İLERİ MÜHENDİSLİK ÇALIŞMALARI VE TEKNOLOJİLERİ DERGİSİ

A Novel Regression Test Selection Method with Graph-Based Genetic Algorithm

Ramazan OZKAN[1], Zeynep ORMAN[2], Ruya SAMLI*[3]

[1]*Nation Defense University, Air Force Academy, Department of Computer Engineering, Istanbul, 34334, Turkey*

[2,3]*Istanbul University-Cerrahpasa, Engineering Faculty, Department of Computer Engineering, Istanbul, 34320, Turkey*

Research Article, Received Date: 11.04.2023, Accepted Date: 05.06.2023

Abstract

Regression test is a re-running test type to ensure that previously developed and tested software is not seriously affected by changes. Testing a software after changes is very important and necessary in order to maintain the software development and maintenance processes. However, repeating all tests after each change is not feasible especially in large-scale projects. Regression test selection which means selection of a subset of tests has emerged as a solution to this issue. This paper presents a GBGA (Graph-Based Genetic Algorithm) with the most compatible neighbor crossover as a solution to the regression test selection problem. In this GBGA, each individual in the population is located on a node of predefined graph structure and the probabilities of the crossover are limited depending on the neighborhood relations to increase population diversity, prevent premature convergence, and refine the convergence performance. This GBGA is applied to this problem to find the minimum set of test cases to enhance the performance of the genetic algorithm by locating populations on graphs and limiting the crossover option with neighborhood connections to increase the diversity. The results show that the proposed GBGA with the most compatible neighbor crossover has superior performance in terms of fitness value when compared to genetic algorithm.

Keywords: Regression test selection, Graph-based genetic algorithm, Compatible crossover, Optimization.

1. INTRODUCTION

A software system evolves during its development and maintenance phases with bug fixing, optimization, enhancement or adaptation activities in time and it must be re-tested after these changes. Regression testing is the activity which is applied to ensure that this evolution does not affect the approved functionality of the software system.

The simplest and safest approach for regression testing is to re-run all test cases, which is called the “re-test all” technique (Rothermel, 1996). However, repeating all test cases, which are previously executed successfully, after each software revision is not practical due to time and budget constraints especially in the case of large-scale software systems.

*Corresponding author ruyasamli@iuc.edu.tr,
[1]rozkan@hho.msu.edu.tr, [2]ormanz@iuc.edu.tr

Therefore, a large amount of research effort has been spent in the literature to be able to select a subset of the test cases, which is called regression test selection, with acceptable cost-benefit balance and several approaches have been proposed for this purpose. The regression test selection process requires a balance between the cost and benefit of regression testing. Providing this balance is an NP-complete optimization problem and it cannot be solved in a reasonable amount of time for large-scale software systems which includes complex test suites (Yamuç et al., 2017). Many heuristic search-based solutions have been used in literature for regression test selection (Li et al., 2007; Mittal & Sangwan, 2018; Panichella et al., 2015; Yadav & Dutta, 2017) and one of them is the Genetic Algorithm.

Genetic Algorithm is a heuristic-based approach to solve problems that cannot be solved with deterministic methods. It mimics the evolution of the species based on natural selection (Mirjalili, 2019) in a population and includes 4 basic steps which are the creation of the initial population, selection, genetic operators and termination. In the first step, a random solution population is created from the search space.

Then, based on a problem specific fitness function which mimics the adaptation level of species in nature, individuals are selected from the population. Genetic operators which are crossover and mutation are applied to selected individuals and a new generation is created. This process is repeated until to reach the termination criteria. Genetic Algorithm is an efficient heuristic search method and there is a huge amount of application in literature including regression test selection (Li et al., 2007; Mittal & Sangwan, 2018; Panichella et al., 2015; Yadav & Dutta, 2017).

On the other hand; the greatest weakness of Genetic Algorithm is the premature convergence due to the loss of population diversity over generations (Ghoumari & Nakib, 2019; Lee et al., 2008; Toffolo & Benini, 2003). Selecting the best individuals in each population creates a population that includes similar individuals, and this may cause immature convergence to a local optimum. In order to resolve this issue, several algorithms have been proposed in the literature (Bryden et al., 2006; Garousi et al., 2018; Lee et al., 2008; Lu et al., 2007; Mirjalili, 2019; Toffolo & Benini, 2003; Whitley et al., 1999). One of these proposals is the GBEA (Graph-based Evolutionary Algorithm) (Bryden et al., 2006) which uses graphs as a geographic structure to locate individuals in the population and indicate the links between them for mating limitations.

As a subfield of evolutionary algorithm, Genetic Algorithm can also be used on graphs for the problems encoded in a series of bit strings and named as GBGA. This paper proposes a tailored version of GBGA with the most compatible neighbor crossover for regression test selection. To our best knowledge, it is the first time that a GBGA has been applied to the regression test selection problem in the literature. This approach enhances the performance of the Genetic Algorithm by locating populations on a graph and limiting the crossover option with neighborhood connections. Another novelty of our paper is that, unlike the GBGA methods used in the literature, the parent selection step of the crossover operator is designed as selecting the most compatible one among the neighbors of the first parent coming from the crossover parent pool as the second parent. With this modification to the crossover operator, the genetic diversity of the GBGA is increased while it is preventing from transforming into a random search algorithm. The proposed GBGA with the most compatible neighbor and Genetic Algorithm are applied to a dataset including 216 test cases, 5610 requirements tested under these test cases and affected requirements lists of five different software versions of a software project used in the study of Garousi et al. (Garousi et al., 2018).The performance of the proposed GBGA is evaluated by comparing it with Genetic Algorithm in terms of fitness value, affected requirement coverage, irrelevant requirement coverage and execution time.
In terms of fitness value, as main comparison criterion, which is calculated with affected requirement coverage rand irrelevant requirement coverage rates, GBGA with the most compatible neighbor crossover gives better results than the Genetic Algorithm.

The rest of the paper is organized as follows. Section 2 overviews the related studies in literature according to underlying goals of this study. Section 3 explains the materials and the methodology and gives the case description and needs for the study. Section 4 presents the proposed GBGA for regression test selection. Section 5 demonstrates and analyzes the results. Finally, conclusions are given in Section 6.

2. BACKGROUND

2.1. Regression Test Selection

The regression test selection process aims to find test cases that are relevant to software changes based on impact analysis. There are several regression test selection methods proposed in the literature. Based on impact analysis differences, these techniques can be placed in three different groups: code analysis-based methods (Aggrawal et al., 2004; Gupta et al., 1992; Jones & Harrold, 2001; Rothermel, 1996; Yamuç et al., 2017), model-based methods (Briand et al., 2009; Engström et al., 2011; Farooq et al., 2007) and requirement analysis-based methods (Aggrawal et al., 2004; Özkan, 2017; Rothermel & Harrold, 1997).

2.1.1 Code Analysis-Based Methods

Most of the regressing test selection methods focused on source code analysis such as execution trace analysis, data flow analysis and control flow analysis (Garousi et al., 2018).

The execution trace of a test case on a program means the execution sequence of program statements that are executed with the test case. In execution trace analysis, execution traces of test cases for old and new versions of the program are compared and test cases that have different execution paths are selected for regression test (Özkan, 2017). Akhin and Itsykson have used this method by identifying the modified software components and extracting the dependency information for test-software component relation (Akhin & Itsykson, 2009). Vokolos and Frankl have proposed a tool named Pythia (Vokolos & Frankl, 1998). This tool uses a slightly different version of execution trace analysis. It keeps a history of the basic blocks executed by each test case and to identify the modified program statements, compares the source files of the old and new versions of the program.

In data flow analysis, data interactions that have been affected by modifications are determined. To find affected interactions, definition-use pairs of the variables are analyzed and test cases executing the path from definition to use of the modified variables are selected for regression. Gupta et al. (1992) proposed a data flow analysis based regression test selection method by using slicing algorithms to explicitly detect definition-use associations that are affected by a program change.

Control flow analysis-based methods are structured based on the analysis of Control Flow Graphs (CFG) differences of original and modified program. A control flow analysis based regression test selection method has proposed by Rothermel and Harold (Rothermel & Harrold, 1997) named as Graph Walk. In this method, control flow graphs of the old and modified program are compared and if any node in the control flow graphs of the old program is not equivalent to the corresponding node in the modified program, all test cases that execute mismatching node are added to test suite.

In these methods, it has been tried to find an optimum test suite that covers the relevance code part based on a time-consuming static analysis of source code. Basically, the cost of regression test selection and execution of selected test cases should be less than rerunning all test cases (Graves et al., 2001). Because of this concern, the application of code-analysis based methods especially for large-scale and complex systems is quite challenging and a limited number of empirical evaluations have been carried out in a real industrial context (Engström et al., 2010).

2.1.2. Model-Based Methods

Model-based regression selection methods use design models like class diagrams, sequence diagrams or case diagrams (Briand et al., 2009). Changes on these models and their impacts on previously verified test suite are analyzed. Farooq et al. (2007) proposed an UML (Unified Modeling Language) based selective regression testing strategy which uses state machines and class diagrams for change identification. Gorthi et al. (2008) proposed a model based approach in their study and used the UML Use Case Activity Diagram to analyze the impacts of changes and select the required test cases.

The application of model-based methods is also limited as the time-consuming static analysis of design models causes similar concerns as code analysis-based methods.

2.1.3. Requirement Analysis-Based Methods

Requirement coverage-based methods aim to find an optimum test set that covers the maximum number of affected requirements which means the requirements affected by software modifications and need to be re-tested. Chittimalli and Harrold (2008) have proposed the basic requirement coverage-based regression test-selection method in which the regression test suite is created by including modification-related requirements. In 2009, Krishnamoorthi and Mary, in 2016 Srikanth et al. and in 2010 Gu et al. have improved this method by using additional factors other than affected requirement coverage such as irrelevant requirement coverage which are the ones that are not affected by the modifications, customer priority, fault impact and implementation complexity.

Regression test selection is an NP-complete optimization problem. There are multiple optimization algorithms used in the literature for regression test selection. Mirarab et al. (2012) have prioritized the selected subset of test cases using a greedy algorithm that maximizes minimum coverage in an iterative manner. Krishnamoorthi and Mary (2009) have proposed a test case prioritization technique using the Genetic Algorithm for a time-constrained execution environment. Li et al. (2007) have presented results from an empirical study of the application of several greedy, metaheuristic, and evolutionary search algorithms to six programs for regression testing.
In 2011, Harman emphasized the regression test selection as a multi-objective optimization problem and in 2018 Garousi et al. have applied this approach to a specific problem by using the genetic algorithm. In their study, the requirement coverage-based regression test selection method is applied the data set used in this study by using a tailored GBGA which has not been applied to regression test selection before in literature.

2.2. Graph-Based Genetic Algorithm

The GBGA initially was used for problems which are already in a graph structure such as NN (Neural Networks) or genetic programming tree. In the study of Miller (1989), the adjacency matrix of a NN is transformed into a binary string by concatenating the adjacency matrix. Genetic programming invented by Koza & Stanford (1990) has swapped sub trees in a tree topology for crossover similar to a one-point crossover in the standard Genetic Algorithm. Korkmaz and Üçoluk (2004) have used the fitness value of sub trees for guiding recombination, not to lose high-value sub trees. In Doerr et al., (2007), unlike the general node-based structure, an edge-based representation has been proposed to improve optimization time. In this method, each edge is stored with its two neighbor edges. Samuel (2008) has proposed a matrix-based crossover and mutation operator by transforming the graph to the adjacency matrix. To be able to eliminate invalid solutions, a connectivity constraint was added to operators.

In the study (Ghoumari & Nakib, 2019), the adaptation of evolution strategy (associations of a crossing operator and a mutation operator) during evolution has been proposed. 20 different evolution strategies are represented in a graph structure and to minimize diversity loss during evolution if the strategy could not improve or protect the diversity it is changed with a new one. Diversity is calculated using Euclidean distance.

All of the above-mentioned methods have been proposed to use evolutionary algorithms on graph type represented problems. On the other hand, a different approach in GBEAs which use graphs to add geography to the population as a solution to premature convergence due to insufficient diversity in evolutionary algorithms was proposed (Bryden et al., 2006). In their approach, a suitable graph structure is selected, and then the evolutionary algorithm is applied to this structure with graph suitable operators. In the standard evolutionary algorithm, individuals that have good fitness values are selected for recombination in each population. Repeating this process decreases the diversity and creates a population that includes similar individuals. GBEA is proposed as a solution to premature convergence due to insufficient diversity in evolutionary algorithms. They use a different type of combinatorial graphs to impose a topology or "geographic structure" on an evolving population. To create a population, each individual is placed on a vertex of the selected graph structure. In order to improve diversity, individuals can be replaced only with the combination of neighbor individuals which are the members of the same edge as a geographic constraint in natural selection. Then a steady-state evolutionary algorithm proposed by Syswerda (1991) is used in which evolution proceeds one mating event at a time. For a mating event, an individual is selected randomly and based on the fitness value a neighbor individual is used for crossover. If the fitness value of the new individual is better than the selected one, it is replaced with the new one.

Based on the same approach, in a study different evolutionary computation problems have been categorized using 15 different connected combinatorial graph structures. A combinatorial graph is composed of vertices and edges which connect vertices as a set of unordered pairs. If any vertex in the graph can be traversed from any other vertex, that graph is defined as a connected graph. Each problem is run on several different graphs and based on the solution time problems are categorized. The choice of graph affects the recombination number for convergence and controls the spread of solutions within the population. It is explained that selecting a suitable graph and tuning can reduce optimization time significantly. Problems with simpler fitness function performed best with highly connected graphs (10 times faster), while problems with difficult fitness landscapes performed better (12 times faster) with less connected graphs. The study (Bryden et al., 2003) applied GBEA to the optimization of heat transfer in a complex system. Specifically, the time to solution and the diversity of the population were examined by using four different graph structures.

As mentioned above, GBEA is used in literature to overcome premature convergence problem of evolutionary algorithm due to the loss of population diversity over generations. Genetic Algorithm which is used for the problems encoded in a series of bit strings is a subfield of evolutionary algorithms and it can also be used on graphs as an evolutionary algorithm. In this study, to benefit from advantage of increasing diversity of graph-based structure, a tailored version of GBGA with three different graph types is proposed and applied to regression test selection for the first time in literature. To increase the diversity of future generations, mating options of individuals in a population are limited with the neighbors of the selected individuals on the graph and the most compatible neighbor is selected for mating as a novelty on GBGA. The proposed approach is applied to a regression test selection problem including 216 test cases, 5610 requirements tested under these test cases and affected requirements lists of five different software versions of a software project used in the study of Garousi et al. (2018) and the application results show that GBGA with the most compatible neighbor has superior performance in terms of fitness value when compared to Genetic Algorithm.

3. MATERIALS AND METHODS

GBGA is a version of the Genetic Algorithm in which the population is placed on a graph inspired by the concept of distance in geography and mating is permitted only between neighboring individuals to be able to keep genetic diversity. In the scope of this study, three different graph types are used as baseline structures for populations of GBGA. Graphs are modeled as neighborhood matrices and these ones which are used to limit crossover possibilities between individuals. Crossover between any two individuals is possible only if there is a connection between their locations on the graph.

In this section of the paper, used graph types and details of GBGA with the most compatible neighbor crossover are explained.

3.1. Used Graph Types

Three different graphs are used as baseline structures for populations of GBGA: torus, Petersen, and 2-pre-Z graphs.

The Torus Graph is a graph whose vertices can be placed on a torus such that no edges cross ("Toroidal graph," 2021).These graphs are grids that wrap at the edges. The $n\,x\,m$-torus (n corresponding to the number of consecutive vertexes on the big circle of torus and m corresponding to the number of consecutive vertexes on the small circle) denoted $T_{n,m}$, has vertex set $Z_n\ x\ Z_m$ (Bryden et al., 2006). Each vertex has edges only with its neighbors. A $12x6$-torus is shown in Figure 1(a).

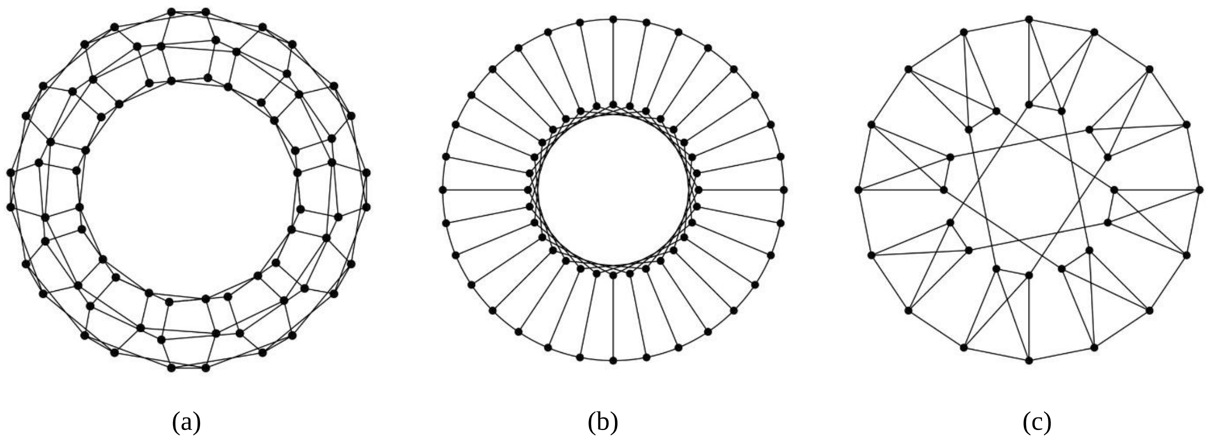


Figure 1. (a)$12x6$ torus graph, (b) 32x5 Petersen Graph, (c) 2-Pre-Z Graph with 32 vertexes (Bryden et al., 2006)

The generalized Petersen graph with parameters n and k (n corresponding to the number of consecutive vertexes on the outside and inside shape which is a circle here and k corresponding to the number of inside vertexes hopped for next edge) is denoted $P_{n,k}$and has vertex set $0, 1, 2, \ldots, 2n-1$(Bryden et al., 2006). An $32x5$-Petersen graph is shown in Figure 1(b).

The 2-Pre-Z graph is a graph obtained as an intermediate product of the simplifying process defined in (Bryden et al., 2006) on a $4x4$ complete graph. A 32 vertex 2-Pre-Z graph is shown in Figure 1(c).

3.2. Graph-Based Genetic Algorithm with The Most Compatible Neighbor Crossover

As in genetic algorithm, GBGA has 4 basic steps which are the creation of the initial population, selection of the individuals, applying the genetic operators (crossover and mutation), and termination. However, a neighborhood matrix which represents the graph structure in GBGA should be generated before the beginning of the search process and a problem-specific fitness function used to evaluate individuals in selection and crossover operations is defined for each problem. GBGA flow diagram can be seen in Figure 2.

3.2.1. Neighborhood Matrix

Neighborhood matrix, an example of which can be seen in Figure 3, is a symmetric (0,1)-matrix with zeros on its diagonal. Each row and column number represents a vertex on the graph and the same row and column number represent the same vertex. 0 means there is no connection between the vertexes that row and column number of the matrix element correspond and 1 means there is a connection. Neighborhood matrix corresponds to undirected adjacency matrix in graph theory and computer science. An adjacency matrix represents a finite graph. Its elements indicate whether pairs of vertices are adjacent or not in the graph.

Before starting GBGA, a $n\ x\ n$ neighborhood matrix (n corresponding to the population size) is created for the graph that will provide the infrastructure for the population. This matrix specifies the connections between individuals which are used as mating possibilities in crossover operations.

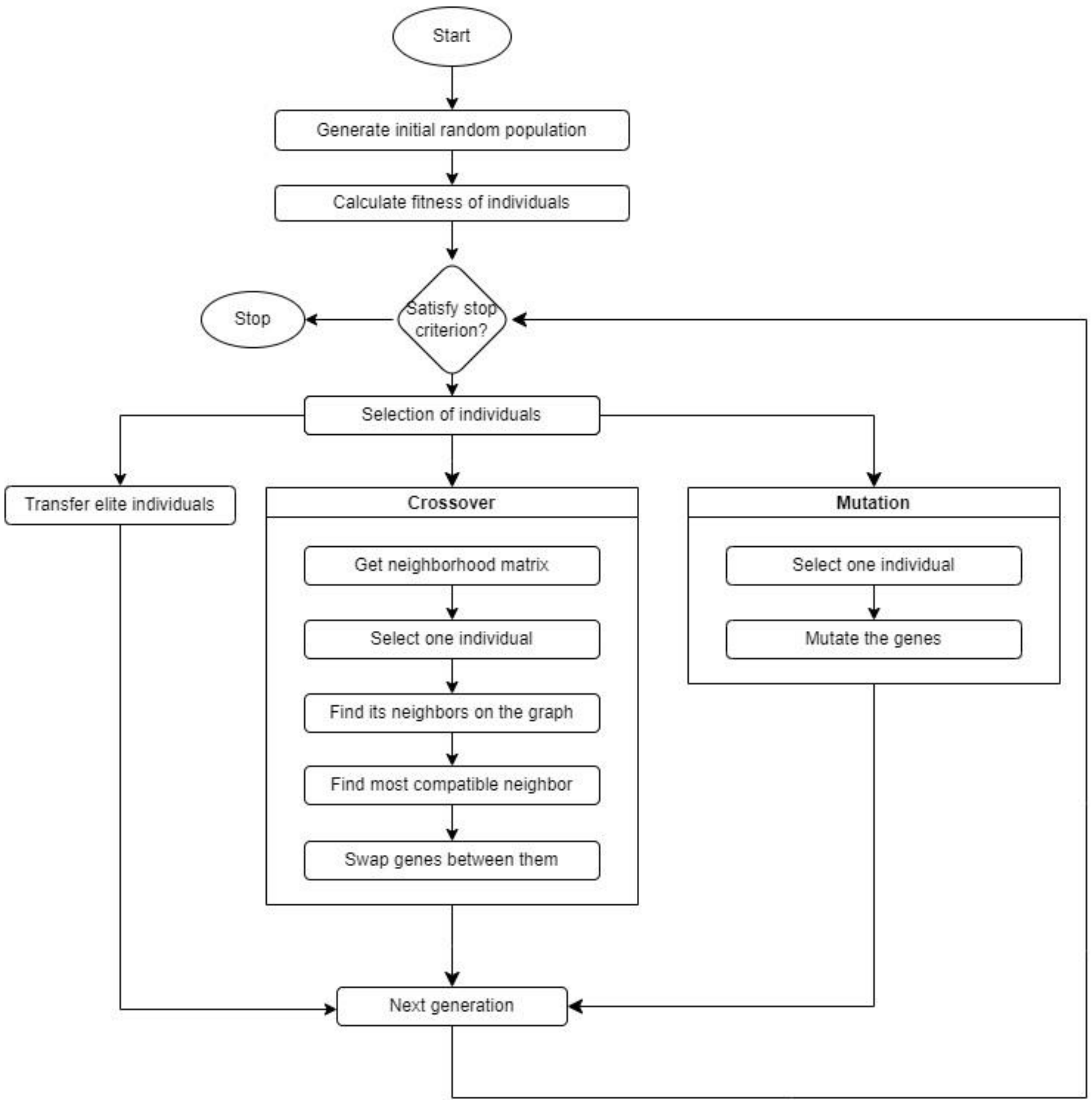
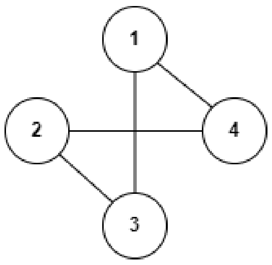
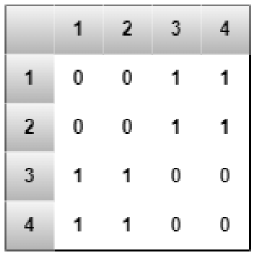


Figure 2. GBGA Flow Diagram





	1	2	3	4
1	0	0	1	1
2	0	0	1	1
3	1	1	0	0
4	1	1	0	0

Figure 3. Sample Neighborhood Matrix

3.2.2. Initial Population Creation:

The initial population is generated randomly based on the problem-specific gene structure. It is assumed that each individual in the population is placed on a vertex corresponding to its number in the selected graph. Graph vertex set should be equal to predetermined population size.

3.2.3. Fitness Function

In the Genetic Algorithm, a portion of the existing population is selected through a fitness function to breed a new generation. As in Genetic Algorithm, a problem-dependent fitness function which measures the quality of the individuals for selection should be determined in GBGA.

3.2.4. Parent Selection for Mating

Selected individuals through fitness function are the parent pool of the next generation. In the Genetic Algorithm, to produce each child individual via crossover operator, a pair from the parent pool is selected and a child is produced by using different crossover methods.

In GBGA with the most compatible neighbor crossover, only one of the parents is selected from the parent pool. Therefore, the crossover parent pool size is half of the Genetic Algorithm. To increase the diversity, the other one is chosen among the selected one's neighbors on the graph via neighborhood matrix. This selection process consists of two steps: *finding neighbors* and *selecting the most compatible neighbor*.

***Finding neighbors*:** As shown in Figure1 vertexes have different edge structures for each graph. Neighbors of the selected parent are determined based on the neighborhood matrix of the used graph. In the neighborhood matrix, a binary representation is used to show edges between vertexes. Each binary value of "1" means the existence of a connection between the corresponding row and column vertexes and "0" means no connection. Therefore, column numbers with "1" values in the row corresponding to the selected individual indicate its neighbors.

***Selecting the most compatible neighbor*:** Inspired natural mating, the most compatible neighbor is selected as the other parent for crossover. "Most compatible" means the similarity between parents. For example, for the problem addressed in this study, regression test selection, "most compatible" means test coverage similarity between parents. The neighbor which covers more common tests with the selected parent than other neighbors is chosen as the second parent.

Limiting the crossover between neighboring increases diversity and choosing the most compatible neighbor prevents the genetic algorithm from turning into a random search.

3.2.5. Crossover and Mutation

After the selection of parents, identical crossover and mutation operators with the genetic algorithm are used for GBGA.

4. PROPOSED ALGORITHM: REQUIREMENT COVERAGE BASED REGRESSION SELECTION BY USING THE GRAPH-BASED GENETIC ALGORITHM WITH THE MOST COMPATIBLE NEIGHBOR CROSSOVER

In the scope of this study, GBGA with the most compatible neighbor crossover is applied to a requirement coverage-based regression test selection problem for three different graphs defined in section 3 and it is compared with the traditional genetic algorithm. The proposed solution is developed in Matlab Global Optimization Toolbox by modifying the genetic algorithm structure provided by this toolbox.

4.1. Dataset

Dataset of the problem which includes a traceability matrix between the test cases and requirements and affected requirements lists of five different software versions of a software project are obtained from (Garousi et al., 2018). Traceability Matrix is the fundamental structure of the requirement coverage-based selection. It represents the relationship between the test cases and requirements in binary matrix format. Each row of the matrix represents a test case, and each column represents a requirement. Each binary value of "1" means that the corresponding row test case covers the corresponding column requirements and "0" means do not cover. This binary matrix representation makes it easy to find out the requirement coverage of candidate solutions in the GBGA search flow.

The size of the traceability matrix has the dimension of $216\ x\ 5610$. 5610 system requirements are tested via 216 different test cases. The relationship between test cases and requirement list is many-to-many which means that a test case covers more than one requirement, and a requirement can be tested in more than one test case.

In addition to traceability matrix, affected requirements lists are used to calculate the requirements coverage performance of GBGA and Genetic Algorithm for five different software versions are represented in $1\ x\ 5610$ size binary vector format.

This dataset is used for empirical evaluation and tuning of the GBGA.

4.2. Proposed Solution

In this part of the paper, the basic steps of GBGA with the most compatible neighbor crossover are explained from an application perspective.

4.2.1. Initial Population Creation

The initial population is created randomly in the structure shown in Figure 4. Each gene as a binary bit represents a regression test case and each chromosome/individual as a $1\ x\ 5610$ size binary vector represents a set of test cases as a possible solution. For each bit, the value of "1" means the existence of the corresponding test case in the solution (regression test-set), and "0" means its absence.

Population size is an important parameter that significantly affects the performance of the Genetic Algorithm and also GBGA. An insufficient number of individuals will cause the Genetic Algorithm to quickly converge to a local minimum (Gotshall & Rylander, 2000). On the other hand, if the population includes too many chromosomes, the Genetic Algorithm may have a performance problem. De (1998) proposed a population size ranging from 50 to 100 chromosomes. Cobb & Grefenstette (1993) recommended a range between 30 and 80. In this study, population size is determined and used as 80 for GBGA and Genetic

Algorithm

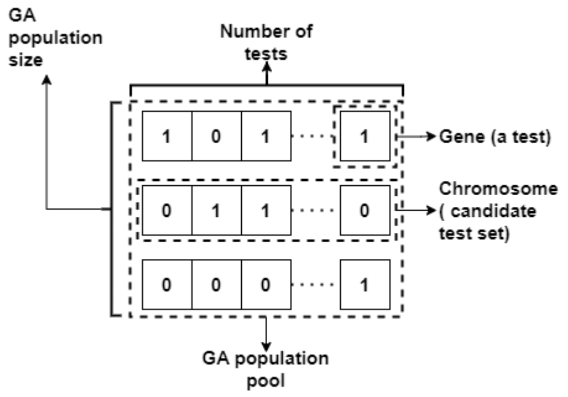


Figure 4. Representation of Population

4.2.2. Fitness Function

In this study, the fitness function is designed on minimizing irrelevant requirement coverage while maximizing the affected requirement coverage. The affected requirement coverage of a regression test set is the proportion of affected requirements covered by that test set to the total number of affected requirements. On the other hand, irrelevant requirement coverage is one minus the proportion of irrelevant requirements covered by a subject test set to the total number of irrelevant requirements. It was formulated as shown in Equation 1 in order to construct it as finding the smallest value. arv is used for affected requirement coverage, irv is used for irrelevant requirement coverage. Identical fitness function is used for classic and GBGAs.

$$Fitness(c) = \frac{1}{arv + irv}$$

Equation 1. Fitness function

4.2.3. Parent Selection for Mating

As defined in section 3, only one of the parents is obtained from the crossover parent pool. The other parent is selected among the neighbors of the first parent on the graph. After determining the neighbors via neighborhood matrix, the most compatible neighbor is selected as the second parent. For the requirement coverage-based regression test selection problem “the most compatible neighbor” means the neighbor which has more common tests with the chosen parent than the other neighbors. After determining the parents,
an identical crossover operator with the Genetic Algorithm is used to generate child individuals.

4.2.4. Termination

Termination criteria determine what causes the algorithm to terminate. In this study, the maximum generation number as 120 and the maximum generation number where the best value does not change more than a certain threshold as 20 are used as the termination criteria for Genetic Algorithm and GBGA. Algorithms stop when reaching any of these two criteria.

4.3. Tuning Parameters of Graph-Based Genetic Algorithm

To be able to reach the best performance, internal parameters of the Genetic Algorithm (e.g., crossover and mutation rates) must be properly tuned for a specific case. Many studies have shown that the tuning of a Genetic Algorithm has a strong impact on its performance. In this study, crossover rate and mutation rate parameters of both the Genetic Algorithm and GBGA for three different graphs are tuned empirically based on the fitness value and execution time. Each GBGA and Genetic Algorithm is executed 100 times for each crossover rate starting from 0.05 to 1 with an increasing value of 0.05 and for each mutation rate starting from 0.01 to 0.2 with an increasing value of 0.01. In order to assess the performance with each crossover rate and mutation rate, average values of fitness values across 100 runs are used. Determined values after this tuning process are shown in Table 1.

Table 1. Tuned parameter values

	Crossover Rate	Mutation Rate
GA	0.6	0.07
Torus GBGA	0.75	0.03
Petersen GBGA	0.8	0.04
2-Pre-Z GBGA	0.7	0.06

5. RESULTS AND DISCUSSIONS

In order to test the performance and compare, GBGA and Genetic Algorithm are used to find the minimum set of tests for regression test selection based on a data set which includes 216 tests, 5610 requirements verified by these tests, and an affected requirement list for five different software versions. Each algorithm is run 100 times for each software version and each run starts with a different initial population and stops when one of the termination criteria is met. The performance of the GBGA with three different graphs and Genetic Algorithm is compared in terms of fitness value, affected requirement coverage, irrelevant requirement coverage, and execution time.

5.1. Fitness Value Comparison

From the fitness value aspect, GBGA with three different graphs provides better results (lower values are better because fitness function is formulated as finding the smallest value) than the Genetic Algorithm for all five software versions as shown in Figure 5. GBGA with Petersen graph is slightly better than the other two GBGAs with torus and 2PreZ graphs. It can be said that the Petersen graph is the best option for requirement coverage-based regression test selection with GBGA.

In this study, fitness value is the one and only determinant for solution selection. Therefore, it is the most important factor to make a comparison in the scope of this study. The fitness value is the primary comparison criterion in this study. As discussed in Section 4, the fitness value of a solution is the combination of affected and irrelevant requirement coverage.

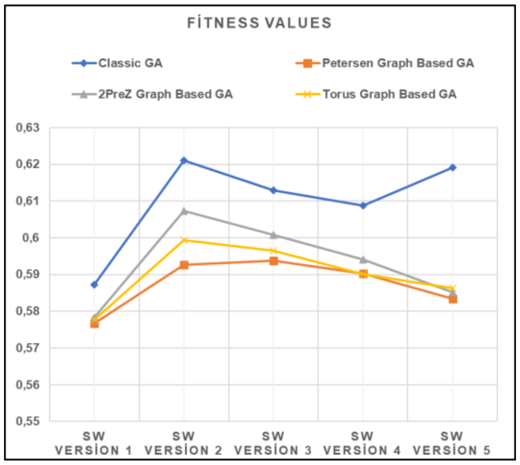


Figure 5. Fitness Value Comparison

5.2. Affected and Irrelevant Requirement Coverage Comparison

In addition to fitness value comparison, coverage ratios of affected and irrelevant requirements are also compared separately. As explained in section 2, affected requirements are those which correspond to the modifications made in the source code across two software versions and irrelevant requirements are the ones that are not affected by the modifications.

Requirement coverage comparison has two different scenarios as shown in Figures6 and 7. Genetic Algorithm has slightly better results in affected requirement coverage perspective than GBGAs, while GBGAs have clearly better results in irrelevant requirement coverage perspective.

When compare GBGAs; GBGA with 2PreZ graph is the best one among GBGAs in affected requirement coverage, while GBGA with Petersen graph is the best in irrelevant requirement coverage. However, affected and irrelevant requirements coverages are secondary criteria, and these comparisons are given as details of fitness value comparison.

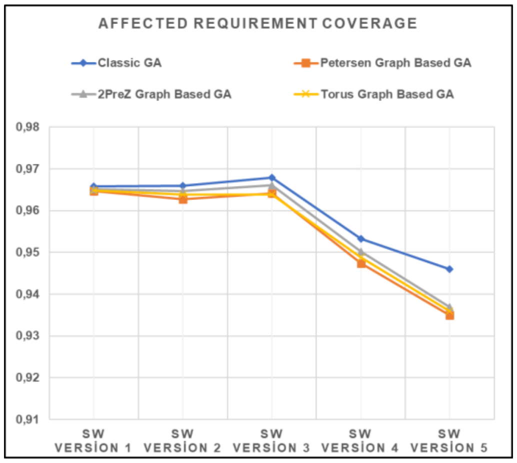


Figure 6. Affected Requirement Coverage Comparison

5.3. Execution Time Comparison

In execution time comparison seen in Figure 8, the Genetic Algorithm requires less execution time than GBGAs as expected. “Finding the most compatible neighbor” process is the reason for this difference. It increases the execution time of the GBGAs linearly depending on the number of neighbors. However, this difference is negligible when compared to the execution of tests cases.

When comparing the GBGAs, it is observed that GBGA with Petersen graph has a lower execution time values than the other two because its average neighbor number for a vertex is less than others.

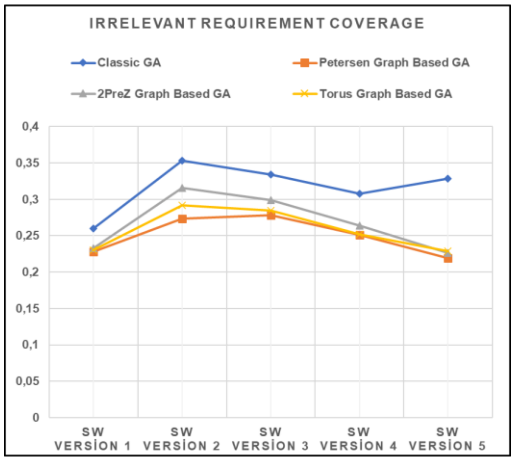


Figure 7. Irrelevant Requirement Comparison

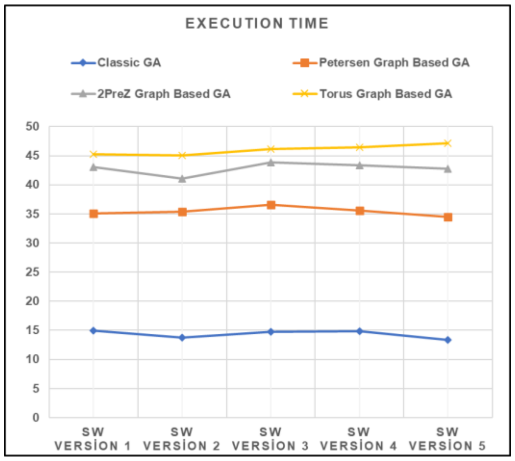


Figure 8. Execution Time Comparison

6. CONCLUSIONS

This paper presents a GBGA modeled with three different graphs named torus, Petersen, and 2PreZ for requirement coverage-based regression test selection. To the best of the authors' knowledge, there is no study in the literature using a GBGA for regression test selection problems.

In order to evaluate the performance of the GBGA with the most compatible neighbor crossover, it is compared with the Genetic Algorithm in terms of fitness value, affected requirement coverage, irrelevant requirement coverage, and execution time.
When evaluated in terms of fitness value, which is the one and only determinant for solution selection as explained in detail in section 5, the GBGA gives better results than the Genetic Algorithm in all five different software versions included in the dataset.

In this study, the GBGA with the most compatible neighbor crossover is proposed as a solution to requirement coverage-based regression test selection. However, the application model of the GBGA provides a general and flexible structure which is applicable to most of the optimization problems by customizing the genetic structure based on problem-specific data and tuning the parameters of genetic operators.

In the scope of this study, three different graph types are used as the baseline structure for GBGA. These graphs are obtained from the study of Bryden, Ashlock, Corns, and Stephen (Bryden et al., 2006). They have reviewed 15 different graphs in their study for GBGA. Inspired by this paper, other graph types which are not used in our study can also be applied to the regression test selection problem.

Application of GBGA to different optimization problems with a newly designed crossover operator would be a step toward understanding what types of problems it is applicable to.

Future work directions can be applying the proposed approach with more graph types (1), and to different optimization problems (2).

REFERENCES

Aggrawal, K. K., Singh, Y., & Kaur, A. (2004). Code coverage based technique for prioritizing test cases for regression testing. *In Proceedings of ACM SIGSOFT Software Engineering Notes*.

Akhin, M., & Itsykson, V. (2009). A regression test selection technique based on incremental dynamic analysis. *In Proceedings of Software Engineering Conference in Russia CEE-SECR.*

Briand, L. C., Labiche, Y., & He, S. (2009). Automating regression test selection based on UML designs. *Information and Software Technology*, *51*, 16–30.

Bryden, Kenneth M., Ashlock, D. A., McCorkle, D. S., & Urban, G. L. (2003). Optimization of heat transfer utilizing graph based evolutionary algorithms. *International Journal of Heat and Fluid Flow*, *24*(2), 267–277.

Bryden, Kenneth Mark, Ashlock, D. A., Corns, S., & Willson, S. J. (2006). *Graph-Based Evolutionary Algorithms*. *10*(5), 550–567.

Chittimalli, P. K., & Harrold, M. J. (2008). Regression Test Selection on System Requirements. *In Proceedings of the 1st India Software Engineering Conference,87–96.*

Cobb, H. G., & Grefenstette, J. J. (1993). Genetic Algorithms for Tracking Changing Environments. *In Proceedings of the 5th International Conference on Genetic Algorithms 523–530.*

De, K. (1988). *Learning with Genetic Algorithms : An Overview*. 121–138.

Doerr, B., Klein, C., & Storch, T. (2007). Faster evolutionary algorithms by superior graph representation. *In Proceedings of the 2007 IEEE Symposium on Foundations of Computational Intelligence, FOCI, 245–250.*

Engström, E., Runeson, P., & Ljung, A. (2011). Improving regression testing transparency and efficiency with history-based prioritization - An industrial case study. *In Proceedings of the 4th IEEE International Conference on Software Testing, Verification, and Validation, ICST.*

Engström, E., Runeson, P., & Skoglund, M. (2010). A systematic review on regression test selection techniques. *Information and Software Technology*, 52(1), 14–30.

Farooq, Q., Iqbal, M. Z. Z., Malik, Z. I., & Nadeem, A. (2007). An approach for selective state machine based regression testing. *In Proceedings of the 3rd International Workshop on Advances in Model-Based Testing - A-MOST, 44–52.*

Garousi, V., Özkan, R., & Betin-Can, A. (2018). Multi-objective regression test selection in practice: An empirical study in the defense software industry. *Information and Software Technology*, *103*.

Ghoumari, A., & Nakib, A. (2019). *Graph based adaptive evolutionary algorithm for continuous optimization*. *September*. http://arxiv.org/abs/1908.08014

Gorthi, R. P., Pasala, A., Chanduka, K. K., & Leong, B. (2008). Specification-based approach to select regression test suite to validate changed software. *Neonatal, Paediatric and Child Health Nursing*, 153–160.

Gotshall, S., & Rylander, B. (2000). Optimal population size and the genetic algorithm. *In Proceedings On Genetic And Evolutionary Computation Conference*, 1–5.

Graves, T. L., Harrold, M. J., Kim, J. M., Porter, A., & Rothermel, G. (2001). An empirical study of regression test selection techniques. *ACM Transactions on Software Engineering and Methodology*, *10*(2), 184–208.

Gu, Q., Tang, B., & Chen, D. (2010). Optimal Regression Testing based on Selective Coverage of Test Requirements. *In Proceedings of International Symposium on Parallel and Distributed Processing with Applications, ISPA.*

Gupta, R., Harrold, M. J., & Soffa, M. L. (1992). An approach to regression testing using slicing. *Software Maintenance, 1992.*

Harman, M. (2011). Making the case for MORTO: Multi objective regression test optimization. *In Proceedings of 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW*, 111–114.

Jones, J. a., & Harrold, M. J. (2001). Test-suite reduction and prioritization for modified condition/decision coverage. *In Proceedings of IEEE International Conference on Software Maintenance, ICSM*, 92–103.

Korkmaz, E. E., & Üçoluk, G. (2004). A controlled genetic programming approach for the deceptive domain. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *34*(4), 1730–1742.

Koza, J. R., & Stanford, K. E. (1990). *Genetic Programming: A Paradigm For Genetically Breeding Populations Of Computer Programs To Solve Problems*.

Krishnamoorthi, R., & Sahaaya Arul Mary, S. A. (2009). Factor Oriented Requirement Coverage Based System Test Case Prioritization of New and Regression Test Cases. *Informaton and Software Technology*, *51*, 799–808.

Lee, Z. J., Su, S. F., Chuang, C. C., & Liu, K. H. (2008). Genetic algorithm with ant colony optimization (GA-ACO) for multiple sequence alignment. *Applied Soft Computing Journal*, *8*(1), 55–78.

Li, Z., Harman, M., & Hierons, R. M. (2007). Search algorithms for regression test case prioritization. *IEEE Transactions on Software Engineering*, *33*(4), 225–237.

Lu, J., Fang, N., Shao, D., & Liu, C. (2007). An improved immune-genetic algorithm for the traveling salesman problem. *In Proceedings of International Conference on Natural Computation, ICNC*, 297–301.

Miller, G. F. . T. P. M. (n.d.). *Designing Neural Networks using Genetic Algorithms*.

Mirarab, S., Akhlaghi, S., & Tahvildari, L. (2012). Size-constrained regression test case selection using multicriteria optimization. *IEEE Transactions on Software Engineering*, *38*(4), 936–956.

Mirjalili, S. (2019). *Evolutionary Algorithms and Neural Networks*. Springer, Cham.

Mittal, S., & Sangwan, O. P. (2018). Prioritizing test cases for regression techniques using metaheuristic techniques. *Journal of Information and Optimization Sciences*, *39*(1), 39–51.

Özkan, R. (2017). *Multi-Objective Regression Test Selection in Practice: An Industrial Case Study*. METU.

Panichella, A., Oliveto, R., Di Penta, M., & De Lucia, A. (2015). Improving multi-objective test case selection by injecting diversity in genetic algorithms. *IEEE Transactions on Software Engineering*, *41*(4), 358–383.

Rothermel, G. (1996). *ERothermel, G. (1996). Efficient, Effective Regression Testing Using Safe Test Selection Techniques. Clemson University.fficient, Effective Regression Testing Using Safe Test Selection Techniques*.

Rothermel, G., & Harrold, M. J. (1997). A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology*, *6*(2), 173–210.

Samuel, W. S. Y. (2008). A Graph Based Evolutionary Algorithm. In *Thesis* (Issue July).

Srikanth, H., Hettiarachchi, C., & Do, H. (2016). Requirements Based Test Prioritization Using Risk Factors. *Informaton and Software Technology*, *69*, 71–83.

Syswerda, G. (1991). A Study of Reproduction in Generational and Steady-State Genetic Algorithms. *Foundations of Genetic Algorithms*, *1*, 94–101.

Toffolo, A., & Benini, E. (2003). Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary Computation*, *11*(2).

Toroidal graph. (2021). In *Wikipedia*.

Vokolos, F. I., & Frankl, P. G. (1998). Empirical Evaluation of the Textual Differencing Regression Testing Technique Outsource Laboratories. *In Proceedings of ACM Sigsoft International Conference on Software Testing and Analysis, ISSTA*, 44–53.

Whitley, D., Rana, S., & Heckendorn, R. B. (1999). The island model genetic algorithm: On separability, population size and convergence. In *Journal of Computing and Information Technology*, 7(1), 33–47.

Yadav, D. K., & Dutta, S. (2017). Regression test case prioritization technique using genetic algorithm. *Advances in Intelligent Systems and Computing*, *509*, 133–140.

Yamuç, A., Cingiz, M. Ö., Biricik, G., & Kalipsiz, O. (2017). Solving test suite reduction problem using greedy and genetic algorithms. *In Proceedings of International Conference on Electronics, Computers and Artificial Intelligence, ECAI*, 1–5.