


ITC 3/50 Information Technology and Control Vol. 50 / No. 3 / 2021 pp. 443-457 DOI 10.5755/j01.itc.50.3.27662	A Prioritization Approach for Regression Test Cases Based on a Revised Genetic Algorithm	
	Received 2020/09/07	Accepted after revision 2021/08/04
	 http://dx.doi.org/10.5755/j01.itc.50.3.27662	

HOW TO CITE: Alrawashdeh, T. A., ElQirem, F., Althunibat, A., Alsoub, R. (2021). A Prioritization Approach for Regression Test Cases Based on a Revised Genetic Algorithm. *Information Technology and Control*, 50(3), 443-457. <https://doi.org/10.5755/j01.itc.50.3.27662>

A Prioritization Approach for Regression Test Cases Based on a Revised Genetic Algorithm

Thamer A. Alrawashdeh

Department of Software Engineering, Alzaytoonah University of Jordan, Airport Street, Amman, Jordan;
e-mail: thamer.r@zuj.edu.jo

Fuad ElQirem

Department of Multimedia, Alzaytoonah University of Jordan, Airport Street, Amman, Jordan;
e-mail: fouad.q@zuj.edu.jo

Ahmad Althunibat

Department of Software Engineering, Alzaytoonah University of Jordan, Airport Street, Amman, Jordan; e-mail: A.thunibat@zuj.edu.jo

Roba Alsoub

Department of Computer Science, Mutah University, Mutah, Alkarak, Jordan; e-mail: robas@mutah.edu.jo

Corresponding author: thamer.r@zuj.edu.jo

The regression testing is a software-based testing approach executed to verify that changes made to the software do not affect the existing functionality of the product. On account of the constraints of time and cost, it is impractical to re-execute all the test cases for software whenever a change occurs. In order to overcome such a problem in the selection of regression test cases, a prioritization technique should be employed. On the basis of some predefined criterion, the prioritization techniques create an execution schedule for the test cases, so the higher priority test cases can be performed earlier than the lower priority test cases in order to improve the efficiency of the software testing. Many prioritization criteria for regression test cases have been proposed in software testing literature; however, most of such techniques are code-based. Keeping in view this fact, this

research work has proposed a prioritization approach for regression test cases generated from software specifications which are based on the criterion of the Average Percentage Transition Coverage (APTC) by using a revised genetic algorithm. This criterion evaluates the rate of transitions coverage by incorporating knowledge about the significance of transitions between activates in the form of weights. APTC has been used as a fitness evaluation function in a genetic algorithm to measure the effectiveness of a test cases sequence. Moreover, in order to improve the coverage percentage, the proposed approach has revised the genetic algorithm by solving the problem of the optimal local solution. The experimental results show that the proposed approach demonstrates a good coverage performance with less execution time as compared to the standard genetic algorithm and some other prioritization techniques.

KEYWORDS: Regression testing, Test Cases Prioritization, Genetic Algorithm, Average Percentage Transition Coverage.

1. Introduction

Software maintenance is considered as a crucial process in the cycle of software development. Often, two-thirds of software development cost is allocated to software maintenance (Pressman, 2005). The software maintenance is frequently carried out to correct errors, to append a new function or improve an existing function of software or to adapt it to new software or hardware (Biswas et al., 2011). Whenever a maintenance activity is executed, the regression test is carried out to verify that the modified parts work correctly and meet the software specifications. Thus, the software testing that includes the regression test makes software robust, more effective and trustworthy.

However, the regression test is still a hard task due to constraints of time and cost. Retesting a software by using the complete range of test cases is expensive and inefficient (Sur et al., 2019, Harikarthik et al., 2019, Yoo and Harman, 2012, Engström and Runeson, 2010). Thus, a prioritization technique should be employed to facilitate the regression test process. Many methods have been proposed in the regression testing literature to overcome such problems. Although the regression test is carried out repeatedly throughout the cycle of software development (Konsaard and Ramingwong, 2015, Ekelund and Engström 2015, Kavitha and Sureshkumar, 2010), most of such techniques are code-based techniques which prove to be useful in unit testing but not in case of functional testing and face the scalability issues with respect to big and complicated software systems (Panda et al., 2019, Sapna and Balakrishnan, 2015). Thus, generation and prioritization of regression test cases from software specifications could be considered as an optimization

in which meta-heuristic methods could be employed one of such methods is Genetic Algorithm (GA).

Genetic algorithm is a robust algorithm which is employed to overcome the optimization problems and it is based on the theory of natural selection and concepts of evolutionary biology (Guo, 2020; Yang et al., 2016). This algorithm is being used in the computing areas especially in the software testing because of its efficiency in providing a right solution for complicated, discrete and nonlinear issues produced by a complicated software (Dey et al., 2020; Vinitha and Preetha, 2018). It could be applied to reduce the effort and cost by creating test cases automatically and thereby Software maintenance is considered as a crucial process in the cycle of software development. Often, two-thirds of software development cost is allocated to software maintenance [29]. The software maintenance is frequently carried out to correct errors, to append a new function or improve an existing function of software or to adapt it to new software or hardware [5]. Whenever a maintenance activity is executed, the regression test is carried out to verify that the modified parts work correctly and meet the software specifications. Thus, the software testing that includes the regression test makes software robust, more effective and trustworthy.

However, the regression test is still a hard task due to constraints of time and cost. Retesting software by using the complete range of test cases is expensive and inefficient [13, 16, 34, 39]. Thus, a prioritization technique should be employed to facilitate the regression test process. Many methods have been proposed in the regression testing literature to overcome such problems. Although the regression test is carried out

repeatedly throughout the cycle of software development [12, 17, 21], most of such techniques are code-based techniques which prove to be useful in unit testing but not in case of functional testing and face the scalability issues with respect to big and complicated software systems [5, 28, 31]. Thus, the generation and prioritization of regression test cases from software specifications could be considered as an optimization in which meta-heuristic methods could be employed one of such methods is Genetic Algorithm (GA).

Genetic algorithm is a robust algorithm which is employed to overcome optimization problems and it is based on the theory of natural selection and concepts of evolutionary biology [15, 38]. This algorithm is being used in the computing areas especially in the software testing because of its efficiency in providing a right solution for complicated, discrete and nonlinear issues produced by a complicated software [11, 36]. It could be applied to reduce the effort and cost by creating test cases automatically and thereby significantly enhances the software testing efficiency. Despite that, the most challenging obstacle that could be encountered in the application of the genetic algorithm is that it could be trapped in the optimal local solution which leads to population ageing [11, 16, 34, 38]. In order to overcome this problem, various methods have been proposed to improve or adjust some factors such as parameter settings, fitness function, genetic operations, and chromosome population [38]. However, the proposed methods have some difficulties that make them unattractive to the software testing including difficulty implementing these methods without extra effort because of the highly complex nature of the enhancements [38]. Therefore, in this research work, a Revised Genetic Algorithm has been proposed to solve the problem of optimal local solution easily and effectively, which is meant to be used for prioritizing the test cases that are generated from the software specifications.

On the other hand, the Unified Modeling Language (UML) is the most popular standards for modeling the software specifications and software design. It includes various models to support software systems development with an object-oriented approach. These models include use case diagram, use case description, sequence diagram, class diagram, activity diagram and state diagram to model both dynamic and static behavior of software systems at different levels of abstraction [10, 37, 6]. As this study includes

functionality level, the activity diagram is employed. Activity diagrams are used to elaborate the scenario related to each use case (functionality) in the software systems. It involves the main, alternate and exception scenarios that deal with the functionality. Thus, the number of test cases that are generated from use case description models by using an automated approach are exhaustive [26]. However, there are relatively few particular techniques proposed to generate test cases from models in analysis and design phases especially the activity diagram [1, 3, 4]. Moreover, extract information from the activity diagram is a complicated task because of the activity diagram provides concepts at a higher abstraction level of a system [35].

A regression test selection is carried out to ensure that developed functionality, both existing and modified, work appropriately by selecting the only subset of test cases that were developed initially to test the software. The regression test selection problem has been introduced as follows: let P be a software program and \tilde{P} a revised version of P . Similarly, let T a test suite that had been developed initially to test a software program during the software development phase. The regression test selection techniques aim to select a sub-set of valid test cases from an initial test suite ($\tilde{T} \subseteq T$) to test that the existing and modified parts of \tilde{P} continue to work properly such that every error is detected when \tilde{P} is executed with T .

The regression test selection aims to select a subset of test cases to be used to test the functionality of the software without affecting the software quality [33, 23]. In this research work, the selected test cases will be considered as a regression test. This regression test will use sub-set of the test cases to verify the functionality of the software with parts of software that have been changed. The premise is that selecting minimum test cases to verify software functionality with respect to changes before elaborately testing the functionality to ensure that all selected test cases are more effective. The regression test selection helps in two ways: it is used to ensure that the functionality that passes the initial test cases is tested further to make sure that no new defects have been produced by the changes in the previously validated functionality; otherwise, the changes that have been done on the functionality are considered void.

Although many research works have proposed different techniques to generate regression test cases, a great

number of such techniques are white-box techniques and a few are black-box techniques. Furthermore, the changes in the functionality of software have not been considered in the proposed black-box techniques. In the same respect, the constraints of cost and offers of software system development provide a scope to propose new ways to enhance the software development processes, including the testing process. Thus, this research work has been conducted to propose an automated approach to prioritize minimal regression test cases generated from UML models (use case description model) by using the Revised Genetic Algorithm. In such an approach, minimum regression test cases are generated to test the changes of software functionality. These minimum test cases suggest the defected software functionality that needs to be reworked before conducting further testing. Another contribution of this research is to revise the genetic algorithm to solve the problem of local optimal solution.

Many research works had employed the heuristic search algorithms to propose an approach to generate or to prioritize the regression test cases [2, 8, 14, 24, 18, 19].

Vinitha and Ramakrishna [36] proposed a multi-objective Regeneration Genetic Algorithm (RGA) approach to enhance the coverage percentage and to reduce the loop statements by applying the coverage and loop statements in the calculations of populations' fitness. This proposed approach met both coverage and loop conditions of conditional statements. As experimental results, this study demonstrated that the proposed approach offers better results in terms of execution time, the number of covered methods and branch coverage.

An important intelligent method, called a Regenerate Genetic Algorithm (RGA) was developed by Yang et al. [38] in the domain of automatic test cases development to solve the problem of population ageing. The proposed method defined the population ageing factor and process to estimate the population ageing degree. This method was utilized to jump out of the optimal local solution, when the population ageing has occurred, to prevent the population ageing and effectively enhance the test coverage. Compared with the standard genetic algorithm, the experimental results of this method showed that it could effectively improve the search efficiency, minimize the number of test cases, enhance test coverage and restrain pop-

ulation ageing. Despite the promising results of this study, there were some limitations; for example, it did not provide a fully automated approach to generate the test cases and does not prioritize the test cases since it only determines a best test case.

A new approach was introduced by Mala et al. [25] to optimize the generation of test cases from the source code by applying the artificial bee colony (ABC) optimization. The proposed approach was used for combining the local search methods executed by employed and onlooker bees with global search methods managed by scouts. Since the three bees technique is employed to accomplish the solution generation faster. This approach was evaluated on the basis of the criteria of coverage-based test adequacy and compared with genetic algorithm-based approaches, random testing and sequential ABC. The experimental results showed that the proposed approach provided better results in the test suite optimization. Although this research work optimized the generated test cases, it neither automated the process of test cases generation nor explained how to utilize the proposed method in the regression test.

Khurana and Chillar [20] developed a new method to generate test cases from a combination of two Unified Modeling Language (UML) models: sequence diagram and statechart diagram. In this method, the sequence diagram and state chart diagram were converted into sequence graph and state graph, respectively. Both graphs were combined to create a system testing graph. Different control flow sequences called test cases were identified from the system testing graph. Finally, the genetic algorithm (GA) was employed to optimize the generated test cases. By implementing this proposed method, all possible test cases could be discovered and various faults could be solved, including pre-post condition faults, integration, error handling, scenario faults and operational faults. However, this research did not evaluate the proposed method and has not prioritized the generated test cases.

Alrawashed et al. [2] developed an automated approach for test suite optimization. In their approach, they generated test cases from the use case description model. This approach was utilized for converting the statements in such a model into the sequence graph. From the graph, various control flow sequences were generated. Consequently, the heuristic search algorithm (genetic algorithm) was applied to optimize the

generated test cases. The experimental results of this research work showed that the proposed approach was efficient and effective in offering a near-optimal test case and high test coverage in the early stage of software development. Despite that, this research work could not solve the main problem of the genetic algorithm which could significantly harm the benefits of the genetic algorithm in the software testing area and increase the effort and cost of the test cases generation. Additionally, the proposed technique in this work did not prioritize the test cases and only provided the optimal path.

Another research work was done for the test suite optimization by Sahoo et al. [30]. In this work, the authors proposed an approach to generate the test cases from state chart and sequence diagrams. The proposed approach converted the sequence diagram and state chart diagram into the sequence graph and state chart graph, respectively. In order to create a system graph, both graphs were combined and from the system graph, different test cases have been discovered and optimized by applying an evolutionary algorithm called hybrid bee colony algorithm. The results of this research showed that the proposed technique minimized the time required to select the best bath and it was more effective in software testing. However, concerns such as how to benefit from the proposed technique in the regression test along with the generated test cases remained unclear and were not prioritized in this study. Another limitation of the study was that the proposed technique remained unevaluated.

Konsaard and Ramingwong [21] proposed an approach-based genetic algorithm to prioritize the generated test cases. This approach generated the test cases from the source code and then the fitness function in the genetic algorithm was modified by using Average Percentage Code Coverage (APCC) to yield maximum code coverage. The result of this research work showed that the proposed approach was efficient in terms of coverage percentage and execution time. Despite the promising results of this approach, it was not a fully automated approach. Additionally, it did not consider the problem of optimal solution in the genetic algorithm.

After reviewing more than fifty studies, it has become evident that no research work has been conducted so far to propose a fully automated approach to generate the regression test cases from software specifications

and to prioritize such test cases. Interestingly, most of such studies proposed approaches to generate the test cases from the software code or software specification to find the optimal test case that could achieve maximum coverage without prioritizing other test cases. Additionally, a few other studies were done to solve the problem of optimal solution that resulted from the application of the genetic algorithm. Therefore, more studies are required to support the results of these studies. Furthermore, most of the proposed prioritization techniques select the test cases on the basis of their ability to cover more faults without considering the test adequacy. For this end, this research work intends to bridge the gap in the related literature by proposing a fully automated and a complete approach to generate the regression test cases from the software specifications and to prioritize these test cases by using the genetic algorithm associated with solving the problem of population ageing and obtaining a maximum test suit coverage. Furthermore, this is the first study that has adapted the criteria of Average Percentage Transition Coverage (APTC) to evaluate the proposed approach in term of coverage percentage. Therefore, the goal of this research work is to provide an answer to the following research question: how does the proposed approach compare in term of time, effort and coverage to the other search-based approaches?

The rest of this research work is organized as follows: section two introduces the proposed technique. Section three addresses the experimentation, results and discussion on the study. Finally, the conclusion and future work are included in section four.

2. The Proposed Approach

This research work aims to propose an automated approach to generate and prioritize the regression test cases from software specifications (activity diagram) on the basis of Revised Genetic Algorithm. In order to construct the proposed approach, the genetic algorithm has been revised to solve the problem of local optimal solution and then used to prioritize the regression test cases. The proposed approach consists of three steps: 1) automatically converting the activity diagram into a Control Flow Graph (CFG) 2) automatically generating the test cases from the CFG 3)

prioritizing the regression test cases using a Revised Genetic Algorithm. Figure 1 shows the proposed approach, where all steps of the proposed approach are discussed in detail below.

Step one: converting the activity diagram into CFG

the activity diagram is employed to model the dynamic behavior of a set of objects in the software systems. Interestingly, the activity diagram represents a set of objects activities, so it could be used to describe the operations in the design stage, the sequence of activities among the involving objects in the control flow, and the relations between activities and objects in the message flow. Furthermore, it details the main, alternative and exception scenarios related to each use case [31]. Thus, the activity diagram allows determining coverage criteria to assure a particular degree of a completeness of the regression test scenarios. The activity diagram involves two types of activities: action activities (events) and control activities including the initial activity, last activity, decisions, merge and fork. Thus, we can make the following definition from the description of the construction of an activity diagram.

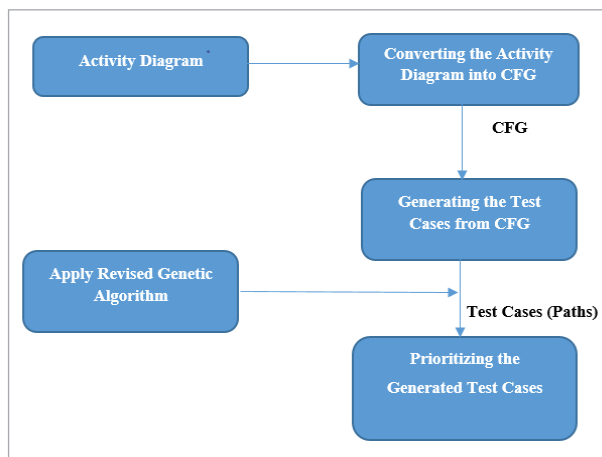
Definition 1: an activity diagram with a set of action activities and control activities can be represented formally as:

$$AD = \{A, T, J, F\}. \quad (1)$$

where A denotes a finite set of activities (a_1, a_2, \dots, a_n); T represents a finite set of control flow (t_1, t_2, \dots, t_n) from an activity to another; J represents a finite set of forks (f_1, f_2, \dots, f_n); and F denotes a finite set of joints (J_1, J_2, \dots, J_n).

Figure 1

Schematic Diagram of the Proposed Approach



In this phase, the scenarios from the activity diagram are extracted to be used in the generation process of regression test cases. Interestingly, these scenarios are converted into control flow graph. Wherein, each node in the CFG corresponds to an action or control activity in the activity diagram and each edge represents the control flow between the activities. Figure 2 presents the algorithm of converting the activity diagram into a control flow graph.

Figure 2

Algorithm of converting an activity diagram into CFG

Input: action and control activities, and control flow ϵAD

Output: Control Flow Graph (CFG)

Identify all Nodes (N) in the CFG: each node in the CFG represents an activity in the activity diagram

Identify the Root (R) of CFG: the first activity (a_1) in the activity diagram

$E1 = R \rightarrow a_2$ // the first edge (E1) connects the first activity with the second one (a_2) (starting from the second activity)

For each a_i

IF $C_i \in a_i$ // if a current activity is a control activity (C)

$E_i = a_i \rightarrow a_{i+1}$

$E_{i+1} = a_i \rightarrow a_{i+2}$

Else

$E_i = a_i \rightarrow a_{i+1}$

End if

Step two: Generating the test cases from CFG

In this phase, the control flow graph of the activity diagram from step one is used as an input. Consequently, the proposed technique generates all possible paths (regression test cases) in a control flow graph. Wherein a path is a finite set of nodes and edges (transitions) from the initial node to the final node. It is worth mentioning that each independent path has at least one new edge in a control flow graph. Additionally, each decision divides the path into two separate paths in the CFG: true path and false path. In software testing, the test cases must traverse each path at least for once through the CFG. Thus, a new concept has been defined as definition 2.

Definition 2: A regression test case could be defined as an execution path from the initial activity to the final activity as the following.

$$tc \in TC, tc = \{a_1, t_1, \dots, a_n, t_n\}. \tag{2}$$

where, tc represents a regression test case; TC denotes a set of regression test cases; a represents a node in the control flow graph and t means an edge in the control flow graph. Figure 3 shows the algorithm of Generating test paths from CFG.

Figure 3

Algorithm of Generating Test Paths from CFG

```

Input: control flow graph of the activity diagram
Output: Regression Test Cases (test paths)
TC: identify all test cases // list all regression test cases
(test paths) TC= tc[1], tc[2]....tc[n] from the start node
to the final node in the control flow graph.
For each path tc[i] ∈ TC
CurrentNode(CN) = StartNode (SN)
While (CN != final node) do
Tc[i].add(CN)
End
    
```

Step three: Prioritizing the generated test cases by using the Revised Genetic Algorithm (RGA)

Genetic algorithm is one of the most popular meta-heuristic techniques used to optimize software testing data [36]. Although there is no a specific definition of GA, the implementations of such algorithm share the same components including: populations, selection, crossover, and mutation operations. The population of GA involves a large number of individuals and subject to the reproduction and mutation.

the standard work of the genetic algorithms consists of five steps. First step: an initial population of n chromosomes (individuals) is randomly produced. Second step: A fitness function is applied to assign a fitness value to each chromosome. In the third step three sub-steps are repeated until m new individuals are produced: 1) the selection operation selects two individuals to reproduce, 2) the crossover operation combines genetic materials from both individuals with probability pc, and 3) the mutation operation changes offspring with probability pm. in the fourth step the replacement operation selects the individuals that will survive to the next generation of the population. Fifth step: if the termination conditions are not met go to the second step [38].

The application of this technique could reduce the effort and improve the test coverage criterion. However, the big challenge associated with implementing the GA is that the population is often trapped in the optimal local solution without continued enhancement of the test coverage (population ageing) [22, and 38]. The following sub-section addresses the population ageing problem that appeared in the software testing which is based on the genetic algorithm.

2.1. The Ageing Problem in Software Testing

There is a big difference between the maturation of the genetic algorithm and the problem of population ageing. Regarding the improvement in the software testing coverage, occasionally when there is a big difference in populations, the improvement in the coverage is slight. On the other hand, a little difference in populations could lead to more extensive coverage. Consequently, in order to further clarify the problem of population ageing, a new concept called ageing factor has been defined giving the ageing degrees of the populations produced during the process of genetic algorithm-based testing.

Assume that $t_{c,i,j}$ is the number of regression test cases in the i th individuals in the j th generation. Additionally, TC_{popt} is the number of individuals in j th generation, in other words TC_{popt} represents the number of regression test cases in the j th generation. The total number of regression test cases in all j th generations could be presented as $\sum_{j=1}^i \sum_{t=1}^{N_{popj}}$. In the same case, the total number of regression test cases in each generation j , generation $j+1$, generation $j+2$... generation $j+n$ are presented in Equation 3:

$$\sum_{j=1}^i \sum_{t=1}^{N_{popj}} nt_{,i} \dots \sum_{j=1}^{i+m} \sum_{t=1}^{N_{popj}} nt_{,i+m}. \tag{3}$$

If the population includes m generations and the population coverage is still the same, this situation indicates that the population is ageing and trapped in the optimal local solution [22, 38]. Equation 4 represents this situation in which q denotes the ratio of the number of newly produced regression test cases that do not improve the coverage promotion in m generations in comparison with a prior number of useful test cases (ageing factor).

$$q = (\text{the increased number of tc}(i) / \text{total number of tc}) * 100\% \quad (4)$$

As shown in Equation 4, the ageing factor is influenced by the increased number of generation (i). The more regression test cases that do not increase the coverage, the higher the degree of population ageing. Consequently, the maturity of the population does not mean that the population ageing will not happen. In the same sense, if the population is not mature, there is a possibility of occurrence of the population ageing on account of the interaction of genetic algorithm with searching space.

On account of the ageing process, regression test cases (individuals) could not improve the transition coverage in software specifications space (activity diagram). In this case, the operations of the genetic algorithm will lose their ability to optimize the transition coverage. So, one of the objectives of this research work is that, after generating a second population, despite the current coverage of the population is stagnate at the same stage, the population evolution process is still anticipated to improve the coverage by using some strategies.

2.2. Transition Coverage Based Software Testing

The target of software testing with respect to the criteria of transition coverage can be presented as follows:

$$TS = \{G, I, A, C, T, Cov, Np(A)\} \quad (5)$$

In Equation 5, G denotes the CFG of an activity diagram under test; I is the input space; A represents the adopted optimization algorithm; C denotes a suite of test cases; T is a set of termination conditions; Cov represents the test coverage and Np(A) represents genetic iterations number. Furthermore:

$$Cov = \{TrCovG(C)\} \quad (6)$$

In Equation 6, the TrCovp(C) represents a transition coverage. Further:

$$A = \{E, SO, M, Sel, Cor, Mut, F\} \quad (7)$$

In Equation 7, E represents the mode of genetic code; P0 denotes the initial population; M denotes the pop-

ulation size; Sel indicates the selection factor; Cor indicates the crossover factor; Mut indicates the mutation factor and F represents the adoption fitness function.

In the black-box software testing, the coverage has been calculated by using Equation 8. In this equation, the TrExecG(C) represents a set of transitions that are covered in a control flow graph (CFG); TrG indicates the set of transitions in a control flow graph (CFG). For the transition coverage of test suite C, TrExecG(C) is defined as the ratio of the transitions that could be executed by the test suite (C) to the total number of transitions in the CFG of a software activity diagram.

$$TrExecG(C) = |TrExecG| / |TrG| \quad (8)$$

In this research work, in order to take the genetic algorithm's advantages of prioritizing the test cases, a fitness criterion has been added. This criterion states that the minimal number of test cases that could achieve a maximum transition coverage TrExecp(C) should be applied. Therefore, the following sub-section presents the Revised Genetic Algorithm that solves the problem of population ageing by prioritizing the test cases on the basis of transition coverage criteria.

2.3. Revised Genetic Algorithm

This section deals with a Revised Genetic Algorithm which is used to solve the problem of population ageing and to prioritize the test cases. Accordingly, if the population ageing occurs when a significant number of populations are produced and the transition coverage is not improved, in this case, the operation of population regeneration should be triggered, so that a new population is produced and the processes of the genetic algorithm are executed successively.

2.3.1. Basic Population in Genetic Algorithm-Based Software Testing

In the GA-based software testing, the population in GA is represented by individuals including the set of test cases as shown in Equation 9 with a transversal vector of $X = \{x_{1,1}, x_{1,2}, \dots, x_{1,m}\}$ indicating the corresponding test cases. Equation 10 shows the total populations with Mpopi denoting the population in ith iteration [27].

$$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,m} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,m} \end{bmatrix} \quad (9)$$

$$M_{popi} = \begin{bmatrix} M_{pop1} \\ M_{pop2} \\ \vdots \\ M_{popi} \end{bmatrix}. \quad (10)$$

After the populations are initialized with respect to the number of individuals, the generated test cases are selected to be transmitted to the next generation on the basis of the value of the selection factor's value. For instance, if the value of the selection factor is set to 0.7. It implies that individuals with fitness value equal to or greater than 70% will be selected to the next generation and individuals with lesser fitness value will not be transmitted [25], so only the best individuals will be transmitted. As shown in Equation 11, each individual with 0.7 as a fitness value will be allocated in

$$X_{tselect} = \begin{cases} X_i^t & \text{if } fitness(X_i^t) \geq fitness(\text{the top } 70\% X^t) \\ \text{null} & \text{else} \end{cases}. \quad (11)$$

The crossover process is a process other than the selection applied in the genetic algorithm to find the optimal solution. This process is governed by the value of the crossover factor. For example, if the value of the crossover factor is set to 0.9, it implies that two individuals with 90% probability of performing crossover operation are randomly selected [25]. Finally, in order to diversify the search into a new area of the search space, the elements of the selected individuals are randomly mutated.

2.3.2. Coverage Oriented Fitness Function

In this paper, the criterion of transition coverage has been used to measure the efficiency of the proposed approach and is employed as the objective fitness value. As shown in Equation 12, in order to achieve individual evaluation, the sum of fitness value ($\sum_{i=1}^{N_{popj}} f_i$) is converted to the standard fitness value ($f_{i,norm}$) [7].

$$f_{i,norm} = \frac{f_i}{\sum_{i=1}^{N_{popj}} f_i}. \quad (12)$$

2.3.3. The Proposed Algorithm

The revised genetic algorithm has been defined as the following equation.

$$NGA = \{Mp, Sel, Cor, Mut, R, Cov, Np(A), T\}. \quad (13)$$

The Mp denotes the individuals of a population; the Sel represents the operation of selection; Cor denotes the operation of crossover; Mut denotes the operation of mutation; R represents the regeneration process; Cov represents the testing coverage; the number of iteration is presented as Np(A); and the termination condition is presented as T.

Figure 4

Algorithm of Revised GA

```

Input: Test Cases (Test Paths)
Output: prioritized test cases
q=0 // q is agent factor
first population p= test cases
while not last test case () do
  while not termination() do
    If not aging
      for i =1 to popSize do
        selection(p)
        offspring ← cross(p)
        offspring ← Mutation(p, offspring)
        fitness (offspring) // using  $f_{i,norm} = \frac{f_i}{\sum_{i=1}^{N_{popj}} f_i}$ 
      end for
    else
      p = regenerate () // new population
    end if
    q = calculate aging factor
  end while
  remove the best test case
end while

```

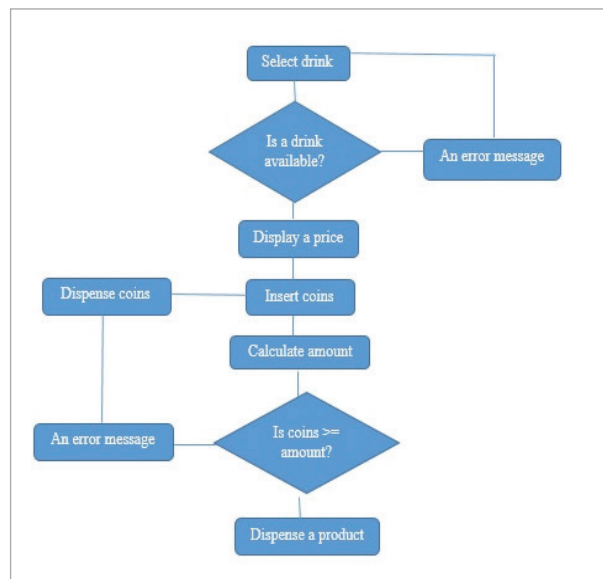
As previously mentioned, when population ageing is detected, the regeneration process is triggered to randomly generate a new population. In this case, the past population is relocated with the new one. However, the

selection is based on crossover and mutation process which are used in the new population only if the new population enhances the transition coverage. If the new population cannot contribute towards the improvement in the transition coverage, the population ageing will be triggered resulting in the elimination of the population, and the generation of a new population until the ageing condition remains inconclusive. Figure 4 shows the new algorithm based genetic algorithm in which the sub-process of the ageing factor calculation and population regeneration process is involved. In order to prioritize the individuals (regression test cases), once the optimal regression test case is obtained, it is removed from the input list and saved in the prioritized regression test cases list. Consequently, the revised algorithm is re-executed on the remaining regression test cases.

3. Results Analysis

In order to introduce and validate the results from the proposed approach, an experimental tool has been constructed. This tool involves hardware layer, operating system layer and application layer (tool). The hardware layer and operating system layer have been implemented by using Windows 8.1 Pro on a PC with i7 2.20 GHz CPU and 4 GB RAM, and the application layer includes the proposed tool which implements the proposed approach by using Java programming language.

Figure 5a
The activity diagram of VMS



3.1. Experimental Results

The activity diagrams of the vending machine and ATM machine have been applied previously in many software engineering research works [9, 35, 32]. Therefore, they have also been used in this research to illustrate the processes of the proposed approach to generate and prioritize regression test cases. As shown in Figure 5a, in the Vending Machine System (VMS) a user (customer) can select a type of drink, then the machine validates the selection and checks for the availability of a product. If a product is not available, the machine displays a message and returns back the selection menu. Else the machine displays a product price and then asks the user to insert the coins. Consequently, the machine calculates the

Figure 5b
The activity diagram of ATM

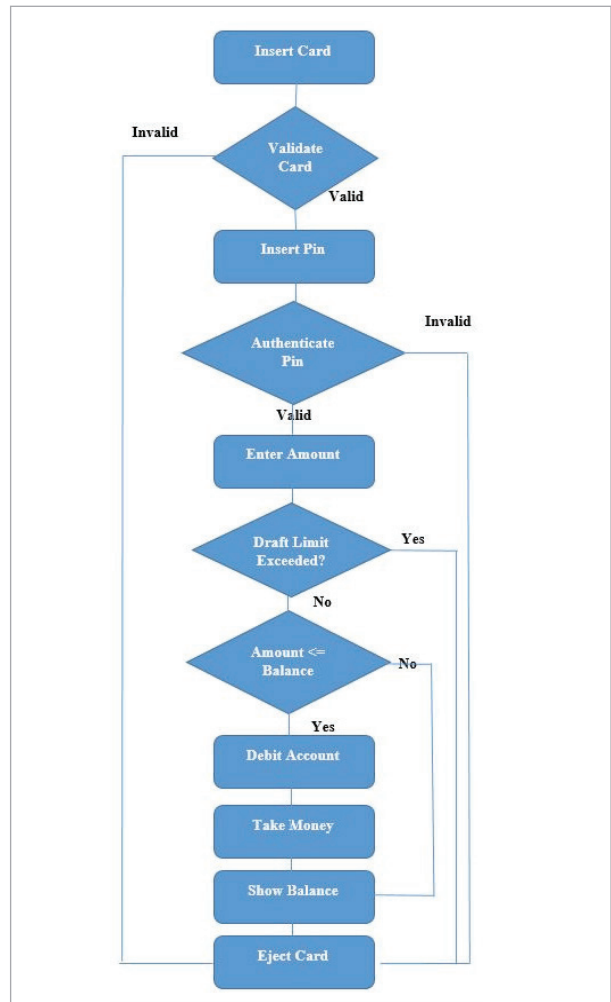


Figure 6b
Control flow diagram of ATM

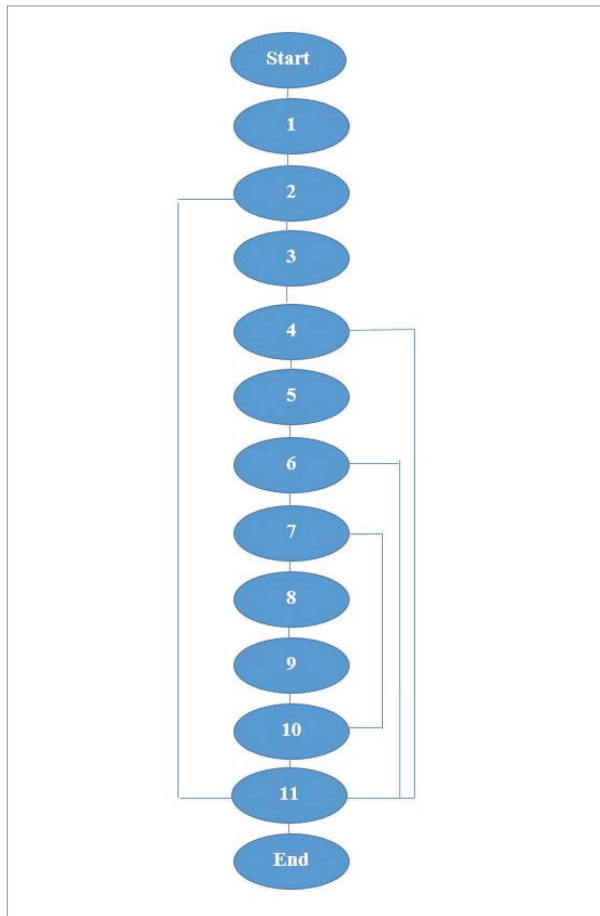
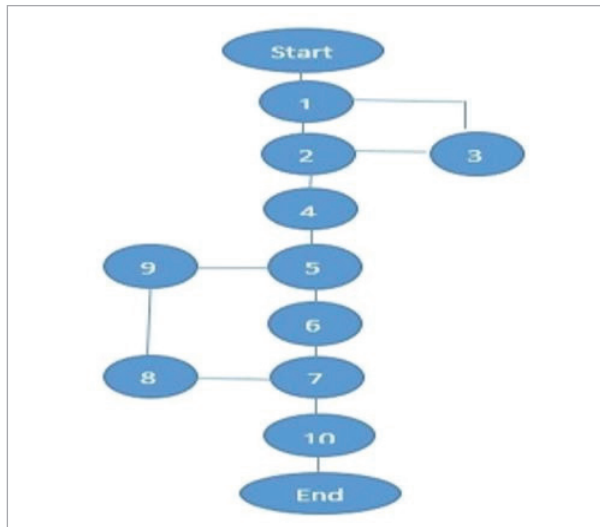


Figure 6a
Control flow diagram of VMS



deposited amount. If the deposited amount is insufficient, the machine displays an error message and dispense back the deposited coins or else, the vending machine dispenses a product and returns back to the main menu. Figure 5a shows the activity diagram of vending machine software, where Figure 5b shows the activity diagram of withdrawal function in the ATM system.

As explained in phase one of the proposed approach, the control flow graph has been automatically generated from the activity diagram. Since each activity in the activity diagram has been represented as a node in the control flow graph and each interaction in the activity diagram has been presented by an edge in the control flow graph, it is worth noting that each decision in the activity diagram has been traversed in two paths in the control flow graph representing the true and false answers. Figures 6a-6b present the control flow graphs generated from the activity diagrams of the vending machine system and ATM system, respectively.

Consequently, the proposed approach provides an algorithm (see the algorithm in Figure 3) to produce all paths in the control flow graph recursively. By applying this algorithm on the control flow graph generated from the activity diagram of the vending machine and ATM, four paths (regression test cases) have been produced from the CFG of VMS and five paths from the CFG of ATM see Figures 7a-7b.

Figure 7a
Regression test cases of VMS

- 1->2->4->5->6->7->10
- 1->2->4->5->6->7->8->9->5->6->7->10
- 1->2->3->1->2->4->5->6->7->10
- 1->2->3->1->2->4->5->6->7->8->9->5->6->7->10

Figure 7b
Regression test cases of ATM

- 1->2->3->4->6->11
- 1->2->3->4->5->6->7->8->9->10->11
- 1->2->3->4->11
- 1->2->3->4->6->7->10->11
- 1->2->11

Finally, the Revised Genetic Algorithm has been applied to prioritize the generated test paths (regression test cases). Figure 8 presents the prioritized regression test cases for both systems produced by the revised genetic algorithm.

Figure 8a

Properitized regression test cases of VMS

1->2->3->4->6->11
1->2->3->4->5->6->7->8->9->10->11
1->2->3->4->11
1->2->3->4->6->7->10->11
1->2->11

Figure 8b

Properitized regression test cases of ATM

1->2->3->4->5->6->7->8->9->10->11
1->2->3->4->6->7->10->11
1->2->3->4->6->11
1->2->3->4->11
1->2->11

3.2. Validation of Results

The prioritized test cases resulted from the proposed approach are evaluated using the average percentage transition coverage (APTC) metric, which quantifies the degree at which prioritized test cases cover the conditions. Equation 14 is used to calculate the APTC.

$$APTC = 1 - \frac{\sum_{i=1}^m TC_m}{mn} + \frac{1}{2n}, \quad (14)$$

wherein T denotes the test suite under evaluation; n represents test cases; m indicates the number of transitions in the control flow graph and TC_i shows the position of the first test case in the test suite T that covers ith transition.

Table 1 includes APTC values and execution time to each prioritization approach, where the proposed

Table 1

APTC values and execution F for the prioritization approaches

Prioritization approach	Values of APTC (%)	Execution time (millisecond)
Optimum order	55%	53
Random order	42%	45
GA	96.4%	53
BCA	94.7%	30
Proposed approach	100%	10

prioritization technique has been compared with the other four prioritization techniques. Here, the first two approaches are considered as a control group where other conditions are identical to other groups.

Optimum order: in this prioritization technique the rate of fault detection is used to prioritize the test cases.

Random order: in this prioritization technique the test cases are prioritized randomly.

GA: in the original genetic algorithm the mutation operators are employed to prioritize the test cases.

Bee Colony Algorithm (BCA): in this technique, the test cases are prioritized to enhance the execution time and coverage [21].

The statistical results in the table show that the proposed approach provides much transition coverage with significantly less execution time in comparison with the other techniques. It is worth mentioning that, the proposed approach is better as it covers the modified transitions in the control flow graph so the modified transitions in the activity diagram of a use case. Hence the proposed approach is better and offers significant help in specific test cases prioritization and providing faults earlier.

From the previous experimental results and analysis of different aspects, the Revised Genetic Algorithm clearly provides a more useful and efficient prioritization approach on average percentage transition coverage (APTC) and execution time in comparison to other prioritization techniques as it achieves more excellent test coverage with minimal regression test cases. As shown in table 1, the original genetic algorithm offers promising results in terms of performance and coverage (53 milliseconds and 96.4%,

respectively). Additionally, the BCA provides also hopeful results in term of execution time and coverage (30 milliseconds and 94.7%, respectively). The optimum order technique achieves only 55% coverage with 53 milliseconds of execution time and the random order technique achieves 42% coverage with 45 milliseconds of execution time, where the run time for each technique integrates the entire process of automatic generating test cases and prioritizing such test cases. However, the proposed technique offers better results in term the execution time and coverage than prioritization techniques. Furthermore, comparing to the Regeneration Genetic Algorithm (RGA) by Yang et al. [38], the proposed technique provides more test coverage rate and less execution time. Interestingly, the coverage rate and the execution time for RGA are 95% and 24 milliseconds, but for the proposed technique 100% and 10 milliseconds.

The possible reason for this enhancement is due to the other techniques apply mutation operators and their functions for exploring the whole search space and sometimes it is hard for the other techniques to find local optimal solutions that exist in a local search space as compared with the proposed technique. On the contrary, as compared with the other techniques, the proposed technique employ the crossover operators to obtain the local optimal solutions, which may be the reason for the better performance of the proposed technique compared with the other techniques.

4. Conclusion

The test cases prioritization is an essential task to reduce the time and effort required in the test regression. In order to obtain maximum transition coverage, this research work has proposed an approach-based revised GA to generate and prioritize the test cases generated from software specifications. five techniques of prioritization of test cases have been empirically stud-

ied and their performances have been compared. The performance of the proposed approach provides promising outcomes on both coverage and time criteria.

The proposed approach takes advantage of ability to generate various test cases from software specifications (activity diagram) and to prioritize such test cases on the basis of revised genetic algorithm. This approach has been automated using the Java Language. The necessity and benefit of applying a new metric APTC as a fitness function in the revised genetic algorithm is also shown in this proposed approach. Finally, the results from the empirical study have been analyzed and compared with the original genetic algorithm and with other techniques based on APTC [25, 38]. It was then found that the proposed approach is better and more efficient in maximizing the coverage with less execution time and it avoids the problem of population ageing that resulted from the application of the genetic algorithm by trigger the population regeneration method when the population ageing detected. Thus, these results provide a good answer for the research question which was formulated as: how does the proposed approach compare in term of time, effort and coverage to the other search-based approaches. Moreover, the experimental results from this research work confirmed also the prior results stated in the software testing literature regarding the good performance of the genetic algorithm [38]. However, the results indicate to some interesting characteristics of the proposed approach including minimizing the execution time and maximizing the transitions coverage.

The results of this study are promising. Thus, further research work is required to support such findings. Additionally, other specifications models such as sequence diagrams and use case description model should be studied and compared with the results from this work. Considering the average of percentage of the coverage, it could be extended to enhance regression test cases selection and prioritization.

References

1. Ahmad, T., Iqbal, J., Ashraf, A., Truscan, D., Porres, I. Model-Based Testing Using UML Activity Diagrams: A Systematic Mapping Study. *Computer Science Review*, 2019, 33(1), 98-112. <https://doi.org/10.1016/j.cosrev.2019.07.001>
2. Alrawashed, T. A., Almomani, A., Althunibat, A., Tamimi, A. An Automated Approach to Generate Test Cases From Use Case Description Model. *Computer Modeling in Engineering and Sciences*, 2019, 119(3), 409-425. <https://doi.org/10.32604/cmescs.2019.04681>

3. Arora, P. K., Bhatia, R. Agent-Based Regression Test Case Generation Using the Class Diagram, Use Cases and Activity Diagram. *Procedia Computer Science*, 2018, 125(1), 747-753. <https://doi.org/10.1016/j.procs.2017.12.096>
4. Barisas, D., Bareiša, E. A Software Testing Approach Based on Behavioral UML Models. *Information Technology and Control*, 2009, 38(2), 119-124. <http://doi.org/10.5755/J01.ITC.38.2.12094>
5. Biswas, S., Mall, R., Satpathy, M., Sukumaran, S. Regression Test Selection Techniques: A Survey. *Informatica*, 2011, 35(3), 289-321. <https://doi.org/10.1109/ICSTW.2011.28>
6. Booch, G., Rumbaugh, J., Jacobson, I. *The Unified Modeling Language User Guide*. Addison-Wesley, Boston, 1999.
7. Chen, T. Y., Zhou, Z. Q. Adaptive Random Testing Through Iterative Partitioning. *Proceedings of International Conference on Reliable Software Technologies*, Berlin, Heidelberg, June 18-22, 2006, 155-166. https://doi.org/10.1007/11767077_13
8. Dai, Y. S., Xie, M., Poh, K. L., Yang, B. Optimal Testing-Resource Allocation with Genetic Algorithm for Modular Software Systems. *Journal of Systems and Software*, 2003, 66(1), 47-55. [https://doi.org/10.1016/S0164-1212\(02\)00062-6](https://doi.org/10.1016/S0164-1212(02)00062-6)
9. Dalal, S., Hooda, S. Automated Test Sequence Generation of Aspect-Oriented Programs Based upon UML Activity Diagram. *International Journal of Engineering and Technology*, 2017, 9(2), 1469-1474. <https://doi.org/10.21817/ijet/2017/v9i2/170902286>
10. De Vito, G., Ferrucci, F., Gravino, C. Design and Automation of a COSMIC Measurement Procedure Based on UML Models. *Software and Systems Modeling*, 2020, 19(1), 171-198. <https://doi.org/10.1007/s10270-019-00731-2>
11. Dey, A., Pal, A., Long, H. V. Fuzzy Minimum Spanning Tree with Interval Type 2 Fuzzy Arc Length: Formulation and a New Genetic Algorithm. *Soft Computing*, 2020, 24(6), 3963-3974. <https://doi.org/10.1007/s00500-019-04166-1>
12. Ekelund, E. D., Engström, E. Efficient Regression Testing Based on Test History: An Industrial Evaluation. In *2015 IEEE International Conference on Software Maintenance and Evolution, (ICSME 2015)*, Bremen, Germany, Sep 29- Oct 1, 2015 , 449-457. <https://doi.org/10.1109/ICSM.2015.7332496>
13. Engström, E., Runeson, P. A Qualitative Survey of Regression Testing Practices. *Proceedings of International Conference on Product Focused Software Process Improvement*, Limerick, Ireland, June 21-23, 2010, 3-16. https://doi.org/10.1007/978-3-642-13792-1_3
14. Girgis, M. R. Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm. *Journal of Universal Computer Science*, 2005, 11(6), 898-915. <https://doi.org/10.3217/jucs-011-06-0898>
15. Guo, K. Research on Location Selection Model of Distribution Network with Constrained Line Constraints Based on Genetic Algorithm. *Neural Computing and Applications*, 2020, 32(6), 1679-1689. <https://doi.org/10.1007/s00521-019-04257-y>
16. Harikarthik, S. K., Palanisamy, V., Ramanathan, P. Optimal Test Suite Selection in Regression Testing with Testcase Prioritization Using Modified Ann and Whale Optimization Algorithm. *Cluster Computing*, 2019, 22(5), 11425-11434. <https://doi.org/10.1007/s10586-017-1401-7>
17. Kavitha, R., Sureshkumar, N. Test Case Prioritization for Regression Testing Based on Severity of Fault. *International Journal on Computer Science and Engineering*, 2010, 2(5), 1462-1466. <https://doi.org/10.24297/ijct.v9i3.6814>
18. Keshanchi, B., Souri, A., Navimipour, N. J. An Improved Genetic Algorithm for Task Scheduling in the Cloud Environments Using the Priority Queues: Formal Verification, Simulation, and Statistical Testing. *Journal of Systems and Software*, 2017, 124(1), 1-21. <https://doi.org/10.1016/j.jss.2016.07.006>
19. Khan, R., Amjad, M., Srivastava, A. K. Generation of Automatic Test Cases with Mutation Analysis and Hybrid Genetic Algorithm. In *2017 3rd International Conference on Computational Intelligence and Communication Technology (CICT 2017)*, Ghaziabad, India, Feb 9-10, 2017, 1-4. <https://doi.org/10.1109/CICT.2017.7977265>
20. Khurana, N., Chillar, R. S. Test Case Generation and Optimization Using UML Models and Genetic Algorithm. *Procedia Computer Science*, 2015, 57(1), 996-1004. <https://doi.org/10.1016/j.procs.2015.07.502>
21. 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 2015)*, Hua Hin, Thailand, June 24-26, 2015, 1-6. <https://doi.org/10.1109/ECTICon.2015.7207103>

22. Kwon, N., Song, K., Ahn, Y., Park, M., Jang, Y. Maintenance Cost Prediction for Aging Residential Buildings Based on Case-Based Reasoning and Genetic Algorithm. *Journal of Building Engineering*, 2020, 28(1), 101006. <https://doi.org/10.1016/j.jobe.2019.101006>
23. Li, Z., Harman, M., Hierons, R. M. Search Algorithms for Regression Test Case Prioritization. *IEEE Transactions on Software Engineering*, 2007, 33(4), 225-237. <https://doi.org/10.1109/TSE.2007.38>
24. Lin, P., Bao, X., Shu, Z., Wang, X., Liu, J. Test Case Generation Based on Adaptive Genetic Algorithm. In 2012 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering, (ICQR2MSE 2012), Chengdu, China, June 15-18, 2012, 863-866. <https://doi.org/10.1109/ICQR2MSE.2012.6246363>
25. Mala, D. J., Mohan, V., Kamalpriya, M. Automated Software Test Optimisation Framework-An Artificial Bee Colony Optimisation-Based Approach. *IET Software*, 2010, 4(5), 334-348. <https://doi.org/10.1049/iet-sen.2009.0079>
26. Mani, P., Prasanna, M. Validation of Automated Test Cases with Specification Path. *Journal of Statistics and Management Systems*, 2017, 20(4), 535-542. <https://doi.org/10.1080/09720510.2017.1395173>
27. Nguyen, C. D., Miles, S., Perini, A., Tonella, P., Harman, M., Luck, M. Evolutionary Testing of Autonomous Software Agents. *Autonomous Agents and Multi-Agent Systems*, 2012, 25(2), 260-283. <https://doi.org/10.1007/s10458-011-9175-4>
28. Panda, N., Acharya, A. A., Mohapatra, D. P. Test Scenario Prioritization for Object-Oriented Systems Using UML Diagram. *International Journal of System Assurance Engineering and Management*, 2019, 10(3), 316-325. <https://doi.org/10.1007/s13198-019-00759-z>
29. Pressman, R. S. *Software Engineering: A Practitioner's Approach*. Palgrave macmillan, London, 2005.
30. Sahoo, R. K., Nanda, S. K., Mohapatra, D. P., Patra, M. R. Model Driven Test Case Optimization of UML Combinational Diagrams Using Hybrid Bee Colony Algorithm. *International Journal of Intelligent Systems and Applications*, 2017, 9(6), 43-54. <https://doi.org/10.5815/ijisa.2017.06.05>
31. Sapna, P. G., Balakrishnan, A. An Approach for Generating Minimal Test Cases for Regression Testing. *Procedia Computer Science*, 2015, 47(1), 188-196. <https://doi.org/10.1016/j.procs.2015.03.197>
32. Sapna, P. G., Mohanty, H. Prioritization of Scenarios Based on UML Activity Diagrams. In 2009 First International Conference on Computational Intelligence, Communication Systems and Networks, (CICSYN 2009), Indore, India, Jul 23-25, 2009, 271-276. <https://doi.org/10.1109/CICSYN.2009.74>
33. Solin, A., Särkkä, S. Hilbert Space Methods for Reduced-Rank Gaussian Process Regression. *Statistics and Computing*, 2020, 30(2), 419-446. <https://doi.org/10.1007/s11222-019-09886-w>
34. Sur, P., Chen, Y., Candès, E. J. The Likelihood Ratio Test in High-Dimensional Logistic Regression is Asymptotically a Rescaled Chi-Square. *Probability Theory and Related Fields*, 2019, 175(1), 487-558. <https://doi.org/10.1007/s00440-018-00896-9>
35. Swain, R. K., Panthi, V., Behera, P. K. Generation of Test Cases Using Activity Diagram. *International journal of computer science and informatics*, 2013, 4(1), 35-44. <https://doi.org/10.47893/IJCSI.2014.1171>
36. Vinitha, K., Preetha, S. A. Multi Objective RGA for Increasing the Coverage and Reducing the Loop Statements. *International Journal of Research and Analytical Reviews (IJRAR)*, 2018, 5(4), 269-276. <https://doi.org/10.6084/m9.doi.one.IJRAR1904027>
37. Vitiutinas, R., Silingas, D., Telksnys, L. Model-Driven Plug-in Development for UML Based Modeling Systems. *Information Technology and Control*, 2011, 40(3), 191-201. <https://doi.org/10.5755/j01.itc.40.3.627>
38. Yang, S., Man, T., Xu, J., Zeng, F., Li, K. RGA: A Lightweight and Effective Regeneration Genetic Algorithm for Coverage-Oriented Software Test Data Generation. *Information and Software Technology*, 2016, 76(1), 19-30. <https://doi.org/10.1016/j.infsof.2016.04.013>
39. Yoo, S., Harman, M. Regression Testing Minimization, Selection and Prioritization: A Survey. *Software Testing, Verification and Reliability*, 2012, 22(2), 67-120. <https://doi.org/10.1002/stvr.430>

