A Metric for Measuring Test Input Generation Effectiveness of Test Generation Methods for Boolean Expressions

Deniz Kavzak Ufuktepe *, Ekincan Ufuktepe*, Tolga Ayav†
* Department of Electrical Engineering and Computer Science, University of Missouri-Columbia, MO 65211, USA
† Department of Computer Engineering, Izmir Institute of Technology Izmir, Turkey
Email: dkyb5@mail.missouri.edu, euh46@missouri.edu, tolgaayav@iyte.edu.tr

***Abstract*—The literature includes several methods to generate test inputs for Boolean expressions. The effectiveness of those methods needs to be analyzed by extensive comparisons. To this end, mutation analysis is often benefited by applying a distinctively selected set of mutants on each test generation method. Mutation analysis provides substantive information about the effectiveness of a test suite by indicating the percentage of killed mutants, which is a common metric. However, as we claim and show in this paper, this metric alone is not sufficient to demonstrate the effectiveness of the methods. For a test generation method, the amount of generated test inputs is also an important attribute to evaluate effectiveness. To the best of our knowledge, there is no metric that measures the effectiveness within a scale taking into account several attributes. In this study, we propose a new metric to measure the effectiveness of test input generation methods, which takes into account both the number of killed mutants and the number of test inputs. We demonstrate our new metric on three well-known test input generation methods for Boolean expressions.**

***Index Terms*—Test input generation, Boolean expressions, mutation analysis, effectiveness**

I. Introduction

There exist many test generation techniques and the amount of these techniques is still growing. Whenever a new method is introduced or a method needs to be selected for a specific goal, effectivenesses of the tests are brought into question. Generally, the decision criteria are based on the amount of generated test inputs, the number of killed mutants, or the number of mutants killed per test. These criteria, however, are not sufficient to represent the effectiveness of a test generation method and could be misleading. For instance, mutation analysis is a method that has been widely used to measure the success of a test generation method, by finding the percentage of the killed mutants by a given set of test inputs. Besides, the number of the test inputs generated by a method, *i.e.* test suite size for that Boolean expression is a valuable metric to show the success of that test generation method. Even though the mutation analysis results and test suite size are used to define a method's success separately, a new metric that combines both the mutant kill ratio and test suite size could be quite beneficial. To the best of authors' knowledge, there is no such metric proposed in the literature. The need for this new metric is due to the fact that standalone metrics can be misleading in certain cases. For instance, a 100% mutant kill ratio can be considered high success if only the test suite is minimal. For sufficiently large test suites, this ratio is not meaningful. Contrarily, if a method generates a small test suite (which makes it very fast and feasible) and the mutant kill ratio is low, then there is no success. Generally, the fraction of the number of killed mutants over the number of test inputs is used to measure the success of a test input set. However, the problem with this approach is that there is no difference between a test input set of one test input and a hundred test inputs, as long as each test input kills a mutant, i.e. both would have the value of 1.0.

978-1-6654-1070-0/21/$31.00 ©2021 IEEE

In this study, a new metric called "effectiveness" is proposed that ranges between a value 0-100%. The effectiveness metric is proposed to be used in order to measure the quality of a test input set that is generated by a test input generation method. The proposed metric combines both mutation kill success of generated test inputs and the number of test inputs a method generates. The number of test inputs is not used directly, the APFD value of the test inputs is used instead, which will give us the opportunity to distinguish the aforementioned cases.

II. Related Work

In the literature, there is no metric that measures the effectiveness of a test generation method. However, we believe that the concept of test minimization is highly related to test generation and the idea of test minimization can be adapted to test generation. The ideas of test minimization and test generation intersect at the point, where they want to achieve the same optimal amount of tests. Let us assume the maximum number of tests that could be generated for a given system is *n*, and the optimal amount of tests that could be generated or minimized is *m*. Based on these information, the difference between test generation and test minimization is that, test minimization tries to approach to *m* from $m \leq x \leq n$. On the other hand, any test generation tries to achieve *m* from zero. Therefore, in this section we provide related work that measure the effectiveness of test minimization methods.

One of the way to measure the effectiveness of test minimization was proposed by Wong et al. [1], where they have studied on 10 common Unix programs using randomly generated test suites. In order to reduce the size of the test suites,

they used a testing tool called ATAC, which was developed by Horgan and London [2]. First, they have created a large pool of test cases by using a random test data generation. After test suites were randomly generated, they have seeded artificial faults into the programs. They have categorized the artificial faults into four groups. The faults in Quartile-I can be detected by [0–25)% of the test cases from the original test suite; the percentage for Quartile-II, III and IV is [25–50)%, [50–75)% and [75–100]%, respectively. To measure the effectiveness of the minimization, they have used the formula given in Equation 1:

$$(1 - \frac{\#\,of\ test\ cases\ in\ the\ reduced\ test\ suite}{\#\,of\ test\ cases\ in\ the\ original\ test\ suite}) * 100\% \quad (1)$$

Then, the impact of minimized test suite is measured by calculating the reduction in fault detection effectiveness, which is given in Equation 2:

$$(1 - \frac{\#\,of\ faults\ detected\ by\ the\ reduced\ test\ suite}{\#\,of\ faults\ detected\ by\ the\ original\ test\ suite}) * 100\% \quad (2)$$

On the other hand, to measure the quality and effectiveness of test cases or a test suite one of the most popular approach is calculating the mutation score, which is proposed by Geist et al. [3]. There are two types of mutation score calculation. The first mutation score calculation does not include equivalent mutant information whereas the second one does. For a given program P and test suite T, to calculate the mutation score of test suite T, the number of mutants M, the number of killed mutants K and the number of equivalent mutants E are required. With respect to the given information the mutation score calculation without the equivalent mutant information is given in Equation 3, while the mutation score calculation with equivalent mutant information is given in Equation 4.

$$MS(P,T) = \frac{K}{M} \quad (3)$$

$$MSE(P,T) = \frac{K}{(M-E)} \quad (4)$$

In the context of test generation, the *Test Effectiveness* (TE) measure has been widely used [4]–[6], which is given in Equation 5. The *Test Effectiveness* measure calculates a positive real number. The smaller the real number is, the effective the tests are. However, the major burden of this measure assumes that equivalent mutants' identification requires approximately additional relative effort with test data generation. Therefore, Papadakis et al [7] proposed an alternative measure to *Test Effectiveness*, which is called *Cost Effectiveness* (CE). The calculation for Cost Effectiveness is given in Equation 6.

$$TE = \frac{number\ of\ test\ cases}{number\ of\ exposed\ faults} \quad (5)$$

$$CE = \frac{\#\,of\ test\ cases + \#\,of\ equivalent\ mutants}{\#\,of\ exposed\ faults} \quad (6)$$

III. Motivation

Mostly, the information of the number of killed mutants by a test input set and the number of test inputs are combined in the following way [3]:

$$success = \frac{\#killed\,mutants}{\#test\,inputs} \quad (7)$$

This ratio makes sense since it provides the number of killed mutants per test, which takes into account both important attributes. Therefore, it seems useful to measure the quality of a test input for a given test input set. However, it is not helpful in all circumstances. Even though a single test input kills many mutants and some of the test inputs kill only one mutant or none, this metric will give us an estimate of the quality of a single test input. The main problem with this metric is that it does not consider the total number of possible tests and the total number of mutants we are dealing with in the mutation analysis phase. In order to understand the possible consequences come with this negligence, assume that there are two different test generation methods, *A* and *B* that generates different amount of test inputs that kills different amount of mutants. Assume that test generation method *A* generated 1 test input -it may be a randomly generated test input- that kills one mutant out of 300 mutants. On the other hand, assume that test generation method *B* has generated 300 test inputs and kills all of the 300 mutants. If we look at the success measure, both will have the same success value of 1.0. However, these two test generation methods cannot be considered equivalent. For test generation method *A*, it has only generated a single test input that exactly kills one mutant, which can be considered as a success in terms of not generating redundant test inputs. The disadvantage of method *A* is that it does not kill all the introduced mutants. Nevertheless, test generation method *B* generated many test inputs, but that kills all of the introduced mutants is also a success. However, test generation method *B* might have generated similar (redundant) test cases, and could have killed all the introduced mutants with fewer test cases. For smaller systems, the disadvantages of *B* might be harmless, but in a relatively large and complex system, execution a large test suite can increase the test execution time drastically. For instance, in an industrial project [8] [9], running the entire test suite is said to take seven weeks.

To measure the effectiveness of the test generation method, we tackle the problem by treating it like f-measurement, which is also defined as a harmonic mean. The f-measurement or the harmonic mean is mostly used to combine two metrics; *precision* and *recall*, which are mostly used in information retrieval and machine learning in order to understand the relevancy. These two metrics both provide valuable, yet independent information. Therefore, these two metrics are combined into a single metric called f-measurement or harmonic mean. In this study, we use two valuable information; number of generated test inputs and number of killed mutants. These two attributes are crucial measurements in the context of test generation. For any test generation method, it is desired to generate few

test inputs. On the other hand, it is also important that the generated test inputs are able to expose all potential faults, in other words, to kill all the mutants.

IV. Proposed Metric

In this section, we define the criteria for the effectiveness measurement of tests. The effectiveness is defined over the two facts: i) To have a minimal amount of test inputs, ii) To kill all the mutants associated to the potential faults. Therefore, for any test generation method, it is essential to generate a minimal test suite that is able to reveal all potential faults. Such a test suite is called effective. This leads to a situation that the test generation technique generates such a small yet effective test inputs that each test input kills more than one mutants. On the other hand, small test suites would be beneficial for regression testing, in terms of reducing the execution time of the entire suite. For a given test generation technique, let;

- m_k = *number of killed mutants*.
- m_t = *number of total mutants*.
- t = *number of generated test inputs*.
- T = *maximum number of test inputs required to reveal all faults in a system. For a 5-input system, maximum number of tests would be* $2^5 = 32$.

We construct our metric combining two different measurements and it is scaled in the range between 0-1. The first input is the ratio of generated test inputs with respect to maximum number of tests and the second input is the mutant kill ratio.

We begin with the definition of the first input, Generated Test Inputs (GTI) that ranges in [0, 1]. When we divide the generated test inputs by the maximum number, small ratios closer to zero mean that the test suite is very small and effective in this manner. We define the most effective test suite, *i.e.* GTI=1.0, such that the test suite contains only one test. The GTI formula is given in Equation 8. Note that the maximum value of GTI is 1.0 whereas its minimum is $1/T$.

$$GTI = 1 - \frac{|t-1|}{T} \tag{8}$$

The second input for the effectiveness calculation is the Mutant Kill Success (MKS) that also ranges in [0, 1]. The MKS values closer to 1.0 mean that the test suite kills almost all mutants. Contrarily, smaller MKS values mean that the test suite is unable to kill the mutants, that is unsuccessful or ineffective. Equation 9 gives the MKS calculation.

$$MKS = \frac{m_k}{m_t} \tag{9}$$

Finally, based on harmonic mean, GTI and MKS metrics constitute the proposed *Test Generation Effectiveness Score* (TGES) metric, as shown in Equation 10.

$$TGES = \frac{2 \cdot GTI \cdot MKS}{GTI + MKS} * 100 \tag{10}$$

In the next section, we evaluate this formula on three well-known test case generation methods for Boolean expressions and compare their effectiveness.

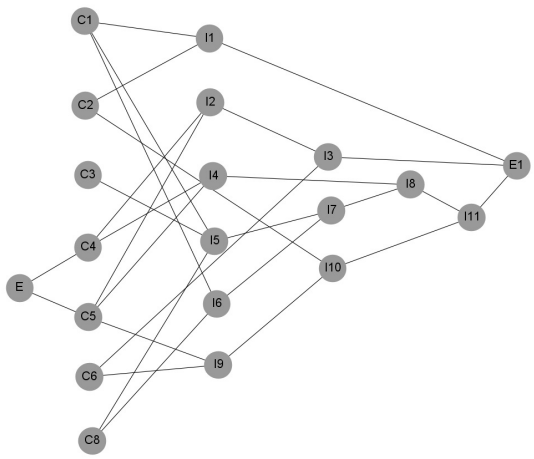

Fig. 1: Cause effect graph of selected requirement of TCAS-2.

V. Experimental Setup

In this section, we present the selected Boolean expression and its corresponding cause effect graph, the selected test case generation methods to evaluate the proposed metric, and finally the tool used to evaluate these methods.

A. Boolean Expression and Cause Effect Graph

TCAS-2 is an avionics system that is used to avoid collisions. The requirements of the TCAS system is often used in experimental studies particularly for testing critical systems [10]–[14], and these requirements can be found in [13]. For evaluations, we select one of the requirements of TCAS-2, given below:

$\overline{(ab)}(d\overline{e}\overline{f} + \overline{d}e\overline{f} + \overline{d}\overline{e}f)(ac(d+e)h + a(d+e)\overline{h} + b(e+f)))$

Cause effect graphs are a graphical presentation of Boolean expressions that can be also used in test input generation. The graph has three main node components; Cause, Intermediate and Effect nodes. The logical and, or, exclusive or and some other defined relationships between different cause nodes can be represented in the graph. From left to right: cause to effect, each component forms an intermediate node which will in the end form together the final effect node. After simplifying by taking similar components in the expression as single nodes, reusing nodes if possible and determining the exclusive or relationship. Then, we get the corresponding Cause effect graph in Figure 1, which is one specification out of a 20 requirements cause effect graph.

B. Selected Test Input Generation Methods

In the experiments, we used three well-known test input generation methods for Boolean expressions: Myers method, MUMCUT and Unique MC/DC.

By refining the cause effect testing proposed by Elmendorf [15], Myers proposed an efficient test input generation method from cause effect graphs of any Boolean expressions [16].

He defined a few rules that will be used to decide which test inputs should be selected from all possible test inputs for that specific Boolean expression. The method analyzes the cause effect graph from the effect nodes to cause nodes. The corresponding formula for each AND and OR relations with their each possible truth value is used to decide the final test input set. This method generates only the test inputs that the rules defined as necessary.

Chen et. al. proposed MUMCUT [17], which is a fault based approach to generate test inputs targeting specific fault types in Boolean expressions given in Disjunctive Normal Form. Disjunctive Normal Form contains components connected by OR logical operation. It is a combination of three sub-methods called Multiple Unique True Point (MUTP), Multiple Near False Point (MNFP) and Corresponding Unique True Point and Near False Point Pair (CUTPNFP). Each of these sub-methods target different types of faults, and the combination of them is used to get MUMCUT test input set. An example of the approach can be given as; for each literal in each component of the Boolean expression, each possible truth value with their effects on the outcome, both in original expression and in the expression where that literal is negated, is considered as different test inputs. Therefore, this method generates a high number of test inputs, with high success in fault detecting.

Unique Modified Condition/Decision Coverage (Unique MC/DC) is an efficient test input generation method for Boolean expressions [18], [19]. It is widely used and accepted as a standard that needs to be met in critical systems. Avionics, automotive and health related software should meet this standard recommended under ISO26262 and DO-178C. This approach considers each literal in each component and their effects on the output directly. It is done by changing the truth value of the selected literal while fixing the truth values of all the other literals. A test input with this characteristics is selected. Then, a second test input is selected with the same characteristics, where the only difference is the truth values of the selected literal and the outcome is the opposite of the first selected test input. These two test inputs are called an independence pair and these pairs are selected for each literal in each component. The aim is to generate test inputs for each possible truth value for each literal with their direct effects on the output, so that any fault related to that specific literal can be caught.

C. Tool

The tool proposed in [20], [21] is used in the experiments to generate test inputs by using the selected methods, and evaluate the mutation success of each. Test input sets are generated by Myers, MUMCUT and Unique MCDC methods and the corresponding mutation kill successes are computed. The mutation types generated and evaluated in the tool are: Operator Reference Fault (ORF), Expression Negation Fault (ENF), Variable Negation Fault (VNF), Missing Variable Fault (MVF), Variable Reference Fault (VRF), Clause Conjunction Fault (CCF), Clause Disjunction Fault (CDF), Stuck at-0 (SA0), Stuck at-1 (SA1). All possible mutants on each given fault type is created and the generated test inputs are run on these mutants to find the number of killed mutants.

VI. Evaluation of Test Generation Methods with TGES

In this section, we evaluate the test generation effectiveness for three test input generation methods; Myers [16], Unique MC/DC [18], [19] and MUMCUT [17]. We perform our evaluations by using *TGES* and *Test Effectiveness* (TE) on including all mutant types and for each mutant type individually. Thereby, we will be able to evaluate the test generation method by their overall and mutant type based effectiveness. In Section V-A, we describe the Boolean expression that we use the experiments. In order to calculate the TGES for each test generation method and to evaluate them we need two prior information; the maximum amount of test inputs that could be generated and the amount of total mutants that are introduced. The Boolean expression has 7 inputs, therefore, the total number of test inputs can be $2^7 = 128$ at most. As for the total number of mutants, we introduce 122 mutants. For calculating TE, we only need the information of total amount of generated tests and the number of exposed faults. Based on these information we can perform TGES and TE calculations. The calculation results and evaluations are given in Tables I and II.

In Table I, we have calculate the TGES results for each test generation method, and with respect to the TGES results, Unique MC/DC ranks first with the highest score 81.96 and MUMCUT ranks third with a 70.54 test generation effectiveness score (TGES). The TE results, also give the same ranking order with TGES. For Unique MC/DC the TE score is calculated as 0.12, 0.37 for Myers, and finally 0.50 for MUMCUT. In literature, Unique MC/DC is known to be an effective test generation method that generates few test inputs that are able to reveal many faults. Furthermore, we can see that Unique MC/DC has only generated 11 test test inputs over 128. On the other hand, MUMCUT is well known to have a higher mutant kill success, but generates too many test inputs as can be seen in Table I. Briefly, for the given Boolean expression, with our proposed test generation effectiveness metric, Unique MC/DC is found to be a more effective test generation method compared to Myers and MUMCUT.

We also perform an experiment using TGES and TE based on the mutant types we introduce, where the results are shown in Table II. We believe that this experiment might show which test generation method is more effective on which type of mutants with respect to the test inputs that are generated. Therefore, in Table II we have given a detailed information for each test generation method and mutant type that includes the number of Killed Mutants (KM) by the test generation method, Mutant Kill Success (MKS), Generated Test Inputs (GTI), TGES and TE. In Table II, one can see that with TGES, Unique MC/DC has a 95.94 effectiveness over mutant type ORF, with respect to the generated test inputs and MKS. On the other hand, with TE Unique MC/DC has a 0.78 effectiveness score. However, in terms of TGES, for mutant

TABLE I: Test Generation Effectiveness Score and Attributes for three Test Generation Methods based on including all mutant types

Test Generation Method	Number of Generated Test Inputs	GTI	Killed Mutants	MKS	TGES	Test Effectiveness (TE)
Unique MC/DC	11	0.9219	90	0.7377	81.96	0.12
Myers	34	0.7422	93	0.7541	74.81	0.37
MUMCUT	52	0.6016	104	0.8525	70.54	0.50

type SA1, MUMCUT has a higher effectiveness than Myers and Unique MC/DC. Although that MUMCUT has the highest effectiveness score for SA1 mutant type with TGES, according to the TE scores, MUMCUT has the lowest effectiveness score and Unique MC/DC has the highest effectiveness score. For mutant type SA0, both calculated with TGES and TE Unique MC/DC has generated effective test inputs than MUMCUT and Myers. On the other hand, we can see that all the test generation methods did not quite generated effective test inputs for mutant type CDF. Among three test generation methods, for mutant type CDF MUMCUT has generated the most effective test inputs, which has 56.54 effectiveness score. However, according to TE scores, MUMCUT has the lowest effectiveness and Unique MC/DC has the highest effectiveness score. For the remaining five mutant types CCF, VRF, ENF, MVF and VNF, with TGES and TE, Unique MC/DC has outperformed MUMCUT and Myers in terms of generating effective test inputs. With respect to the experimented Boolean expression and the TGES results in Table II, Unique MC/DC has generated effective inputs over seven mutant types (ORF, SA0, CCF, VRF, ENF, MVF and VNF), and MUMCUT has generated effective inputs over two mutant types (SA1 and CDF). However, the TE scores shows that Unique MC/DC has generated effective inputs over all of the mutant types. Based on the TGES and TE results that are specifically measured for each mutant type, we can have a brief idea about the rules of the test generation methods and effectiveness over specific mutant types.

VII. Discussion

We perform the experiments on a single Boolean expression given in Section V-A, and calculate the effectiveness of the generated test inputs with our proposed metric TGES and compare it with a widely used effectiveness measure TE (Test Effectiveness).

According to the outcomes given in Section VI, TGES provides general score within a given scale, while TE provides varying range of effectiveness score. On the other, TE does not provide whether the test generation method is really effective or not, but still can be used for comparing test generation methods. However, even in comparing test generation methods TE can be misleading, since that it does not include the surviving mutants. This could be seen, for the mutant type SA1, where Unique MC/DC has the least amount of test inputs generated, but low mutation score for SA1. In addition, we can see that MUMCUT has a higher mutation score for SA1, but higher test inputs generated. As for the final outcome based on mutant type SA1, MUMCUT is slightly more effective than Unique MC/DC. In Table II, another example can be seen for mutant types ORF and VNF. It can be seen that TE has generated the same test effectiveness scores. However, they do not share the same mutation score. Intuitively, they should not share the same effectiveness score, the test generation method with the highest mutation score must have a higher score. Table II shows that for ORF and VNF the effectiveness scores by TGES are different, and the mutant type with a higher mutation score has also a higher effectiveness score.

Furthermore, another advantage of TGES is that, if a test generation method fails to kill any of the existing mutant, the TGES score for the test generation method will be 0, which indicates that the test suite is entirely ineffective.

VIII. Threats to Validity

In this section, we discuss the limitations of our overall case study design, setting and the proposed metric. The metric we propose is more adequate for critical systems, where requirements are clearly defined. However, this metric may not be appropriate in software systems with complex inputs, such as objects types and free text content inputs. These inputs increase the complexity of calculating all possible test inputs. However, the complexity might be decreased by using domain partitioning and sampling.

Common metrics such as *Test Effectiveness* and *Mutation Score* are the best alternatives that take into account equivalent mutants under consideration. In our study, we do not construct TGES score based on equivalent mutants. Therefore, our experimental study does not dwell upon further investigation about which mutants are equivalent.

There are not enough real life examples of Boolean experiments or open source critical systems that can be used for evaluation purpose. Most of the studies that focus on test generation for Boolean expressions forge their own Boolean expression in their case study. TCAS-II seems to be a unique and common real example. The evaluations should be repeated with more real systems.

IX. Conclusion and Future Work

The existing metrics that measure the effectiveness of a test generation method, provides a useful insight about the quality of the generated tests. However, these metrics can be unreliable, when they exclude the surviving mutants or faults. In our motivation, we have given an example where two different test generation method can have the same effectiveness results even though they are obviously different. Therefore, in this study, we propose a new metric that measures the effectiveness of test generation methods, namely TGES. The metric provides an percentage score that ranges between 0-100. To calculate the effectiveness score, the metric requires

TABLE II: Mutant type based TGES and TE results

Mutant Type	Total Number of Mutants	Myers Test inputs = 34 GTI = 0.7422				Unique MC/DC Test inputs = 11 GTI = 0.9219				MUMCUT Test inputs = 52 GTI = 0.6016			
		KM	MKS	TGES	TE	KM	MKS	TGES	TE	KM	MKS	TGES	TE
ORF	14	14	1.0	**85.20**	2.42	14	1.0	**95.94**	0.78	14	1.0	**75.12**	3.71
SA1	15	7	0.4667	**57.31**	4.86	7	0.4667	**61.97**	1.57	10	0.6666	**63.24**	5.2
SA0	15	14	0.9333	**82.69**	2.42	12	0.8	**85.66**	0.91	14	0.9333	**73.16**	3.71
CDF	15	6	0.4	**51.98**	5.66	5	0.3333	**48.96**	2.2	8	0.5333	**56.54**	6.5
CCF	15	8	0.5333	**62.06**	4.25	8	0.5333	**67.57**	1.37	11	0.7333	**66.10**	4.72
VRF	15	12	0.8	**77.00**	2.83	12	0.8	**85.66**	0.91	15	1.0	**73.16**	3.46
ENF	11	11	1.0	**85.20**	3.09	11	1.0	**95.94**	1.0	11	1.0	**75.12**	4.72
MVF	7	7	1.0	**85.20**	4.85	7	1.0	**95.94**	1.57	7	1.0	**75.12**	7.42
VNF	15	14	0.9333	**82.69**	2.42	14	0.9333	**92.76**	0.78	14	0.9333	**75.12**	3.71

two types of information. The first required information is the ratio of generated test inputs divided by the total amount of test inputs that could be generated. The second required input is mutation kill success ratio. With these two information, we calculate a test generation effectiveness score.

We try our metric on a TCAS requirement and compare it with a common metric called *Test Effectiveness* (TE). Furthermore, we compared TGES and TE on three popular test generation methods over 9 types of mutants. We perform two evaluations with TGES and TE. The first evaluation was on the overall effectiveness of test generation methods. In the first evaluation, TGES and TE provide the same ranking among the three test generation methods. However, it is important to mention that TGES provides a general effectiveness of a test generation method, and does not require a comparison if a test generation method if effective or not. On the other hand, TE provides a positive real number, that can be used for comparison, but does not provide an output if the test generation method is effective based on a scale. The second evaluation was on measuring the test generation methods' effectiveness over mutation types. TE inferred the Unique MC/DC is effective on every mutant type. However, TGES inferred that Unique MC/DC is effective on seven out nine mutant types, and MUMCUT effective on the remaining two types of mutants.

There might be two possible extensions to this work in future. The first one is that, our metric does not take into account the test case generation time. For instance, MUMCUT is known to have a higher complexity and longer runtime compared to other test generation methods. Sometimes the time spent on generating a test suite can be considerable, which plays a factor in choosing a test generation method. Therefore, this time can also be included in the effectiveness. The second potential extension of our work could be taking equivalent mutants into account. Existing metrics such as mutation score and test effectiveness have alternative metrics that take equivalent mutants under consideration. Thereby, the TGES formula can be extended by including equivalent mutants.

REFERENCES

[1] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness," *Software: Practice and Experience*, vol. 28, no. 4, pp. 347–369, 1998.

[2] J. R. Horgan and S. London, "A data flow coverage testing tool for c," in *[1992] Proceedings of the Second Symposium on Assessment of Quality Software Development Tools*. IEEE, 1992, pp. 2–10.

[3] R. Geist, A. J. Offutt, and F. C. Harris Jr, "Estimation and enhancement of real-time software reliability through mutation analysis," *IEEE Transactions on Computers*, no. 5, pp. 550–558, 1992.

[4] E. J. Weyuker, "More experience with data flow testing," *IEEE transactions on software engineering*, vol. 19, no. 9, pp. 912–919, 1993.

[5] M. Papadakis and N. Malevris, "An empirical evaluation of the first and second order mutation testing strategies," in *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 2010, pp. 90–99.

[6] N. Li, U. Praphamontripong, and J. Offutt, "An experimental comparison of four unit test criteria: Mutation, edge-pair, all-uses and prime path coverage," in *2009 International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 2009, pp. 220–229.

[7] M. Papadakis and N. Malevris, "An effective path selection strategy for mutation testing," in *2009 16th Asia-Pacific Software Engineering Conference*. IEEE, 2009, pp. 422–429.

[8] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on software engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[9] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE transactions on software engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[10] M. F. Lau and Y. T. Yu, "An extended fault class hierarchy for specification-based testing," *ACM Transactions on Software Engineering and Methodology*, vol. 14, no. 3, pp. 247–276, 2005.

[11] T. Y. Chen, M. F. Lau, K. Y. Sim, and C. Sun, "On detecting faults for boolean expressions," *Software Quality Journal*, vol. 17, no. 3, pp. 245–261, 2009.

[12] U. Badhera, P. G.N., and S. Taruna, "Fault based techniques for testing boolean expressions : A survey," *International Journal of Computer Science & Engineering*, vol. 3, no. 1, pp. 81–90, 2011.

[13] E. Weyuker, T. Goradia, and A. Singh, "Automatically generating test data from a boolean specification," *IEEE Transactions on Software Engineering*, vol. 20, no. 5, pp. 353–363, 1994.

[14] A. Gargantini and G. Fraser, "Generating minimal fault detecting test suites for general boolean specifications," *Information and Software Technology*, vol. 53, no. 11, pp. 1263–1273, 2011.

[15] W. R. Elmendorf, "Cause-effect graphs in functional testing," [Poughkeepsie, N.Y.] : IBM, Tech. Rep., 1973.

[16] G. J. Myers, *The Art of Software Testing*, 1979.

[17] T. Chen, M. Lau, and Y. Yu, "Mumcut: A fault-based strategy for testing boolean specifications," in *Asia-Pacic Software Engineering Conference*, 1999, p. 606.

[18] K. Foster, "Sensitive test data for logic expressions," *ACM SIGSOFT software engineering notes*, vol. 9, no. 2, pp. 120–125, 1984.

[19] J. Chilenski and S. Miller, "Applicability of modified condition/decision coverage to software testing," *Software Engineering Journal*, vol. 9, no. 5, pp. 193–200, 1994.

[20] D. Kavzak Ufuktepe, T. Ayav, and F. Belli, "Test input generation from cause–effect graphs," *Software Quality Journal*, pp. 1–50, 2021.

[21] D. Kavzak Ufuktepe, "Test case generation from cause effect graphs," Master's thesis, Izmir Institute of Technology, 2016.