3D Res (2018) 9:32
https://doi.org/10.1007/s13319-018-0182-y

3DR EXPRESS

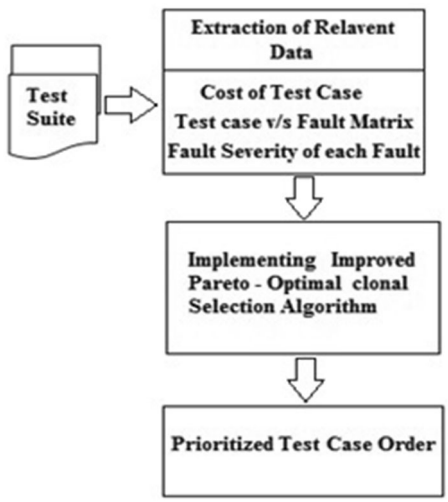
Multi-objective Test Case Prioritization Using Improved Pareto-Optimal Clonal Selection Algorithm

Megala Tulasiraman · Nivethitha Vivekanandan · Vivekanandan Kalimuthu

Received: 26 April 2018 / Revised: 26 June 2018 / Accepted: 29 June 2018 / Published online: 23 July 2018

Abstract Regression test plays a vital role in software testing by ensuring the quality and stability of the developed software. During regression test a large number of test cases are involved thus making the process expensive and difficult. In order to reduce the cost and time of regression test, test case prioritization is applied. However in real world scenario multiple testing criteria and constraint are evolved, such as to detect all faults within minimum time, and to detect most severe faults earlier. This takes the test case prioritization problem turn into multi-objective test case prioritization paradigm. In this paper an improved pareto-optimal clonal selection algorithm is proposed to generate test case order depending on three objective such as minimum execution time, maximum severity fault identification and cost-cognizant average percentage of fault detected. The experimental analysis is conducted over an industrial project with seven different versions for which the proposed approach generates scheduled test case order. And it is concluded that the performance of proposed approach is better than other tested algorithms like random approach, weighted genetic algorithm, greedy and NSGA-II.

Graphical Abstract



Keywords Pareto-optimal · Multi-objective test case prioritization · Clonal selection algorithm

M. Tulasiraman (✉) · V. Kalimuthu
Department of Computer Science, Pondicherry Engineering College, Pondicherry, India
e-mail: Megelaganesh@pec.edu

V. Kalimuthu
e-mail: kvivekanandan@pec.edu

N. Vivekanandan
Department of Computer Science, Vel Tech Institute of Science and Technology, Avadi, Chennai, India
e-mail: nive.nivethitha@gmail.com

1 Introduction

In recent software development, the software product developed undergoes frequent functionality updates

and modification to meet customers demand. The test engineers are in need to ensure that the functionality updates and modification do not affect the other unmodified part of software. Regression testing is one of the software testing activities guarantees that the recent change introduced to the software do not affect the unchanged part [13]. The basic approach followed in regression test is to run the entire test suite to identify faults. Since running entire test suite is time consuming and expensive, test case management technique such as test case minimization, selection and prioritization has emerged. The test case minimization and selection technique selects a subset of test cases from entire test suite to reduce the complexity of regression test, but at times it might reduce the fault detecting capability due to reduction of test case. In test case prioritization the test cases are scheduled in an order to meet certain performance goal without reducing the number of test cases.

Earlier, the objective of test case prioritization is to find out an order that exposes the faults as earlier as possible [10]. However the faults considered are of equal severity and execution time. In real world scenario, multiple test criteria and constraints are involved in testing. Since there is no specified criteria for ideal testing different criteria like code coverage, fault detection, requirement coverage have been widely used [14]. Apart from testing criteria, there also exists multiple constraints. Most of the previous research considered cost as important constraint. Cost can be measured in terms of execution time, human effort, hardware resource but predominantly execution time is considered for cost. There also situation arises involving multiple conflicting testing criteria for instance, in a resource constrained environment, a tester is in need to find out faults with higher severity within minimum execution time which forms a multi objective test case prioritization problem. Previously most of existing prioritization were single objective approaches and very few with multi-objective test case prioritization. Malishevsky et al. [2] formulated two—objective test case prioritization considering code coverage and cost. Harman et al. [7] proposed the first multi-objective test suite selection and minimization problem. In this paper we propose improved pareto-optimal clonal selection algorithm (I-POCSA) for solving multi objective test case prioritization. The proposed algorithm combines pareto-optimality approach with clonal selection algorithm to solve conflicting objective effectively. Clonal selection algorithm have produced better result for multi-objective problem from various fields and also the representation of proposed problem in clonal selection algorithm is much easier. Therefore this gives the motivation for choosing clonal selection algorithm. The contribution of the paper is as follows

- Clonal selection algorithm is introduced for the first time in solving multi-objective test case prioritization
- An improved pareto-optimal clonal selection algorithm for solving test case prioritization with three objective formulation that caters for execution time, fault severity and APFDc is proposed
- The paper compares the proposed algorithm performance with random algorithm, greedy algorithm, weighted genetic algorithm and NSGA-II and the result show that the proposed approach outperform the other algorithm

The rest of the paper is organized as follow: Sect. 2 review the related works and in Sect. 3 the proposed approach is illustrated. Section 4 presents the empirical study and the results of the study are analyzed in Sect. 5. Finally Sect. 6 draws some conclusion with future work.

2 Related Works

In recent time, several researches show that around 50% of total cost of software development has been spent for testing activities [15]. To optimize the cost and time spent over the testing activities elbaum et al. proposed test case prioritization [1]. Test case prioritization approaches arranges the test case in order based on some performance goal. There are numerous methods available in literature for finding an order for given test suite. Most of the test case prioritization technique prioritize test case based on coverage information obtained from source code. For example, in functional coverage the test case are prioritized based on the quantity of function executed by the test case. Other than code coverage, test case are also prioritized based on fault exposing potential, that is test case which is capable of finding more faults are executed earlier. Test cases are also ordered based on other software artifacts. Krishnamoorthi and Mary [9] proposed a prioritization technique considering factors

such as requirement coverage, implementation complexity, requirement volatility and fault prone.

In modern software development different parameters such as code and requirement coverage, faults severity, available execution time, requirement important to client, and so on are considered for prioritization which lead to multi-objective test case prioritization. In recent research, test case prioritization problem is formulated as multi-objective optimization problem and solved using variants of multi-objective evolutionary algorithm [21]. Yoo et al. [8] defined the first multi-objective test case selection and reduction problem considering three conflicting objective such as fault—coverage, code coverage and cost and solved it using variant of NSGA-II. Mondal et al. [6] solved the multi-objective test case selection problem with maximum code coverage, maximum diversity and minimum test case execution time using NSGA-II. Zheng et al. [12] proposed multi-objective test case minimization problem with two conflicting objectives, such as maximum code coverage with minimum execution time. In his experimental study, the performance of four algorithm such as greedy algorithm, NSGA-II, MOEA/D, and MOEA/D with fixed value parameter c is compared and the results prove that MOEA/D performs better than other algorithm. Wang et al. [18] proposed approach for solving multi-objective optimal resource allocation problem using harmonic distance based multi-objective evolutionary algorithm. Marchento et al. [5] schedules the test case in an order based on multiple objective such as discovering the faults earlier within minimum execution time by using information from software artifacts. Megala et al. [19] proposed history based cost cognizant test case prioritization, the approach uses clonal selection algorithm for generating test case order based on historical information of test case such as execution time, fault identified and severity of fault identified. Recently, Khanna et al. [20] prioritized test cases in a multi objective environment for web testing and the performance of NSGA-II is compared with random algorithm, 2-opt algorithm, greedy algorithm, weighted genetic algorithm. The recent other multi-objective problem in software testing activities include multi-objective test data generation [4], multi-objective test suite reduction [11], multi-objective requirement engineering [16] and software product management [17].

Thus, the study of literature reveals that very little work has been carried out on multi-objective test case prioritization. NSGA-II, MOEA/D were widely used to solve multi-objective problem and none of the paper used immune based algorithm to address multi-objective prioritization problem. This gives the motivation to apply immune based clonal selection algorithm for first time in solving multi-objective test case prioritization. The main aim of proposed problem is to generate an optimal test case order based on multiple conflicting objective. The clonal selection algorithm could not handle multiple conflicting objective effectively therefore pareto-optimal concept is hybridized with clonal selection algorithm to effectively optimize all conflicting objective simultaneously and proposes an improved pareto-optimal clonal selection algorithm. To generate optimal test case order the proposed algorithm takes parameter such as test case execution time, rate of fault detection per execution of test case, rate of Severity Detection for generating scheduled test case order.

3 Proposed Prioritization Approach

In this section the proposed multi-objective prioritization approach is illustrated. The main goal of this proposed approach is to find out an order for executing the test case considering the objective such as within minimum execution time, maximum faults with high severity should be identified and also the proposed order should have a maximum APFDc value. Since the proposed problem is a multi-objective NP-hard problem a nature-inspired meta-heuristic clonal selection algorithm is used in the proposed approach. This paper proposes an improved—Pareto optimal clonal selection algorithm combining clonal selection algorithm and pareto-optimality concept. Clonal selection algorithm have been used in many engineering problem and proved its efficacy. Pareto-optimality is a notation derived from economics and been used widely to simultaneously solve multi-objective optimization problem.

The proposed approach initially extracts the information about test suite such as execution time of each test case, fault identified by a test case and the severity of detected fault. A fault versus test case and fault versus severity matrix is constructed from the extracted information. Software profiling tools such

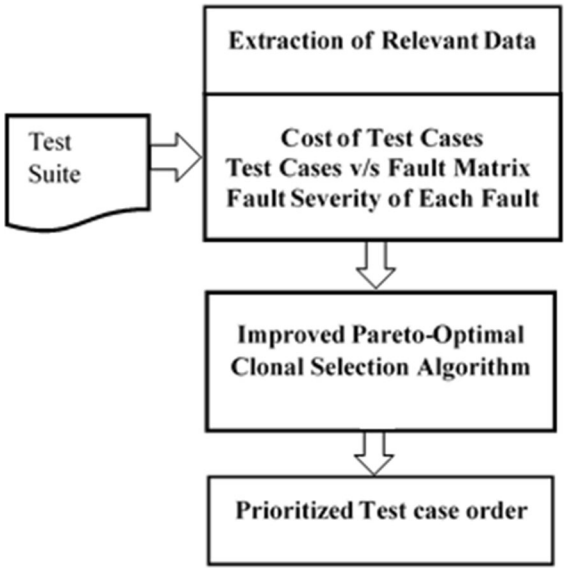


Fig. 1 Architecture of proposed approach

as Emma are used to calculate execution time. The architecture diagram of proposed approach in given in Fig. 1.

In proposed I-POCSA, the test suite is taken as input and generates set of solution called pareto-optimal solution as output considering all the three objective. Since the proposed approach generates a set of solution as output. All the solutions generated are equally important and non-dominated to each other. The proposed I-POCSA algorithm uses the constructed matrix and information about test case to find the test case order which simultaneously satisfy all three objective. In clonal selection principle, the problem to be solved is taken as antigen and the candidate solution are considered as antibody. As

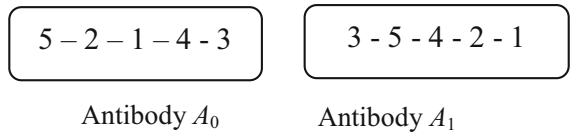


Fig. 2 Antibody Representation

same way in I-POCSA, the optimal set of test case order to be found is considered as antigen and the randomly generated sequence of test cases is considered as antibody. In-POCSA, the antibody is encoded using permutation encoding, in which each antibody is a sequence of test case number that represents the execution order. For example antibody A_0 represents the test case execution order Tc5–Tc2–Tc1–Tc4–Tc3 and antibody A_1 is represents test case execution order as Tc3–Tc5–Tc4–Tc2–Tc1. The antibody representation is shown in Fig. 2.

The proposed I-POCSA is shown in Algorithm 1. Initially the I-POCSA initializes the antibody using the widely used random initialization technique and for each initialized antibody APFDc is calculated. The whole population is sorted using the fast non-dominated sorting algorithm of deb [23] and the solution are placed on different dominance level For Example, solution on the current pareto frontier are assigned dominance level of 0 and after taking out these solution, the fast dominant sorting algorithm calculates the pareto-frontier for remaining solution; solution which lies on the current pareto-frontier are assigned dominance level of 1 and as same way the whole population is assigned a dominance level which becomes the basis for selecting the solution for next generation.

Algorithm: Test case prioritization using improved multi objective immune algorithm

Input:
Ts:test suite
NP: population size
β:Multiplying factor
Gmax:Maximum generation
Mp:mutationprobability

Output:a set of Non-dominated scheduled test case order.

1. Gather test case cost, fault identified, fault severity for given test suite.
2. Generate random antibody population P_0, ngen=0
3. While ngen ≥ Gmax
4. {
5. Evaluate the APFDc value for antibody in population P_0
6. Sort the population base on non-dominance
7. Find front for each non-dominated antibody
8. Select antibody based on rank and crowding distance At
9. Clone the selected antibody
10. Apply hypermutation
11. Combine parent and offspring antibody and proceed from step 2.
12. Repeat step 2 to 11 untill maximum generation reached
13. Return Toutput

Algorithm1. Proposed Improved-Pareto-optimal clonal selection algorithm

3.1 Selection

In general clonal selection principle, the antibody are selected from whole population using selection operator. Selection is a process of selecting a fittest candidate solution for recombination process from population, different selection method are available in literature. In this paper we employ a selection method which selects the antibody based on rank and crowding distance. A Crowding distance is said to be the normalized sum of distance of an individual solution with other individual solution in respect to each objective. The selection operator randomly selects two antibody from whole population and compares with each other based on rank and the candidate with better rank is selected. If both the antibody are of same rank the antibody is compared based on crowding distance. The antibody with higher crowding distance is selected so that it leads to wider pareto-frontier. The recombination operator is applied over selected antibody to generate new population.

3.2 Cloning

After selecting set of antibody from population, cloning is applied over antibody to reproduce new non-dominated solution. Cloning is the process of replicating the copy of antibody. The number replication is dependent on the affinity of the antibody. As in I-POCSA algorithm, we do not assign individual affinity value for antibody the rank of the antibody is utilized for cloning. When higher the rank of antibody then more number of copy is cloned. The clones are generated for selected antibody based on the following Eq. (1).

$$N_c = round\left(\frac{\beta.Population\ Size}{i_f}\right) \quad (1)$$

N_c denotes the total number of clones generated for each antigen, β is a multiplying factor, Population size is the total number of antibodies in a population, round(.) is the operator used to round its argument into nearest integer, i_f is the rank of antibody.

3.3 Hyper-Mutation

Once the cloned population is generated, hyper-mutation operator is applied on the cloned population to produce new offspring. Hyper mutation is the process of creating a new solution by mutating the antibody depending upon the affinity value. The antibody with lower affinity value are mutated at higher rate and vice versa. There are different type of mutation operator available in literature such as static hyper mutation, inverse mutation and proportional hyper mutation. In static hyper mutation the antibody is mutated randomly without considering the affinity value of antibody but in inverse and proportional mutation the antibodies are mutated depending on fitness value. Since the proposed I-POCSA does not assign fitness value to clone population we choose static mutation type. A pair-wise mutation belonging to static type mutation is chosen as hyper-mutation in proposed algorithm, in which the antibody are mutated by choosing two random point P1 and P2 and the position of the chosen point are interchanged resulting in newer antibody and Fig. 3 shows an example of pair wise mutation. Once the new generation is generated the parent and child offspring is combined to form a new population and the step 2–11 of algorithm is repeated until the maximum generation is reached. The set of non-dominated solution which lies on the dominance level 0 are the solution of the proposed problem.

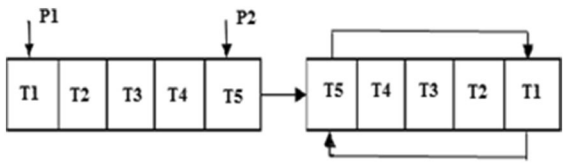


Fig. 3 Pairwise Mutation

4 Discussion on Parameter

To prioritize the given test suite three parameters are chosen such as execution time, early detection of fault with maximum severity and average percentage of fault detected. And this paper do not argue only selecting these parameter will lead to good result. To show the importance of chosen parameter a test suite of five test cases is considered and the test cases are scheduled based on three objective. The first objective is to prioritize the test case in an order which detect all the faults within minimum execution time. The second objective is to prioritize the test case in an order that rate of fault detection per execution of test case is maximized the third objective is to schedule the test case in order that rate of detection of severe fault per execution of test case is maximized. The parameters considered are execution time, Rate of fault detection per execution of test case and Rate of severity detection (Table 1).

(I) *Execution time* The Test case execution time is taken as the time taken to execute the test case. Tools such as Emma is used to measure the test case execution time. In given scenario, execution time is the total time taken by the sequence of test cases order to detect all the faults. And Table 2 shows the execution time of test cases

(II) *Rate of fault detection per execution of test case* The rate of fault detection of test case order is measured by the average percentage of fault detected with cost and it is the standard metric used to measure the performance of test case order [25]. If d is the number of faults and n is the number of test cases then APFDc is defined as [20]

Table 1 Sample test case versus fault matrix

Test cases	Fault			
	F1	F2	F3	F4
Tc1	X		X	
Tc2		X		
Tc3	X			X
Tc4		X		
Tc5	X		X	X

Table 2 Test cases versus execution time

Test case	Execution time
Tc1	3
Tc2	1
Tc3	2
Tc4	1
Tc5	4

Table 3 Fault versus severity

Fault	Severity
F1	2
F2	1
F3	4
F4	3

$$APFD_C = \sum_{i=1}^{d} \frac{\left(s_i \times \left(\sum_{j=pf_i}^{n} c_j - (1/2)pf_i\right)\right)}{\sum_{j=1}^{n} c_j \times \sum_{i=1}^{m} s_i} \quad (2)$$

In Eq. (2), *Si* is the severity of fault *i* and *Cj* is the cost of the test case and *pfi* is the position of the test case in the antibody that detect the fault *i* first. The APFDc values for sequence 1 in Table 4 is shown below.

$$\frac{\left(\frac{10}{10}\right) \times 9}{1} + \frac{\left(\frac{1}{10}\right) \times 1}{2} + c(0) + (0) + (0) = 9.05$$

The importance of all three parameter is illustrated with complete scenario below. A test suite with 5 test cases is considered, the test case versus fault matrix is shown in Table 2, the severity of each fault and execution cost of each test case in shown in Tables 2 and 3. The complete performance

$$APFD_C = \frac{2\left(\sum_{J=1}^{5} C_J - \left(\frac{1}{2}\right)(4)\right) + 1\left(\sum_{J=2}^{5} C_J - \left(\frac{1}{2}\right)(1)\right) + 4\left(\sum_{J=1}^{5} C_J - \left(\frac{1}{2}\right)(4)\right) + 3\left(\sum_{J=1}^{5} C_J - \left(\frac{1}{2}\right)(4)\right)}{(4+1+2+1+3) \times (2+1+4+3)} = 0.7590$$

(III) *Rate of severity detection* This parameter measure the severity of fault detection per execution of test case. The reason to choose this parameter is that most severe fault will have more impact over business criticality. So earlier detection of severe faults will improve the performance. To measure the performance of proposed test case sequence in terms of severity detection is given below [20]. *n* is the number of test case, *sud* is the undetected severity, *sd* is the severity detected and *PT* position of test case j in test suite. The severity detection for sequence 1 in Table 4 is shown below.

$$Severity\ detection = \sum_{j=1}^{n} \frac{SUD(\%) \times SD}{PT} \quad (3)$$

matrix of scheduled test case order is shown in Table 4. From Table 4 it is observed that test sequence 1 is best in respective to all three objective and test sequence 4 is worst execution order in terms of all three objective. Test case sequence 2 proves to be best in terms of APFDc and severity detection but has an increased execution time.

5 Experimental Setup

To evaluate the performance of the proposed approach, an empirical study with sample test cases is taken for illustrated and inventory management system (IMS) a real time application developed using java platform consisting of 28 modules, and approximately 10,000 lines of code, the software was developed by soft key technologies private limited, Pondicherry, India. Seven version of the software is

Table 4 Example of sample test case

Test case execution order (TEi)	APFDc observed	Rate of severity detection	Execution time
TC5–TC4–TC3–TC2–TC1	0.7590	9.05	5.0
TC1–TC2–TC3–TC4–TC5	0.75	6.5	6.0
TC3–TC4–TC2–TC1–TC5	0.731	5.65	7.0
TC5–TC4–TC3–TC2–TC1	0.6368	4.13	7.0
TC3–TC1–TC2–TC5–TC4	0.777	6.26	6.0

considered for experimental study. In each version an addition of functionality, modification and deletion is done for study. Up to 30 faults of different severity level were manually injected randomly into the code. The test suite is generated using the selenium tool which generates test case which uncover the manually seeded fault. And the execution time of test case is calculated using Emma. The severity of the fault were decided by taking a survey from it professional having more than 5 years of experience. Twenty five test engineers were given a questionnaire and based on the response severity of fault was decided. Initially a fault free inventory management system is taken has software under test and randomly fault were seeded into the system the proposed approach and other existing approach were applied to analyze the performance of the various algorithm. The approach of manual fault seeding and thereafter identifying the faults has been widely accepted by the research group and various studies have followed the same approach for performance evaluation [20]. The fault verses test case, execution time matrices and fault severity matrices were generated with help of software profiling tools. To evaluate the performance of the proposed approach four existing approaches is been considered and compared. All the algorithm were implemented using Mat lab.

5.1 Random Prioritization

Random Prioritization: In Random test case prioritization the test cases are randomly arranged [3], the efficiency of the generated test case order is calculated using Eqs. (2) and (3). The technique blindly orders the test case due to which for most of the problem random prioritization does not propose an optimal result.

5.2 Simple Greedy

Greedy algorithm have been widely used in solving single objective test case prioritization problem and also proved its competence [20]. The greedy approach schedules the test case based on the factor value of each test case given by (summation of severity/test case execution time). The test case are sorted in decreasing order of factor and the efficiency of the generated test case order is calculated using Eqs. (2) and (3). The greedy algorithm at many times lead to sub-optimal solution which is one of the major issue to be considered.

5.3 Weighted Genetic Algorithm

Weighted genetic algorithm solves multi-objective problem by assigning weights to each of the objective, it works similar to normal genetic algorithm. Since the proposed problem consists of three objective its considers three weight w1, w2, w3 and the fitness function is defined as follows

$$\textit{fitness function} = W1 \times \left(1 - {}^{a}/_{b}\right) + W2 \times (c) + W3 \times (x/y) \tag{4}$$

where a, is the execution time taken by scheduled order of test cases to discover all the faults; b, is the total execution time of all test cases in test suite and c, is the value of APFDc; x, rate of severity of faults detected of the scheduled order of test cases which exposes all the faults; y, maximum possible severity rate. In this equation, all the parameters are given equal importance. However, if the tester is in need to give importance to any one parameter the corresponding weights ($w1$, $w2$ and $w3$) are modified. During the empirical study, all the weights are assumed to be 1 and the values of x/y, z and a/b are normalized between

0 and 1, where x/y is to be maximized while z and a/b are to be minimized. After the execution of all iterations, the generated solutions are the result of weighted genetic algorithm sorted based on each of all three objective resulting. Hence, the 3 solutions are generated in which one solution has highest APFDC, another solution has highest severity detection rate and the last solution has least test cases execution time to expose all the faults. But the major issue of wga is the optimal solution distribution is not uniform.

5.4 NSGA-II

NSGA-II is the most popular and commonly used algorithm by the research community to solve multi objective optimization problem. NSGA-II is simultaneously optimize all the objective even if the objectives are conflicting and works based on genetic algorithm and uses a fast non-dominated sorting technique to find and sort the non-dominated solution. The proposed I-POCSA is similar to NSGA-II, the difference is instead of genetic algorithm the proposed algorithm employ clonal selection algorithm.

NSGA-II simultaneously optimize all the objective even if the objectives are conflicting in nature. The algorithm returns a set of non-dominated solution which dominates all other solution in any one of the objective. For given problem there may exist many pareto-optimal solution which are equally important and non-dominated to each other non-dominant solution. In this empirical study, NSGA-II is implemented as published earlier in [22–24]; however after the last iteration, the solutions of the first front, which are non-dominated among themselves and moreover dominating all the elements of remaining fronts, are sorted in the decreasing order of the value of APFDc is considered as the output of the proposed problem. Similarly, the elements are also sorted on the basis of the remaining two parameters; however, APFDc is the only standard method followed for measuring the efficacy of prioritized test sequence.

6 Result Analysis and Discussion

From the related work it is evident that only two works have been conducted in multi objective test case prioritization and published in reputed journal [5, 20]. The parameter considered in both the papers are code coverage, requirement coverage, execution time in [5, 20] cost, fault severity and APFDc is considered. The parameters considered are cost, fault severity and APFDc. Since the aim of this work is to propose a new approach to generate a prioritized test case order in multi-objective environment and does not argue upon the parameter selected.

During the experimental study, the performance of all five algorithm are analyzed and the best results obtained by executing the five algorithm are shown in Tables 3, 4 and 5. It has been noted that among the five existing algorithm, the proposed I-POCSA and NSGA-II were able to generate optimal solutions for all the version. While analyzing the behavior of I-POCSA, the intelligent way of using historical information and searching the solution based on multiple objective help to generate optimal solutions. 40 solution were generated in first front out of 9 solution were unique and in second and front out of 69 solution 13 solution were unique and in third front 18 solutions were unique When analyzing the solution in first front of I-POCSA all the solution are non-dominant and found to be the optimal solution. When comparing the solution which lies on first front of NSGA-II and I-POCSA both the algorithm able to find a set of non-dominant solution. In certain version I-POCSA is able to generate best optimal solution. the test case order generated by WGA and SG are near to optimum value only for certain version and Random algorithm is able to generate near optimum value only for one version out of seven version. In modern software development there since the algorithm generates non-dominant set of solution, each solution is dominating on different parameter. For instance, a solution may generate test case with maximum APFDc and another solution which uncover severe faults earlier than any other solution. Therefore the non-dominant solution are dominating one another in any of the parameter. Therefore depending upon the tester need the solution from non-dominant solutions are selected. And the algorithm are compared based on how far the solution generated contributes to different parameter. When considering the APFDc parameter Table 5 represents the best APFDc value obtained in executing the test case order generated by all the algorithm, inferring from Table 5 it is evident that the proposed I-POCSA has generated a test case sequence with highest APFDc value above 98.5 for five version

Table 5 Performance of algorithms in terms of APFDc applied on all versions of software under test

Version	Proposed I-POCSA APFDc	NSGA-II APFDc	Weighted genetic algorithm APFDc	Simple greedy APFDc	Random APFDc	Standard deviation (σ)
V1	**98.73**	97.6723	96.546	95.324	94.348	1.5722
MD	2.20594	1.14824	0.02194	− 1.20006	− 2.1760	
V2	98.96	**99.1**	97.271	98.0131	93.5147	2.039
MD	1.5882	1.72824	− 0.10076	0.64134	− 3.8570	
V3	**99.3241**	98.76	98.514	98.88	95.6284	1.3228
MD	1.1028	0.5387	0.2927	0.6587	− 2.5929	
V4	**99.977**	99.765	98.257	99.196	97.731	0.8648
MD	0.9918	0.7798	− 0.7282	0.2108	− 1.2542	
V5	97.816	**99.1432**	97.987	96.5564	95.261	1.3290
MD	0.46328	1.79048	0.63428	− 0.79632	− 2.09172	
V6	**99.513**	98.4969	98.379	95.614	98.6406	0.5119
MD	1.3843	0.3682	0.2503	2.5147	0.5119	
V7	**99.9125**	99.676	99.437	97.2314	96.6111	1.3715
MD	1.3389	1.1024	0.8634	1.3422	− 1.9625	

Bold value represents the best APFDc value achieved from all the compared algorithms for each version

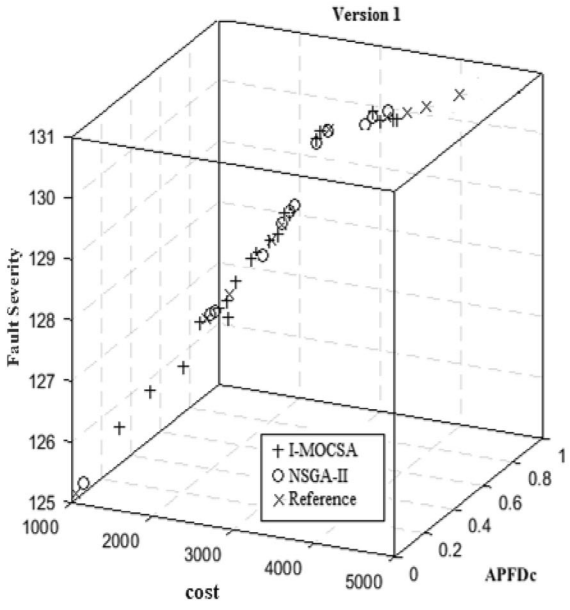
Table 6 Performance of algorithms in terms of severity applied on all versions of software under test

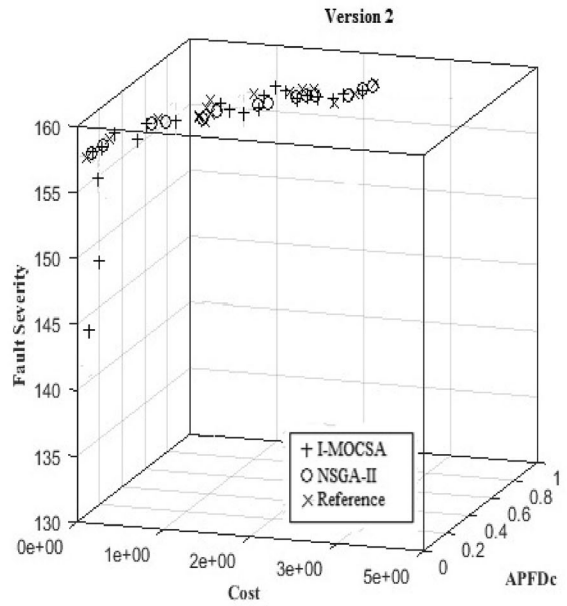
Version	Proposed I-POCSA Severity	NSGA-II Severity	Weighted genetic algorithm Severity	Simple greedy Severity	Random Severity	Standard deviation (σ)
V1	**130.89**	129.4989	128.213	122.938	107.218	4.099
MD	6.4652	2.1552	− 0.1308	− 3.3638	− 5.1258	
V2	**169.647**	169.302	165.905	150.321	140.721	4.210
MD	7.2678	1.9228	− 4.4742	− 3.0582	− 1.6582	
V3	**176.034**	**176.032**	**176.112**	166.07	142.453	13.0311
MD	8.6938	8.6918	8.7718	− 1.2702	− 24.8872	
V4	**183.999**	182.357	180.432	160.432	159.418	11.0059
MD	10.6714	9.0294	7.1044	− 12.895	− 13.9096	
V5	**178.319**	177.871	177.233	162.297	169.318	6.2939
MD	5.3114	4.8634	4.2254	− 10.710	− 3.6896	
V6	**183.434**	181.67	177.271	150.013	125.713	11.9140
MD	11.8138	10.0498	5.6508	− 11.607	− 15.9072	
V7	**183.999**	182.17	175.907	167.879	163.876	7.84801
MD	9.2328	7.4038	1.1408	− 6.8872	− 10.8902	

Bold value represents the highest severity value achieved from all the compared algorithms for each version

and NSGA-II generated a test case sequence with best optimal value for two versions.

When considering severity as a parameter, the proposed I-POCSA has generated the best optimal values for all version and also the NSGA-II has generated near optimal values for all version suppressing the weighted genetic algorithm based on severity have also generated near optimal values for all version. The greasy and random approach were not able to generate test case with near optimal value. The performance of all the algorithm in terms of severity is shown in Table 6.





When considering execution as a parameter, the proposed I-POCSA produces best result than all other algorithm for all versions. It is observed that NSGA-II performs better than greedy, random and weighted genetic algorithm and produces near optimal value for all versions. It is also noted that weighted genetic algorithm have also performed well for most version. The performance of all the algorithm in terms of severity is shown in Table 6. To qualitative analyses the proposed approach a reference pareto-frontier is constructed and used in comparing different algorithm based on pareto-front they produce.

The reference-pareto frontier, is constructed by combining the best of each obtained from all

Table 7 Performance of algorithms in terms of cost applied on all versions of software under test

Version	Proposed I-POCSA Cost	NSGA-II Cost	Weighted genetic algorithm Cost	Simple greedy Cost	Random Cost	Standard deviation (σ)
V1	**91.89**	92.89	100.23	102.38	103.118	4.9675
MD	− 6.2116	− 5.2116	2.1284	4.2784	5.0164	
V2	**109.47**	109.302	113.95	115.21	120.821	4.2490
MD	− 4.2806	− 4.4486	0.1994	1.4594	7.0704	
V3	**76.04**	77.032	84.92	86.07	92.35	6.0694
MD	− 7.2424	− 6.2504	1.6376	2.7876	9.0676	
V4	**83.999**	84.357	85.432	90.02	99.48	5.8237
MD	− 4.6586	− 4.3006	− 3.2256	1.3624	10.8224	
V5	**118.319**	116.871	117.243	122.297	139.18	8.422
MD	4.463	− 5.911	− 5.539	− 0.485	16.398	
V6	**83.41**	85.67	86.271	90.013	115.13	9.7494
MD	− 7.6888	− 5.4288	− 4.8278	− 1.0858	19.0312	
V7	**100.9**	102.17	105.117	107.879	103.36	2.4329
MD	− 2.9852	− 1.7152	1.2318	3.9938	− 0.5252	

Bold value represents the minimum execution time achieved from all the compared algorithms for each version

approaches result in a hybrid pareto-frontier. Figure 3 shows the pareto-frontier for version 1 and version 2 obtained by implementing proposed I-POCSA, NSGA-II and reference pareto-frontier constructed. From the Fig. 3 it is evident that the proposed method is able to reach the reference pareto-frontier and also able to find solution that are not found by other existing methods (Table 7).

7 Conclusion

Most of the existing prioritization technique aims to order the test case considering a single objective such as to increase the fault detection rate. In this paper, we prioritize test case based on multi-objectives, the test case are scheduled considering three objective such as to maximize the fault detection rate and rate of severe fault detection within minimum execution time. This paper introduces the clonal selection algorithm for first time to solve multi-objective test case prioritization and proposes an improved-pareto optimal clonal selection algorithm. This paper also describes the benefits of combining Pareto-optimal concept and clonal selection algorithm. Since the proposed algorithm uses pareto-optimal concept, it generates a set of non-dominated test case order considering all three conflicting objective, based on the testers preference and objective test case order is selected from the generated test case order. In our experimental study, we have used a real-time software application to evaluate the proposed technique and the empirical results shows that the proposed methodology satisfyingly performs better in comparison with genetic algorithm based approaches, greedy and random approach. Finally there are still some research issues to improve the effectiveness of proposed system by considering other software artifacts.

References

1. Elbaum, S., Malishevsky, A. G., & Rothermal, G. (2002). Test case prioritization: a family of empirical studies. *IEEE Transactions on Software Engineering, 28*(2), 159–182.
2. Malishevsky, A. G., Ruthruff, J. R., Rothermel, G., & Elbaum, S. (2006). Cost-cognizant test case prioritization, technical report TRUNL-CSE-2006-0004. Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln.
3. Rothermal, G., Untch, R., & Harrold, M. (2001). Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering, 27*(10), 929–948.
4. Kavita, C., & Purohit, G. (2014). A multiobjective optimization algorithm for uniformly distributed generation of test cases. In *IEEE international conference on computing for sustainable global development* (2014).
5. Marchetto, A., Islam, M., Scanniello, G., Asghar, W., & Susi, A. (2016). A multi-objective technique to prioritize test cases. *IEEE Transactions on Software Engineering, 42*(10), 918–940. https://doi.org/10.1109/TSE.2015.2510633.
6. Mondal, D., Hemmati, H., & Durocher, S. (2015). Exploring test suite diversification and code coverage in multi-objective test case selection. In *IEEE conference*.
7. Yoo, S., & Harman, M. (2007) Pareto efficient multi-objective test case selection. In *ISSTA 2007*. London: ACM.
8. Yoo, S., & Harman, M. (2010). Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *Journal of Systems and Software, 83*(4), 689–701.
9. Krishnamoorthi, R., & Mary, S. S. A. (2009). Factor oriented requirement coverage based system test case prioritization of new and regression test cases. *Information and Software Technology, 51*(4), 799–808.
10. Catal, C., & Mishra, D. (2013). Test case prioritization: A systematic mapping study. *Software Quality Journal, 21*(3), 445–478.
11. Marchetto, A., Islam, M., Scanniello, G., & Susi, A. (2013). A multiobjective technique for test suite reduction. In *The eighth international conference on software engineering advances*. IARIA.
12. Zheng, W., Hierons, R., Li, M., Liu, X., & Vinciotti, V. (2016). Multiobjective optimization for regression testing. *Information Sciences, 334–335,* 1–16. https://doi.org/10.1016/j.ins.2015.11.027.
13. Mathur, A. (2012). *Foundations of software testing, seventh impression*. New York: Pearson Education.
14. Chauhan, N. (2010). *Software testing principles and practices* (1st ed.). Oxford: Oxford University Press.
15. Singh, Y. (2012). *Software testing* (1st ed.). Cambridge: Cambridge University Press.
16. Zhang, Y., Harman, M., & Mansouri, S. (2007). The multi-objective next release problem. In *GECCO'07*. London: ACM.
17. Ruiz, M., Roderiguez, D., Riquelme, J., & Harrison, R. (2011). Multiobjective simulation optimization in software project management. In *Proceedings of the 13th annual conference on genetic and evolutionary computation GECCO 2011* (pp. 1883–1890). ACM.
18. Wang, Z., Tang, K., & Yao, X. (2000). Multi-objective approaches to optimal testing resource allocation in modular software systems. *IEEE Transactions on Reliability, 59*(3), 563–575.
19. Tulasiraman, M., & Kalimuthu, V. (2018). Cost Cognizant history based prioritization of test case for regression testing using immune algorithm. *Journal of Intelligent Engineering Systems, 11*(1), 221–228. https://doi.org/10.22266/ijies2018.0228.23.

20. Khanna, M., Chauhan, N., Sharma, D., Toofani, A., & Chaudhary, A. (2017). Search for prioritized test cases in multi-objective environment during web application testing. *Arabian Journal for Science and Engineering*, *43*, 1–23.
21. Li, Z., Harman, M., & Hierons, R. M. (2007). Search algorithms for regression test case prioritization. *IEEE Transactions on Software Engineering, 33*(4), 225–237.
22. Shapiai, M. I., Ibrahim, Z., & Adam, A. (2017). Pareto optimality concept for incorporating prior knowledge for system identification problem with insufficient samples. *Arabian Journal for Science and Engineering, 42*(7), 2697–2710.
23. Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182–197.
24. Deb, K. (2010). *Multiobjective optimization using evolutionary algorithms* (1st ed.). New Delhi: Wiley India Pvt Ltd.
25. Nayak, S., Kumar, C., & Tripathi, S. (2017). Enhancing efficiency of the test case prioritization technique by improving the rate of fault detection. *Arabian Journal for Science and Engineering*. https://doi.org/10.1007/s13369-017-2466-6.