FUDMA Journal of Sciences (FJS)
ISSN online: 2616-1370
ISSN print: 2645 - 2944
Vol. 9 No. 9, September, 2025, pp 129 – 138
DOI: https://doi.org/10.33003/fjs-2025-0909-2901

COST-COGNIZANT TEST CASE PRIORITIZATION FOR SOFTWARE PRODUCT LINE USING GENETIC ALGORITHM

***[1]Bello, A. and [2]Alhassan, H. M.**

[1]Department of Computer Science, Usmanu Danfodiyo University, Sokoto.
[2]Department of Computer Science, Federal Polytechnic Nasarawa.

*Corresponding authors' email: Kiyawa99@gmail.com Phone: +2348035534197

ABSTRACT

Software Product Line (SPL) is a family of related software systems with some commonalities and variabilities concerning features and relationships. SPL tends to reduce development and maintenance costs, increase quality, and decrease time to market. Due to its unique features, most software companies are moving from single software to SPL. A feature describes the behavior and capability of SPL displayed in a Feature Model. Moreover, testing of SPL is a difficult task as compared to a single system, based on this, a test case prioritization is needed to order test cases best on its importance. In this study, a cost-cognizant test case prioritization for software product line that uses path-based testing to identify the possible execution approaches was proposed. The path was extracted from a feature model of SPL obtained from FeatureIDE. Further, a Genetic Algorithm (GA) was used to prioritized test cases based on the rate of fault detection per unit test cost. The approach was evaluated using the Average Percentage of Fault Detection per Cost (APFDc) metric across four program objects (Video Store Versions VS1, VS2, VS3, and VS4). For the result, the proposed approach achieves higher performance compared to the existing methods namely CEC, RNDP and NOP. Specifically, APFDc scores reached 94.48% for VS1, 91.84% for VS2, 90.93% for VS3, and 71.81% for VS4, confirming the effectiveness and efficiency of the method in improving fault detection while reducing testing cost.

Keywords: Test Case Prioritization, Test Suite, Software Product Line, Genetic Algorithm

INTRODUCTION

Software engineering is mostly about developing a set of single software products rather than developing a set of related software product lines (SPLs) by reusing a common set of features. On the other way round, SPL is among the widely used means for capturing and handling commonalities and variabilities of many applications of a target domain (Ensan et al., 2011). The SPL differences and commonalities are presented through a Feature Model (FM). A FM describes the behavior and capability of a software system present in a tree-like structure whose nodes are the domain features, and the edges are the interaction between the features (Qureshi, 2018). Figure 1 shows the example of a feature model representing an SPL Electronic shop (E-shop). The automated analysis of FM based on the information extracted from the computer was presented and only a valid combination of features otherwise known as configuration undergoes configuration processes. Given this, the analyses of SPL reduced the development and maintenance cost, increased quality, and decreased time to market.

However, the possible number of configurations of FM increases exponentially based on the size of the feature model and as such, the time and effort needed for testing of SPL will increase which might lead to combinatorial explosion. Outside the context of SPL, testing of a single software application is easier as compared to SPL, to deal with such a scenario, it is important to ensure that, fault is free by the analysis of all its potential products comprehensively to ensure the use of tools that have been developed for penetration testing with the purpose of raising the level of security strength (Aminu et al., 2020). For example, testing the e-shop model obtained from SPLOT repository required testing of more than 1 billion of products (Sanchez et al., 2014) which makes testing a difficult task. In the context of this, there have been different attempts used to reduce the stress of testing by using regression testing techniques (Sanchez et al., 2014; Bello, 2019; Al-Hajjaji et al., 2014; Devroey et al., 2016).

Regression testing (RT) can be performed when changes are made to existing software functionality in such a way that the behaviors of the existing software are not altered. Among the RT are Test Suite Minimization (TSM), Test Case Selection (TCS), and Test Case Prioritization (TCP). TSM techniques aim to identify redundant test cases and remove them from the test suite to reduce the size of the test suite. TCS is concerned with the problem of selecting a subset of test cases that will be used to test the changed parts of the software and TCP deals with the identification of the ideal ordering of test cases based on some performance goals. One performance goal is the rate of fault detection which is to measure how fast test cases revealed faults (Yoo & Harman., 2010). The most promising technique is the TCP technique which can schedule test cases for execution so that attempt to increase their effectiveness or efficiency at meeting some performance goal (Raju, 2012). E.g., to increase the rate of fault detected, a tester needs to order test cases based on the number of faults detected by the test cases in the preceding executions of a test suite. As regards to benefit derived from using TCP, it is proving to provide a perfect result combined with an evolutionary algorithm (Bello, 2019).

To this end, An Evolutionary Algorithms (EAs) as a search-based methods that mimic the natural biological evolution and/or the social behavior of species was presented. EA have been developed to arrive at near-optimum solutions to large-scale optimization problems (Elbeltagi et al., 2005). More specifically, The Genetic algorithm (GA) for this study, It is among the types of EA that increases the population of chromosomes by continuously replacing one with another based on fitness function assigned to each chromosome. Furthermore GA start with generation of test cases/chromosomes for product line and gradually improve the quality of the test cases by evolving them in a control manners (Ensan et al., 2012).

In this paper, An Evolutionary cost-cognizant test case prioritization for SPLs was proposed. This approach aims to detect faults while minimizing cost. To demonstrate the implementation of this study, a path-based testing approach was applied to source code of an extracted feature model and its coverage information. The approach works by taking the executing test case cost presented in Table 1 and the severities of the exposed fault depicted in Table 2 as input from feature models. Each test case information is collected from the previous regression testing activities. Feature/Test case repository holds the generated optimal order of the existing test suite to enable reuse in future regression testing. As soon as prioritization is applied, the proposed study inputs the information about the test cases to the ECRTCP algorithm whereby the algorithm will search for that order of test cases effectively using the information retrieved from the repository.

For the implementation of the approach, a prototype was developed for Regression test case prioritization for software product lines using Genetic Algorithm. The feature model of the illustrative example was extracted from featureIDE of Eclipse. After the extraction, the feature models are updated and used a path-based testing approach to select the paths, then the prioritized test cases were encoded. A Genetic Algorithm (GA) was applied for software product line. Starting with the generation of an initial population, followed by Crossover, and finally Mutation operator. The fitness functions were determined using the Award value formula and the percentage score from the Average Percentage of Fault Detection Per Cost (APFDc).

The approach was evaluated using an SPL object program with four different versions. The Mujava tool, based on (Fischer et al., 2018) was used to introduce mutants into the source code of the program objects. The results showed that the proposed approach achieved the highest scores compared to other methods.

Related Works

The field of regression test case prioritization for Software Product Lines (SPLs) has seen several key developments that aimed at improving test case cost and fault detection rate. A common goal across many studies is to select and prioritize the most important test cases that can reveal faults in a timely manner.

Several authors have proposed different approaches to address the mentioned challenges. Ensan et al., (2011) proposed a goal-oriented approach that selected the most important features to reduce the test space and prioritize the remaining test cases based on vital features. by identifying those features that are more important and need to be tested. This method aims to make the testing process more manageable and effective. Similarly, Raju, (2012), presented a different framework called Prioritization Factors (PF) for SPLs. The framework considers a concrete factor such as test case length, code coverage, data flow, and faults proneness, as well as abstract factors like perceived code complexity and severity of faults found by prioritized test cases. Consequently, other researchers have explored different approaches for prioritization. (Sanchez et al., 2014) presented a TCP technique that shows an increase in the rate of early fault detection for SPL. The technique was based on five prioritization criteria to schedule the execution of test cases to allow faster feedback and decrease the debugging efforts. Al-Hajjaji et al., (2014) introduced a similarity-based prioritization technique that help to reduce the number of products under test by generating new ones. They also (2017) proposed a delta-oriented modeling approach on similarity measures to find the differences among products of SPLs. Both of these methods focus on leveraging product similarities to enhance fault detection. (Elbaum & Rothermel., 2001; Malishevsky et al., 2006) proposed new metrics for accessing the rate of fault detection of prioritized test cases. The metric assumed that test case cost and fault severity vary, while in practice it does.

As a significant area to focus is cost-cognizant for SPL. Kumar, (2017) proposed a cost-based test case prioritization technique for a software product line. This approach, applied to test case prioritization, that considers varying test case costs and fault severity. This study highlights the need to consider economic factors in the prioritization process.

Genetic Algorithm

Researchers have also explored the use historical data and algorithms to improve prioritization,

Tulasiraman and Kalimuthu, (2018) proposed a cost-cognizant history-based test case prioritization approach that utilizes the historical information of test cases, such as cost of test cases, fault identified by test cases, and severity of faults for prioritization. Bello, (2019) proposed a cost-cognizant test case prioritization for object-oriented programming software. The approach used path-based integration testing to identify and extract the possible execution paths from the Java system dependence graph (JSDG) method. Test cases were prioritized by evolutionary algorithm. Sabharwal et al., (2010) and Li et al., (2010) independently proposed a genetic algorithm (GA)-based approach to identified the optimal test case path. Additionally, Bello et al., (2018) extended this by proposing an evolutionary cost-cognizant regression testing approach. They found that prioritized test cases based on severity of fault detection rate connected with dependency effectively detects severe faults. The current research plans to use this methodology within the context of SPLs, leveraging an open-source repository and a feature model from FeatureIDE.

MATERIALS AND METHODS

Automated analysis of Feature Models

An example of feature model representing an elevator system SPL shown in Figure 1, was generated using FeatureIDE. This model illustrate how test cases/features for SPL are prioritized through a path-based testing approach, with fitness functions are derived from test case cost and fault severity, as summarized in Tables 1 and 2.

Table 1: Assigning Cost to Test Cases

Test Case	Cost
t1 = t6 = t8	5
t2 = t9 = t11	4
t3= t7 = t10	3
t4 = t13	2
t5 = t12	1

Table 2: Assigning Severities to Faults

Faults	Severities
Fault (1) = Fault (6) = Fault (10)	2
Fault (2) = Fault (8) = Fault (12)	4
Fault (3) = Fault (7) = Fault (13)	5
Fault (4) = Fault (9)	3
Fault (5) = Fault (11)	1

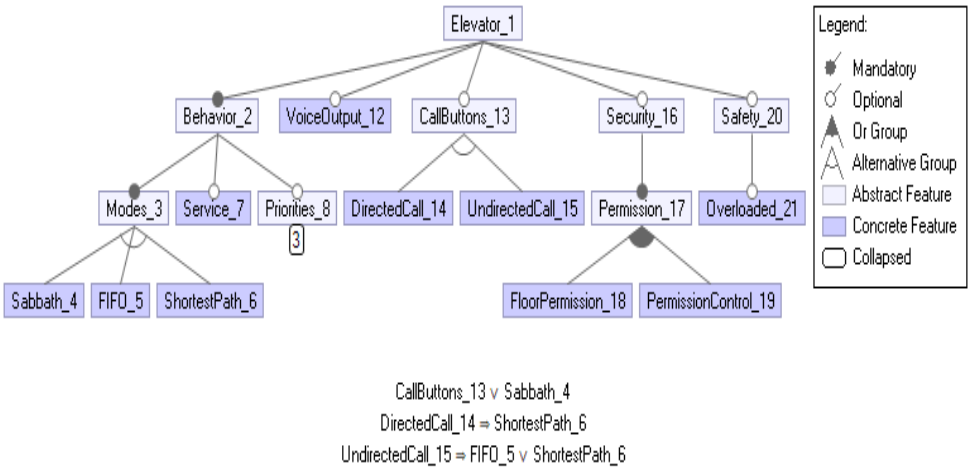

Figure 1: A Feature Model Sample

The study describes a running example for a study on an Elevator System Design (ES). This example has 21 features, each with its own relationships. The relationships are grouped into Mandatory group: which consist of elevator, behavior, and modes. Optional group: service, priorities, voiceoutput, callbuttons, security, safety and overloaded. Or relationship: floorpermission and permissioncontrol. Alternate group: sabbath, FIFO, shortestpath, Directedcall and undirectedcall and lastly, cross tree constraint relationship: directedcall require shortestpath and undirectedcall require FiFO or shortestpath. In addition, feature Elevator is the basic framework of the SPL that all variants have in common. In our running example, firstly, we extracted the FM from FeatureIDE for eclipse, then we updated it by assigning a path to each features/test cases as presented in Table 3.

Table 3: Results for SPL Feature Model in Figure 1

Test case	Extracted paths
t1	Elevator, behavior, modes, Sabbath
t2	Elevator, behavior, modes, FiFo
t3	Elevator, behavior, modes, shortestpath
t4	Elevator, behavior, service
t5	Elevator, behavior, priorities, rush Hour
t6	Elevator, behavior, priorities, floorpriority
t7	Elevator, behavior, priorities, person Priority
t8	Elevator, voiceoutput
t9	Elevator, cellbuttons, direct cells
t10	Elevator, cellbuttons, undirect cells
t11	Elevator, security, permission, floorpermission
t12	Elevator, security, permission, permissioncontrol
t13	Elevator, safety, overloaded

However, after the selection of test cases, it then continued with the prioritization of the selected test cases using the selected test case encoder for representing it properly as shown in Table 4.

Table 4: SPL Feature Model Paths Encoded to Integer

Test case	Paths
1	1, 2,3,4
2	1,2,3,5
3	1,2,3,6
4	1,2,3
5	1,2,7
6	1,2,8,9
7	1,2,8,10
8	1,2,8,11
9	1,13,14
10	1,13,15
11	1,16,17,18
12	1,16,17,19
13	1,20,21

The encoded solution of the test cases/features which serves as the initial population was obtained from T = [t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13] to T` = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. The initial population was randomly generated by the encoded test cases based on the generated random population algorithm as seen in Table 5. The award value formula was used for the computation of fitness functions.

Table 5: Initial Population

S/No	Chromosome
1	3, 2, 1, 6, 5, 4
2	2, 1, 5, 3, 4, 6
3	5, 3, 6, 2, 4, 1
4	1, 3, 4, 2, 6, 5
5	5, 6, 4, 1, 2, 3

Test Case Prioritization (TCP)

Test case prioritization is among the techniques of regression testing. The techniques schedule test cases for execution in an order that attempts to increase their effectiveness at meeting some performance goal (Rothermel et al., 2001). In such scenario, running all test cases in an existing test suite can be challenging due to cost and time constraint. The author above reported that, an industry having an application of 20,000 line of code required seven weeks to run its test suite. To resolve this issue, TCP technique was employed by several researchers (Kwon et al., 2014; Ramasamy et al., 2008; Sulaiman et al., 2021).

Evolutionary Algorithm (EA)

EAs are inspired by biological evolution, which consists of reproduction, selection, crossover, and mutation operators, hence they focused on evolution as a search strategy. The main logic of this is to find a near optimal solution by gradually mutating and recombining these existing solutions into newer solution so as a fitness function is improved in the process (Ensan et al., 2012). There are numerous search algorithms, includes the renowned Genetic Algorithm (GA) that inspired natural selection processes. Also, it provides the best solution to optimization and search problems. An objective function (fitness function) is well-defined to guide the search based on the objectives (Markiegi et al., 2017). In addition, GA can simply be implemented by generating initial population randomly, selection operator, crossover operators and mutation operators.

Evolutionary Cost-Cognizant Test Case Prioritization for Software Product Lines

In this section, we propose the application of regression test case prioritization techniques in the context of Software Product Line (SPL). As part of the proposal, we extracted a feature model sample from a FeatureIDE, and coverage information were generated from the source code of the feature model sample by applying path-based testing approach. This approach tends to prioritize test cases in such a way that, the test case cost and severity of fault varies, which gives room for detecting of fault based on its severity using less cost. Hence, it could provide evidence that when the testing process is halted, those severe faults will be detected using less cost. In view of this, it gives faster feedback about the system under test and detection of fault will be done earlier (Elbaum et al., 2002).

Figure 2 shows an overview of the test case prioritization process for SPL and how our approach fits on it. In the beginning, the approach started with the feature model extractor where the already designed feature model with its source code for this study was selected from FeatureIDE. The feature model was updated via its source code to suit our study. In addition, the selected test cases served as an input to the prioritization components whereby the test case encoder served as an encoder that convert test cases to numeric integer. Hence, the encoded integer now served as the solution to the problem where the initial population was generated randomly from the encoded integers. To obtain the size of the initial population, a random population of that size of test cases was generated. In that case, there is a need to compute the fitness value of the population by assigning costs to test cases and severities to faults, information from history of the previous TCP was employed. Fitness value computation can be achieved based on the cost of test cases that revealed faults and the severities of the revealed faults executed during the preceding TCP. Based on this, the initial population which can be referred to as (set of chromosomes) and the affected statements for each selected test cases were used to computes the fitness of each chromosome. Test case award value formula and APFDc are used for the computation and

evaluation of the fitness functions respectively. The implementation of the proposed approach is presented in the next section.

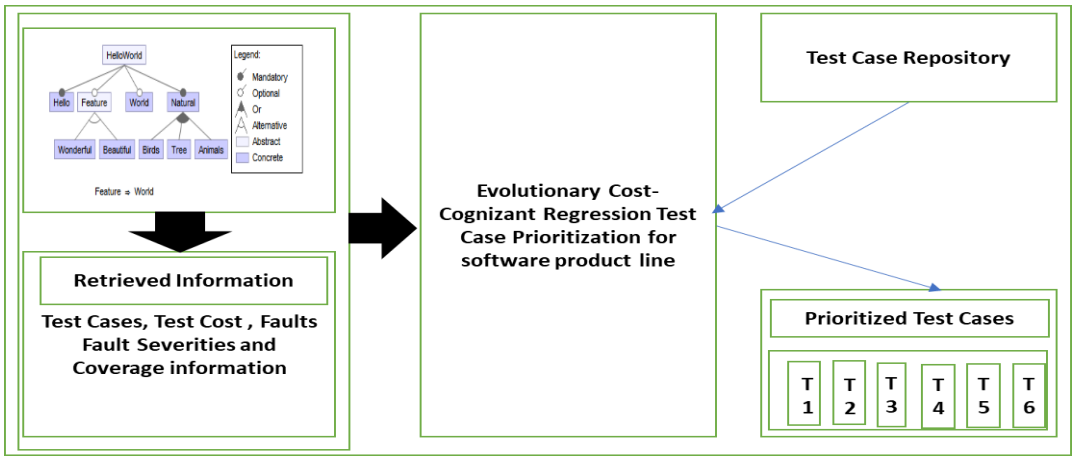

Figure 2: Overview ECRTCP for SPL

Test Case Prioritization Problem

Several TCP approaches for SPL are studied and their efficacies in terms of faults detection are evaluated. Most of the earlier published work (Malishevsky et al., 2006) assumed that all test cases are equally expensive, and all faults are equally severe, but in practice it is not. TCP can simply be defined as:

Given: A set of test suite Ts, A set of permutation P of Ts, and A function f, from $PTs \rightarrow R$ to the real numbers (default).

Problem: find $Ts'' \in PTs$ such that, $(\forall Ts'')(Ts'' \in PTs)(Ts'' \neq Ts')[f(Ts') \geq f(Ts'')]$.

Where: PTs is the set of possible prioritization orders of Ts and f is an objective function that applied to any order which yield an award value for that order.

Test Case Award Value (A_k)

The award value of test case (A_k) determines the position of test cases. When calculating for K-th, valid configuration was needed for software product line to calculate the award value of each feature/test case (gene) in a chromosome which is calculated by the below equation (1)

$$A_k = f_k \times \frac{df_{crit_{k-1}}}{cost_{k-1}} \quad (1)$$

Where: $f_{k:}$ - number of faults executed by test case in the kth configuration

$df_{critk\text{-}1}$: - total severities of the faults exposed by the test case and

$cost_{k\text{-}1:}$ - total cost of test cases in $k_{th\text{-}1}$ configuration. Test case with the highest award value will be executed first follow by the next.

Average Percentage of Fault Detection Per Cost (APFDc)

Since the goal of the proposed approach is to find the order of test suite that detects more severe faults at a lower cost. The cost-cognizant metrics APFDc was calculated by the formular in equation (2)

$$APFDc = \frac{\sum_{i=1}^{m}(f_i \times (\sum_{j=TF_i}^{n} t_j - \frac{1}{2}t_{TF_i}))}{\sum_{i=1}^{n} f_i \times \sum_{j=1}^{m} t_j} \quad (2)$$

Let f_i be the fault severity of fault i for ordering Ts', t_j be the cost of j_{-th} test case in the chromosome and TF_i be the first test case in the chromosomes that detects the fault i.

Implementation of the Proposed Approach

The main idea of this work is to ensure that, the proposed approach, evaluate fitness function and compute-award-value formula works together to prioritize test cases during TCP.

Test Case Award Value Computation

Test case award value in equation 1 determine the position of test cases whereby, as it input, it takes in CVRInfo (coverage information), changed objects, test case cost, test suite, feature criticalities, test case criticality and finally faults locations. Test case award value was returned as its output.

In the beginning, the Award Value formula was used to compute the award value of each gene (test case, feature) in a chromosome. The below examples give the detailed explanation on how the genes were computed. In chromosome1 i.e., C1 = {3, 2, 1, 6, 5, 4} are computed by collecting the feature information based on the coverage of the genes that formed the chromosomes.

The first genes (feature, test case) A3 of chromosome1 transverses 4 paths i.e., t3 = {Elevator, behavior, modes, shortestpath}

A3 = 4 * the criticality of the revealed faults/ cost of executing gene 3

According to Table 4.1, the executing cost of t3 (gene3) is 3, i.e

A3 = 4 * (1+2+3+6)/3 = 16

Gene2 (feature2) executes 4 statements i.e., t2 = {Elevator, behavior, modes, FiFo}

A2 = 4* (1+2+3+5)/4 = 11

Gene1 (feature1) executes 4 statements i.e., t1 = {Elevator, behavior, modes, sabbath}

A1 = 4 * (1+2+3+4)/5 = 8

Gene 6 (feature6) executes 4 statements i.e., t6 = {Elevator, behavior, priorities, floorpriority}

A6 = 4 * (1+2+8+9)/5 = 16

Gene5 (feature5) executes 3 statements i.e., t5 = {Elevator, behavior, priorities, rush Hour}

A5 =3* (1+2+7)/1 =30

Gene 4 (featue4) executes 3 statements i.e., t4 = {Elevator, behavior, service}

A4 = (1+2+3)/2 = 9

The award value of the selected 6 test cases (features, genes) are calculated in Table 6.

Table 6: The Award Value of Calculated Features

S/No	Gene (Test Case/ Feature)	Award value (A)	Prioritized Test Cases
1	1	8	t5
2	2	11	t6
3	3	16	t3
4	4	9	t2
5	5	30	t4
6	6	16	t1

The prioritized test cases T' = {5, 6, 3, 2, 4, 1}

APFDc Computation

Using the APFDc formula in equation 2, the approach computes the fitness values of the chromosomes in Table 5 as shown in Table 7

Table 7: Initial Population and Their Fitness Values

S/No	Chromosome	Fitness value (VcVs)
1	3, 2, 1, 6, 5, 4	0.596
2	2 ,1, 5, 3, 4, 6	0.717
3	5, 3, 6, 2, 4, 1	0.596
4	1, 3, 4, 2, 6, 5	0.492
5	5, 6, 4, 1, 3, 2	0.492
6	5, 6, 3, 2, 4, 1	0.642

Test Case Selection

Here, the best two chromosomes taken to the next generation for production are selected. The selected chromosomes with higher fitness functions are excellent for next generation chromosomes. The initial population chromosome and their respective fitness values are shown in Table 7. Based on the given table, chromosomes 2 and 6 have the highest fitness value and therefore, are considered fit for production in the next generation.

P1 = 2, 1, 5, 3, 4, 6

P2 = 5, 6, 3, 2, 4, 1

Crossover

In this approach, i.e., ECRTCP used a single point crossover on the selected chromosomes (P1 & P2). The crossover was done by generating an integer number n between 1 and the chromosomes size. Based on the size of our chromosomes which is 6, the n (random integer number) was chosen between n-1 and 6 to be the point of crossover. The child chromosome (C1, C2) produced from the parent chromosomes (P1& P2) are shown in figure 3. Hence, the crossover point as regard to this case was chosen to be k=3.

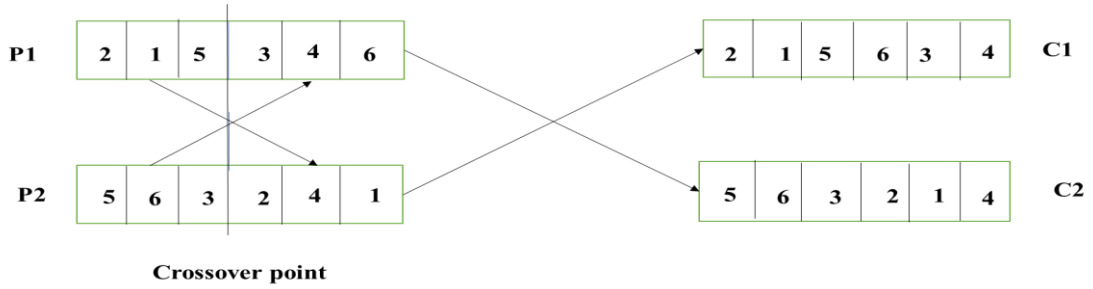

Figure 3: Diagrammatic Representation of Crossover

C1 = child 1 chromosome = {2, 1, 5, 6, 3, 4} and

C2 = child 2 chromosome = {5, 6, 3, 2, 1, 4}

Mutation

In this approach i.e., ECRTCP used Mutation to selects two random integer numbers from 0 to the size of the chromosomes minus 1, which will be from 0 to 5. Two integer numbers 1 and three were selected for which the number of the position selected are switched as shown in Figure 4. Hence, based on the mutation operation carried out, two new child chromosomes were produced (C1 and C2).

Figure 4: Diagrammatic Representation of Mutation Operation on Childs' Chromosomes

C1 = {2, 3, 5, 6, 1, 4} and
C2 = {5, 1, 3, 2, 6, 4}

Finally, we compute the fitness values of the newly chromosomes produced. Table 8: shows the first generation and their fitness values.

Table 8: First Generation and their Fitness Values

S/No	Code	Chromosome	Fitness Value
1	P1	2, 1, 5, 3, 4, 6	0.717
2	P2	5, 6, 3, 2, 4, 1	0.675
3	C1	2, 3, 5, 6, 1, 4	0.590
4	C2	5, 1, 3, 2, 6, 4	0.718

After the completion of the fitness values score of the newly produced chromosomes, the two best chromosomes were selected for reproduction. Whereas the remaining worse chromosomes are discarded from the population. This process continued until it reached termination stage.

RESULTS AND DISCUSSION

In this section, an experiment is presented to answer the following research questions;

RQ1: How can a GA-based evolutionary approach be used to propose test case prioritization for software product line?

RQ2: How can the proposed approach increase the Average Percentage of Fault Detection per Cost (APFDc) of test case prioritization for software product line?

Firstly, the experimental settings are described, followed by the explanation of the experimental results.

Experimental Settings

To assess our approach, we developed a prototype implementation for the proposed approach i.e., Test Case Prioritization for Software Product Lines using Genetic Algorithm (GA). The prototype inputs a program object and TCP approach and generates an ordered set of test cases. We used an open-source repository to obtain the program objects for this study (Ritterman & Klein, 2009). All the experiments performed were carried out on the same computer having the following specifications. HP, 2.00 GHz, 4.00GB RAM, 64-bit OS, x64-based processor, Java SE Development Kit (64-bit), FeatureIDE for Eclipse, and Mujava tool.

1. **Models:** For our experiment, we selected 1 program object having four different versions from open source repository (Ritterman & Klein, 2009). Table 9 lists the characteristics of the model namely: Object Program, Line of Code (LOC), Number of Classes (NOC), Number of Methods (NOM), Number of Tests (NOT), Traditional Mutants (#TM), Class Mutants (#CM), and All Mutants (#AM) are presented.

Table 9: Program Objects Used in our Experiments

Video Store	Program Object						
	LOC	NOC	NOM	NOT	#TMs	#CMs	#Ams
V1	218	3	18	18	235	3	238
V2	241	3	18	18	387	3	390
V3	260	4	22	22	396	4	408
V4	243	4	26	26	Nil	Nil	Nil

2. **Seeding of Fault:** In order to measure the efficiency and effectiveness of the approach, a mutation operator named Mujava (μjava) was used to seed faults into the code of SPL. Several researchers (Ma & Offutt, 2016; Bello, 2019) used it to evaluate their work. Similarly, the source program of the selected program objects was compiled and the .class file was placed in classes sub directory under Mujava directory. After executing the Mujava tool, the original program was mutated into a modified program, and it was stored in a result subfolder under Mujava directory. All the changes obtained when running the program are in the result folder as mutation log.
3. **Evaluation Metric:** To evaluate how fast severe faults are detected with minimal cost during testing, the Average Percentage of Fault Detection per Cost

(APFDc) metric was used (Bello, 2019; Askarunisa et al., 2010; Bello et al., 2018; Tulasiraman & Kalimuthu, 2018). This metric has been discussed in section III.

Experimental Setup

For each model presented in Table 9, a series of steps were performed, consisting of a test to treatment on the selected measures. A Randomized Complete Block Design (RCBD) was chosen for the study, with the experiment blocked on each object to ensure that each object used all the treatments only once. Each of the four treatments was randomly assigned to four program objects to run the experiment. Additionally, the faulted features for each program object were generated and compiled only once, which allowed for other approaches to used. The generated faulted features are stored temporary in features pool for reuse. Finally, for the analysis, the APFDc metric was used to compare the presented approach with an existing one with the help of descriptive statistical analysis

Experimental Results

Table 10 presented the experimental data of APFDc together with four versions of SPL video store used in this study. The best score in each row is highlighted in boldface and the average scores are shown in the last column. As illustrated, among all the experimental object used for the experiment, ECRTCP was the approach with the highest APFDc measure having the highest percentage scores as 94.48 for VS4, 91.84% for VS3, 90.93% for VS2 and 71.81% for VS1.

Furthermore, for the second-best approach, CEC had the following scores, VS2 (79.01%), VS1 (76.72%),), VS3 (76.39%) and VS4 (76.16%). followed by RNDP had the highest score as (80.77%) VS4, followed by the remaining scores as (72.93%) VS3, (69.82%) and (50.44%). The program object with least APFDc had a highest score for VS4 as (62.5%), followed by VS1 with (58.33%), then, VS3 (51.84%) and VS2 (48.34%). The approach with the least APFDc results were NOP and RNDP having the lowest APFDc results on three program objects. This suggested that those with highest APFDc score used cost in their approaches.

Table 10: Experimental Data of APFDc

Video store	Program Object							APFDc			
	LOC	**NOC**	**NOM**	**NOT**	**#TMs**	**#CMs**	**#AMs**	**NOP**	**RNDP**	**Km2017**	**ECR TCI**
Version 1 (VS1)	218	3	18	18	235	3	238	58.33	50.44	76.72	78.51
Version2 (VS2)	241	3	18	18	387	3	390	48.34	69.82	79.01	90.93
Version3 (VS3)	260	4	22	22	396	4	408	51.84	72.93	76.39	91.84
Version4 (VS4)	243	4	26	26	NIL	NIK	NIL	**62.50**	**80.77**	**79.16**	**94.48**
Average								**44.40**	**68.49**	**77.82**	**88.94**

Furthermore, the comparison of APFDc observed from the experimental data in Table 10 was illustrated in Figure 5 as a means of statistical approach.

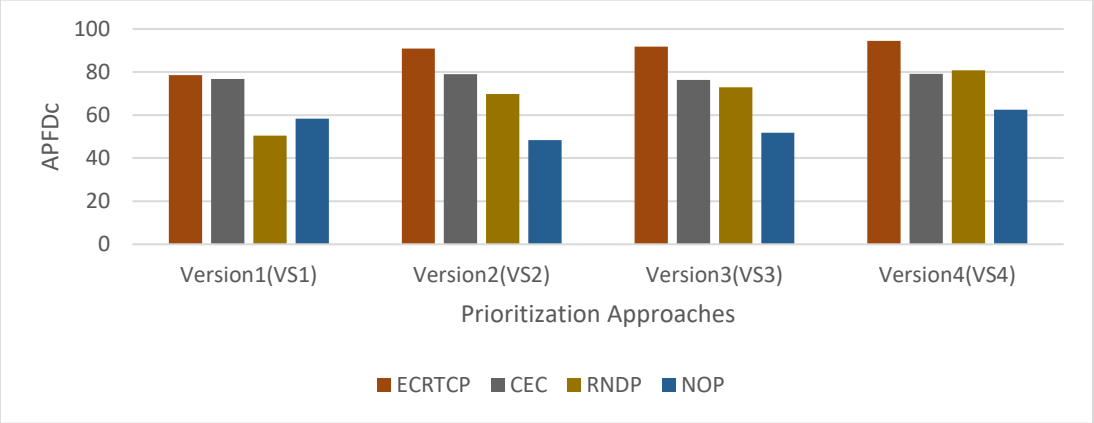

Figure 5: APFDc Analysis

Hence, with the results obtained from our computations on all the object programs, we decided to further the study based on the APFDc average scores on all the approaches. Bar chart was employed for this statistical analysis on all the approaches represented in Figure 6.

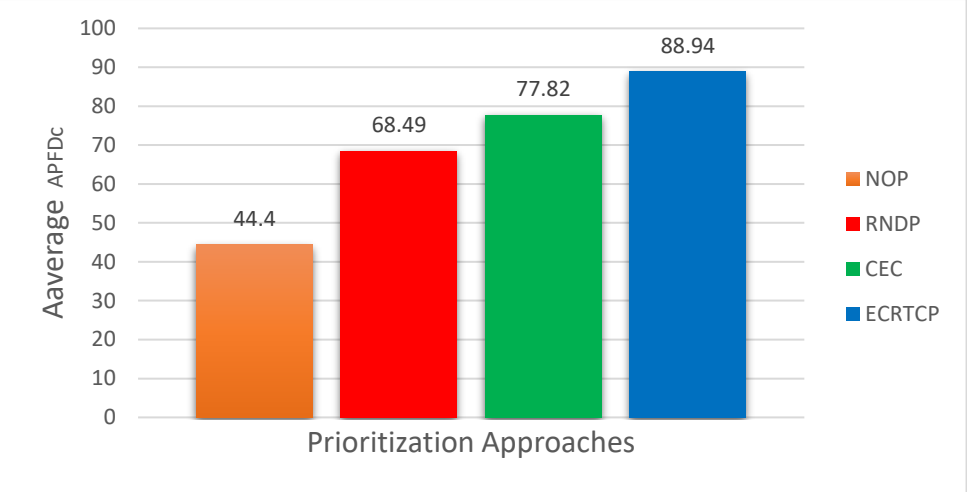

Figure 6: Average APFDc

The answer to RQ1 and RQ2 was solved from the results obtained. As regards to RQ1, the results shows that GA-based evolutionary approach is effective for prioritization of test cases for software product line. Concerning RQ2 the results shows that the proposed approach increases the Average Percentage of Fault Detection per Cost (APFDc) of test case prioritization for software product line as compared with other approaches which shows that, our approach has the highest percentage score.

Threat to Validity

The study's validity is threatened by several key assumptions;
a. Issues of reuse without modification, it assumes that the features, faults, and test pools, which were not selected based on any principles, are sufficient for the experiment. There's no assurance that the test pool has the same coverage and fault detection rate as a rigorously selected one. Secondly, the Mujava tool was to automatically seed faults, which assumes that the programs won't have similar complexity, regardless of how this is measured. This creates a risk that the experimental results could vary depending on the specific fault injected.
b. To determine the efficacy of the prioritization approach, APFDc was not the only measures that could achieved that. The approach does not count for the possibilities that faults and test cases may have different severities and costs, it can partially capture the essential aspect of prioritization.
c. An evolutionary search and optimization based on genetic algorithm concept designed for finding the optimal ordering of test cases was used. The algorithm may not only find the true optimal ordering of the experiment. This is because, the approach was conventional and intentionally ignored several factors: the cost of human intervention, the cost of debugging, and the time spent on prioritizing test cases.
However, it would have been more effective to use real faults and faults injected by hand. Although, the analysis assumes that all parts of the program objects were considered.

CONCLUSION

In this paper, we presented an automated regression test case prioritization for software product line. The approach was implemented in java and used the FeatureIDE tool to create the feature model. It utilized a Genetic Algorithm based on foundational theoretical concepts.
Similarly, the approach ECRTCP was empirically evaluated and compared against three other methods: CEC, RNDP and NOP. The result showed that the proposed method ECRTP for SPL had the highest APFDc, indicating that, it effectively prioritized costs and fault severities.
For the future work. Firstly, is by continuing research in cost-cognizant for software product lines, by creating a fully automated regression test case prioritization for software product lines and lastly, to evaluate this study with a larger program object.

REFERENCES

Al-hajjaji, M., Lity, S., Lachmann, R., Th, T., Schaefer, I., & Saake, G. (2017). Delta-Oriented Product Prioritization for Similarity-Based Product-Line Testing. https://doi.org/10.1109/VACE.2017.8

Al-Hajjaji, M., Thüm, T., Meinicke, J., Lochau, M., & Saake, G. (2014). Similarity-based prioritization in software product-line testing. ACM International Conference Proceeding Series, 1(September), 197–206. https://doi.org/10.1145/2648511.2648532

Bello, A., Sultan, A. M., Ghani, A. A. A., & Zulzalil, H. (2018). Evolutionary cost cognizant regression test prioritization for object-oriented programs based on fault dependency. International Journal of Engineering and Technology(UAE), 7(4), 28–32. https://doi.org/10.14419/ijet.v7i4.1.19486

Bello abdulkarim. (2019). Evolutionary cost-cognizant regression test case prioritization for object-oriented programs abdulkarim bello fsktm 2019 6.

Devroey, X., Perrouin, G., Legay, A., Schobbens, P., & Heymans, P. (2016). Search-based Similarity-driven Behavioural SPL Testing.

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

Elbaum Sebastian; Rothermel, G. (2001). Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization.

Elbeltagi, E., Hegazy, T., & Grierson, D. (2005). Comparison among five evolutionarElbeltagi, E., Hegazy, T., & Grierson, D. (2005). Comparison among five evolutionary-based

optimization algorithms. 19, 43–53. https://doi.org/10.1016/j.aei.2005.01.004y-based optimization algorithms. 19, 43–53. https://doi.org/10.1016/j.aei.2005.01.004

Ensan, A., Bagheri, E., Asadi, M., Gasevic, D., & Biletskiy, Y. (2011). Goal-oriented test case selection and prioritization for product line feature models. Proceedings - 2011 8th International Conference on Information Technology: New Generations, ITNG 2011, 291–298. https://doi.org/10.1109/ITNG.2011.58

Ensan, F., Bagheri, E., & Gaˇ, D. (2012). Evolutionary Search-Based Test Generation for Software Product Line Feature Models. 613–628.

Fischer, S., Lopez-Herrejon, R. E., & Egyed, A. (2018). Towards a fault-detection benchmark for evaluating software product line testing approaches. Proceedings of the ACM Symposium on Applied Computing, 2034–2041. https://doi.org/10.1145/3167132.3167350

Kumar, S. (2017). cost-based test case prioritization technique for software product line. 40(115).

Kwon, J., Ko, I., Rothermel, G., & Staats, M. (2014). Test Case Prioritization Based on Information Retrieval Concepts. https://doi.org/10.1109/APSEC.2014.12

Li, S., Bian, N., Chen, Z., You, D., He, Y., & Prioritization, A. T. C. (2010). A Simulation Study on Some Search Algorithms for Regression Test Case Prioritization. 72–81. https://doi.org/10.1109/QSIC.2010.15

Ma, Y., & Offutt, J. (2016). Description of muJava ' s Method-level Mutation Operators. 1–4.

Malhotra, R. (2015). Empirical research in software engineering : concepts, analysis, and applications. CRC press.

Malishevsky, A. G., Ruthruff, J. R., Rothermel, G., & Elbaum, S. (2006). Cost-cognizant Test Case Prioritization. Department of Computer Science and Engineering University of NebraskaLincoln Techical Report, TR-UNL-CSE-2006-0004, 1–41. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.131.6542&rep=rep1&type=pdf

Markiegi, U., Arrieta, A., Sagardui, G., & Etxeberria, L. (2017). Search-based product line fault detection allocating test cases iteratively. 123–132. https://doi.org/10.1145/3106195.3106210

MS. A. Askarunisa, MS. L. Shanmugapriya, D. N. R. (2010). Cost and Coverage Metrics for Measuring the Effectiveness of Test Case Prioritization Techniques.

Panigrahi, C. R., & Mall, R. (2014). A heuristic-based regression test case prioritization approach for object-oriented programs. Innovations in Systems and Software Engineering, 10(3), 155–163. https://doi.org/10.1007/s11334-013-0221-z

Qureshi, S. A. (2018). Test Case Prioritization for Software Product Line Testing. CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD.

Raju, S. (2012). Factors Oriented Test Case Prioritization Technique in Regression Testing using Genetic Algorithm. 74(3), 389–402.

Ramasamy, K., Ieee, M., & A, S. A. S. A. M. S. (2008). Incorporating varying Requirement Priorities and Costs in Test Case Prioritization for New and Regression testing. 2.

Ritterman, J., & Klein, E. (2009). OOP Lab 3 Exercises : Instances , Arrays , and Encapsulation Exercises Exercise 1 : Looking at the Weather. 1–11.

Rothermel, G., Untcn, R. H., Chu, C., & Harrold, M. J. (2001). Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering, 27(10), 929–948. https://doi.org/10.1109/32.962562

Sabharwal, S., Sibal, R., & Sharma, C. (2010). Prioritization of test case scenarios derived from activity diagram using genetic algorithm. 2010 International Conference on Computer and Communication Technology, ICCCT-2010, 481–485. https://doi.org/10.1109/ICCCT.2010.5640479

Sahak, M., Jawawi, D. N. A., & Halim, S. A. (2017). An Experiment of Different Similarity Measures on Test Case Prioritization for Software Product Lines. 9(3), 177–185.

Sanchez, A. B., Segura, S., & Ruiz-Cortes, A. (2014). A Comparison of test case prioritization criteria for software product lines. Proceedings - IEEE 7th International Conference on Software Testing, Verification and Validation, ICST 2014, 41–50. https://doi.org/10.1109/ICST.2014.15

Sulaiman, R. A., Jawawi, D. N. A., & Halim, S. A. (2021). A Dissimilarity with Dice-Jaro-Winkler Test Case Prioritization Approach for Model- Based Testing in Software Product Line. 15(3), 932–951.

Tulasiraman, M., & Kalimuthu, V. (2018). Cost Cognizant History Based Prioritization of Test Case for Regression Testing Using Immune Algorithm. Journal of Intelligent Engineering and Systems, 11(1), 221–228. https://doi.org/10.22266/ijies2018.0228.23

Wohlin, C., Runeson, P., H¨ost, M., Ohlsson, M. C., Regnell, B., & Wessl´en, A. (2012). Experimentation in SOftware Engineering. London: Springer Heidelberg New York Dordrecht. In Springer Heidelberg New York Dordrecht. london. https://doi.org/10.1007/978-3-642-29044-2

Yoo, S., & Harman, M. (2010). Regression testing minimization, selection and prioritization: a survey. Software Testing, Verification and Reliability, n/a-n/a. https://doi.org/10.1002/stvr.430