JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 11(2), JUNE, 2023
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.net

Quality-Aware Genetic Algorithm Based Cost Cognizant Test Case Prioritization for Object-Oriented Programs

Hassan Abubakar[1], Fatima Umar Zambuk[2], Usman Mahmud Ahmed[2], Abdulsalam Ya'u Gital[1]
[1]*Faculty of Science,*
Department of Mathematical Sciences,
Abubakar Tafawa Balewa University, Bauchi, Nigeria
[2]*Faculty of Management Science,*
Department of Management & Information Technology,
Abubakar Tafawa Balewa University, Bauchi, Nigeria

Abstract
Regression testing is essential for validating changes made to software. A new testing technique known as Quality-aware GA-based cost cognizant test case prioritization (QAG-TCP) is presented to improve the prioritization of test cases for object-oriented application regression testing. Actual fault severities, actual test cost, and the latest result of the number of faults per test case detected during regression testing will be collected to ensure the use of a scenario closer to reality than the use of mutants. The technique also suggests apart from the use of test case costs, fault severity in fitness function of some previous work, coupling, and cohesion ratings will be incorporated on a GA to improve and ensure effectiveness and efficiency in regression test case prioritization of object oriented programs.

ARTICLE INFO
Article History
Received: March, 2023
Received in revised form: May, 2023
Accepted: May, 2023
Published online: June, 2023

KEYWORDS
Regression testing, Quality-aware, Prioritization, test cases, coupling, and Cohesion

INTRODUCTION

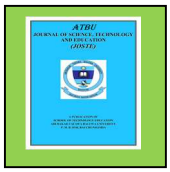
Regression testing is essential for validating changes made to software as it evolves through a sequence of modifications. Regression testing becomes convenient if we can pinpoint the program components that will probably be impacted by the maintenance activity-related changes to the programs. The new testing technique known as Quality-aware GA-based cost cognizant test case prioritization (QAG-TCP) is presented in this section. The technique was explicitly created to improve the prioritization of test cases for object-oriented application regression testing. This seminar achieves this research's primary goal, i.e., to develop a Quality-aware GA-based test case prioritization for object-oriented programs that consider coupling and cohesion. Actual fault severities, actual test cost, and the latest result of the number of faults per test case detected during regression testing will be collected to ensure the use of a scenario closer to reality rather than mutants. The technique also suggests a fitness function that uses test case costs, fault severity, coupling, and cohesion ratings on a GA to improve and ensure effectiveness and efficiency in regression test case prioritization. This constituted the first and second objectives of this research work

RELATED WORKS

Ahmed et al., (2022) presents a value-based cost-cognizant test case prioritization (TCP) technique for regression testing. The technique takes into account the severity of faults and the cost of test cases in the prioritization process. The technique is evaluated using a case study of a real-world software system. The results of the case study show that the technique is able to significantly improve the effectiveness of regression testing.

Raamesh et al., (2022) reported a brand-new method for prioritization and test cases selection called HBR2O. The Battle Royale Optimization (BRO) and Remora Optimization

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi



(RO) algorithms are combined to create HBR2O. A test case selection and prioritizing technique that is both effective and efficient is produced by combining these two methods. The findings demonstrated that HBR2O outperformed other state-of-the-art test case selection and prioritizing algorithms in finding high-quality test cases while using less time and memory. A new technique for prioritization and test cases selection is the HBR2O algorithm. The researchers believe that HBR2O has the potential to completely change how software is tested for the better.

Yadav, and Dutta, (2020) in their research indicates that, it might be challenging to pinpoint the set of test cases that need to be re-executed since object-oriented software is often more complex than procedural software. Their suggested technique is founded on the utilization of a dependency graph, which shows the connections between the various components of a software system. It was tested against several object-oriented software systems, and it passed. The proposed technique is a cost-effective strategy to decide which test cases to prioritize for regression testing of object-oriented software.

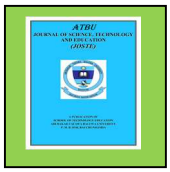
Bajaj et al., (2019) reported the usage of the genetic algorithm in test case prioritization is examined and categorized in this research. It chooses the 20 studies, out of 384, that are the most pertinent, and compares them on the basis of their research methodologies, prioritization techniques, dataset specifications, and test suite sizes, types of genetic algorithms employed, performance metrics, and validation standards. This analysis is thorough. According to the assessment, genetic algorithms offer a lot of potential for tackling test case prioritizing issues, and there is still room for advancement in this field. The recommendations made in the review can be used by future scholars to close the research gaps. A genetic algorithm-based test case prioritizing method for regression testing can be one of the efficient ones when used in the right way.

Musa, (2021) proposed a mutated regression test case prioritization technique that takes selected test cases and orders the test case using GA (named mutGA). This technique applies mutation operation on test cases to produce mutants; it reduces the severity of a fault if the previous test cases have executed it. This technique can be used for regression testing of OOP. The proposed technique (mutGA) used existing tools (such as ESDG) in some of its execution phases, while some other steps were manually executed using Java. The illustrated example shows that the proposed approach achieved better without the typical crossover operation in most GA-based regression test case prioritization techniques. The evaluation of the proposed technique indicates that the technique outperforms the values of APFD for the techniques used for comparison, i.e., retest-all and randomly ordered test cases techniques. The proposed technique (mutGA) APFD value of 70.8% shows a higher percentage of faults detection than the APFD values of retest-all and randomly ordered test cases techniques (56.2% and 65.2%, respectively).

Id et al., (2021) indicates the recent confidence developed by researchers on Machine Learning (ML) techniques towards improving the effectiveness of Test case Selection and Prioritization (ML-based TSP). A relatively accurate prediction model is achieved using test cases information from both partial and imperfect sources with the help of this technique. Ranking models, NLP-based, RL, and clustering are the most commonly used ML techniques in TSP. Combining two of these techniques, for example, Ranking models and NLP-based were reported severally by researchers. Incorporating the two techniques improve test case prioritization performance; even though the study reports a lack of standard evaluation procedures and appropriate publicly, it hindered ML-based TSP performance assessment. They directly affect the current open problems and the current state-of-the-art technique. Researchers such as (Ramírez et al., 2022) have put effort into addressing this general issue.

In another study Samad et al., (2021), the researchers proposed a multi-objective approach using swarm optimization (MOPSO). The MOPSO algorithm uses code Coverage, execution time, and fault detection to prioritize test

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi



cases. The researchers report an outstanding performance in terms of less cost, extreme fault detection, and highest coverage in comparisons with reverse ordering, random ordering, and no ordering.

Mishra et al., (2019), the researchers implement a multi-objective genetic algorithm in test case optimization and prioritization. Statement coverage, requirement priority, execution time, and risk exposures were the factors associated with test cases for a specific SUT at a stipulated time constraint. The proposed technique uses a multi-criteria based GA to prioritize and optimize test cases by considering a specified time to execute test cases.

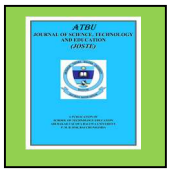
Paterson (2019) presented an approach based on defect prediction; it analyses code features to predict the likelihood of any given Java class having a bug in it. The researcher further illustrates that if defect prediction can correctly predict likely buggy classes, a tool can prioritize tests to quickly detect defects in that class. An empirical evaluation was conducted with six real-world Java programs containing 395 real faults. It was found that an average reduction of 9.48% of the number of test cases was achieved to find a fault compared to the existing coverage-based strategies and 10.5% in comparison with existing history-based strategies

Although Paterson et al., (2019) argue that mutants faults inflate results, this work presents a technique with some level of improvement in terms of fault detection in the context of OOP. Still, the researchers indicate that this technique is meant for Java users; it can only be extended for other OOP users such as C# and C++. This clearly shows the limitation of this technique to static OOP programs while the Dynamic OOP programs users cannot use this technique for Regression testing purposes. Furthermore, the focus of the researchers is on the effectiveness of the technique in terms of fault detection; the severity and cost of the faults are not considered.

Another development Huang et al., (2020) proposed a new approach that takes the concepts of code coverage and combination coverage and applied the duo on Regression Test Case Prioritization named code combinations coverage based prioritization (CCCP). Empirical studies conducted to compare testing efficiency and effectiveness of the proposed technique with four popular Regression Test Case Prioritization techniques, namely total, additional, adaptive random, and search-based test prioritization, indicate better performance. Furthermore, the proposed CCCP technique considers dynamic coverage and test cases as inputs for a fair comparison. Fourteen versions of four Java programs were used and the PIT mutation tool. The Fault Tracer tool was used to collect the coverage information for each program version. The result shows the difference in mean and median APFD values between the selected technique and the proposed technique to be less than 5%. It further indicates that the proposed technique (CCCP) with a higher combination strength (sufficient testing resources) can be more effective than all other prioritization techniques on APFD and APFDc. The researchers take a multi-objective approach in their work, the work indicated a relatively good improvement, but this is highly dependent on combination strength which might be relative.

Bello et al., (2019) proposed a multi-criteria evolutionary regression test Prioritization for dynamic object-oriented programs. The technique implements a genetic algorithm with a single objective multi-criteria fitness function to find the best possible ordering of the test suite to a very round of repetition and then produce the prioritized test suite as the solution. Considering more than one parameter (criterion) is an additional future of this technique. These parameters are then assigned weight and later summed up to compute the overall weight considered as the fitness function value of every test case in the test suite. The fitness function values are then used to prioritize the Test Cases. Although the technique has not been implemented, talkless of it has been evaluated (Bello et al., 2019). The research direction is not cost-cognizant because the performance matrix they plan to use is APFD, which considers the cost and severity to be uniform.

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi



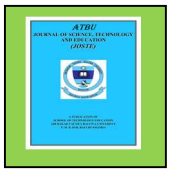
The experiments of Paterson et al., (2018) describes that the type of fault (real or mutant) in a program has a meaningful influence on the effectiveness of a prioritized test suite. This research shows that for an individual fault, an average of 659 extra tests need to be executed to detect a real fault compared to a mutant fault. The experiments further reveal that the effectiveness of prioritization is affected by the number of faults in a program. Additionally, prioritization has a similar effect for single and multiple faults on real faults, contrary to mutant faults. The higher the number of mutant faults, the better the effectiveness of its prioritization. The collection of the results of this research demonstrates that mutant faults are not a substitute for real faults; this shows the necessity for future evaluations of test prioritization to use real and mutant faults. The average percentage of faults detected (APFD) metric was used during the research evaluation. The evaluation results of this experiment show that APFD scores get inflated if prioritization is executed using a single mutant compared to using a single real fault. If the inflation is known before the prioritization run, mutant faults could have been a good substitute for real faults. But the evaluation results indicate that the inflation is unpredictable; moreover, the APFD scores depend on the test case prioritizer and the project. The researchers finally concluded that real faults are difficult for prioritized test suites to detect, and mutant faults lead to APFD scores getting inflated unpredictably. Even though the researcher represented substantive evidence on the need for evaluation using real fault, the research community has to depend on the mutant fault because of lack of respiratory for real faults.

Bello et al., (2018) proposed an evolutionary regression test cases prioritization for object-oriented programs based on the dependency among faults. The severities of dependent fault are used to assign a value to a test case while other information about the test cases is retrieved from a repository. This approach will pass the complete details on the test cases as input into the ECRTP algorithm. The result will be a scheduled test case order that can detect severe faults earlier during regression testing, and the best-ordered existing test suite will be saved for further reuse in a test case repository. A holistic analysis was conducted to compare this approach with three existing approaches using 8 Java programs to ascertain the most effective cost-cognizant approach. Test effort efficiency, fault detection effectiveness, and APFDc measurements were used as the basis of the experimental data, statistical analysis, evaluation, and comparisons of the proposed approach. All the results clearly show that the ECRTP approach outperforms the other approaches, hence proving the worthiness of ECRTP in Regression testing. However, the researcher indicates the need to extend this work using the latest regression testing information. Also, to consider other object-oriented metrics such as Coupling and Cohesion for better evaluation and incorporate multi-objective evolutionary processes to increase the effectiveness of this technique.

Panigrahi & Mall, (2014) assumes to come up with a model (i.e., a test case that executes statements previously has a greater chance of detecting a fault in the executed statement than the test case that executes it later). Using this model, they proposed a heuristic-based regression test case prioritization technique based on the analysis of the dependence model for object-oriented programs. They use an approach that involves the construction of an intermediate dependency model for a program from the program source code. The model will get updated to accommodate the changes made if the program is updated. The nodes affected by the change are determined by constructing the forward slices' union that correlates with each changed element. The selection criteria for regression tests are test cases that cover one or more affected nodes. The weight of a test case is computed by assigning weight to the affected nodes. The selected test cases are prioritized by computing the assigned weight value. The technique has been experimented on seven java programs. The experiment result shows that the proposed technique outperformed the moderated techniques with an average increase in the APFD metric value by 9.01 %. Although this research

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 11(2), JUNE, 2023
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.net



indicates an increase in fault detection effectiveness, it is not cost-cognizant.

METHOD AND MATERIALS

Genetic Algorithms

Darwin's Theory of Evolution analysis forms the foundation of Genetic Algorithms; GA is used to find the fittest candidates group (Test cases). Chromosome refers to an individual candidate in the group as it is in the biological name of this original approach. The initial population of the group contains several possible individuals, or chromosomes are selected from the search domain. A better set of chromosomes is selected to evolve on every round of reiteration. After the whole evolution, some change is expected on the selected chromosome. Since after a Chromosome has been revolved, it is analyzed if it is better than some existing Chromosomes before it is added to the group or population; It can be presumed that the population's fitness should increase over time (Paterson, 2019).

Genetic algorithms overtime attracted numerous researchers and analysts in computer science. Since the late 1950s and mid-1960s, by 1962, many researchers had derived many computations for machine adaptation and capacity improvement, but little was reported in the literature (Musa et al., 2016). The main idea of a population was later introduced in more recent versions of GA (Mitchell, 1998).

Encoding

The need to convert the potential answer to a form the computer will use in GA brings about the Encoding procedure. The potential answer is encoded using an encoding function related to the problem then GA is applied to work on the problem. Mostly the encoding process produces strings of binary. But there is a similar approach of encoding that produces integers or decimal numbers (Peter et al., 2002). Individuals or chromosomes can equally be represented in strings of a letter. (Musa et al., 2016)

Fitness Value

The fitness value of a chromosome in a given population has a vital role in a genetic algorithm because it determines the level of importance; it is dependent on the problem. It determines how good a chromosome is. It is the determining factor of the Chromosome participation in the next generation of the population (Musa et al., 2016).

Selection

The selection process in the Genetic algorithm makes use of many different techniques to choose chromosomes that are needed in the next generation. Some of the techniques highlighted by (Marczyk, 2004) are scaling selection, tournament selection, rank selection, generational selection, steady-state selection, Elitist selection, fitness-proportionate selection, roulette-wheel selection, and hierarchical selection(Musa et al., 2016). Fitness-proportionate selection is widely used in Regression testing

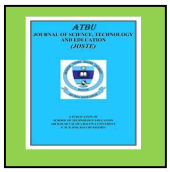
Crossover

Crossover is an integral part of the Genetic Algorithm search. It involves altering two chromosomes randomly to anticipate improving their fitness values (Marczyk, 2004). Moreover, the outcome of alteration is expected to be artificial children that contain partial solutions from both parents (Paterson, 2019). Crossover can be implemented in many ways ranging from the simple swap of the section of their code (Musa et al., 2016); others include single-point, two-point, partially matched crossover (PMX), and three-parent crossover (Paterson, 2019).

Mutation

Mutation occurs after a successful crossover operation. A genetic operator called mutation operator ensures the diversity from one generation of a chromosome population to the next is maintained. (R.C., 2010) shows that altering chromosomes in one or two of its genes by mutation from its initial state can create new gene values added to the gene pool. The added gene values can make the genetic algorithm arrive at a better solution than previously possible (Musa

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi



et al., 2016). Mutation further helps in reducing the number of undetected faults by creating many such faults as the results of changes made to the software, which are referred to as mutants. The test suite is evaluated against each of those mutants; this will show how well the test suite can handle the changes to the codes (Paterson, 2019). Flip bit, boundary, non-uniform, uniform, and Gaussian are among many mutation operators in use (Musa et al., 2016).

PROGRAM REPRESENTATION

The need for codes to be represented so that the codes can be traced easily brings about graphs. It has made traceability easy, an added advantage in software testing. Different graphs are in use, such as Program Dependency graph, System Dependency graph, Control Flow Graph, Data Flow Graph, Call Graph, and more, depending on the type of testing and approach. (Musa et al., 2016).

Control Flow Graph (CFG)

Control flow graph (CFG) is used for the intermediate representation of programs that uses data flow analysis and a lot of code optimization transformations such as copy propagation, loop invariant code motion, and common sub-expression elimination (S. Panda, 2016). The CFG represents the initial statement to the end statement as nodes while a flow of control from one node to another node as edges of a program on the graph (Musa et al., 2016).

Program Dependence Graph (PDG)

The PDG is used to represent data dependencies and control dependencies explicitly. Data dependencies represent data flow relations, while control dependencies represent control flow relationships of the program. (Musa et al., 2016). As most graphs, PDG represent statements of a program as nodes while control dependencies and data dependencies are represented by edges (Bello, 2019; Najumudheen et al., 2009)

System Dependence Graph (SDG)

SDG is an extension of PDG that contains the PDG of each procedure in the system and the main method (Sultan et al., 2014). PDG only models a single procedure and doesn't cover inter-procedural calls. Several types of vertices and edges in SDG are not available in PDG (Bello, 2019).

EXTENDED SYSTEM DEPENDENCE GRAPH (ESDG)

The Extended System Dependence Graph (ESDG) extension of SDG represents all those dependencies resulting from object-relations such as association, inheritance, and polymorphism; this includes control and data dependencies and information about those OOP attributes mentioned above. ESDG also helps identify changes at basic statement levels during analysis at statement levels. An extension of SDG was presented to model procedural programs (Bello, 2019)

Call Graph

A call graph is a directed graph that represents calling relationships between modules in a program wherein nodes are modules or units and a directed edge from one node to another node means one module has called another module. Coupling information between modules can be portrayed in a Call graph (Kumar et al., 2015).

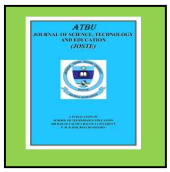
Module Dependency Matrix

Module Dependency Matrix is a quantitative measure of the dependence of modules used to find out the level of stability of the design. MDM is specifically affected by Coupling and Cohesion. Numerous values are assigned for various types of module coupling and Cohesion (Kumar et al., 2015)

PROGRAM SLICING TECHNIQUES

Program slicing is a computational task or technique used to separate program statements for different software engineering activities, including debugging, program integration, testing, maintenance, etc.

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi



(Sasirekha1 et al., 2011). (Weiser, 1981) introduces the concept of a program slice for program debugging, automatic parallelization, and program integration (Bello, 2019).

Static and Dynamic Slicing

The process of slice calculation performed without considering program input is called static slicing. The program slicing technique meant for static programs was originally proposed by (Weiser, 1981).

On the other hand, the dynamic slicing technique considers the input values in the computation of slices. Decades later (Sasirekha1 et al., 2011) proposed dynamic slicing as the need for the dynamic slicing technique grows exponentially due to exponential growth in the number of dynamic program users. Dynamic slicing was made been discussed since the '90s by (Agrawal H. & Horgan, 1990) when the idea of Dynamic Dependence Graph (DDG) based on the PDG was introduced (Musa et al., 2016).

Backward and Forward Slicing

Backward Slice simply consists of all statements, conditions, and inputs (attribute) to the program that can affect the values or slice criteria (Xu et al., 2005). While forward Slice consists of the statements that can affect the attributes value (Musa et al., 2016). The actual areas that backward slicing is applied are: program comprehension, debugging, algorithmic debugging, dead code removal, program segmentation, program analysis, program differencing, software maintenance, testing, program parallelization, module cohesion analysis, and program integration. (Bello et al. 2018). Both Backward Slice and forward slicing share some common application areas, such as dead code removal and software maintenance (Bergeretti et al., 1985). The unique area of application of forward slicing is program differencing, program comprehension, program analysis (Bello, 2019; Weiser, 1984).

Quality Indicator

The Quality Indicator phase is the new phase introduced by this research to the overall regression testing; it stands in as the third phase. This stage was the product of several components that worked together and performed different functions. The Original Program of a given java program is passed into the javac command that compiles the classes in the original program into separate class files. Then the CKJM tool accesses the class files to calculate the Object-Oriented metrics by processing the bytecode of compiled Java files. The program calculates the six Object-Oriented metrics (WMC, NOC, CBO, RFC, LCOM, Ca, and NPM) for the given classes. Then the output is stored in a text file (OCKJM.txt).

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 11(2), JUNE, 2023
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.net

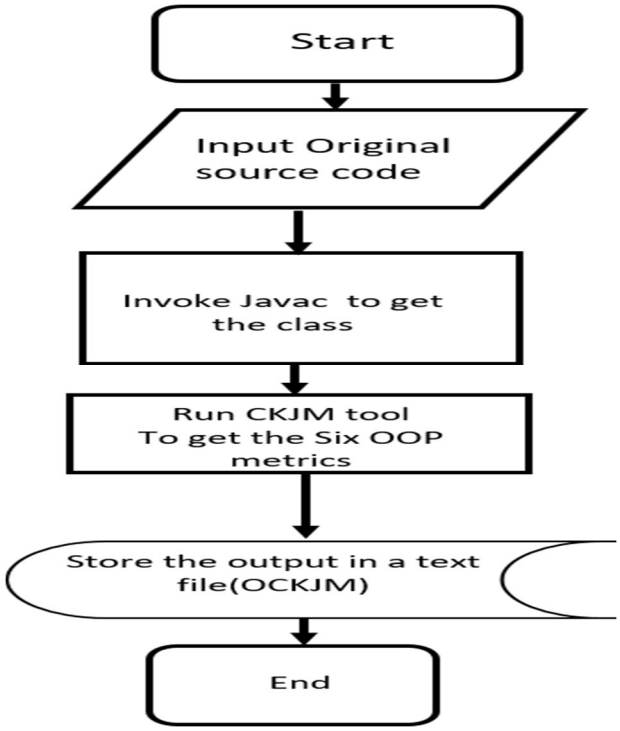


Figure 6: CKJM-IFN

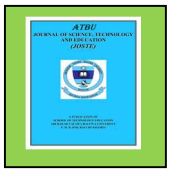
Algorithm: (CKJM-IFN)
Input: J, C_i
Output: CK
Begin getCKmetrics(J, C_i, CK)

1. $C_i \leftarrow$ NULL
2. ck $\leftarrow$ NULL
3. OOMETRIC $\leftarrow$ ARRAY[0:8] OF STRING
4. $COUNT$, $i \leftarrow 0$
5. OPENFILE “myJavaProgram.java” for READ
6. WHILE NOT EOF(“myJavaProgram.java”)
7. READFILE “myJavaProgram.java”, J
8. C_i = JAVAC(J)
9. COUNT = COUNT + 1
10. ENDWHILE
11. OPENFILE “OCKJM.txt” for WRITE
12. For each $i <$ COUNT DO
13. ck_i = ckjm(C_i)
14. oometric(i) = ck_i
15. WRITEFILE “OCKJM.txt” , oometric(i)

end getCKmetrics

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 11(2), JUNE, 2023
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.net



PROPOSED TECHNIQUE

Unlike some regression test case prioritization with three components, i.e., identification of Affected statements, selection, and Prioritization techniques (Bello, 2019; Musa et al., 2016), QAG-TCP will consist of four components, with the fourth one dealing with coupling and cohesion properties of OOP. The fourth component becomes necessary as it determines the quality of the software (Kumar et al., 2015), hence the prioritization technique that recognizes it will be more effective in ensuring all types of faults, including those related to Coupling and Cohesion are detected and handled which in return ensures quality is not lost during modification and updates.

QAG-TCP, like other TCPs, starts with the affected statements identification; this involves passing the affected statements to the Graphviz tool to produce a java dependency graph (JSDG). At the same time, passing the affected statement is forwarded to a mutation procedure to create a faulty version of the program P'. A function called JSDG updater updates JSDG with all the changes that occurred during mutation. Then the process of program slicing (forward slicing) is carried out, which aims to get the directly affected statements as the result of controls, data, or object-relational dependence. The same changed statements are used as a slicing criterion.

Unlike other TCPs, The program test suite T and the original program S are passed into the second component, which will, in turn, run a path-based integration testing to generate coverage information and store it in CovInfo. A test case selector uses affected statements and coverage information to select the test cases that execute at least one affected statement. Then selected test cases are encoded into a numerical integer, serving as the prioritization component input. The use of GA for prioritization is based on the survival of the fittest. Therefore, there is a need to compute the initial population's fitness value, which Advance Fitness Function does. The fitness computation is based on the cost of the test that discovered the faults, the severity of the discovered faults executed during the preceding regression testing, and the computed value of the software quality from the Object-Oriented Metric that indicates the highest priority based on coupling and cohesion. The fitness value determines the participation of the chromosomes in the next generation. Crossover operation comes next, which involves genes swapping between two chromosomes. A check (Ordered Crossover) is implemented to avoid non-existing integers not being introduced in the integer crossover process. Two offspring are produced, followed by Swap mutation on the offspring. The fitness value of the offspring is computed and then added to the initial population; simultaneously, the worst chromosomes are removed from the population. Termination criterion is checked; if true, the fittest chromosomes are returned as the prioritized test cases; else, the operation is repeated.

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 11(2), JUNE, 2023
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.net

FLOWCHART QAG-TCP

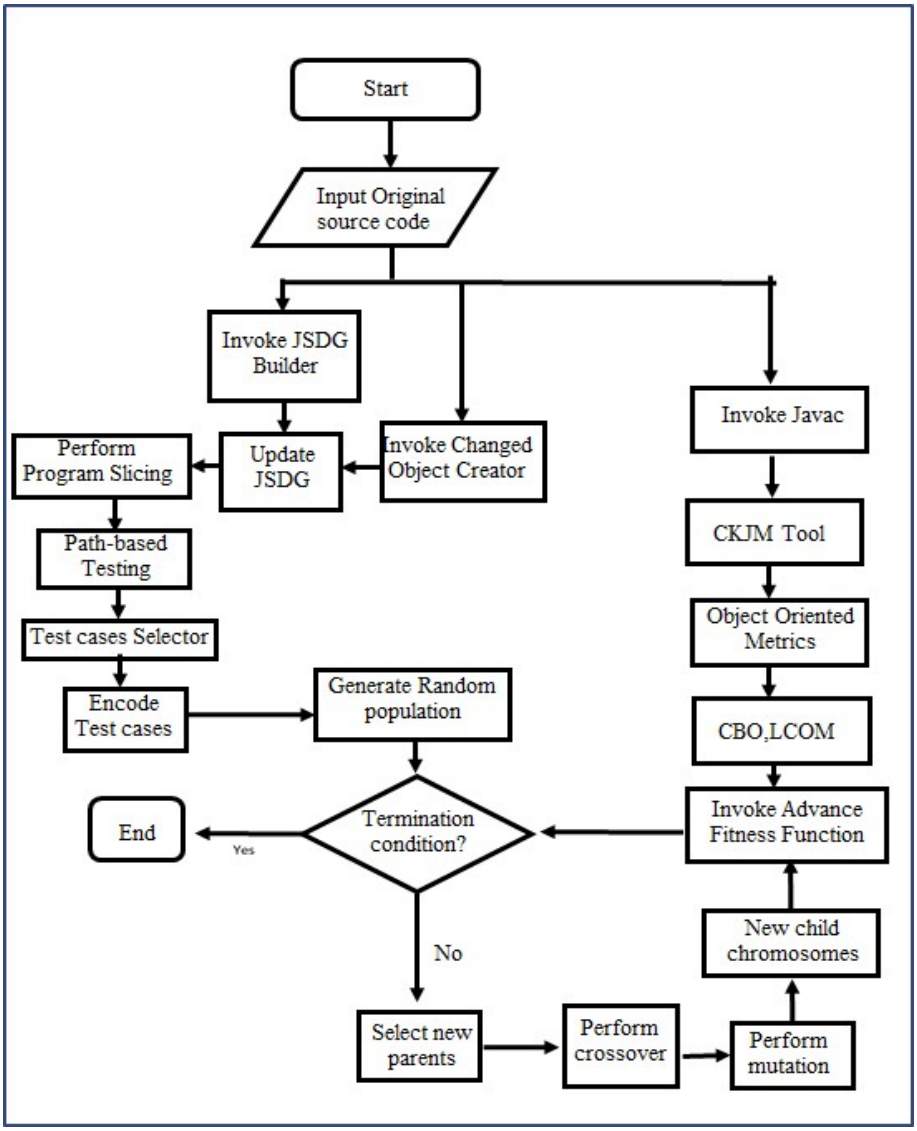
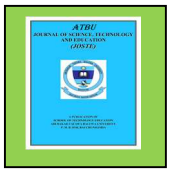


Figure 8: FLOWCHART QAG-TCP

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 11(2), JUNE, 2023
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.net



LGORITHM OF QAG-TCP

Algorithm: QAG-TCP
input: Java Source code, P , Test suite, T
output: prioritized T
declaration: *M'* :*updated Model*
T' : select test suite {tc\tc ⊂ ξT}
begin OAG-RTP
1. Construct JSDG model M of P
2. Load Test suites from a repository, *T*
3. Update the JSDG model, *M'*
4. Get all the affected statement, *AffectedS*
5. Get the test case coverage information ***covInfo***
6. **AffectedTestCasesSelect** (*M'*, T, *AffectedS, covInfo,* T')
7. **TestCaseEncode**(T')
8. *initialPopulation* = **InitialPopulationRGen**(T')
9. **computeQFitness(***initialPopulation*)
10. Select two-best chromosomes *C1* and *C2* base on fitness computation for crossover
11. WHILE NOT TERMINATION
12. Perform crossover on the two selected chromosomes
13. Perform mutation of the chromosomes formed after crossover
14. **computeQFitness (***C1,C2*)
15 Add two-best chromosomes to the population
16 ENDWHILE
17 return prioritized T

end QAG-TCP

CONCLUSION

This research presents an improved test case prioritization technique for object-oriented programs. We proposed a Quality-aware test case prioritization technique that is cost-cognizant using GA which considers coupling and cohesion among other object-oriented properties that are significant to the quality of software under test. This research involve the development of an advance fitness function for GA and employ the use of tools that capture the object-oriented properties. A lot of challenges is expected as the work on object-oriented test case prioritization is still not wide. Much work is expected in the future including the implementation of this proposed technic and possible extension of the technique to cover dynamic object-oriented paradigm

REFERENCE

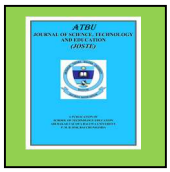
Ahmed, F. S., Majeed, A., Khan, T. A., and Bhatti, S. N. (2022). Value-based cost-cognizant test case prioritization for regression testing. *PLOS ONE*, *17*(5), e0264972. https://doi.org/10.1371/JOURNAL.PONE.0264972

Bajaj, A., and Sangwan, O. P. (2019). A Systematic Literature Review of Test Case Prioritization Using Genetic Algorithms. *IEEE Access*. https://doi.org/10.1109/ACCESS.2019.2938260

Bello, A., Md. Sultan, A. B., and Shehu, S. (2019). Multi-Criteria Evolutionary Regression Test Prioritization for Dynamic Object-Oriented Programs. *International Journal of Advances in Electronics and Computer Science*, *6*(1), 14–18.

Bello, A., Sultan, A., Abdul Ghani, A. A., and Zulzalil, H. (2018). Evolutionary Cost Cognizant Regression Test Prioritization for Object-Oriented Programs Based on

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi



Fault Dependency. *International Journal of Engineering and Technology*, *7*(4.1), 28–32.

Huang, R., Zhang, Q., Towey, D., Sun, W., and Chen, J. (2020). Regression test case prioritization by code combinations coverage. *Journal of Systems and Software*, *169*, 110712. https://doi.org/10.1016/j.jss.2020.110712

Id, D., Pan, R., Bagherzadeh, M., Ghaleb, T. A., and Briand, L. (2021). *Test Case Selection and Prioritization Using Machine Learning: A Systematic Literature Review*. *1*, 1–32.

Mishra, D. B., Mishra, R., Acharya, A. A., and Das, K. N. (2019). Test case optimization and prioritization based on multi-objective genetic algorithm. In *Advances in Intelligent Systems and Computing* (Vol. 741). Springer Singapore. https://doi.org/10.1007/978-981-13-0761-4_36

Musa, S. (2021). *IJSGS FUGUSAU VOL. 7(1) MARCH 2021 WEBSITE: http://journals.fugusau.edu.ng*. *7*(March), 1–8.

Panigrahi, C. R., and 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

Paterson, D., Abreu, R., Kapfhammer, G. M., Fraser, G., Mcminn, P., Lisbon, U., College, A., and Prioritization, A. T. C. (2019). *An Empirical Study on the Use of Defect Prediction for Test Case Prioritization*. https://doi.org/10.1109/ICST.2019.00041

Paterson, D., Kapfhammer, G. M., Fraser, G., and Mcminn, P. (2018). *Using Controlled Numbers of Real Faults and Mutants to Empirically Evaluate Coverage-Based Test Case Prioritization*. 57–63.

Raamesh, L., Radhika, S., and Jothi, S. (2022). A cost-effective test case selection and prioritization using hybrid battle royale-based remora optimization. *Neural Computing and Applications*, *34*(24), 22435–22447. https://doi.org/10.1007/S00521-022-07627-1/METRICS

Ramírez, A., Informática, D. De, and Insti-, U. D. C. (2022). ATaxonomy of Information Attributes for Test Case Prioritisation: Applicability,Machine Learning. *ACM Transactions on Software Engineeringand Methodology*, *1*(1). https://doi.org/10.1145/3511805

Samad, A., Mahdin, H. Bin, Kazmi, R., Ibrahim, R., and Baharum, Z. (2021). Multiobjective Test Case Prioritization Using Test Case Effectiveness: Multicriteria Scoring Method. *Scientific Programming*, *2021*. https://doi.org/10.1155/2021/9988987

Yadav, D.K., Dutta, S. (2020). Regression test case selection and prioritization for object oriented software. *Microsyst Technol*, *26*, 1463–1477.

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi