Proposed Test Case Generation Model using Fuzzy Logic (TCGMFL)

AHMED ALTHUNIBAT[1], MUSTAFA MAHMOOD[1], HEND ALNUHAIT[2], SALLY ALMANASRA[2], HANEEN A. AL-KHAWAJA[3]
[1]Faculty of Science and Information Technology,
Al-Zaytoonah University of Jordan, Amman,
JORDAN

[2]Faculty of Computer Studies,
Arab Open University, Riyadh,
SAUDI ARABIA

[3]Faculty of Business,
Amman Arab University, Amman,
JORDAN

Abstract: - This research addresses the pressing need to enhance software testing, specifically focusing on white-box testing and basis path generation. Software testing is a linchpin in the software development process, ensuring software operates flawlessly and aligns with its intended objectives. However, this phase often incurs substantial time and resource investments. The primary aim of this study is to introduce an efficient and automated approach for basis path generation, a crucial component of white-box testing. The model commences by transforming source code into a tailored control flow graph (CFG), streamlining the automated generation of test paths. Central to this model is an algorithm for generating test paths (AGTP), meticulously traversing CFG nodes from source to destination. The algorithm's design aims to comprehensively cover all test paths within the CFG. To enhance testing process efficiency, the model employs k-means clustering to generate and cluster inputs. Path coverage is rigorously assessed for each cluster, and fuzzy logic is used to determine the optimal path. The overarching goals of this research are to reduce time and financial costs associated with software testing while maintaining precision and efficiency. The model's effectiveness in generating test cases is confirmed through the examination of multiple examples, underscoring its valuable contribution to software testing. This study marks a significant advancement toward more effective and cost-efficient software testing methodologies.

Key-Words: - Software Testing; Fuzzy logic (FL); Test Case Generation; k-mean clustering; Basis Path Testing; Automatic test case generation; Automatic Testing

Received: June 17, 2022. Revised: September 19, 2023. Accepted: October 21, 2023. Available online: November 22, 2023.

1 Introduction

Software testing (ST) is an important component in the software development process, and it is one of the essential stages in the software development life cycle (SDLC), [1]. The main goal of this process is to create efficient software that is free of bugs, faults, and failures by executing the software with good test cases. Choosing test cases is an important step in software testing, and successful testing is supported by good test selection to make testing more efficient. All features in the software that should be checked must be covered by the test cases in the software, [2].

The process of designing and validating a large number of test cases that involve software testing is considered a complicated process due to the effort, time, and cost involved, [3]. One of the issues faced by software testing is software accuracy, which necessitates a high number of test cases and their appropriateness, [4], [5], As previously stated, [6], this phase of the Software development life cycle (SDLC) is the most expensive since it necessitates a significant amount of time and effort. The tester requires a sufficient number of test cases to ensure that the testing is thorough. When a product has a complicated branching structure, the tester must

usually pick test cases manually. However, manually determining test cases becomes incredibly laborious, necessitating the use of automated test case creation, [7].

As a result, automation of Software testing approaches plays a critical role in the development of test cases to save time, effort, and money, as well as to provide more reliable results than manual testing methods that are prone to human error, particularly the generation of test data. The development of testing data, in particular, improves the efficiency of software testing as a whole. Rather than creating data from scratch to test, [8].

Structure-oriented test methods are extensively used and define test cases based on internal program structures. During software development, one of the most essential structure-oriented test methodologies is the path-based test. How to construct linear independent paths automatically and efficiently is a critical challenge in path testing in software engineering, [9].

Structural testing (sometimes known as white box testing) is a type of program testing that is commonly utilized. Many test criteria, such as statement and branch coverage, necessitate the discovery of a collection of executable paths that match certain criteria. We must discover appropriate values for the input variables to run each path. This issue is known as test data generation. If the path cannot be exercised with any set of input data, the path is called an infeasible path, [10].

2 Problem Statement

In the software development process, software testing is an essential phase. The test case design process is an important phase in software testing. Effective testing requires a good test case the test cases in the set must cover all features that should be tested, and the number of test cases should not be too large. the path coverage appears as the issue via time complexity due to the height number of paths which will increase the time in the testing process and the size of a test suite, view aspects use the concept of machine learning (ML) such as fuzzy logic regarding fuzzy logic (FL) the outcome more complexity versus other rules mechanism.

We developed an approach that automates the process of test case generation in software testing by proposing a technique that first generates all test paths from the corresponding control flow graph (CFG) automatically by developing an algorithm that takes the control flow graph as input and generates a set of the test path. the algorithm traverses each node from source to destination to find these paths, moreover, the technique generates test data in a given domain and group data into the cluster using k-mean which help in determining the basis path set by using a path coverage as a test adequacy criteria with the help of fuzzy logic optimal path has determined, [11].

This new approach will help to complete path testing and determine the optimal path to prioritize the process of test case selection. Also, the new approach will reduce the time and cost in the testing phase. The general objectives of this research are: to investigate the terms test case, and fuzzy logic in the software testing phase, to develop the model to reduce the time complexity and improve the testing process, and to evaluate the model using statistical analysis measures, [12].

3 Literature Review

Software testing is the process of running software against a set of test cases to find flaws. The various testing approaches are defined by the artifact that is used to generate Test cases. Functional – or black-box – testing gets its name from the fact that it tests the functionality of a system. Test cases are derived from a program's specification or description; structural – or white-box – testing is derived from the program's specification or description. Test cases are derived from implementations; fault-based testing is derived from implementations. Test cases are derived from fault models based on typical programming errors, while model-based testing is derived from fault models. Models of system specifications are used to create test cases, [13].

To determine whether a software system is right, one might examine every piece of the system's input domain and compare the result to the intended outcome, [14].

Aside from testing methodology and criteria, there are other parts to the testing process. For example, manual testing of programs is prohibitively expensive; as a result, software testing typically relies on tools to automate the generation, execution, and collection of test results when flaws are discovered during program testing, they must be located, and corrected, [15], [16].

Software Testing (ST) Estimating testing efforts, finding a competent test team, building Test cases, running the program with those Test cases, and reviewing the results produced by those executions are all part of the software testing process. According to studies, testing accounts for 40-50 percent of the cost of software development, [17], with the amount for testing important software being much higher, [14]. As a result, software

testing (ST) is defined as the process of validating and verifying a software product.

Software testing is the practice of putting a program through its paces with well-designed input data to find flaws, [18], [19].

Software testing techniques assist in the construction of better test cases as follows:

*Structural Testing: The major purpose of these tests is to trigger specific areas in the software, such as specific statements, program branches, or paths to be run. White Box testing is a sort of testing that is used to test the structure of code. The expected performance is determined using coverage metrics such as path coverage, branch coverage, and data-flow coverage, [20], [21].

*Functional Testing: is a sort of software testing that validates a software system's functional requirements/specifications. Black box testing is another name for it. Functional tests are used to evaluate each function of a software application by giving appropriate input and comparing the output to the functional requirements In other words, functional testing isn't concerned with the application's source code, [21], [22].

Computer programs have now infiltrated every part of life, enabling the manipulation of a wide range of complex applications. Many of these applications are large, complex, and life-threatening. As a result, highly dependable software, or software with a high level of reliability, is required. Software testing is an important and commonly used methodology, in addition to the many other techniques for boosting reliability, [23].

The major goal of this process is to generate efficient software free of bugs, faults, and failure, [1]. Despite the fact that there are several resources for assuring software quality through testing, the majority of software products are not sufficiently tested. Insufficient testing will result from a large number of unhandled failures, resulting in increased software development time and expenses. The software development lifecycle entails more than just developing code. Testing occurs after the process, but it is just as important. Testing uncovers defects and verifies software, allowing for bug fixes and modifications. The phrase "fuzzy logic" has been applied in two ways. In a strict sense, fuzzy logic is a logical system for reasoning under uncertainty that generalizes conventional two-valued logic. Fuzzy logic, in a wide sense, refers to any theories and methods that use fuzzy sets, or classes having unsharp bounds, [24].

[25], [26], proposed fuzzy set theory, which gave rise to the name fuzzy logic, [25], [26], [27], Fuzzy Logic (FL) is a multivalued logic that allows for the definition of intermediate values between conventional evaluations such as true/false, yes/no, high/low, and so on. To apply a more human-like way of thinking in computer programming, concepts like quite tall or very fast can be mathematically expressed and processed by computers, [26].

The truth value of variables in fuzzy logic can be any real number between 0 and 1, both inclusive. It's used to deal with the concept of partial truth, where the true value can be somewhere between 100 percent true and 100 percent false, [28]. The truth values of variables in Boolean logic, on the other hand, can only be the integer values 0 or 1. [27], established the theory of fuzzy sets, which gave rise to fuzzy logic. A fuzzy set assigns a degree of membership to elements of a universe, which is commonly a real number from the interval [0, 1] [0, 1].

By assigning degrees of truth to propositions, fuzzy logic emerges. The standard set of truth values (degrees) is [0,1] [0,1], with 00 being "completely false," 11 denoting "completely true," and the remaining digits denoting "partial truth," i.e., intermediate degrees of truth

Not only does fuzzy logic provide a meaningful and powerful representation for measuring uncertainties, but it also provides a meaningful representation of muddled concepts stated in everyday language. Fuzzy logic is a mathematical theory that deals with the idea of ambiguity in defining concepts and meanings. Expressions like "low" and "high," for example, include ambiguity or fuzziness since they are imprecise and relative. As a result, the variables analyzed are referred to as "fuzzy" rather than "crisp." Fuzziness is merely one way of expressing uncertainty; (FL) employs a non-binary logic (i.e., true or false),

Fuzzy set theory has been proved to be a valuable tool for describing scenarios involving imprecise or ambiguous data. Fuzzy sets deal with these problems by assigning a degree of belonging to a set to each object, [25], [26], [27]. The fuzzy set is a variation of the traditional set. The membership of items in classical crisp set theory follows a binary logic: either the element belongs to the crisp set or it does not.

While a fuzzy set does not have a crisp, clearly defined boundary, it can contain elements with degrees of membership ranging from 0 to 1, [29]. This is because a fuzzy set does not have a crisp, clearly defined boundary, and its fuzzy boundary is described by membership functions, which make the degree of membership of elements range from 0 to 1. [26], presented membership functions in the first study on fuzzy sets A membership function (MF) is

a curve that specifies the feature of a fuzzy set by assigning a membership value, or degree of membership, to each element. It converts each point in the input space into a membership value in the [0, 1] closed unit interval.

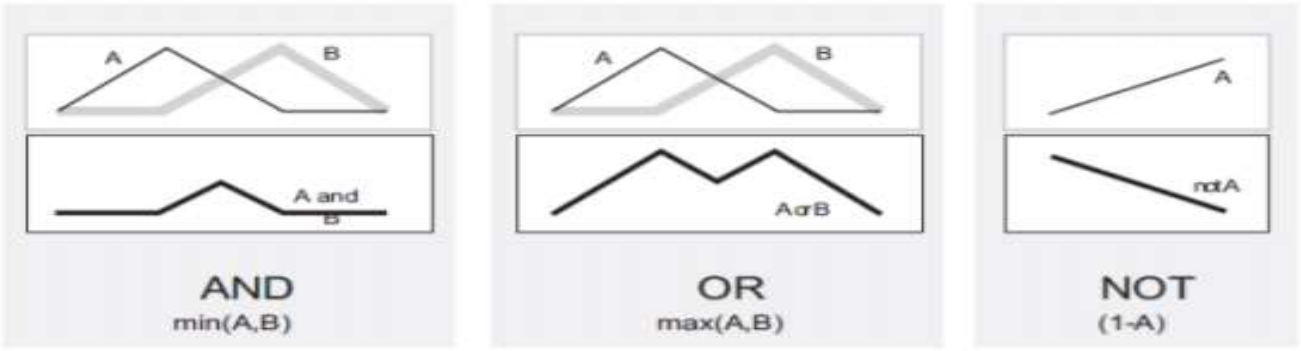

Fig. 1: Fuzzy logical operations

A general membership function curve is shown in Figure 1. The horizontal axis represents an input variable x, while the vertical axis shows the input variable x's corresponding membership value (x). The range in which the input variable will have a nonzero membership value is explained by the Support of Membership Function Curve. When x is any point between point a and point d in this diagram, (x) = 0. While the Core of Membership Function Curve interprets the range in which the input variable x has a full degree of membership ((x) = 1), in other words, any point within the interval [b, c] belongs to a fuzzy set described by this membership function.

Because standard binary logic is a specific example of fuzzy logic in which the membership values are always 1 (totally true) or 0 (entirely false), fuzzy logic must have the same consistency as standard logic. AND, OR, and NOT are the most fundamental logical operations. The operands A and B, unlike conventional logical operations, are membership values in the interval [0, 1]. The logical AND is expressed by the function min in fuzzy logical operations, therefore the phrase A AND B is equal to min (A, B). Because the function max defines logical OR, A OR B becomes identical to max (A, B). The logical NOT transforms operation NOT A into operation 1 – A.

A test case is a set of inputs, execution circumstances, and expected outputs that are designed to accomplish a specific goal, such as exercising a specific scenario, a specific scenario sequence, or validating compliance with a set of arguments, [30].

A test case, according to, [31], is a set of conditions that a tester must test to determine whether the System under Test (SUT) is satisfied with the expected outcome appropriately.

The problem of developing test cases that meet arbitrary test criteria is difficult, and many researchers have focused on automating this activity. For the automatic development of test cases, they used a variety of approaches, [32] Hand-written testing entails a significant amount of effort, a long testing period, and a large number of flaws. It is a prominent research topic to automatically generate testing scenarios.

According to, [33], identifying appropriate test cases is dependent not only on the faults found but also on how errors have been defined using test adequacy criteria to evaluate a test case. When all of the stated criteria have been met, the testing procedure comes to a close, [34].

A test adequacy criterion assists in identifying test objectives or goals to be accomplished during software testing. For example, path coverage requires that each test case be performed at least once for each path.

Path testing is a test-case design method and is a sort of method of white-box testing, [35]. During the software unit testing stage, it is commonly utilized. In fact, among all white-box testing methods, the path testing method has the best error detection capacity during the unit testing stage, [35]. Although the path testing method can detect most software errors during software testing, the number of program paths will increase if a program contains branch or loop statements, [36].

Control flow graphs, also known as program graphs, are commonly used to describe the control flow of programs in software analysis, [37]. Program statements are the nodes of a control flow

graph, and the edges reflect the control flow between the statements. Except for the entry and exit nodes, the control flow graph is a graph in which each node (apart from the entry and exit nodes) corresponds to one statement in the program code.

The measure of cyclomatic complexity refers to the number of fundamental cycles that appear in connected and undirected graphs, [38]. Figure 2 shows a section of the various pseudocodes and the control flow graph.

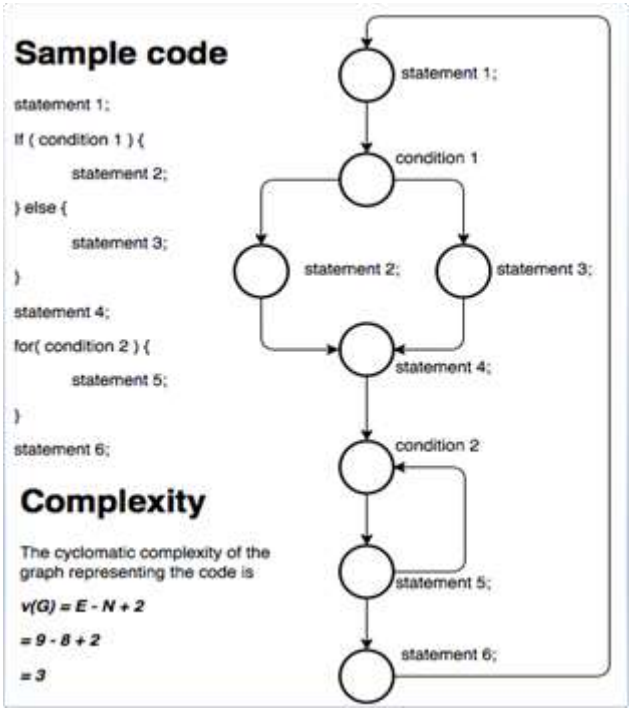

Fig. 2: Example of pseudo-codes and corresponding control flow graph.

According to, [39], Path coverage is one of the most accurate code coverage metrics because it focuses on a sequence of branch decisions rather than a single branch decision. Because even if a particular branch choice or statement has already been handled, the combinations with other branch decisions (or statements) may not have been tested, path coverage is better than statement and branch coverage. Unfortunately, due to the limitless number of paths, path coverage is difficult, if not impossible, to measure.

A path is essentially a collection of nodes connected by edges in a finite number of ways. A path that begins at the first node, travels via some nodes between the first and last nodes and ends at the last node, [40]. There are numerous obstacles to overcome.

1. **Test cases are incomplete:** It's frequently impractical to create test cases that cover all possible paths, especially for real-world applications. This is because I the total number of possible pathways in a program is enormous: at least exponential to the code size, and possibly infinite due to loops and recursive functions; and (ii) producing an input to cover any specific path is potentially incomputable, [39], Although symbolic unit test generation, [41], and dynamic test generation, [41], has made significant progress recently, they are still constrained by the above-mentioned unbounded computation complexity problem and have many difficult-to-handle cases, such as loops.
2. **Other states' dependency:** When the control path is also dependent on system states (such as the time of day), hardware states (such as disk condition), resource states (such as how much virtual memory has been allocated), and application states, the situation becomes much more difficult (e.g., the total number of outstanding requests). While fault injection-based testing can help with this issue to some extent, it is usually quite expensive and does not cover all conceivable state combinations, [7].
3. **Efficiency of Testing and Human Effort:** In real-world programs, enforcing each test case can take a significant amount of human labor and time. As a result, the number of examples that can be evaluated in the real world is restricted; it's crucial to reduce the number of test cases without reducing overall test coverage, [41], [42].

[6], explain how to automatically generate test cases using an evolutionary structural testing technique. The genetic algorithm (GA) is an evolutionary method that is used to automatically generate test cases to cover its def-use associations. The GA conducts its search by creating new test data using previously generated test data that has been determined to be effective. The technique generates a set of randomly generated test cases within a domain, which are then classified (clusters) using the K-Mean clustering algorithm. Each cluster identifies a collection of paths that have been covered, refines these clusters based on the paths that have been covered, and uses a genetic algorithm to generate new test cases in each cluster to increase path coverage. The Genetic Algorithm is used to create fresh inputs for path coverage. When compared to random generation, the Genetic Algorithm provided 100 percent path coverage.

[8], proposes an approach for discovering the error-prone path in software that combines Genetic Algorithms and Binary Search (BSGA). The BSGA

improves software testing by combining the Genetic Method (GA) with the Binary Search (BS) algorithm, which employs the BD as input values for program path coverage. The BSGA is a robust nonlinear search approach that provides a higher-quality answer, resulting in cost savings in the software testing industry. The results of the experiments show that using the BS to improve the performance of the GA in terms of finding appropriate test cases and test data for the input Big Data domain values has a positive influence. These results, on the other hand, reduce the cost of testing.
[2], proposes a genetic algorithm-based method for creating test cases. The paths of the graph are described by following path coverage criteria when a flow graph is built from the program source code. The genetic algorithm is then utilized to create test cases. Finally, we employ mutation testing to assess the test cases' effectiveness. The experiments show that the tests created are quite effective at detecting errors. In addition, based on the proposed strategy, the research has developed a test case generation tool.

By combining the evolutionary algorithm technique with the static technique for test case creation, it will be possible to eliminate all faults and increase the quality of software in the future.

Clustering is the division of a data set into subsets (clusters) so that the data in each subset share certain common qualities, such as proximity according to some defined distance metric. Clustering is vital in our lives since we live in a data-driven society where we encounter a tremendous volume of data.

Classifying or grouping these data into a set of categories or clusters is an important part of working with them. Clustering is used in a variety of fields. Data clustering, for example, is a common statistical data analysis technique utilized in a variety of domains such as machine learning, data mining, pattern recognition, image analysis, and bioinformatics. Clustering is also used to find relevant knowledge in data, [43].

Clustering is an unsupervised classification strategy that seeks to arrange things into clusters so that objects in the same cluster are extremely similar and objects in other clusters are substantially different. Cluster analysis is a well-known concept in the field of data mining, [44]. It's the first step toward an amazing new world of information.

Data is separated into different clusters in hard clustering, with each data element belonging to only one cluster.

3.1 The k-mean Clustering Algorithm

For prediction analysis, the k-mean clustering technique is used to group similar types of data. The likelihood of the most relevant function is estimated in the k-mean clustering procedure, and the functions are grouped using the Euclidian distance formula.

The fundamental k-means clustering algorithm, [45], is based on the partitioning method and is utilized for various clustering applications, notably with low-dimension datasets. It divides n objects into k clusters using k as a parameter so that objects in the same cluster are similar to one another but distinct from objects in other clusters.

3.2 The c-mean Clustering Algorithm

[46], introduced the Fuzzy C-Means clustering approach, which is an extension of the Hard C-Mean clustering method. FCM is an unsupervised clustering approach that can be used to solve a wide range of feature analysis, clustering, and classifier design challenges, [46].

Agricultural engineering, astronomy, chemistry, geology, image analysis, medical diagnostics, and form analysis and target recognition are just a few of the fields where it's used, [47], the FCM clustering algorithm, which is based on Ruspini Fuzzy clustering theory, was proposed in the 1980s with the growth of fuzzy theory. This algorithm is used to perform analysis based on the distance between different data points in the input. The clusters are constructed based on the distance between data points, and each cluster has its cluster center. The clustering algorithms used in test case generation are presented in Table 1.

Table 1. Clustering algorithms used in test case generation

Clustering algorithms	Research usage	Result	Authors
k-mean	Divide the test cases into clusters, each with a group of paths covered, and use a genetic algorithm to generate new test cases in each cluster to increase path coverage.	The experiment yielded a promising outcome.	[6]
K-mean	Eliminating redundant test cases to reduce the number of test suite	The experimental results show improved clustering accuracy and a large reduction in redundant test cases.	[48]
K-mean	Target Path Selection is accomplished by separating paths into groups using the K-means algorithm so that paths with a high degree of similarity are grouped. Then choose the cluster centers as targets, making sure that the target paths chosen have a greater degree of differentiation.	The experimental findings show that the proposed method is effective.	[49]
k-mean	Effective Test Cases are identified by separating test cases into two groups: effective and non-effective test cases.	The clustering-based test case classification can discover effective test cases with a high recall ratio and a significant accuracy percentage, according to an empirical study.	[50]
C-mean	Reduction in the number of tests required the main goal is to reduce the number of test cases and thus the amount of time spent testing.	The methodology's final test suite will produce good results in terms of conditions and path coverage.	[1]

4 Methodology

The Test Case Generation Model using Fuzzy Logic (TCGFL) is designed to automate and improve the test case generation process in the context of software testing. The primary objective is to reduce the time complexity and enhance the effectiveness of the software testing phase within the system development life cycle (SDLC). Here is a detailed description of how the TCGFL model operates:

1. Control Flow Graph (CFG) Construction: The process begins by automatically constructing a Control Flow Graph (CFG) from the source code. A CFG is a graphical representation of the program's control flow, illustrating how the program's execution moves through various branches and decisions.
2. Variable Extraction: The TCGFL model extracts variables from the source code. These variables are crucial for creating meaningful test cases that assess the behavior of the software with different input values.
3. Input Generation: Within a specified domain, the model generates inputs. These inputs are used as test data to evaluate the software. It's important to cover a broad range of input values to test various scenarios and uncover potential issues.
4. Basis Path Generation: The model proceeds to generate basis paths from the constructed control flow graph. Basis paths represent fundamental paths through the program, and they are essential for thorough testing.
5. K-Means Clustering: The TCGFL model employs a k-means clustering algorithm for the extracted variables. Clustering helps group similar variables together, making it easier to handle and assess the data. This step contributes to more organized and efficient testing.
6. Path Coverage Assessment: The clustered data is used to assess path coverage for each basis path. Path coverage is a critical metric in software testing, ensuring that all possible paths through the program are tested. The clustered data allows for a more systematic and comprehensive path coverage analysis.
7. Fuzzy Logic Optimization: Finally, fuzzy logic is used to determine the optimal path. Fuzzy logic is employed to make nuanced decisions and select the most appropriate path based on various criteria, improving the overall quality of test case selection.

The process and workflow of the TCGFL model are summarized in Figure 3, providing a visual representation of how the various components interact to automate the test case generation process. This research aims to make software testing more efficient, reduce testing time and costs, and ensure thorough testing, contributing to more effective software development within the SDLC.

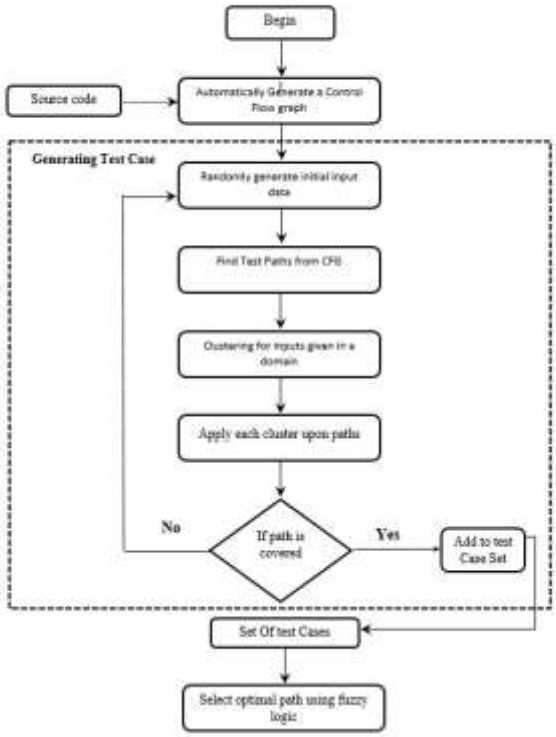

Fig. 3: Approach of Proposed Test Case Generation Model (TCGMFL)

Triangle problem, [48], [49], [50], [51], has been chosen as an ongoing case study as previous study and benchmark data, as well as a program to check Prime Numbers to demonstrate the application of our proposed methodology.

The triangle program accepts three variables as input and returns the type of triangle formed as an output on the other hand Prime program accepts one variable as input and returns if the input is a prime number or not to execute the program, we have used an integrated development environment (IDE) for python called visual studio code version 1.57.1 It is the most widely used IDE that provides a base workspace and can be easily customized due to its extensible plug-in system.

The analysis of a program under test (PUT) written in Python is the first step in the TCGFL model to understand the flow inside a program. Through transforming the code into a set of nodes and edges, The Control flow graph (CFGs) is a direct graph that usually focuses on a graph with a set of edges and a set of nodes, with each node focusing on a collection of sequential statements that form a primary block, and edge connections between the nodes. Because CFGs may precisely reflect the flow inside a program, they are commonly used in static analysis and compiler applications. Every node in CFG represents a statement in the program under test, allowing every path of statements to be sequenced. We introduced an algorithm that takes a list generated through a control flow graph (CFG) as input and a list containing different paths for a given program as output to find the possible test paths of the CFG starting from the first node and ending at the exit node.

With the help of cyclomatic complexity which is calculated by the algorithm, the number of independents is determined. A number of paths can be determined for any graph but even so, doesn't ease our work as there are several infeasible paths in the control flow graph. The cyclomatic complexity V(G) = E-N+2. Where N is the number of nodes and E is the number of edges, the algorithm traverses each node from source to destination to find all possible paths into corresponding CFG this algorithm is used for the Python programming language.

In this phase, the prepared datasets are imported into Python for clustering we employed the k-means clustering algorithm that was previously explained in section 2.2.1 Clustering Algorithms The algorithm was used because it is suitable for the scope of this work.

We employed k-mean on the datasets to group test data into the different clusters, clustering is applied on different data sets 1-dimensional space for the prime program and 3-dimensional space for the triangle program after all datasets are transformed into numbers, and the data can be grouped using the K-Mean Clustering method. Several procedures must be completed to group these data into different clusters. Test Case Generation for each path is accomplished through symbolically executing each path through a program the algorithm traverses a test path across the program's flow graph from the source node to the sink node, visiting each node along the path. When a node is visited, the algorithm symbolically executes the program's corresponding statement.
As inputs to the test case, the algorithm employs the clustered data created in the previous phase. The expected outputs along the execution path are presented in terms of the test case's inputs and the test case's symbolic inputs can be turned into a set of actual inputs by using such expressions to evaluate their intended results, [7].

The current study introduces a novel approach for test case generation in the context of software testing using the Test Case Generation Model using Fuzzy Logic (TCGFL). While the topic of software testing and test case generation has been explored in

existing literature, the current study distinguishes itself in several ways:

- Use of Fuzzy Logic: The primary differentiator is the incorporation of fuzzy logic into the test case generation process. Fuzzy logic is employed to make nuanced decisions, particularly in determining the optimal test path. This approach is not commonly found in conventional test case generation methods and offers a more sophisticated way of selecting the most suitable test cases.
- Application to Specific Programs: The study focuses on two specific programs, the Triangle Problem and a Prime Number checker, which serve as ongoing case studies. This targeted approach allows for practical and real-world application of the proposed methodology, making it more relevant and actionable.
- Integration of Clustering: The study utilizes k-means clustering to group test data, thereby enhancing the organization of test cases and making the process more efficient. Clustering is applied to different datasets for the two programs, demonstrating adaptability and customization.
- Comprehensive Path Coverage: The TCGFL model aims to cover all possible test paths within the control flow graph. It addresses the issue of infeasible paths and ensures thorough testing. This level of comprehensive path coverage is a notable feature of the proposed methodology.
- Python Programming Language: The study focuses on the Python programming language, and the algorithm and approach are tailored specifically for Python. This specificity allows for a more in-depth exploration of test case generation within this language, which may have unique characteristics and challenges.
- Detailed Algorithm Description: The study provides a detailed explanation of the algorithm used for generating test paths and symbolically executing test cases. This transparency in the methodology helps readers understand the process clearly.

In summary, the current study distinguishes itself from existing literature by incorporating fuzzy logic, applying the approach to specific programs, using clustering, addressing infeasible paths, focusing on the Python programming language, and providing a comprehensive description of the algorithm. These unique aspects make the TCGFL model a valuable contribution to the field of software testing and test case generation.

5 Conclusion

Many researchers have recently become interested in the problem of automating the software testing process. The majority of currently available tools do not cover all aspects of the testing phase. Furthermore, these techniques are not fully automated, implying that human work, whether from the customer or the developer, is required to complete the testing or design test cases. The approach used in this study was aimed to assist developers or testers in automatically generating test cases.

In this study, a control flow graph is automatically created and weighted each edge to use in optimal path determination then the source code converts into XML to extract the variables and generate random data in the given domain for thus variables and all possible paths are extracted through traversed each node in the flow graph from source to destination a clustering is done for the variables test data to check the feasibility and coverage of paths. Finally, the fuzzy logic will determine the optimal path.

Triangle and Prime programs are used as a case study to show how our test case generation tool (TCGFL) was implemented with a completely automated approach from the first phase to produce test cases and evaluate paths. Furthermore, we use an existing approach to analyze the model using metrics of "cyclomatic complexity." The results show that the proposed automated approach TCGFL is more efficient in terms of testing time, cost, and effort the findings indicate that the generated basis path from the suggested approach will be compatible with the complexity of the software that will represent the case of this study (prime and triangle programs) the cost and effort necessary for testing will be reduced and test cases are generated efficiently in addition, the prioritizing the selection of test case through determining the optimal path.

6 Future work and Limitations

Our methodology can be improved in several respects in future work, we aim to improve the model by providing a solution for generating new test data for the cluster not having 100 percent path coverage.

Furthermore, more complex programs are required for assessing our model and applying it to other programming languages such as Java, c#, and C++, as well as more research into test case generation optimization methods such as hybrid GA and PSO. Furthermore, the approach assumes that the program unit under test isn't extremely complex

and thus lacks control dependencies. However, if the program being tested has control dependencies, it won't be able to divulge them, therefore the test cases developed won't be able to completely test the program.

Acknowledgment:
The authors would like to thank the Arab Open University and Al-Zaytoonah University for providing the necessary scientific research supplies to implement the research

References:

[1] G. Kumar and P. Bhatia, "Software testing optimization through test suite reduction using fuzzy clustering," *CSI Transactions On ICT*, vol. 1, no. 3, pp. 253-260, 2013. doi: 10.1007/s40012-013-00233.

[2] N. Chuaychoo and S. Kansomkeat, "Path Coverage Test Case Generation Using Genetic Algorithms", [Online]. Journal.utem.edu.my (Accessed Date: March 15, 2021).

[3] T. A. Alrawashed, A. Almomani, A. Althunibat, and A. Tamimi, "An automated approach to generate test cases from use case description model," *Computer Modeling in Engineering & Sciences*, vol.119, no. 3, pp.409-425, 2019.

[4] I. Septian, R. S. Alianto, and F. L. Gaol, "Automated Test Case Generation from UML Activity Diagram and Sequence Diagram Using Depth First Search Algorithm," *Procedia Computer Science*, vol.116, no. 1, pp.629-637, 2021.

[5] A. Kalaee and V. Rafe, "An Optimal Solution for Test Case Generation Using ROBDD Graph and PSO Algorithm," *Quality and Reliability Engineering International*, vol.32, no. 7, pp.2263-2279, 2016.

[6] S. R. Khan, A. Nadeem, and A. Awais, "TestFilter: a statement coverage-based test case reduction technique," in *Multitopic Conference*, Dec 2006, pp.275-280, 2006.

[7] T. K. Wijayasiriwardhane, P. G. Wijayarathna and D. D. Karunarathna, "An automated tool to generate test cases for performing basis path testing," *2011 International Conference on Advances in ICT for Emerging Regions (ICTer)*, Colombo, Sri Lanka, 2011, pp.95-101, doi: 10.1109/ICTer.2011.6075032.

[8] A. Alhroob, W. Alzyadat, A. Imam, and G. Jaradat, "The Genetic Algorithm and Binary Search Technique in the Program Path Coverage for Improving Software Testing Using Big Data," *Intelligent Automation and Soft Computing*, 2020. doi: 10.31209/2020.100000168.

[9] L. Barqawi, "The Impact of Using Artificial Intelligence in Pharmaceutical Companies," *Al-Zaytoonah University of Jordan Journal for Legal studies*, vol.4, no. 1, pp.217-236, 2023. doi: 10.15849/ZUJJLS.230330.11.

[10] R. Omeish, "Messing with the Blockchain Technology to Commit a Crime," *Al-Zaytoonah University of Jordan Journal for Legal Studies*, vol.3, no. 1, pp.91-108, 2022. doi: 10.15849/ZUJJLS.220330.06.

[11] A. Alshehadeh, G. Elrefae, A. Belarbi, A. Qasim, and H. Al-Khawaja, "The impact of business intelligence tools on sustaining financial report quality in Jordanian commercial banks," *Uncertain Supply Chain Management*, vol.11, no. 4, pp.1667-1676, 2023.

[12] G. Y. Quba, H. Al Qaisi, A. Althunibat, and S. AlZu'bi, "Software requirements classification using machine learning algorithms," in *2021 International Conference on Information Technology (ICIT)*, pp.685-690, IEEE, July 2021.

[13] M. Soda, M. Makhlouf, Y. Oroud, A. Alshehadeh, R. Omari, and H. Al-Khawaja, "Does the audit quality have any moderating impact on the relationship between ownership structure and dividends? Evidence from Jordan," *Uncertain Supply Chain Management*, vol.11, no. 4, pp.1789-1800, 2023.

[14] Lemos O. A. L., Ferrari F. C., Eler M. M., Maldonado J. C., and Masiero P. C., "Evaluation studies of software testing research in Brazil and in the world: A survey of two premier software engineering conferences," The Journal of Systems and Software, vol. 86, pp. 951–969, 2012. [Online]. Available via: The Journal of Systems and Software. [Accessed 6 April 2020].

[15] H. Al-Shafei, "Computerization of Programs for Teaching Arabic to Non-Native Speakers: Android Applications as a Model," *Al-Zaytoonah University of Jordan Journal for Human and Social Studies*, vol. 3, special issue, pp.301-323, 2022. doi: 10.15849/ZJJHSS.220508.15.

[16] A. Kinani, "The Arabic Language in Social Networking Sites, Between the Arabic and

Latin Letters," *Al-Zaytoonah University of Jordan Journal for Human and Social Studies*, vol.4, no. 1, pp.89-106, 2023. doi: 10.15849/ZJJHSS.230330.05.

[17] Wang X., Jiang, Y. and Tian W. (2015). An Efficient Method for Automatic Generation of Linearly Independent Paths in White-box Testing. *International Journal of Engineering and Technology Innovation*, vol.5, pp.108-120.

[18] R. Mall, "Fundamentals of software engineering," 2nd ed. New Delhi: *Prentice-Hall of India Ltd*, 2008.

[19] B. Alsawareah, A. Althunibat, and B. Hawashin, "Classification of Arabic Software Requirements Using Machine Learning Techniques," in *2023 International Conference on Information Technology (ICIT)*, Amman, Jordan, 2023, pp. 631-636, doi: 10.1109/ICIT58056.2023.10225789.

[20] A. P. Mathur, "Foundations of Software Testing," *Addison-Wesley Professional*, USA, 2008.

[21] B. Beizer, "Software Testing Techniques," 2nd edn. *Van Nostrand Reinhold Co.*, New York, 1990.

[22] ISO/IEC/IEEE International Standard - Systems and software engineering. ISO/IEC/IEEE 24765:2010(E). Dec. 2010, pp.1–418.

[23] R. Amro, A. Althunibat, and B. Hawashin, "Arabic User Requirements Classification Using Machine Learning," in *2023 International Conference on Information Technology (ICIT)*, Amman, Jordan, 2023, pp. 483-488, doi: 10.1109/ICIT58056.2023.10225936.

[24] Kumar M., Lata Misra and Gyan Shekhar. "A Survey in Fuzzy Logic : An Introduction." (2015).

[25] L. A. Zadeh, "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes," 1973.

[26] L. A. Zadeh, "Making computers think like people," IEEE Spectrum, vol.8, pp.26-32, 1984.

[27] L. A. Zadeh, "Fuzzy sets," Information and Control, vol.8, no. 3, pp.338-353, 1965.

[28] V. Novák, I. Perfilieva, and J. Močkoř, "Mathematical principles of fuzzy logic," *Dordrecht: Kluwer Academic*, 1999.

[29] D. Dubois and H. Prade, "Fuzzy Sets and Systems," *Academic Press*, New York, 1988.

[30] R. V. Binder, "Testing Object-Oriented System Models, Patterns, and Tools," *NY: Addison Wesley*, 1999.

[31] R. Ibrahim, A. Bani Amin, S. Jamel, and J. Wahab, "EPiT: A Software Testing Tool for Generation of Test Cases Automatically," *International Journal of Engineering Trends and Technology*, vol.68, no. 7, pp.8-12, 2020.

[32] P. Wang, X. Hu, N. Qiu, and H. Yang, "White-Box Test Case Generation Based on Improved Genetic Algorithm," in *Recent Advances in Computer Science and Information Engineering*, 2012, pp.489-495.

[33] J. B. Goodenough and S. L. Gerhart, "Toward a theory of test data selection," in P*roceedings of the international conference on Reliable software*, 1975, pp.493–510.

[34] A. Andrews, R. France, S. Ghosh, and G. Craig, "Test adequacy criteria for UML design models," Software Testing, *Verification and Reliability*, vol.13, pp.95–127, 2003.

[35] B. Beiter, "Software system testing", *Van Nostrand Reinhold Company*, 1990.

[36] P. G. Frankl and E. J. Weyuker, "An Applicable Family of Data Flow Testing Criteria," *IEEE Transactions on Software Engineering*, vol.14, no. 10, pp.1483-1498, 1988.

[37] N. Vijay K and V. K. S. Kumar, "Automated Test Path Generation using Genetic Algorithm," *International Journal of Engineering Research*, vol.6, Issue 7, 2017.

[38] A. Watson and T. McCabe, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric," Special Publication, *National Institute of Standards and Technology*, August 1996.

[39] Roper and Marc, "Software Testing," McGraw-Hill, 1994.

[40] M. R. Girgis, "Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm," *Journal of Universal Computer Science*, vol.11, no. 6, pp.898-915, 2005.

[41] N. Gupta, A. Mathur, and M. Soffa, "Automated test data generation using an iterative relaxation method," *ACM SIGSOFT Software Engineering Notes*, vol.23, no. 6, pp. 231-244, 1998. doi: 10.1145/291252.288321.

[42] Y. Al-Kasabera, W. Alzyadat, A. Alhroob, S. Al Showarah, and A. Thunibat, "An Automated Approach to Validate Requirements Specification," *Compusoft*, vol.9, no. 2, pp.3578-3585, 2020.

[43] Shaheda Akthar and SkMd Rafi, "Improving the software architecture through fuzzy clustering technique," I*ndian Journal of Computer Science and Engineering*, vol.1, no. 1, pp.54-57, 2010.

[44] Daljit Kaur and Kiran Jyot, "Enhancement in the Performance of K-means Algorithm," *International Journal of Computer Science and Communication Engineering*, vol.2, no. 1, 2013.

[45] K. A. Abdul Nazeer and M. P. Sebastian, "Improving the Accuracy and Efficiency of the k-means Clustering Algorithm," in *Proceedings of the International Workshop on Computer Education*, July 1-3, 2009, London, U.K.

[46] J. C. Bezdek, "Pattern Recognition with Fuzzy Objective Function Algorithms," *New York: Plenum Press*, 1981.

[47] Y. Yong, Z. Chongxun, and L. Pan, "A Novel Fuzzy C-Means Clustering Algorithm for Image Thresholding," *Measurement Science Review*, vol.4, no. 1, 2004.

[48] A. Pandey and A. Malviya, "Enhancing test case reduction by k-means algorithm and elbow method," *International Journal of Computer Sciences and Engineering*, vol.6, no. 6, pp.299-303, 2018.

[49] Z. Yan, Q. Li, W. Xingya, C. Jingying, and L. Xuefei, "Automatic Software Testing Target Path Selection using K-Means Clustering Algorithm," *International Journal of Performability Engineering*, vol.15, no. 10, p.2667, 2019.

[50] Y. Pang, X. Xue, and A. Namin, "Identifying Effective Test Cases through K-Means Clustering for Enhancing Regression Testing,", *12th International Conference on Machine Learning and Applications*, 2013.

[51] L. Briand, Y. Labiche, and Z. Bawar, "Using Machine Learning to Refine Black-Box Test Specifications and Test Suites," *The Eighth International Conference on Quality Software*, 2008.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

- Ahmed Althunibat carried out the framework, Research Methodology, and Conceptualization.
- Mustafa Shuker Mahmood has implemented the Literature review, Previous Studies, Validation.
- Hend AlNuhait has organized and executed the Discussion, Conclusion, and Supervision.
- Sally Almanasra was responsible for the Data curation, Software, Results, and analysis.
- Haneen A. Al-Khawaja wrote the main Hypothesis, Writing - original draft, Visualization.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself The authors extend their appreciation to the Arab Open University for funding this work through AOU research fund No. (AOURG-2023-021)

Conflict of Interest

The authors have no conflicts of interest to declare.