 1 Test cased generated for Accept_saving

Test case id	Input data	Output
Tl	(111, 'Savin2', 0)	Enter correct account number
T2	(12345, 'Current', 0)	Enter correct account type
T3	(12345, 'Saving', 0)	Enter correct amount
T4	(12345, 'Saving', 5000.00)	Print the balance

7 Factors Considered for Prioritization

The different factors like total statement coverage, requirement priority factor value, and total risk exposure value are considered for prioritization. For minimization, the total execution time has been taken as one of the proposed multi-objective constraint.

7.1 Requirement Priority Factor

From the Java coding shown in Fig. 5, the following requirements are needed:

Table 2 The priority factor value for requirements

Requirement	Manager	Developer	Customer	Total
R1	10	10	10	30
R2	9	9	10	28
R3	7	7	9	23
R4	8	8	9	25

- R1—If the user inputs wrong account no then the message should display to input correct account number.
- R2—For wrong account type input the proper message should display.
- R3—If the user inputs a wrong amount the appropriate message should display to input correct value.
- R4—Finally if accept saving is successful then the message should display indicating the operation success and display the total balance.

The requirement priority factor values are assigned by manager, developers, and customers from 1 to 10 depending on their importance. It may be same or different for different persons and the priority values of different requirements are recorded in Table 2.

7.2 Risk Exposure

In software development life cycle, each module is tested by analyzing the potential of risks. The testers use risk analysis to select most crucial test cases. So test cases are prioritized by some potential problems occurs during software development. There are mainly four types of risks which may occur during software development [5], such as loss of power (LP), corrupt file data (CFD), unauthorized user access (UUA), and slow throughput (ST). Risk exposure can be calculated using Eq. (1)

$$\text{Risk Exposure} = \text{UFP} \times \text{RI} \tag{1}$$

where UFP is the uncertainty factor of the potential problem with a scale of 1 (low) to 10 (high) and RI is the risk impact values from 1 (low) to 10 (high). The risk exposure for each requirement is computed by using Eq. 1 and shown in Table 3.

7.3 Total Statement Coverage

It counts the number of statements covered by each test case. In our case study, the statement coverage of test cases $T1$, $T2$, $T3$ and $T4$ is (5, 7, 9, 12) respectively. Table 4 shows the total number of statement covered and the time of execution of

Table 3 Total risk exposure of each requirement

Requirement	Uncertainty factor/risk impact	Potential factor value				Risk exposure
		LP	CFD	UUA	ST	
R1	UF	4	5	7	10	226
	RI	9	8	10	8	
R2	UF	8	9	6	3	155
	RI	7	3	9	6	
R3	UF	6	4	8	5	163
	RI	3	7	9	9	
R4	UF	6	7	9	7	211
	RI	9	6	5	10	

Table 4 Statement coverage of different test cases

Test case	Total statement covered	Total execution time
T1	5	15
T2	7	21
T3	9	27
T4	12	36

each test case according to the java code for Accept_ Details ( ) shown in Fig. 5. The execution time for each statement is taken as 3 milliseconds (ms).

8 Experimental Setup and Result Analysis

The initial population is created by using permutation encoding and the fitness function is designed keeping in mind the factors such as total statement coverage, requirement priority factor value and execution time, which is shown in Eq. (2). The test case weight is divided by the order of the test case to decrease the weight when the order of a particular test case increased. The test suite with high fitness value will be selected for next generation. Next crossover and mutation operators are applied to the selected chromosomes to generate new offspring (test suite). Finally, the number of test cases is minimized by deleting the duplicates which gives the optimum result.

$$\text{Maximize } F(x) = \sum_{i=1}^{n} \frac{\text{TCW}_i}{\text{TCO}_i}$$

$$\text{Subject to: } T1 + T2 + T3 + T4 \leq 80 \tag{2}$$

$$\text{TCW} = \text{Total Statement Covered} + \text{Requirement Priority Factor} + \text{Risk Exposure} \tag{3}$$

Table 5 Prioritized test cases and their coverage

Test case factors (coverage)	Test case			% covered
	T1	T4	T3	
Total statement covered	5	12	9	93.33%
Requirement covered	30	25	23	74%
Risk covered	226	211	163	79.47%
Total execution time	15	36	27	≤80 ms

where TCW represents the test case weight and it is calculated by using Eq. (3), n represent the total number of test cases and TCO_i indicates the test case order of ith test case. Table 5 shows the resultant test case after minimization by considering the time factor and we get the prioritization order of test cases as $T1$, $T4$ and $T3$ according to the total factors covered by each test case.

9 Conclusion

In this paper, a new approach is used for test case prioritization by considering total statement coverage, requirement priority factor value, risk exposure, and execution time. The proposed method is implemented on a small case study namely Accept Saving Details for an Income Tax Calculator. Further, the test cases are optimized based on the time constraints.

In future, it is planned to implement the proposed algorithm to prioritize and optimize the test cases for large and complex software.

References

1. Prakash, N., Gomathi, K.: Improving test efficiency through multiple criteria coverage based test case prioritization. Int. J. Sci. Eng. Res. (2014)
2. Raju, S., Uma, G.V.: Factors oriented test case prioritization technique in regression testing using genetic algorithm. Eur. J. Sci. Res. **74**(3), 389–402 (2012)
3. Krishnamoorthi, R., Mary, S.S.A.: Factor oriented requirement coverage based system test case prioritization of new and regression test cases. Inf. Softw. Technol. **51**(4), 799–808 (2009)
4. Stallbaum, H., Metzger, A., Pohl, K.: An automated technique for risk-based test case generation and prioritization. In: Proceedings of the 3rd International Workshop on Automation of Software test, pp. 67–70. ACM (2008)
5. Rhmann, W., Zaidi, T., Saxena, V.: Test cases minimization and prioritization based on requirement, coverage, risk factor and execution time

6. Jeyaprakash, S., Alagarsamy, K.: A distinctive genetic approach for test-suite optimization. Procedia Comput. Sci. **62**, 427–434 (2015)
7. Acharya, A.A., Mohapatra, D.P., Panda, N.: Model based test case prioritization for testing component dependency in cbsd using uml sequence diagram. IJACSA Int. J. Adv. Comput. Sci. Appl. **1**(6), 108–113 (2010)
8. Sharma, N., Purohit, G.N.: Test case prioritization techniques "an empirical study". In: 2014 International Conference on High Performance Computing and Applications (ICHPCA), pp. 1–6. IEEE (2014)
9. Konsaard, P., Ramingwong, L.: Total coverage based regression test case prioritization using genetic algorithm. In: 2015 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), pp. 1–6. IEEE (2015)
10. Kavitha, R., Sureshkumar, N.: Test case prioritization for regression testing based on severity of fault. Int. J. Comput. Sci. Eng. **2**(5), 1462–1466 (2010)
11. Ansari, A., Khan, A., Khan, A., Mukadam, K.: Optimized regression test using test case prioritization. Procedia Comput. Sci. **79**, 152–160 (2016)
12. Chauhan, N.: Software Testing: Principles and Practices. Oxford University Press, Oxford (2010)
13. Kaur, A., Goyal, S.: A genetic algorithm for regression test case prioritization using code coverage. Int. J. Comput. Sci. Eng. **3**(5), 1839–1847 (2011)
14. Umbarkar, A.J., Sheth, P.D.: Crossover operators in genetic algorithms: a review. ICTACT J. Soft Comput. **6**(1) (2015)
15. Sharma, N., Sujata, M.: Model based test case prioritization for cost reduction using genetic algorithm
16. Kumar, A., Gupta, S., Reparia, H., Singh, H.: An approach for test case prioritization based upon varying requirements. Int. J. Comput. Sci. Eng. Appl. **2**(3), 99 (2012)
17. Goldberg, D.E.: Genetic Algorithms: In Search, Optimization and Machine Learning. Addison Wesley, MA (1989)
18. Mishra, D.B., Mishra, R., Das, K.N., Acharya, A.A.: A systematic review of software testing using evolutionary techniques. In: Proceedings of Sixth International Conference on Soft Computing for Problem Solving, pp. 174–184. Springer, Singapore (2017)
19. Ahmed, A.A., Shaheen, M., Kosba, E.: Software testing suite prioritization using multi-criteria fitness function. In: 2012 22nd International Conference on Computer Theory and Applications (ICCTA), pp. 160–166. IEEE (2012)
20. Boopathi, M., Sujatha, R., Kumar, C.S., Narasimman, S.: The mathematics of software testing using genetic algorithm. In: 2014 3rd International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), pp. 1–6. IEEE (2014)
21. Mishra, D.B., Bilgaiyan, S., Mishra, R., Acharya, A.A., Mishra, S.: A review of random test case generation using genetic algorithm. Indian J. Sci. Technol. **10**(30) (2017)