Mitigating the Effects of Equivalent Mutants with Mutant Classification Strategies

Mike Papadakis[1], Marcio Delamaro[2] and Yves Le Traon[3]

Abstract— **Mutation Testing has been shown to be a powerful technique in detecting software faults. Despite this advantage, in practice there is a need to deal with the equivalent mutants' problem. Automatically detecting equivalent mutants is an undecidable problem. Therefore, identifying equivalent mutants is cumbersome since it requires manual analysis, resulting in unbearable testing cost. To overcome this difficulty, researchers suggested the use of mutant classification, an approach that aims at isolating equivalent mutants automatically. From this perspective, the present paper establishes and empirically assesses possible mutant classification strategies. A conducted study reveals that mutant classification isolates equivalent mutants effectively when low quality test suites are used. However, it turns out that as the test suites evolve, the benefit of this practice is reduced. Thus, mutant classification is only fruitful in improving test suites of low quality and only up to a certain limit. To this end, empirical results show that the proposed strategies provide a cost-effective solution when they consider a small number of live mutants, i.e., 10-12. At this point they kill 92% of all the killable mutants.**

Keywords- Mutation Analysis, Mutants' Impact, Equivalent Mutants, Mutant Classification, Mutation Testing strategies

1. Introduction

Mutation testing aims at detecting software defects by injecting artificial errors, called *mutants*, in the examined software [5], [17]. By introducing a defect, two versions of the same program are produced: the *original* and the *mutant*. The technique relies on the underlying assumption that injecting and detecting mutants is already a strong adequacy criterion that forces tests to be effective for finding real faults. Mutants are introduced by making alterations to the source code of the program under test based on a set of simple syntactic rules called *mutant operators*.

Mutants are used to assess the ability of test cases to reveal them. Researchers have provided evidence that mutants, despite being artificially seeded, behave as realistic faults [3]. Thus, mutants can be effectively used as a testing criterion. Such a criterion can be established by requiring selected test cases to distinguish the behavior of the mutated and the original program versions. In practice, this requirement is fulfilled by comparing the programs' outputs when executed with the selected test cases. A mutant is said to be *killed* when it produces outputs that are different from the original program

[1] Interdisciplinary Center for Security, Reliability and Trust (SnT), University of Luxembourg, Michail.Papadakis@uni.lu
[2] Computer Systems Department, Universidade de Sao Paulo Sao Carlos, SP, Brazil, delamaro@icmc.usp.br
[3] Interdisciplinary Center for Security, Reliability and Trust (SnT), University of Luxembourg, Yves.LeTraon@uni.lu

while running the same test case. Otherwise, it is called *live*. In terms of fault revealing power, the higher the number of killed mutants, the higher is the test case power. However, while some mutants are killable, some mutants cannot be killed. In that case, such mutants are qualified as *equivalent* ones. As a result, an equivalent mutant forms a program version functionally equivalent to the original program, i.e., there exist no test case able to distinguish this mutant from the original program.

An equivalent mutant plays the role of a parasite in the testing process. Indeed, while it is expected to be killable, it remains always live. Even worse, a tedious effort could be uselessly dedicated to improving tests with no hope of killing it (in a way similar to covering infeasible statements or branches w.r.t. code coverage criteria [16]). As a consequence, mutation testing requires the removal of these mutants. However, discarding equivalent mutants tackles an even harder problem, since judging programs' functional equivalence is known to be undecidable [4]. As a result tedious manual analysis is required. It has been empirically found that it takes approximately 15 minutes [22] to identify an equivalent mutant in a real world application. Since a vast number of equivalent mutants exist, the application cost of mutation is escalated. As a consequence, the criterion used to stop the testing process requires the identification of most if not all of the existing equivalent mutants.

To decrease the undesirable effects of equivalent mutants, heuristic methods appear to be promising. Schuler and Zeller [21, 22] suggested a method to automatically classify mutants into the likely killable and the likely equivalent. The underlying idea of this approach is to measure the effects, called the *impact*, introduced by mutants on the runtime program execution. It has been found that mutants with an impact are more likely to be killable than those with no impact. Although mutant classification has been suggested [22], it is not evident how it could be applied in practice. Furthermore, it is not evident what the practical benefits of its utilization are. The present paper defines strategies that take advantage of the mutant classification. We call these strategies as *mutant classification strategies*. These turn mutants' impact as a guide towards improving a test suite.

The primary aim of the present work is to define and evaluate the relative effectiveness and efficiency of mutant classification strategies. In other words, we address the question of how the mutation testing process should be performed in order to take advantage of mutant classification. To this end, we seek to identify the practical benefits of these strategies compared to the "traditional" mutation testing approach, i.e., the usual way of applying mutation testing (without mutant classification); see Section 3.3 for a definition of the "traditional" and the mutant classification approaches.

The above intentions were investigated based on a set of industrial programs written in C and using the Proteum [6] mutation testing system. It has been found, that the examined strategies are cost-effective in improving the quality of a test suite. This improvement can lead to mutation scores of 92%-95% by examining up to twenty mutants per program. However, reaching higher mutation scores is not fruitful with the examined strategies. Therefore, in these cases the traditional approach should be undertaken. Despite this weakness, employing these strategies can lead to substantial benefits with respect to the equivalent mutants' identification in non-critical cases. Thus, mutant classification can provide a cost-effective solution to the equivalent mutants' problem only up to a certain limit. To this end, empirical results show that when examining only a small number of mutants, i.e., (10-12) of mutants, the

proposed strategies are beneficial and they kill more than 92% of all the killable mutants. Additionally, the obtained results show that the methods' effectiveness loss is a straight consequence of the accuracy of the proposed classification schemes; therefore, pointing out possible directions for future research.

The present paper extends the previous work on mutant classification strategies [24] in several dimensions. First, by considering a wider range of programs. Second, by studying and analyzing (manually) the equivalent mutants for all C-language mutant operators. Third, by studying the behavior of the strategies when random test cases are used as initial ones. Finally, a cost-effective analysis of the proposed strategies and a comparison with the traditional mutation approach is also performed.

The remainder of this paper is organized as follows: Sections 2 and 3 present the concepts underlying the present work and the mutant classification strategies. Section 4 presents and details the experimental setup. The empirical results are presented and analyzed in Section 5. Section 6 discusses the findings of the present study and considers some threats to the validity of the conducted study. Finally Sections 7 and 8 respectively present the related work and the conclusions of the paper.

2. Problems Caused by Equivalent Mutants

Using mutation as a testing criterion [17] requires a way of measuring the adequacy of testing. Generally, test adequacy can be measured based on the exact mutation score, which is defined as follows:

Exact MS = #killed mutants / (#Mutants-#Equivalent)

However, calculating the exact mutation score is hard due to the existence of the so-called equivalent mutants [16]. These mutants cannot be killed and hence, they must be somehow identified in order to compute the mutation score. To this end, practitioners have two choices: a) to manually analyze the live mutants, which in practice is very hard, if not practically infeasible and b) to approximate the exact mutation score. The later approach is based on the actual mutation score, which is defined as follows:

Actual MS = #killed mutants / #mutants

Using Actual MS has two main drawbacks. First, it requires from the tester to decide which value of the score is satisfactory for completing the process. Without a manual analysis, this choice is more or less arbitrary, resulting in a degraded confidence in the testing process. Second, it does not give any guidance to the tester as to which mutants should be targeted first, to increase the score. To address these difficulties, the present paper examines the use of mutants' impact [22], [21] as a possible way to automatically identifying the killable mutants. It also specifies the order of mutants that should be targeted. In other words, mutants' impact allows deciding whether live mutants can be killed or not. Therefore, the tester can concentrate only on these (likely killable) mutants according to the provided order. Additional details regarding the concept of mutants' impact are given in the following section.

3. Mutant classification and strategies

This section details the mutant classification strategies. It fist introduces the concept of mutants' impact in section 3.1, that underlies the examined approaches. Then section 3.2 describes how mutants' impact is measured and implemented in the conducted experiments. The section 3.3 specifies the examined strategies and section 3.4 demonstrates their application through an example. Finally, section 3.5 describes the problems addressed by the present paper.

3.1 The Mutants' Impact

Schuler and Zeller [22] advocated "if a mutant impacts internal program behavior, it is more likely to change external program behavior and thus, it impacts the semantics of the program". In other words, they proposed to identify the differences in the internal program behavior between the original and the mutant programs. These differences are attributed to the introduced mutants and referred to as the mutants' impact.

Generally, mutants with an impact are more likely to be killable than mutants without an impact [22]. Hence, mutants can be classified as likely killable, i.e., mutants with an impact, and likely equivalent, i.e., mutants without an impact. But, how can the mutants' impact be determined? In other words, what to compare between the two test executions, i.e., the original and mutant executions, to effectively consider the mutant as likely killable? This question has already been investigated in the literature [22], [21] by considering various impact measures. These works have found that among the various examined measures, the coverage impact [22] is the most appropriate measure for mutant classification. Therefore, the present paper considers only mutant classification based on coverage impact. Figure 1 gives an example of the coverage impact computation. In this example we consider a mutant on statement 3, i.e., the statement "b = S2(a);", of the original program is mutated to "`b = S2(a++);`". Mutants' impact is a dynamic measure and thus, it is computed based on some test cases. We consider three test cases in our example (named as Test1, Test2 and Test3). To compute the impact of mutants we execute the mutants with the available test cases and record their traces (record the execution frequency of each program statement per considered test case). In Figure 1 the trace information of the three example test cases is recorded in the subcolumns "Test1", "Test2" and "Test3" of the "Original Program" and "Mutant" columns for the original and mutant programs respectively. By comparing the execution traces of the original and the mutant programs we compute the impact. In other words the impact is the number of statements for which the original and the mutant program have different execution frequencies. We used two impact measures, the Whole Impact, which is the number of differences in the whole program, and the Non-Local Impact, which is the number of differences in the whole program except for the method that contains the examined mutant. In Figure 1 the columns "Whole Impact" and "Non-Local Impact" record the impact information for the three example test cases. More details on how we calculate the coverage impact is given in the following section.

Original Program					Mutant						Whole Impact			Non-Local Impact		
		Test1 (-1)	Test2 (1)	Test3 (5)		Test1 (-1)	Test2 (1)	Test3 (5)			Test1 (-1)	Test2 (1)	Test3 (5)	Test1 (-1)	Test2 (1)	Test3 (5)
1.	S1(int a){				S1(int a){											
2.	int b;	1	1	1	int b;	1	1	1								
3.	**b = S2(a);**	1	1	1	**b = S2(a++);**	1	1	1								
4.	if (b < 0)	1	1	1	if (b < 0)		1	1			*					
5.	b = -b;	1			b = -b;						*					
6.	printf("%d", b);	1	1	1	printf("%d", b);	1	1	1								
7.	}				}											
8.	int S2 (int x){	1	1	1	int S2 (int x){	1	1	1								
9.	switch (x){	1	1	1	switch (x){	1	1	1								
10.	case1:	1	1	1	case1:	1	1	1								
11.	return x*2;		1		return x*2;							*			*	
12.	case2:	1		1	case2:	1	1	1				*			*	
13.	return x;				return x;		1					*			*	
14.	case0:	1		1	case0:	1		1								
15.	return x+1				return x+1	1					*			*		
16.	}				}											
17.	return -1;	1		1	return -1;			1			*			*		
18.	}				}											
Output		1	2	1		1	2	1		Impact	4	3	0	2	3	0

Figure 1. Mutants' Impact, decide whether live mutants can be killed or not. The example mutant (line 3) is not killed by any test case and has an impact, i.e., the two program versions (original and mutant) have differences in their execution traces, when executed by the Test1 and Test2. The mutant has no impact, i.e., have the same program execution traces on the two program versions (original and mutant), when executed by Test3.

3.2 Mutant classification schemes

This paper considers mutant classification strategies based on coverage impact. Generally, impact on coverage is measured as the difference on the statement coverage between the original and mutant programs. Therefore, the coverage impact is a number that represents the maximum difference in covered statements between the original and the mutant program versions when executed with a set of tests.

In this paper, coverage impact is calculated as suggested in [22], by counting how many times every program statement is executed per test case. Thus, for each test, the execution frequency of each statement per program statement is computed. To this end, two variations of this metrics are used:

Approach A (Whole Impact, WI): The coverage impact on all the program statements.

Approach B (Non-Local Impact, NLI): The coverage impact on all the program statements except the ones belonging to the method of the examined mutant.

Generally, the NLI approach focuses on non-local impact while the WI approach focuses on both local and non-local impact. The NLI approach is based on the lines suggested in [22] and it is expected to give more accurate results than the WI. Using these metrics, the live mutants are classified in different categories: the likely equivalent, i.e., those with no impact, and the likely killable, i.e., those with an impact.

3.3 Mutant classification strategies

Applying mutation testing in practice requires the fulfillment of several steps. These steps form what we call the mutation testing process. Following the regular mutation testing steps results in the process presented in the Figure 2. The process starts from an initial test set called *initial test cases*, executes it with the mutants and computes the mutation score. Then a mutant from the live mutants is randomly selected and analyzed. If this mutant is killable a test case is generated and executed with all live mutants. If this mutant is equivalent it is discarded. This process iteratively continues until reaching a specific score threshold. We call this process as the "*traditional*" mutation approach.

Extending the traditional process we define a generic mutation testing process taking advantage of the mutant classifiers, referred to as a *mutant classification strategy* or simply as a *strategy*. This process is presented in Figure 3 and involves the regular mutation testing process' steps by introducing the mutant classification (step e). This step approximates the mutation score evaluation by considering only the likely killable set. Additionally, it ranks in a decreasing order all the mutants according to their impact values. This order is then used for analyzing the mutants with the highest impact [22] (step f.). In order to be practical, such a classification scheme must provide accurate estimations of the actually killable mutants during the whole testing process.

Towards defining possible mutation testing strategies that take advantage of mutant classifiers and thus aim only at likely killable mutants, two main issues arise. *First*, since mutant classification requires the existence of some test cases before the classification process, what should these tests be? *Second*, when should the classification process be performed? The first issue is presented in Figure 3 in step b) when no process iterations have been performed. The second issue is presented in the same figure in step e). Setting these two parameters are important for the effectiveness of the examined strategies.

Considering the initially used test set, the present study adopts two testing approaches. The first is statement coverage and the second is random testing. The reason for choosing statement coverage test sets is twofold. First, statement coverage forms the minimum testing requirement that should be employed by a tester. Second, achieving statement coverage ensures that all mutants will be executed by the tests, thus, enabling their classification [22]. Note that if a mutant is not executed by any test, it has no impact. Thus, it will be classified as equivalent, hindering the effectiveness of the studied approaches. The reason for choosing a randomly selected test sets is twofold. First, it conforms to the usual practice, which is to produce tests in an ad-hoc way, i.e., without the use of a testing criterion. Second, random tests fail to execute all mutants and thus, the strategies will classify these mutants incorrectly. However, by aiming at the mutants with an impact result in evolving the test suite, i.e., by adding new test cases. These tests might traverse different program parts and result in executing some

mutants not executed before. This fact causes some of these mutants to have an impact and eventually to be correctly classified as killable by the strategies. So the question that it is investigated here is whether the initially used test suite has a great effect on the strategies.

Generally, mutant classification relies on the ability of the employed tests to trigger the mutants' impact. Therefore, to reduce the sensitivity of the strategy to the employed test, mutant classification should be performed after the execution of each utilized test. Thus, at each iteration of the process (Figure 3), all live mutants are executed with all the employed test cases. This process relies on mutant classification schemes (WI and NLI) that are presented in section 3.2. Therefore, two possible strategies are considered and denoted as: approach A (WI) and approach B (NLI).

Mutant classification strategies provide the advantage of prioritizing the live mutants according to their impact. Thus, the mutants can be considered based on their likelihood of being killable and not at random as in the traditional approach. In other words, the tester can select and analyze more killable mutants when he uses the impact as guidance than when he selects them at random. Therefore, the tester loses less time in analyzing equivalent mutants. This is an important issue since analyzing equivalent mutants is a very costly, manual activity [21], [22].

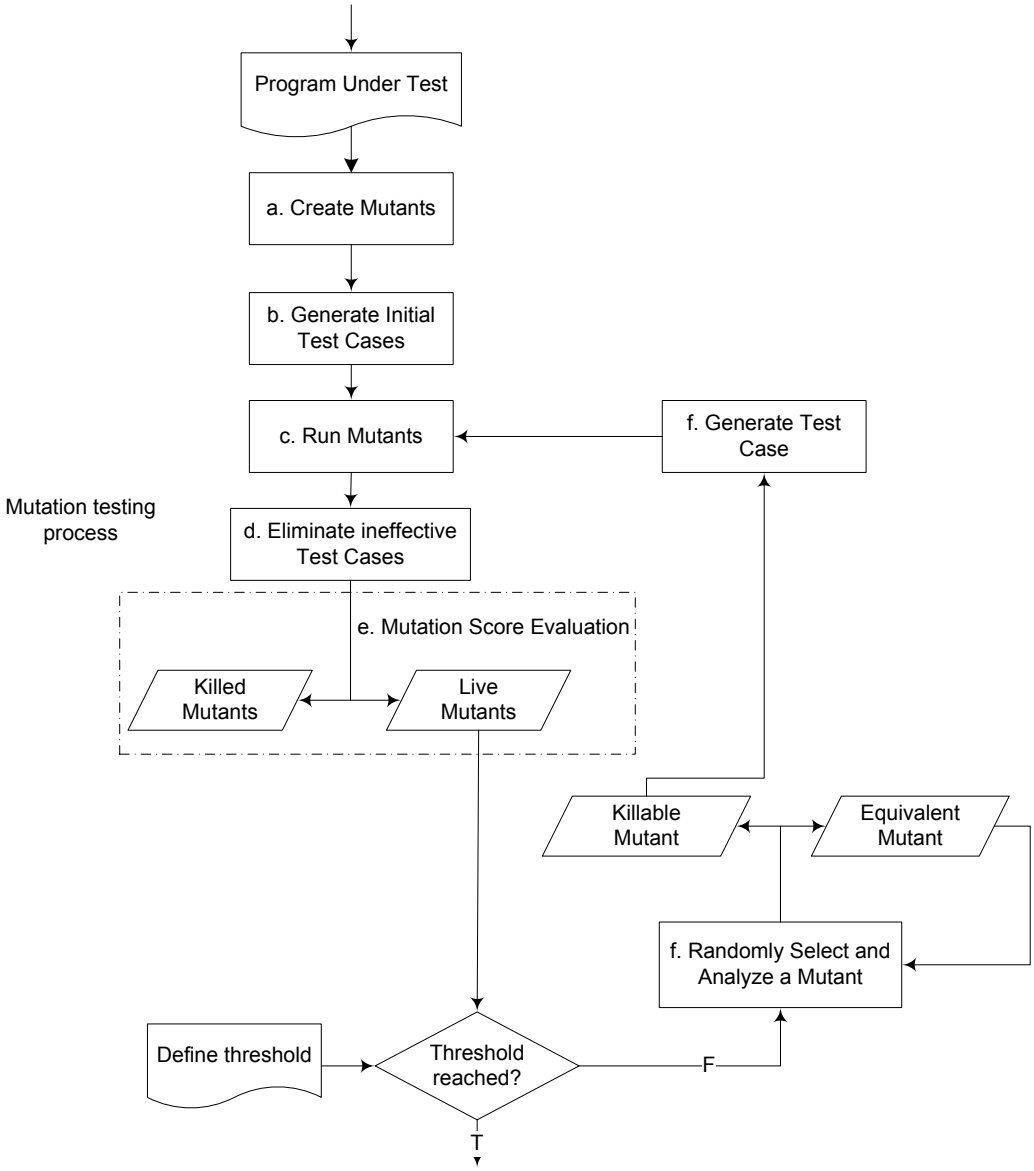


Figure 2. The "traditional" mutation testing process

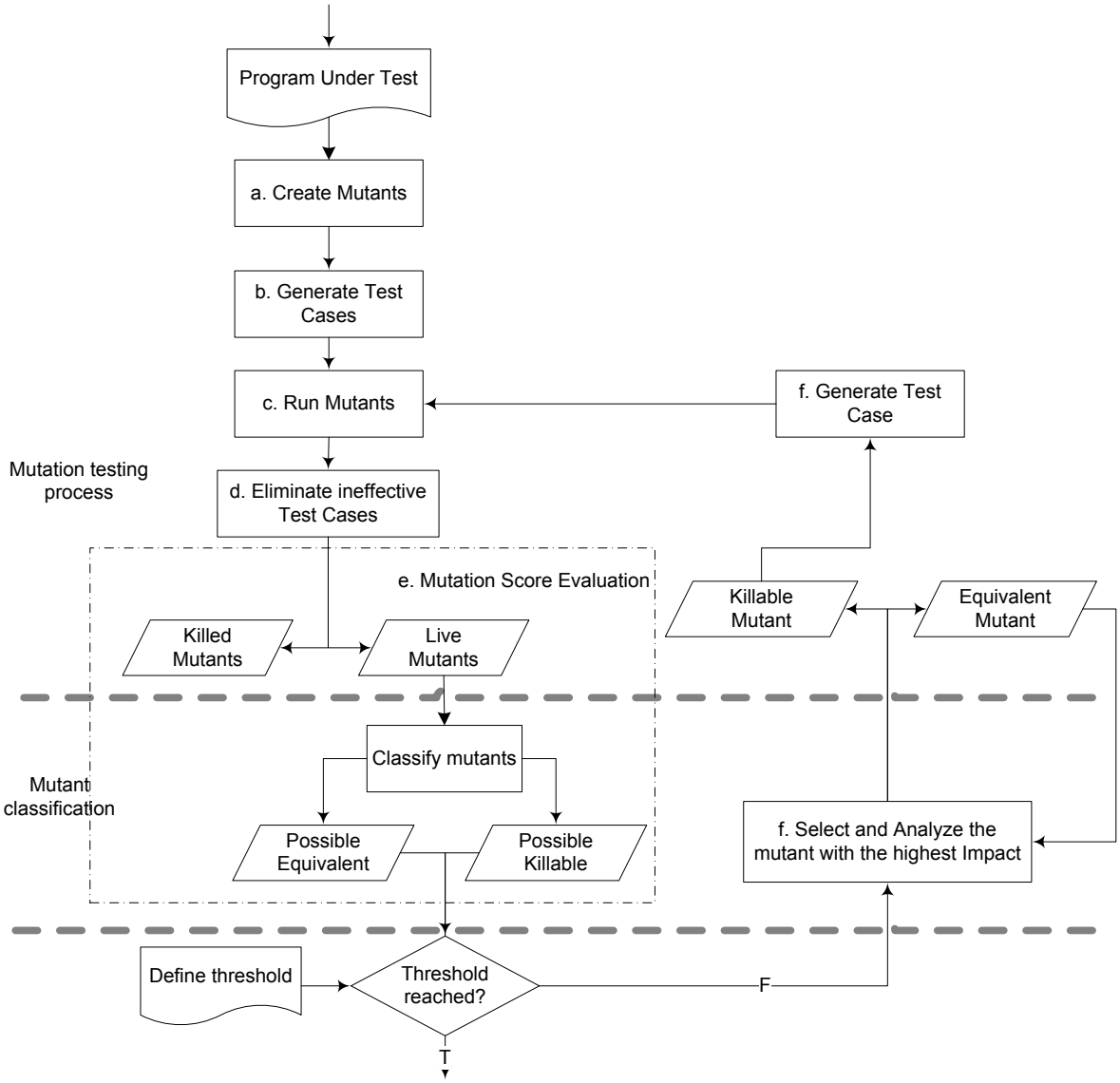


Figure 3. A generic mutant classification strategy

3.4 Example of the traditional and mutant classification approaches

To demonstrate how the "traditional" and mutant classification approaches work consider the following scenario. We have a program that has been tested by three test cases named as Test1, Test2 and Test3. These form our initial test cases. The purpose of using mutation is to improve the quality of the initial test set. Therefore, we will need to design additional test cases in order to kill all the mutants of the program under test. Consider that the program has ten mutants named as "M1, M2, M3, M4, M5, M6, M7, M8, M9, M10" and that Mutants "M5, M8, M9" are equivalent. Now consider that Test1 kills the mutants M1, M2, the Test2 kills the M2, M4 and that Test3 kills the M1, M3. Thus, by executing the initial test cases mutants M1, M2, M3 and M4 are killed and M5, M6, M7, M8, M9, M10 are live.

When applying the "traditional" approach the tester randomly selects one mutant from the live ones and analyzes it with the aim of designing a test case that kills it. For the purpose of our example, say that the tester selects mutant M8. This mutant is equivalent so the tester marks the mutant as

equivalent, i.e., removes it from the live mutants, and proceeds by selecting another one. It is noted that the effort put by the tester in analyzing the equivalence of the mutant was wasted since no test case was designed. In our example the probability of selecting an equivalent mutant is 50% since three out of the six live mutants, i.e., M5, M6, M7, M8, M9, M10, are equivalent. The process proceeds by randomly selecting another mutant, say M6. This is a killable mutant and thus the tester designs a test case, say Test4. Test 4 kills only mutant M6 and thus the live mutant set is composed of (M5, M7, M9, M10). The process continues with the tester selecting one mutant at random, say M9. This is equivalent so the tester selects mutant M7 and design Test5 that kills it. Then the tester selects another mutant at random, say M5, which is equivalent. Finally the tester selects the last mutant; say M10 and designs Test 6 which kills it. These steps are depicted in Figure 4

Applying mutant classification the tester has to determine the mutants with an impact. Lets say that the Test1 triggers an impact on M3 and M5 mutants, Test2 on M6 and Test3 on M7. Thus, after executing the initial tests only the M5, M6 and M7 mutants from the live mutants have an impact. Then, the tester selects the mutant with the highest impact, say M6 and analyzes it. So he designs Test 4, which kills the M6 mutant. It is noted that the probability of selecting a killable mutant is 66% since two out of the three mutants with an impact are killable. Then the tester analyzes the mutant with the highest impact from the remaining ones. Lets say that this mutant is M7 and that Test6 is design. Test6 kills the M7 and M10 mutants and does not trigger an impact on any of the live mutants. At this point since there is no live mutant with an impact the process stops. Figure 5 depicts the steps of the mutant classification approach as already described.

The advantages of mutant classification process are due to the fact that mutants with an impact are more likely to be killable than those without. So selecting a mutant from those with an impact results with a higher probability in a killable mutant than a live one.

			Killed Mutants	**Live Mutants**
1	Initial Test Cases	Test1	M1, M2	M3, M4, M5, M6, M7, M8, M9, M10
		Test2	M1, M2, M4	M3, M5, M6, M7, M8, M9, M10
		Test3	M1, M2, M3, M4	M5, M6, M7, M8, M9, M10
2	Select a live mutant at random (M8)	Equivalent Mutant	M1, M2, M3, M4	M5, M6, M7, M9, M10
3	Select a live mutant at random (M6)	Test4	M1, M2, M3, M4, M6	M5, M7, M9, M10
4	Select a live mutant at random (M9)	Equivalent Mutant	M1, M2, M3, M4, M6	M5, M7, M10
5	Select a live mutant at random (M7)	Test 5	M1, M2, M3, M4, M6, M7	M5, M10
6	Select a live mutant at random (M5)	Equivalent Mutant	M1, M2, M3, M4, M6, M7	M10
7	Select a live mutant at random (M10)	Test 6	M1, M2, M3, M4, M6, M7, M10	-

Figure 4. Example of applying the "traditional" Mutation testing approach

			Killed Mutants	Live Mutants	Mutants with an Impact
1	Initial Test Cases	Test1	M1, M2	M3, M4, M5, M6, M7, M8, M9, M10	M3, M5
		Test2	M1, M2, M4	M3, M5, M6, M7, M8, M9, M10	M3, M5, M6
		Test3	M1, M2, M3, M4	M5, M6, M7, M8, M9, M10	M5, M6, M7
2	Select the mutant with the highest impact (M6)	Test4	M1, M2, M3, M4, M6	M5, M7, M8, M9, M10	M5, M7
3	Select the mutant with the highest impact (M7)	Test 6	M1, M2, M3, M4, M6, M7, M10	M5, M8, M9	M5
4	Select the mutant with the highest impact (M5)	Equivalent Mutant	M1, M2, M3, M4, M6, M7, M10	M8, M9	-

Figure 5. Example of applying Mutation testing using mutant classification

3.5 Problem Definition

Generally, mutants' impact is a probabilistic approach that depends on the quality of the underlying test cases. Due to being probabilistic, it results in selecting some equivalent mutants and ignoring some killable mutants. In previous works [21], [22] and [14] this approach was evaluated by measuring the number of equivalent mutants that are selected and the number of mutants that are not selected and are killable. However, doing so has several drawbacks. *First*, since the approach is dependent on the initial test suite, these measures will be affected when different test cases are used. *Second*, the evolution of a test suite, by adding test cases, has the following two consequences: a) additional mutants are killed; b) additional mutants have an impact. Since the interest is on the live mutants, a) results in a set composed of less killable mutants than before. Thus, it is harder to categorize these mutants. b) results in identifying additional mutants as likely killable. Hence, the abovementioned measures change every time a test is added. *Third*, generating a test to kill one mutant results in collaterally killing some others. Therefore it is impossible to estimate how many mutants without an impact can be killed collaterally. Additionally, it is unclear whether killing a mutant with an impact results in collaterally killing approximately the same number of mutants than a randomly selected one. In other words, selecting a mutant at random might result in killing more mutants collaterally and thus, having a higher effectiveness.

From the above discussion it should be clear that the mutants' impact might fail to kill some mutants. But, how many they are? Additionally, mutants' impact provide an ordering to the aimed mutants by giving priority to those that are more likely to be killable. Mutants' impact also results on the one side to select less equivalent mutants but also, on the other side to kill fewer mutants. So, how cost-effective is this approach? To answer the first question we evaluate the achieved mutation score by the proposed strategies. This provides evidence about the strengths of such an approach. To answer the second question we compare the approach with the traditionally followed one in a cost-effective fashion. The number of the mutants requiring manual analysis with respect to each one of the approaches represents

the cost and the mutation score achieved represents the effectiveness. Thus, conclusions regarding the practical benefits of the approach can be drawn.

4. Empirical Study

The present study empirically investigates the use of mutant classifiers within the testing process. This is based on the strategies detailed in section 3.3. This section describes the undertaken experiment. It first defines the goals of the experiment in section 4.1. It then introduces the chosen benchmarks (section 4.2) and the employed tools. In section 4.4 the procedure followed during the study is detailed. Finally, section 4.5 gives information regarding the manual analysis performed during the experiment.

4.1 Definition of the Experiment

Generally, the present study seeks to investigate the relative effectiveness and efficiency of the mutant classification strategies. Additionally, issues regarding the classification ability of the underlying classification schemes are also considered. These issues are summarized based on the following research questions:

- **RQ1:** *How effective are the examined strategies? Comment: The effectiveness refers the mutation score achieved by the strategies compared to the traditional mutation approach.*
- **RQ2:** *How efficient are the examined strategies? Comment: The efficiency refers the number of the equivalent mutants encountered by the strategies and whole the number of mutants that must be manually analyzed by the strategies compared to the traditional mutation approach.*
- **RQ3:** *Which strategy is the most cost-effective? Comment: the relation between the number of mutants to be analyzed to the number of mutants killed by the strategies and the traditional mutation approaches is compared.*
- **RQ4:** *How the classification ability of the mutant classifiers is affected by the increase in the actual mutation score? Comment: how well the ability of the mutant classifiers performs when test suite evolves.*

Answering the abovementioned questions help practitioners in choosing an appropriate strategy to apply. We seek to identify the circumstances that make the examined strategies profitable. The aim is to investigate whether these strategies can form a cost-effective alternative solution to mutation. To this end, a comparison with the traditional mutation testing approach is undertaken. The focus of the experiment is the manual effort put by researchers when performing mutation. Thus, the cost comparison basis is the number of mutants requiring analysis by the examined strategies and the traditional approach. It is noted that this cost measure is composed of the killable mutants requiring analysis plus the equivalent ones.

Based on the following experiment and its respective analysis the main results regarding the above research questions are summarized in following points:

- ***(RQ1)*** The results suggest that on average a mutation score of 97.09% and 95.35% is respectively achieved by the WI and NLI strategies when statement based test cases are employed as the initial ones. In the case of random selected test cases, the WI and NLI strategies achieve a mutation score of 96.10% and 93.28% respectively.
- ***(RQ2)*** The results also suggest that a small number of equivalent mutants require analysis if the classification strategies are employed. Specifically, 14.56% and 6.09% of the whole number of equivalent mutants require analysis, if the WI and NLI strategies based on statement tests are used, respectively. In the case of random tests, the WI and NLI strategies require 13.88% and 5.87% respectively. The number of mutants requiring analysis is found to be 15.55% and 8.89% of the whole number of mutants that require analysis, if the WI and NLI strategies based on statement tests are respectively used. In the case of random tests, the WI and NLI strategies require 15.43% and 9.03% respectively.
- ***(RQ3)*** The cost-effective analysis suggests that WI is generally better than NLI. It is also more cost-effective than the traditional approach when analyzing a small number of mutants. However, why we observe this behavior? This is a matter investigated and explained by the RQ4. The results suggest that by analyzing up to 10-12 mutants WI is more-cost effective than the traditional approach. Beyond this point it loses ground and gradually becomes less cost-effective than the traditional approach. In particular, at that point, WI approximately achieves a mutation score of 95% while the traditional approach achieves 94%. This is for the case of statement tests. In the other case, random tests, WI scores at 93% while the traditional approach scores at 92%.
- ***(RQ4)*** The results suggest that classifying equivalent mutants based on coverage depends on the percentage of mutants that are killed by the utilized tests. Surprisingly, when a test suite evolves (based on mutation) the classification ability decreases. Hence, the guidance provided by the classification process also decreases. This fact explains why the mutant classification approaches fail to be cost-effective beyond a certain limit.

4.2 Subject Programs

The conducted study uses in total ten subjects, the seven programs of the Siemens suite [10] and two larger ones named Space and Flex. Table 1 records the details about them. It records the program names, column Subject Program, the programs' lines of code, column Lines of Code, and their associated test suite pool size. These programs were chosen because they were in C and are available from the SIR repository [7] along with their accompanied test suite pools. Additionally, they have been extensively used in empirical studies, involving mutation, such as [10], [19], [18], [8], [3], [11]. They can thus, be considered as benchmarks.

Each of these programs is associated to a comprehensive test pool. This provides the advantage that we can simulate well the test generation process by sampling the test pool. Regarding the test pool of the Siemens suite, it is noted that several researchers using a combination of techniques produced it. These techniques include random, category-partition, all statements, all edges and all definition-use pairs.

More details about the construction of these pools can be found in Harder et al. [8]. Regarding the test pool of the two larger programs (Space and Flex) it is noted that other researchers produced it using informal specifications and category partition techniques. Since the associated test suites were produced independently of the present study and consist of a large number of high quality tests (since they were developed according to many testing criteria) they are well suited for the conducted experiment.

Table 1. Subject programs

Subject Program	Lines of Code	Test Pool Size
Schedule	296	2650
Schedule2	263	2710
Tcas	137	1608
Totinfo	281	1052
Replace	513	5542
Printtokens	343	4130
Printtokens2	355	4115
Space	9,564	13,585
Flex	14,120	567

4.3 Mutation analysis and Utilized tools

The present study used a mutation analysis tool named Proteum [6]. This tool employs a comprehensive set of mutant operators for the C programing language as defined by the Agrawal et al. [2]. A widely used GNU structural coverage analysis tool named gcov was also employed. Proteum and gcov tools were selected since they have been successfully used in many software testing experiments like [15], [25]. To implement the examined approaches, Proteum was used to produce the mutants and gcov for gathering the required statement coverage information. A prototype was developed in order to compile mutants, execute them, analyze the execution traces, determine mutants' impact and implement the examined strategies.

Mutation analysis requires vast computational resources to produce and execute the sought mutants. Therefore, to complete the experiment with reasonable resources all the mutants were produced and analyzed only for the seven programs of the Siemens suite. For the larger programs (Space and Flex) two main restrictions were adopted. First, only the mutant operators belonging to the general class of "operators" [2] were considered. This class is composed of 44 mutant operators, which introduce discrepancies on the various source code operators uses. Similar restrictions have also been undertaken in [3] and [18]. Second, a 10% sampling of the mutants was made. This was forced by the fact that these programs involve a huge number of mutants, e.g., 22,500 mutants, are introduced for the space program. The selection of these mutants was performed based on their production order, i.e., every 10th produced mutant was considered. The same approach was also applied on similar studies such as [3] and [18]. It is noted that in the present paper, we focus on examining the effectiveness of the

studied strategies with respect to a set of mutants. Thus, the use of the above restrictions affects only the initial set mutants and not their impact assessment.

4.4 Experimental procedure

To address the stated RQs, we report the results derived from the application of the proposed mutant classification strategies. Two experiments were conducted, one using the programs of the Siemens suite and one using the two larger programs, i.e., Space and Flex.

First experiment: the present experiment uses two approaches to select the initial test sets. These are the statement-based and the random. To this end, ten test sets were independently selected for the two approaches, i.e., five for the statement-based and five for the random. To construct sets of test cases adequate with respect to statement testing per subject program, gcov was employed. These sets were constructed based on a random test case selection from the accompanied test suite pools. All the selected tests that do not increase the sets' coverage with respect to their selection order were removed from the sets. This was done in order to discard redundant test cases that might coincidentally kill mutants and influence the impact measures. The five random test sets were created by randomly selecting a number of tests equal to the average of the five statement-based selected tests.

The experimental process: The experiment follows the mutation strategies described in Section 3 and presented in Figures 2 and 3.

Regarding the *mutant classification strategies*, after executing all mutants with the initial tests, live mutants are classified either as "likely killable" or "likely equivalent" ones according to the two classification approaches. Then, the first mutant in the resulting ranked list of likely killable mutants (mutant with the highest impact) is selected. If the mutant is equivalent, it is removed from the "likely killable" list and the process continues with the next mutant. In the opposite case, a test case able to kill the selected mutant is chosen at random from the test suite pool. Then, the process continues by executing this test case with all live mutants and removing those that are killed by the newly selected test. The mutant classification process is repeated until none of the live mutants have an impact (the likely killable mutants' set becomes empty).

Regarding the *traditional mutation approach* after executing all mutants with the initial tests, a mutant from the live ones is randomly selected. If this mutant is equivalent, it is discarded and the process continues with the next mutant. In the opposite case, a test case able to kill the selected mutant is chosen at random from the test suite pool. Then, the process continues by executing this test case with all live mutants and removing those that are killed by the newly selected test. The mutation testing process is repeated until all live mutants have been killed or discarded.

To avoid any bias due to (1) the initially selected test cases and (2) the random selection test cases that kill the selected mutants, five independent repetitions of the experiment were performed.

To assess the classification ability of the employed mutant classifiers two measures were used:

- **Recall:** the ratio of the correctly classified mutants as killable to all the existing killable mutants.

- **Precision:** the ratio of the correctly classified mutants as killable to those classified as killable.

These measures evaluate the ability of the classifier to categorize mutants as killable [21], [22]. To compute these values, manual analysis is required in order to determine all the killable mutants. Therefore, equivalent mutants were identified based on manual analysis. A mutant is identified as being equivalent based on the programs' output. In this study we used Proteum, which by default considers as program output everything that it is printed on the standard and error output of the tested program. Proteum also consider timeout (for infinity looping mutants) and abnormal termination of mutants. Based on this analysis, killable mutants that are not killed by the associated test suite were also identified. For these mutants test cases were manually constructed in order to kill them. These tests were added to the associated test suite pools in order to ensure that the examined strategies can select tests to kill these mutants.

Second experiment: Performing complete analysis on the larger programs is practically infeasible since it requires manual analysis of hundreds of thousands of mutants. Although conclusions about the application of the examined strategies can be drawn based on the first experiment, we seek to check whether they hold on larger programs. If a similar trend of the results is observed, the confidence on the validity of the findings of the present study is increased. To this end, we investigate the cost-effectiveness trend of the examined strategies compared to the traditional when non-adequate test pools are used. Thus, the mutants that were left live after being executed with the whole test suite pool were treated as being equivalents. This practice fulfills the purpose of the experiment, which is to examine whether a similar trend with the effectiveness results of the first experiment can be recorded. Effectiveness refers to the number of mutants killed by the utilized strategies. Thus, our interest is on whether testing based on these strategies results in killing approximately the same number of mutants than testing by using all mutants. Since the same mutants and test pools are used in both cases (testing based on the examined strategies and by using the all mutants), the killable mutants who have been left live after their exercise with the whole test suite pool are common in both cases. Finally, it is noted that the same practice has been undertaken in many similar empirical studies such as [3], [15] and [18]. For this experiment the initially used test cases were constructed by randomly selecting five test cases from the test pools. This experiment was also repeated independently five times.

To address RQ1, for each subject program we computed the mutation scores achieved by each strategy, when no mutants with an impact remain. At the same point, to answer the RQ2, the ratio of the mutants that were analyzed is also recorded. Regarding the RQ3, for each strategy we compute the average mutation score achieved after analyzing every mutant. The number of mutants requiring analysis represents the cost of the approach while the mutation score expresses the effectiveness measure of the process. Since the number of the equivalent mutants per program and strategy vary, to enable a fair comparison, all the examined approaches were stopped at the same point, i.e., the point that one strategy had no more mutants to analyze. Finally, for RQ4, the respective recall and precision values were calculated every time the classification process was performed, (see Figure 3).

4.4 Mutation Analysis on the Siemens suite

The present study aims at studying the ability of the proposed strategies at reducing the side effects caused by the equivalent mutants. Therefore, for the conducted experiment there is a need to detect all the equivalent mutants. This was manually performed for all the employed mutants of the Siemens suite. Table 2 records the total number of the introduced mutants (col. Number of Mutants), the number of the identified equivalent mutants (col. Number of Equivalent Mutants), the ratio of the equivalent mutants to the total number of mutants (col. Equivalent Mutants % of all Mutants) and the number of the manually generated test cases (col. Manually Generated Test Cases) per subject program.

To identify the equivalent mutants, all the test cases of the provided suite were executed in order to determine the killed mutants. These mutants are certainly killable and so there is no need to analyze them. The remaining (live) mutants were manually analyzed using the Proteum tool. The tool presents the original and the mutated programs so the tester can decide and mark a mutant as being equivalent or design a test case to kill it. In the later case, a new test was added to the test set and the live mutants were executed with it. Proteum also allows the tester to create the source code for the mutant so the tester can execute the mutant with some input data. This practice helped some cases in deciding whether some mutants are equivalent and to choose appropriate test cases to kill them. As an extreme measure, sometimes debugging the mutant may also help to understand its behavior and to decide if it should be marked as equivalent or not. There is no real control about this process and this is a potential threat to the validity of the study (see section 6.1. for a discussion about the potential threats to the validity of the present study). It is possible that non-equivalent mutants have been marked as equivalent; however, these mutants should be few. This is due, to the fact that a high quality test suite was used and the mutants where manually analyzed and additional test cases were designed. Additionally, the data of the present experiment (description of the mutants, the list of the identified equivalent ones and the list of the designed test cases) were made publically available in the following link in order to enable the replication of the present study:

https://sites.google.com/site/mikepapadakis/home/tools-and-data-sets/scp2014-data

Overall, the performed analysis included 33,462 mutants in total out of which 5,589 were found to be equivalent. This suggests that on average the16.7% of the introduced mutants are equivalent. However, the number of them is program dependent and ranges from approximately 9% to 20%. We also measure the number of equivalent mutants per mutant operator. Figure 6 presents the number of introduced mutants and the number of the equivalent mutants per mutant operator. From this graph it can be observed that the equivalent mutants are distributed in all mutant operators. None of the operators produces significantly more equivalent mutants than the others. Therefore, it is impossible to alleviate this problem by just removing some operators.

Table 2. Subject programs

Subject Program	Number of Mutants	Number of Equivalent Mutants	Equivalent Mutants % of all Mutants	Manually Generated Test Cases
Schedule	2106	338	16.05%	8
Schedule2	2590	526	20.31%	8
Tcas	2872	515	17.93%	12
Totinfo	6405	572	8.93%	17
Replace	10683	2122	19.86%	14
Printtokens	4249	725	17.06%	9
Printtokens2	4557	791	17.36%	4
Sum	**33,462**	**5,589**	**16.70%**	**72**

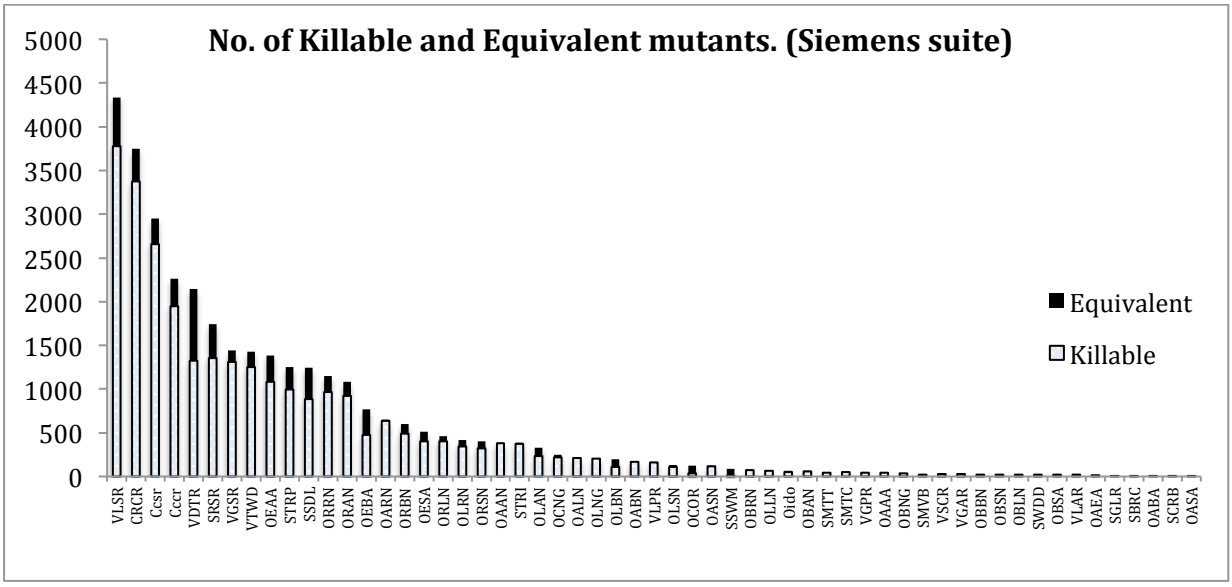


Figure 6. Number of mutants and equivalent mutants per mutant operator for the Siemens program suite.

5. Empirical Evaluation

The results of the conducted experiment are reported and analyzed in this section. The section first presents the results of the first experiment. These are analyzed according to the RQ1-RQ4 in the sections 5.1-5.4. Then in section 5.5, the results of the second experiment are presented.

5.1 Approach Effectiveness (RQ1)

This section considers the relative effectiveness of the examined strategies. The results are summarized in tables 3 and 4. These tables record the number of tests, the actual mutation scores and their variation according the examined methods, i.e., "Statement", "Random" and "Strategy", per classification scheme (Approach A (WI) and approach B (NLI)) per subject program. Table 3 refers to the

experiment conducted by using the statement test cases as an initial test set. Table 4 refers to the results when randomly selected tests are used. From these results, it can be realized that a high variation on the effectiveness of the examined approaches between the subject programs is recorded. Generally, the use of the WI and NLI schemes achieve mutations score from 92% to 99%. Comparing the results of the tables 3 and 4, it can be seen that the initial test suites do not play an important role to the effectiveness of the approach. This is a surprising result, since the proposed strategies aim only at mutants with an impact and the initial test suites differ significantly in the number of mutants that they kill. Additionally, since the initial test suite affects less the WI, it can be stated that WI forms a more stable scheme than NLI.

Another interesting result is that in almost all the cases WI scores better than NLI. However, in none of the studied subject a mutation score was very close to 100%. Achieving a score close to 100% is highly desirable in some cases. Thus, the examined strategies are good for improving a test suite but only up to a certain limit. Here, it should be noted that generally, the effectiveness is lower than 100% due to mutants with no impact.

Table 3. Effectiveness results based on Statement test cases

Subject Program	**Method**	**#Tests**	**Exact MS**	**Std. Deviation**	**#Tests**	**Exact MS**	**Std. Deviation**
		Approach A (WI)			**Approach B (NLI)**		
Tcas	Statement	7.2	66.71%	2.13%	7.2	66.71%	2.13%
	Strategy	27.8	92.91%	1.33%	27.4	92.86%	1.57%
Schedule	Statement	2.6	85.63%	2.42%	2.6	85.63%	2.42%
	Strategy	10.2	93.82%	1.07%	10.4	93.31%	1.09%
Schedule2	Statement	7.4	84.53%	2.04%	7.4	84.53%	2.04%
	Strategy	22.8	97.17%	0.57%	20	92.47%	1.16%
Totinfo	Statement	6.8	89.07%	2.73%	6.8	89.07%	2.73%
	Strategy	21.8	99.18%	0.42%	16.8	94.60%	1.07%
Replace	Statement	16.6	89.86%	1.98%	16.6	89.86%	1.98%
	Strategy	69.4	99.21%	0.16%	59.8	98.51%	0.31%
Printtokens	Statement	15.8	96.04%	0.42%	15.8	96.04%	0.42%
	Strategy	22.2	98.09%	0.50%	18.6	96.71%	0.63%
Printtokens2	Statement	11.1	97.98%	0.42%	11.2	97.98%	0.42%
	Strategy	21	99.27%	0.62%	14.6	99.00%	0.43%
Average	**Statement**	**9.64**	**87.12%**	**10.28%**	**9.66**	**87.12%**	**10.28%**
	Strategy	**27.88**	**97.09%**	**2.67%**	**23.94**	**95.35%**	**2.72%**

Table 4. Effectiveness results based on Radom test cases

Subject Program	Method	#Tests	Exact MS	Std. Deviation	#Tests	Exact MS	Std. Deviation
		Approach A (WI)			Approach B (NLI)		
Tcas	Random	7	50.03%	10.36%	7	50.03%	10.36%
	Strategy	29	91.34%	2.90%	25.6	90.96%	1.68%
Schedule	Random	3	77.85%	9.86%	3	77.85%	9.86%
	Strategy	9.4	91.86%	0.51%	10.6	92.22%	0.38%
Schedule2	Random	7	80.42%	6.47%	7	80.42%	6.47%
	Strategy	23.4	95.30%	0.80%	17.4	88.61%	3.75%
Totinfo	Random	7	78.44%	9.95%	7	78.44%	9.95%
	Strategy	21.8	99.22%	0.37%	17.8	94.54%	0.63%
Replace	Random	17	80.42%	6.47%	17	80.42%	6.47%
	Strategy	78.6	98.87%	0.93%	65.2	98.04%	0.77%
Printtokens	Random	16	72.80%	8.60%	16	72.80%	8.60%
	Strategy	25.4	96.49%	1.04%	20.8	90.78%	4.69%
Printtokens2	Random	11	88.71%	2.71%	11	88.71%	2.71%
	Strategy	21.8	99.61%	0.29%	16.2	97.82%	0.75%
Average	**Random**	**9.71**	**75.52%**	**12.20%**	**9.71**	**75.52%**	**12.20%**
	Strategy	**29.91**	**96.10%**	**3.44%**	**24.8**	**93.28%**	**3.64%**

6.2. Approach Efficiency (RQ2)

The efficiency of the examined approaches is greatly influenced by the accuracy of the employed classification schemes. This is due to the guidance that the classification scheme provides towards killable mutants. Therefore, in general, using a classification scheme with higher accuracy results in encountering less equivalent mutants. To this end, the obtained results are presented in Tables 5 and 6 for the two used initial test suites (statement and random) respectively. These tables record the percentage of the mutants that require manual analysis when following the classification schemes WI and NLI compared to the traditional mutation approach per program. The respective variation of these percentages is also given. Intuitive, NLI is expected to be more efficient than WI since it targets non-local impact [22]. Indeed, the NLI classification scheme encounters on average 8.89% and 9.03% mutants, while, WI encounters on average 15.55% and 15.43% when statement and random initial test sets are used respectively.

To provide further details about the cost of the classification schemes we record in Tables 7 and 8 the percentages of the encountered equivalent mutants by the examined classification schemes for the two used initial test suites (statement and random) respectively. Thus, NLI encounters on average 6.09% and 5.87% equivalent mutants, while, WI encounters on average 14.56% and 13.88% when statement and random initial test sets are used respectively.

Table 5. Efficiency results, measured on the ratio of mutants requiring manual analysis when applying the Whole Impact (WI) and Non-Local Impact (NLI) approaches compare to the traditional approach, based on Statement test cases

Subject Program	% of Manually Analyzed Mutants	Std. Deviation	% of Manually Analyzed Mutants	Std. Deviation
	Approach A (WI)		Approach B (NLI)	
Tcas	35.70%	0.64%	35.35%	0.42%
Schedule	5.17%	0.32%	5.22%	0.50%
Schedule2	10.89%	0.65%	6.93%	0.84%
Totinfo	20.88%	7.05%	4.01%	2.25%
Replace	18.04%	0.15%	5.48%	0.34%
Printtokens	8.12%	0.12%	1.18%	0.11%
Printtokens2	10.02%	0.28%	4.03%	0.30%
Average	15.55%	1.32%	8.89%	0.68%

Table 6. Efficiency results, measured on the ratio of mutants requiring manual analysis when applying the Whole Impact (WI) and Non-Local Impact (NLI) approaches compare to the traditional approach, based on Random test cases

Subject Program	% of Manually Analyzed Mutants	Std. Deviation	% of Manually Analyzed Mutants	Std. Deviation
	Approach A (WI)		Approach B (NLI)	
Tcas	38.16%	1.07%	37.04%	0.68%
Schedule	5.03%	0.26%	5.38%	0.33%
Schedule2	10.37%	0.50%	4.89%	2.26%
Totinfo	17.28%	1.59%	4.82%	2.91%
Replace	18.11%	1.07%	5.68%	0.33%
Printtokens	8.79%	0.23%	1.21%	0.62%
Printtokens2	10.28%	0.16%	4.17%	0.06%
Average	15.43%	0.70%	9.03%	1.03%

Table 7. Ratio of the encountered equivalent mutants when applying the Whole Impact (WI) and Non-Local Impact (NLI) approaches based on Statement test cases

Subject Program	% of Equivalent Mutants	Std. Deviation	% of Equivalent Mutants	Std. Deviation
	Approach A (WI)		Approach B (NLI)	
Tcas	35.38%	0.29%	35.07%	0.32%
Schedule	3.25%	0.00%	3.25%	0.00%
Schedule2	8.48%	0.68%	5.10%	0.78%
Totinfo	19.02%	7.06%	1.92%	2.22%
Replace	16.13%	0.04%	3.62%	0.11%
Printtokens	7.45%	0.00%	0.83%	0.00%
Printtokens2	8.98%	0.00%	3.54%	0.00%
Average	14.56%	14.24%	6.09%	7.88%

Table 8. Ratio of the encountered equivalent mutants when applying the Whole Impact (WI) and Non-Local Impact (NLI) approaches based on Random test cases

Subject Program	% of Equivalent Mutants	Std. Deviation	% of Equivalent Mutants	Std. Deviation
	Approach A (WI)		Approach B (NLI)	
Tcas	35.34%	0.55%	34.80%	0.37%
Schedule	3.25%	0.00%	3.25%	0.00%
Schedule2	7.45%	0.09%	3.00%	1.20%
Totinfo	15.21%	1.46%	2.73%	2.53%
Replace	15.60%	0.98%	3.50%	0.11%
Printtokens	7.56%	0.12%	0.55%	0.38%
Printtokens2	8.98%	0.00%	3.54%	0.00%
Average	13.88%	13.90%	5.87%	7.87%

5.3. Cost Effective application (RQ3)

The results presented in the previous sections study the cost and effectiveness measures in isolation. The results show that the examined approaches achieve to reduce the number of the mutants requiring analysis and the encountered equivalent mutants but with the expense of losing some of the approach effectiveness. Is this a good tradeoff? This section examines this issue by performing a comparison of the cost-effectiveness of the examined methods.

The examined strategies provide the advantage of prioritizing the live mutants according to their impact. Thus, the mutants can be considered based on their likelihood of being killable. In view of this, the proposed strategies can be efficiently applied by stopping them before examining the whole list of mutants (list of mutants with an impact on coverage). This way a stopping rule, e.g., stop the testing

process when a set of a predefined number of mutants are evaluated, can be adopted. It is noted that the same rule can be applied to the traditional approach. However, in this case the order is made at random.

Using mutants' impact it is possible to achieve a good trade-off between cost and effectiveness of the approaches. We are interested in maximizing the benefits compared to the traditional approach. The question that it is raised here is when the best cost-effective point to stop the process is. To examine this issue, the cost (number of mutants that require analysis) and effectiveness factors of the examined strategies were put together. Hence, the average mutation score achieved by the WI, NLI and traditional mutation approaches per mutant requiring manual inspection was computed. To this end, the graphs of Figures 7 and 8 record the difference in the scores achieved by the WI and NLI with the traditional (denoted as Trad) approaches, y-axis, per mutant requiring analysis, x-axis, for the statement and random initial test sets respectively.

The results of Figures 7 and 8 suggest that both strategies have an advantage when a small number of mutants are to be analyzed. In particular for the case of WI, a mutation score difference of approximately 0.5%-1.5% and 1.0%-1.7% is experienced when 2-6 mutants are to be analyzed when statement and random test cases are used respectively. For the NLI approach the difference is 0.7%-1.2% and 0.8%-1.2% when analyzing 2-11 mutants for statement and random tests. From this point, the difference starts decreasing markedly. Surprisingly, the WI approach is more cost effective when compared with the NLI approach. This is in contrast to what expected since it encountered many more equivalent mutants than the NLI. Generally, WI approach is fruitful if it is applied up to analyzing 19-20 for the statement test cases. For the case of the random initial tests WI is fruitful when analyzing up to 16 mutants. In contrast, NLI is beneficial up to 15 mutants, when it is applied with statement-based test cases (Figure 7). In the case of random tests (Figure 8) it is beneficial up to 11 mutants. By comparing the two approaches, it can be argued that NLI has no advantage compared to WI.

Figures 9 and 10 present the average mutation score achieved by the WI, NLI and traditional mutation approaches per mutant requiring manual inspection. From these graphs it becomes evident that a mutation score higher to 92% can be achieved by the strategies with advantages over the traditional one. This is achieved by manually examining 10-12 mutants per program. However, as stated before this advantage has a limit, which was found to be approximately 94% and 92.5% for the WI approach, when statement and random test suites are used. For the NLI approach the limits were found to be 92% and 87% for the statement and random test suites respectively.

Conclusively from the reported results it can be argued that mutant classification is beneficial for improving the mutation score if a small number of equivalent mutants is about to be analyzed. In the cases that a higher mutation score is required, the traditional mutation approach provides a better solution.

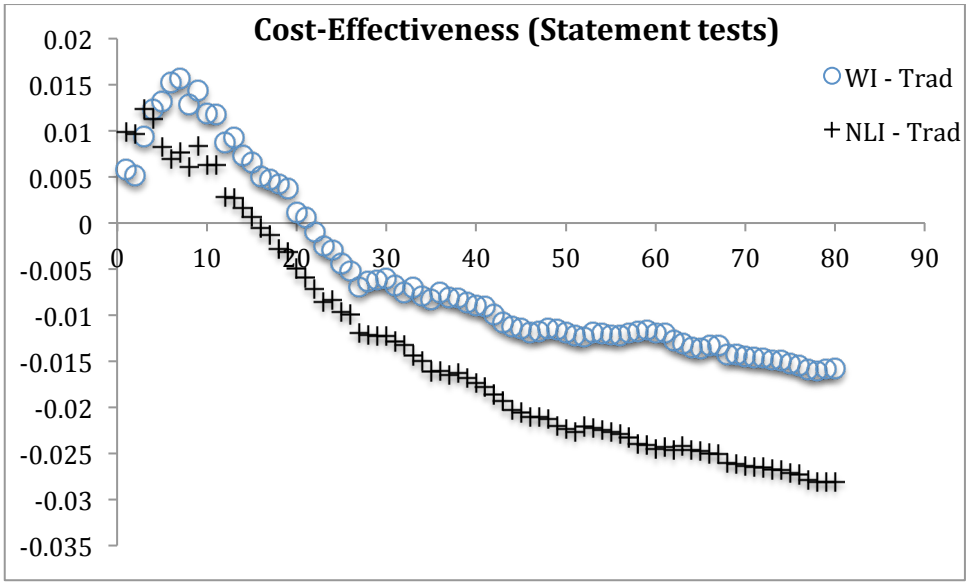


Figure 7. Cost-Effectiveness comparison when statement test cases are initially used. The x-axis represents the number of mutants requiring analysis. The y-axis records the difference in mutation scores between the strategies and the traditional approach ($MS_{Strategy}$ - MS_{Trad}).

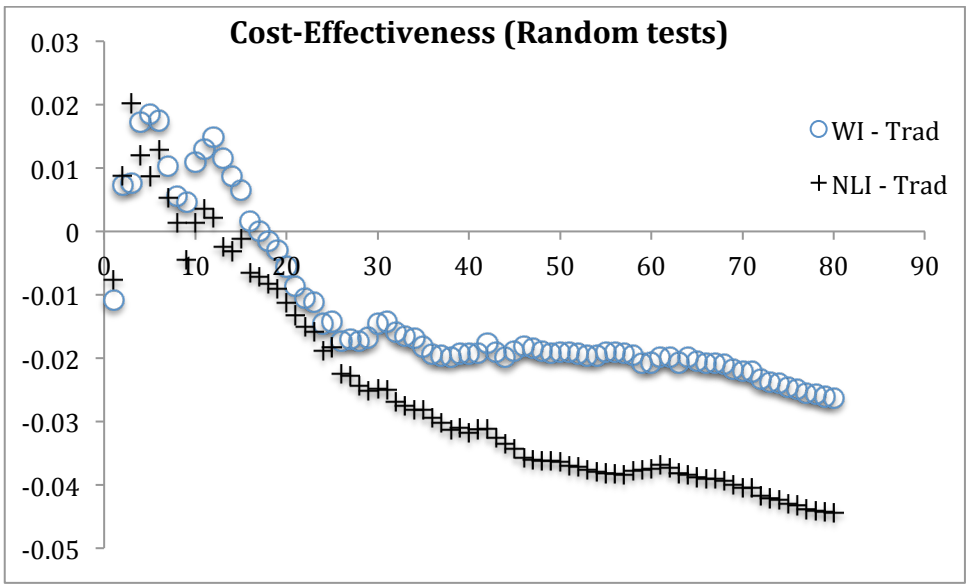


Figure 8. Cost-Effectiveness comparison when statement test cases are initially used. The x-axis represents the number of mutants requiring analysis. The y-axis records the difference in mutation scores between the strategies and the traditional approach ($MS_{Strategy}$ - MS_{Trad}).

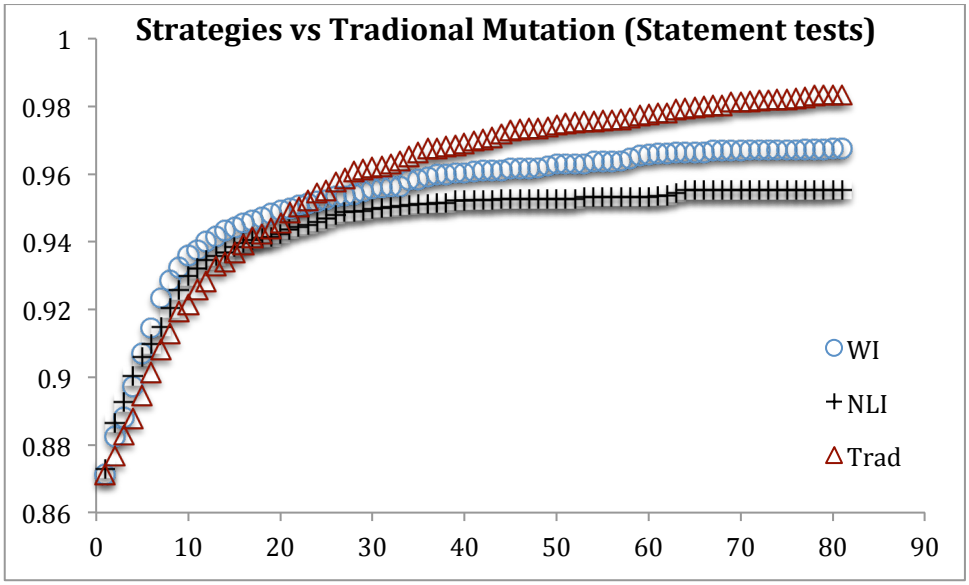


Figure 9. WI, NLI VS traditional approach comparison when statement test cases are initially used. The x-axis represents the number of mutants requiring analysis to achieve the mutation score recorded in y-axis.

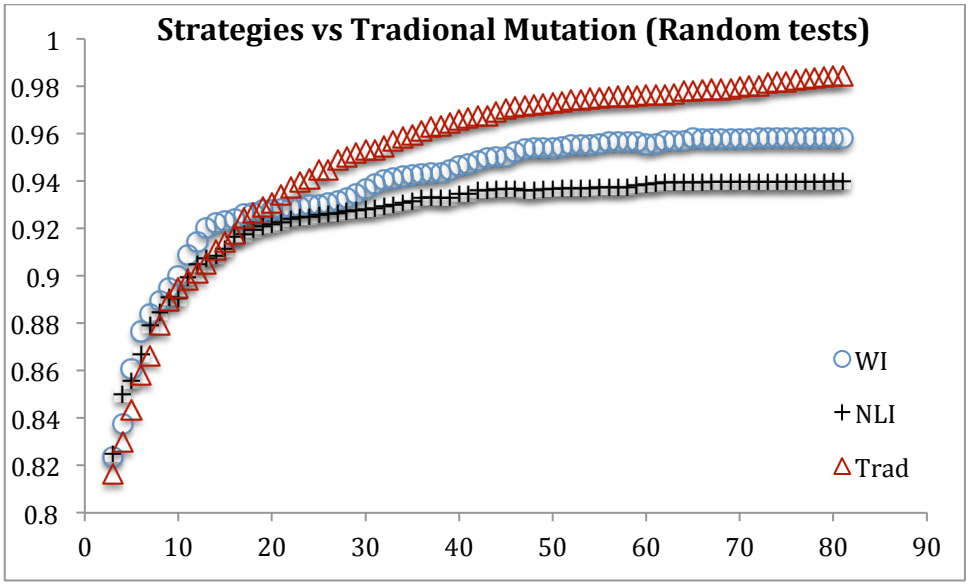


Figure 10. WI, NLI VS traditional approach comparison when random test cases are initially used. The x-axis represents the number of mutants requiring analysis to achieve the mutation score recorded in y-axis.

5.4. Classification Ability (RQ4)

The ability of the proposed approach to classify mutants as likely killable is assessed based on its respective precision and recall values. Table 9 records these values per utilized program for both the considered initial test sets. These results indicate that the statement test suites classify the live mutants on average with 55.66% precision value and 28.26% recall value when using WI. In the case of NLI,

precision and recall values of, respectively, 67.63% and 19.23% are achieved. These values suggest that the mutant sets selected by WI and NLI are composed of 55.66% and 67.63% killable mutants respectively. These mutants (killable) are the 28.26% and 19.23% of all live killable mutants. Similarly, regarding the random test suites, they classify on average with 69.35% precision value and 30.46% recall value when using WI. The NLI achieves a value of 83.70% precision and 19.24% of recall. These results suggest that NLI is more precise (achieves a better precision) than the WI. However, WI applies on a larger number of mutants (achieves a better recall value).

To evaluate whether the employed classification schemes provide a systematic guidance towards the killable mutants a comparison with a random selection approach is needed [22]. In view of this, Table 9 also records the killable mutants' ratios. These values represent the probability of randomly selecting a killable mutant. Both WI and NLI approaches categorize killable mutants much better than a random selection process. For example the 55.66% of the mutants categorized by WI are killable. Similarly, the 69.35% of those categorized by NLI while, the whole set has 35.21%. These results seem to be quite successful since they indicate that mutant classification provides a systematic way to select killable mutants [22]. This can be performed with a higher probability than a random selection process (of more than 20%). However, the results presented in the previous sections suggest that the mutant classification approaches have advantage when a small number of equivalent mutants are to be analyzed.

Table 9. Classification ability (Recall Precision)

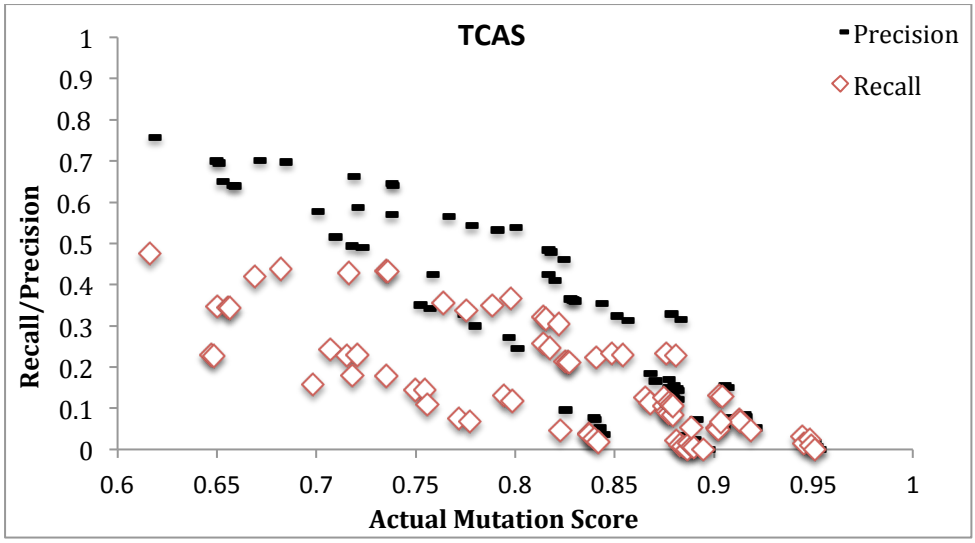
Subject Program	**Initial Test Set**	**Mutation Score**	**Killable Ratio**	**Precision**	**Recall**	**Precision**	**Recall**
				Approach A (WI)		**Approach B (NLI)**	
Tcas	Statement	66.71%	60.33%	62.87%	34.03%	63.14%	33.98%
	Random	50.03%	69.11%	73.78%	88.96%	79.09%	39.16%
Schedule	Statement	85.63%	42.66%	68.70%	10.70%	66.30%	9.71%
	Random	77.85%	51.60%	77.64%	16.80%	76.96%	15.79%
Schedule2	Statement	84.53%	37.65%	60.34%	26.06%	78.54%	24.42%
	Random	80.42%	42.65%	53.98%	12.84%	67.49%	10.77%
Totinfo	Statement	89.07%	52.08%	79.01%	26.02%	100.00%	5.60%
	Random	78.44%	65.67%	81.17%	18.27%	100.00%	5.75%
Replace	Statement	89.86%	28.85%	57.84%	49.77%	86.61%	43.53%
	Random	80.42%	42.65%	75.44%	39.41%	94.25%	37.57%
Printtokens	Statement	96.04%	16.13%	43.28%	30.27%	67.71%	12.76%
	Random	72.80%	55.74%	58.43%	5.84%	91.71%	3.86%
Printtokens2	Statement	97.98%	8.74%	17.60%	20.95%	11.09%	4.64%
	Random	77.85%	34.56%	65.02%	31.08%	76.40%	21.80%
Average	**Statement**	**87.12%**	**35.21%**	**55.66%**	**28.26%**	**67.63%**	**19.23%**
	Random	**73.97%**	**51.71%**	**69.35%**	**30.46%**	**83.70%**	**19.24%**

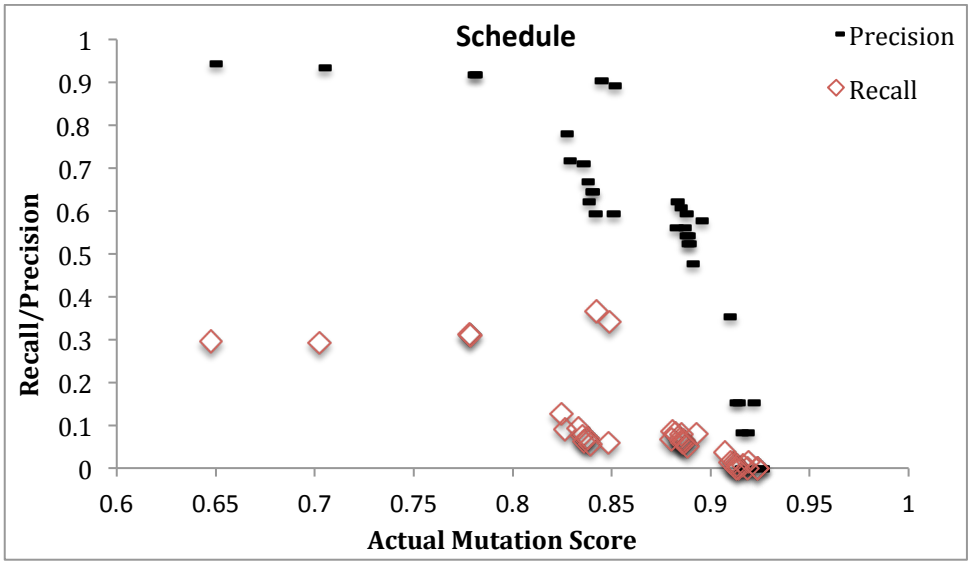
To investigate this issue, Figure 11 records the precision and recall values according to the WI classification scheme (see Section 3.2 for details) for all subjects when the initial test sets are selected at random. From these graphs, it can be deduced that both the recall and precision values decrease when the actual mutation score increases. Additionally, it can be clearly seen that both recall and precision are decreased with approximately the same trend. Consequently, the classification ability of the examined approaches decreases when the test suite evolves, i.e., new tests are added, according to these strategies.

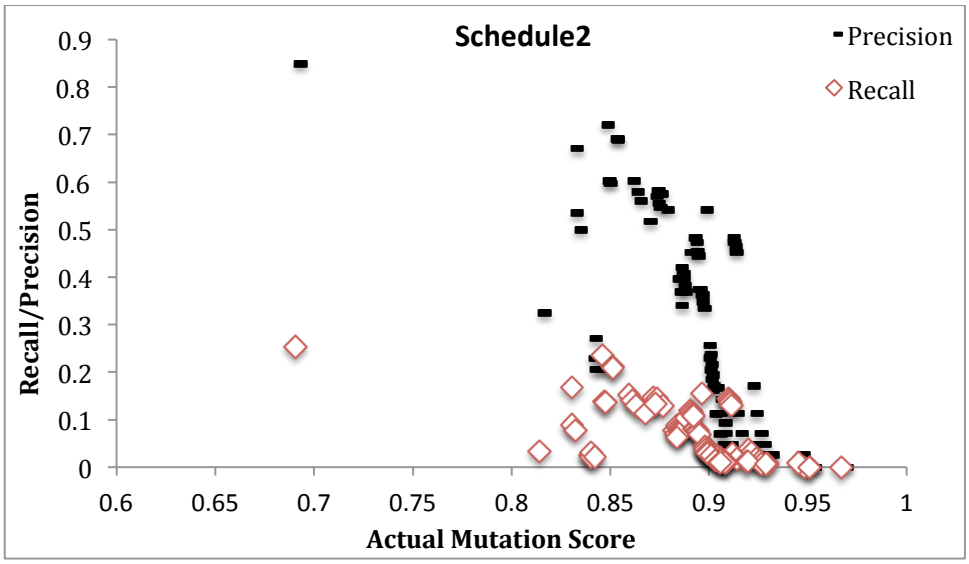
At first sight, this trend might seem to be counter-intuitive due to the fact that better tests should provide fewer classification mistakes. To explain this situation we have to consider that: a) mutant classification is performed on the live mutant set and not the whole set of mutants, and b) that there is an amount of equivalent mutants that have an impact [22]. This situation is depicted on Figure 12. Therefore, when the test suite evolves, mutants with an impact are killed. Recall, that the test suite evolves by repeatedly aiming at the mutants (among the live ones) with the highest impact. Thus, the produced tests kill those mutants (with the highest impact). This results in decreasing the number of the killable mutants with an impact while leaving the number of equivalent mutants with an impact constant. Perhaps now some killable mutants might have an impact due to the test suite evolution, but this is a very small number (as observed by the present experiment). Additionally, this is the case for equivalent mutants, i.e., more equivalent mutants have an impact, which results in increasing the error rate. As a consequence, both recall and precision values decrease.

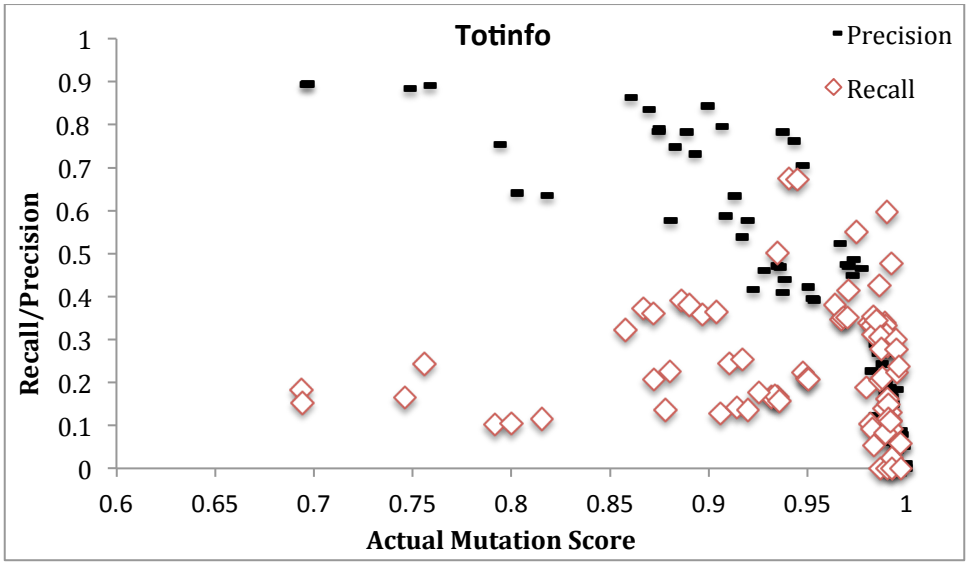
In practice, the decreasing trend explains why the effectiveness of the examined approaches is lower to 100% (subsection 5.1 presents the effectiveness results). It also explains why the mutant classification are not more cost effective beyond a specific point. Since, the classification precision is reduced when the test suite evolves; less guidance towards killable mutants is provided. Actually, beyond a certain point there is no guidance at all, fact suggesting that there are applicable limits on the number of mutants that can be killed by using the coverage impact. These observations, confirm the conclusion drawn by the results reported the previous sections.

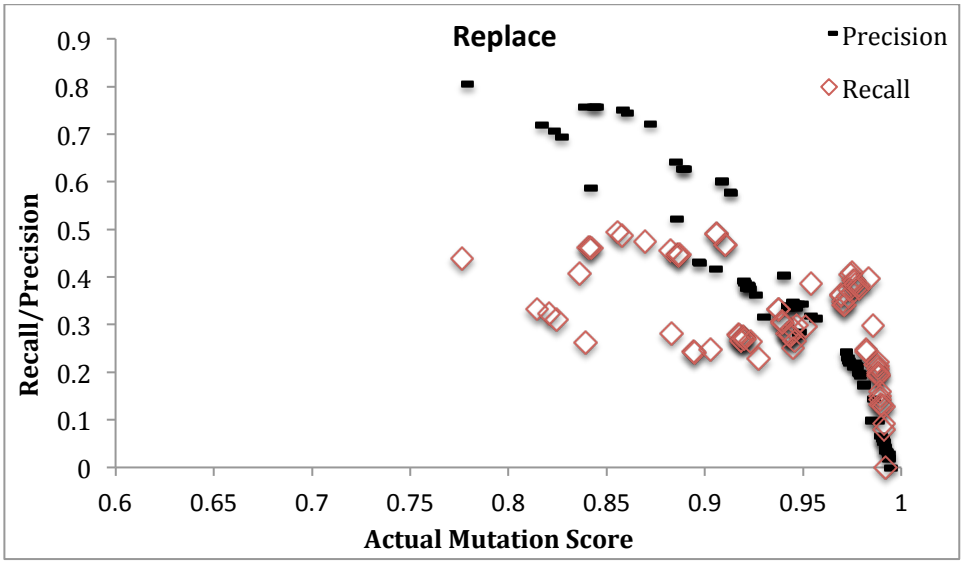
To further investigate the validity of the abovementioned observations we recorded the number of equivalent mutants with an impact when mutation adequate test cases are used. The respective results are recorded on Table10. These results indicate that a significant number of equivalent have an impact. This number is approximately the 16% and 7% of all the equivalent mutants for the WI and NLI approaches respectively. These mutants represent the 2.68% and 1.22% of all the introduced mutants. Recall that equivalent mutants were found to be approximately 17% of all the introduced mutants. Therefore, the equivalent mutants' problem has been reduced from the 17% to something less than 3%.

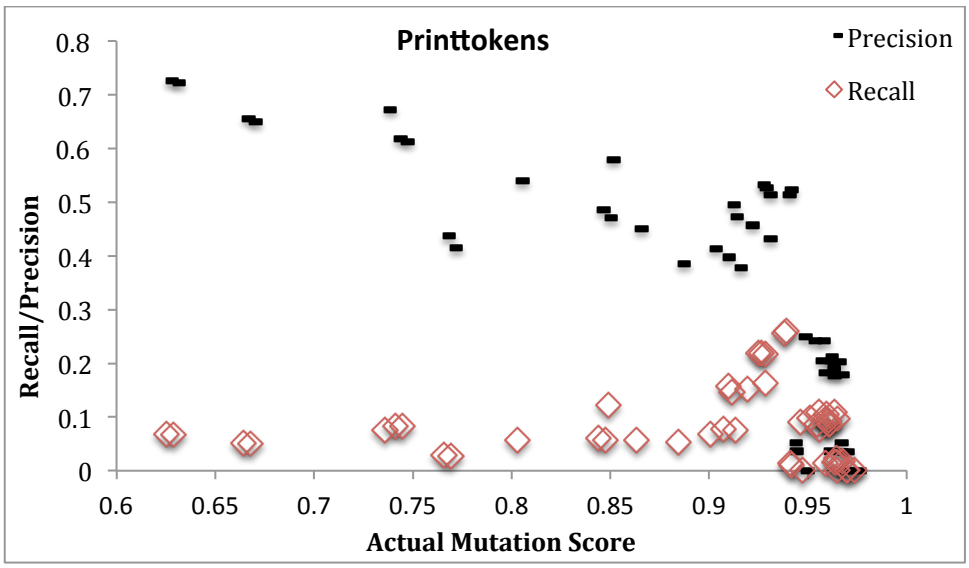

TCAS
Precision
Recall
Recall/Precision
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
Actual Mutation Score


Schedule
Precision
Recall
Recall/Precision
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
Actual Mutation Score


Schedule2
Precision
Recall
Recall/Precision
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
Actual Mutation Score


Totinfo
Precision
Recall
Recall/Precision
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
Actual Mutation Score


Replace
Precision
Recall
Recall/Precision
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
Actual Mutation Score


Printtokens
Precision
Recall
Recall/Precision
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
Actual Mutation Score

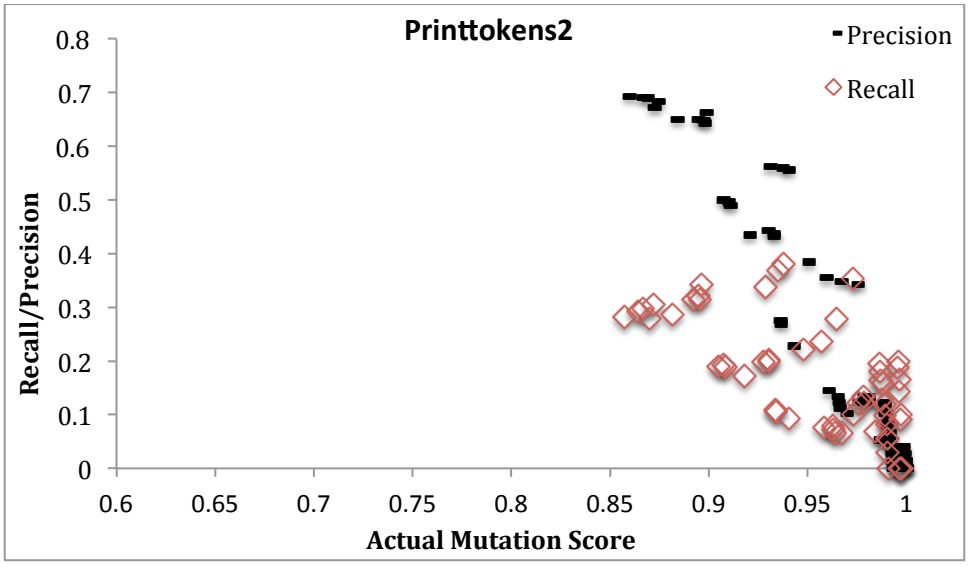


Figure 11. Recall and Precision values VS actual mutation score per utilized program for the Whole Impact classification approach (WI)

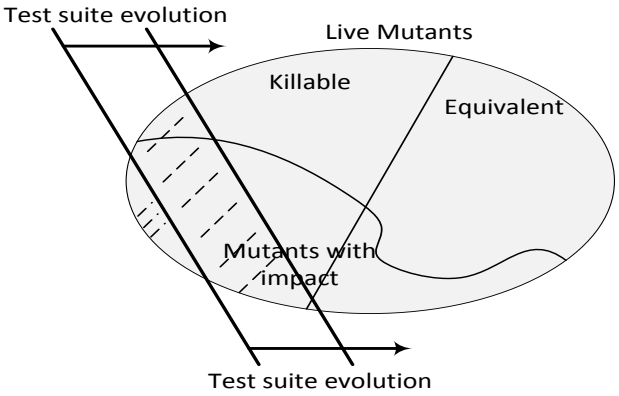


Figure 12. Classifying mutants (live) using the classification strategies. When tests are added, the killable mutants with an impact are killed. Therefore, the percentage of killable mutants with an impact decreases.

Table 10. Equivalent Mutants with an Impact

Subject Program	Number of Equivalent Mutants	Number of Equivalent Mutants with an Impact WI	Number of Equivalent Mutants with an Impact NLI
Schedule	338	19 (5.62%)	12 (3.55%)
Schedule2	526	55 (10.46%)	35 (6.65%)
Tcas	515	193 (37.48%)	191 (37.09%)
Totinfo	572	159 (27.80%)	60 (10.49%)
Replace	2122	343 (16.16%)	77(3.63%)
Printtokens	725	57 (7.86%)	6 (0.83%)
Printtokens2	791	71 (8.98%)	28 (3.54%)
SUM	**5,589**	**897** (16.05%)	**409** (7.32%)

5.5 Application to Larger Programs

The results presented in the previous sections were based on the Siemens suite and on the complete manual analysis performed on them. Performing complete analysis on larger programs requires vast manual effort. Therefore, for the two large programs (Space and Flex), we turned to evaluate the approach based on the mutants killed by the associate test suite. If we observe a similar trend with the findings of the Siemens suite, we can gain confidence regarding their validity. In view of this, the cost effective relation of the examined approaches is drawn and presented in Figures 13 and 14 for the Space and Flex programs respectively. Interestingly on both subjects the WI approach is cost-effective in analyzing a small number of mutants. These results agree with those of the Siemens suite giving confidence that the tester should stop the testing process after analyzing a few mutants. For the case of the Space program the classification approaches seem to be more cost-effective than the traditional approach for a higher number of mutants. However, this is not true for the case of the Flex subject. Additionally, it is noted that these results also suggest that approach WI is almost always better than the NLI.

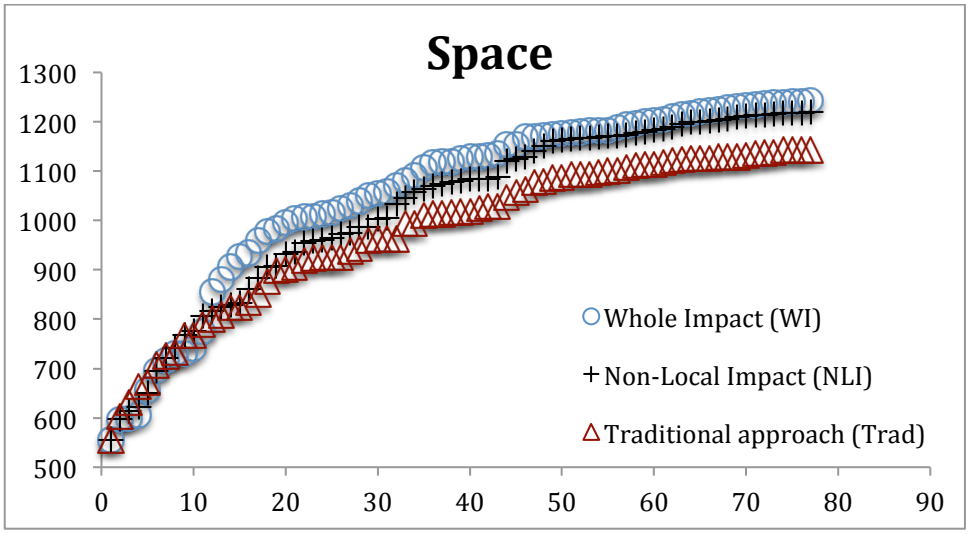


Figure 13. Cost-Effectiveness comparison for the Space program. The x-axis represents the number of non-killed mutants to kill the number of mutants recorded in y-axis.

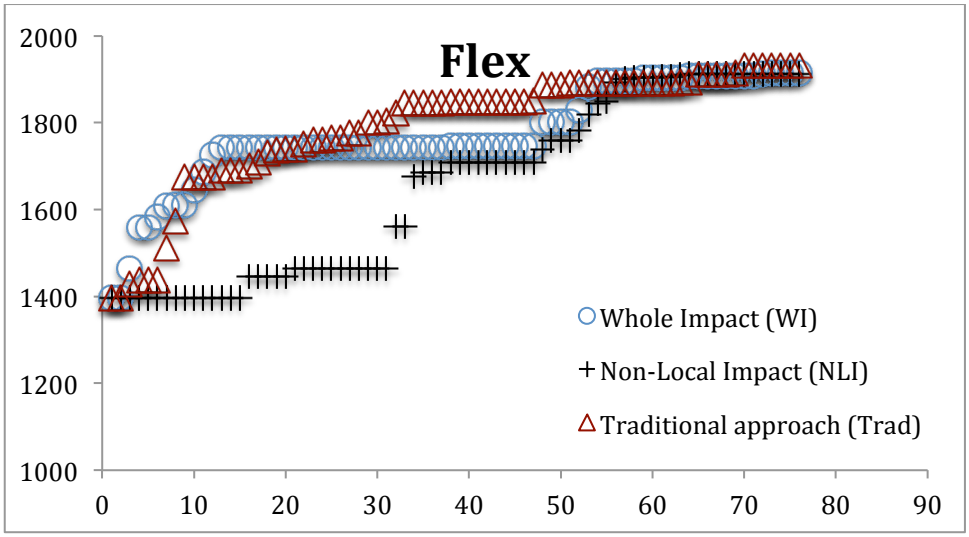


Figure 14. Cost-Effectiveness comparison for the Flex program. The x-axis represents the number of non-killed mutants to kill the number of mutants recorded in y-axis.

6. Discussion

The conducted study suggests that mutant classification strategies have a lower effectiveness when compared to the traditional testing approach. However, they have advantages regarding their ability to isolate equivalent mutants [22] and [21]. The present paper is the first to the authors' knowledge that investigates the benefits of this approach compared to the traditional one. We found that mutant classification is beneficial only for improving a low quality test suite and only up to a certain limit. The discussion on section 5.4 gives an intuitional explanation about that and validates it based on the experimental results. Based on the cost-benefit evaluation, section 5.3, it can be argued that the proposed strategies are fruitful if they are applied until analyzing up to 8-12 mutants per program. Generally, at this point a mutation score higher than 92% is achieved by the examined approaches. Regarding the WI approach, which gives the best results, the achieved score is approximately 94%.

Surprisingly the NLI approach performs always worst than the WI. Since NLI has a higher precision than the WI, we expected it to be more cost-effective [22]. By comparing the precision and recall values of the WI and NLI approaches we can infer that that WI has approximately 10% higher recall value and 10% less precision. These results suggest that mutant classifiers having a higher recall value might be more efficient to those having higher precision. In view of this, it can be argued that mutant classification research should focus on developing mutant classifiers with higher recall value.

Another outcome of the present study is the fact that the initial test suite has a small effect on the effectiveness of the mutant classification strategies. This is an interesting result since mutants that are not executed by the initial test cases cannot have an impact. Thus, it is expected that these mutants will be ignored by the strategies. However, in practice it turns out that the mutants that are executed and have an impact force the generation of new tests. In turn, these tests execute additional mutants and cause them to have an impact. Therefore, at the end, the strategy does not need to ensure statement coverage at the first place.

6.1 Threats to Validity

Several threats to the validity of the findings of the present study can be identified.

With respect to the *internal validity,* i.e., factors that might affect the reported results without the authors' knowledge, some issues can be recognized. Issues might arise from the utilization of the selected subjects, their associate test cases and the tools used. However, as already mentioned the selected subjects and their associate tests are not only large in size, they are also of high quality since they were produced based on a combination of techniques [8]. In addition, manually constructed test cases were also added and the experiment was independently repeated five times. Furthermore, many empirical studies, such as [17], [19], [20], [22], utilized the same tools and subjects. Thus, we believe that these particular threats are not of a great importance.

The selected operators introduce additional validity threats. Hence, different operators may behave differently. However, the present study, with regard to the first experiment, considers all C unit-level mutant operators. These operators were carefully designed by considering all C language constructs [2]. The second experiment is also based on a relatively large set of operators. It is composed of 44 operators and involves all the language operators. Other threats could be attributed to the manual identification of the equivalent mutants. Hence, it is possible that some mutants have mistakenly been identified. However, the Siemens suite is composed of high quality test cases and thus, even if there are some mistakes, only a few such cases should exist. Additionally, to reduce this threat we make these data publically available. Other threat that is identified is due to the implementation of the impact measure. To reduce this threat, a careful manual check was performed.

The *external validity* of the experiment, i.e., representativeness of the selected subjects, can be questioned due to the use of the selected subjects. Clearly more experiments are in need to support the representativeness of these findings. This is a common problem to all empirical studies. Yet, all the selected subjects are benchmark programs, widely used in similar experiments [11]. Additionally, the results from the second experiment increase the confidence on the reported findings.

The *construct validity*, i.e., whether the intended properties are represented by the employed measurements, can be questioned due to the use of the mutants to analyze and mutation score as measures of the strategies cost and effectiveness. With respect to the cost measure, it is noted that designing test cases to kill the live mutants and identifying the equivalent ones form a manual activity and thus, they dominate the automated parts of the mutation process. Perhaps the use of heuristics, e.g., [9], that identify some equivalent mutants may influence the findings. However, in practice these approaches fail to identify most of the equivalent mutants and their application to real-world programs is questionable. This is due to the fact that there is no tool or study showing the feasibility of any heuristic in identifying equivalent mutants in real-world applications. In any case, it is believed that their application will not influence the findings of the present study since these approaches are complementary to mutant classification [22], [24]. Regarding the effectiveness measure, i.e., the mutation score, it is noted that the examined approaches form alternatives to the traditional approach. Therefore it is natural to evaluate them in the terms of the original one. Additionally, the use of this measure has been employed in many empirical studies such as [11], [12].

7. Related Work

The dynamic reduction of the side effects caused by equivalent mutants is a new research topic that helps automating effectively the mutation testing process. One such approach has been suggested by Adamopoulos et al. [1] by using an evolutionary method. In their approach, the evolution method seeks for both mutants and test cases with the aim of selecting at the same time a small and killable set of mutants. Although this approach achieves to produce killable mutant sets, it relies on the quality and ability of selecting and the producing test cases and thus, the adequacy of the testing process is uncertain. On the contrary, the present approach tries to isolate equivalent mutants in an attempt to both perform mutation testing efficiently and to assess the adequacy of testing.

Schuler et al. [21] proposed the use of mutants' runtime behavior as a measure of the mutants' killability likelihood based on dynamic program invariants. In the same study, it was found that mutants are likely killable when they break dynamically introduced invariants. The idea behind this approach was then used to assess mutations based on coverage impact [22] as discussed in the present paper. Empirical comparison between the abovementioned approaches [22] revealed that the impact on coverage is more efficient and effective at assessing the killability of mutants. Therefore, the present paper empirically investigates the application of the coverage impact classification scheme within the mutation testing process. Additionally, its application effectiveness and efficiency were also examined.

Generally, the automatic identification of equivalent mutants has been proven to be an "undecidable" problem [4]. As a result, no approach able to detect all equivalent mutants can be defined. Fortunately, heuristics for detecting some cases exist [17]. Offutt and Pan [16] suggested another approach to identify equivalent mutants with the combinational use of a constraint-based technique. Empirical evaluation of this technique reported that 45% of equivalent mutants could be identified on average. Other approaches aiming at identifying equivalent mutants employ program slicing [9] to assist the identification process. All the aforementioned techniques aim at detecting equivalent mutants, but they don't focus on their ability to be likely equivalent or killable ones. Thus, these approaches are orthogonal to the presently examined ones [22]. This allows employing them first to detect equivalent mutants and then assess the remaining ones based on their impact.

Another approach to tackle the equivalent mutants' problem is by using higher order mutants [12], [14], [13] and [18]. Generally, the main idea underlying these approaches is to produce a set of higher order mutants and use them instead of the first order ones. Higher order mutation strategies such as [13] and [18] produce considerably less equivalent mutants, thus alleviating the problems that they introduce. Contrary to the present approach, the higher order strategies produce mutant sets containing a few equivalent mutants without aiming at isolating them. A comparison between these approaches and the presently proposed one is a matter open for further investigation.

A static approach to deal with the equivalent mutants has been proposed in [23]. This approach focuses on the cost-effective generation of test cases aiming at killing mutants. The underlying idea is to select program paths that kill a targeted mutant and attempt to produce tests that cover these paths. Failing to

do so for more than 50 times results in inefficient solutions and thus, the process should stop. This way, mutants are categorized as killable and likely to be equivalent. Contrary, the present approach tries to dynamically select mutants that are likely to be equivalent. It also does not aim at producing test cases but to provide guidance towards killable mutants. Therefore, the tester and or an automated method for producing test cases can benefit from this guidance. Additionally, the presently presented solution is independent to the test generation process. Thus, it does not suffer from scalability issues and the other limitations of the test generation techniques.

8. Conclusions

The present paper investigates the use of dynamic strategies aiming at reducing the effects of equivalent mutants during the mutation testing process. Towards this direction, mutant classification strategies were defined and evaluated. The innovative part of the proposed strategies is their ability to effectively produce and evaluate test suites by considering only a small set of equivalent mutants. Doing so gives the advantage of performing mutation by manually analyzing only a small number of mutants. The undertaken experiments suggest the following conclusions.

- Mutant classification strategies provide substantial benefits in improving an existing test suite by reducing the effects of equivalent mutants. However, the strategies are efficient to improve only a low quality test suite.
- Mutant classification is in general less cost-effective than the traditional mutation process. Despite the promising results of the previous studies, e.g., [21] and [22], the present experiment strongly supports the conclusion that the mutant classification is beneficial only up to a certain point. Thus, it is not cost-effective to analyze more than 10-20 mutants per program.
- Classification ability was found to be dependent on the percentage of mutants that are killed by the utilized tests. Surprisingly, when a test suite evolves (based on mutation) the classification ability (both recall and precision values) decreases. Hence, the guidance provided by the classification process decreases.

Future work includes conducting additional experiments to statistically revalidate the findings of the experiment. Additionally, an evaluation of the proposed strategies with the use of other impact measures, e.g., [14], is under investigation. Further, a comparison between the examined approaches and other mutation testing strategies [21] are also planned.

The data of the present study are publically available and can be found the companion website of the present paper:

https://sites.google.com/site/mikepapadakis/home/tools-and-data-sets/scp2014-data

9. References

[1] Adamopoulos, K., Harman, M. and Hierons, R.M. 2004. How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-evolution. In GECCO 2004, 1338-1349.

[2] Agrawal, H., DeMillo, R.A., Hathaway, B., Hsu, W., Hsu, W., Krauser, E.W., Martin, R.J., Mathur, A.P. and Spafford, E. 1989. Design of Mutant Operators for the C Programming Language. Purdue University.

[3] Andrews, J.H., Briand, L.C. and Labiche, Y. 2005. Is mutation an appropriate tool for testing experiments? In Proceedings of the 27th international conference on Software engineering (ICSE '05), 402-411.

[4] Budd, T.A. and Angluin, D. 1982. Two notions of correctness and their relation to testing. Acta Informatica. 18, 1 (1982), 31-45.

[5] DeMillo, R.A., Lipton, R.J. and Sayward, F.G. 1978. Hints on Test Data Selection: Help for the Practicing Programmer. Computer. 11, 4 (1978), 34-41.

[6] Delamaro, M. and Maldonado, J.C. 1996. Proteum - A Tool for the Assessment of Test Adequacy for C Programs. In Proceedings of the Conference on Performability in Computing Systems (PCS '96), 79–95.

[7] Do, H., Elbaum, S. and Rothermel, G. 2005. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact. Empirical Software Engineering journal. 10, 4 (2005), 405-435.

[8] Harder, M., Mellen, J. and Ernst, M.D. 2003. Improving test suites via operational abstraction. In Proceedings of the 25th International Conference on Software Engineering (ICSE '03), 60-71.

[9] Hierons, R.M., Harman, M. and Danicic, S. 1999. Using Program Slicing to Assist in the Detection of Equivalent Mutants. Software Testing, Verification and Reliability. 9, 4 (1999), 233-262.

[10] Hutchins, M., Foster, H., Goradia, T. and Ostrand, T. 1994. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In Proceedings of the 16th international conference on Software engineering (ICSE '94) 191-200.

[11] Jia, Y. and Harman, M. 2011. An Analysis and Survey of the Development of Mutation Testing. IEEE Trans. Software Engineering. 37, 5 (September 2011), 649-678.

[12] Jia, Y. and Harman, M. 2009. Higher Order Mutation Testing. Information and Software Technology journal ,51, 10 (2009), 1379-1393.

[13] Kintis, M., Papadakis, M. and Malevris, N. 2010. Evaluating Mutation Testing Alternatives: A Collateral Experiment. In Proceedings of the 2010 Asia Pacific Software Engineering Conference (APSEC '10), 300-309.

[14] Kintis, M., Papadakis, M. and Malevris, N. 2012. Isolating First Order Equivalent Mutants via Second Order Mutation. In Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST '12), 701-710.

[15] Namin, A.S., Andrews, J.H. and Murdoch, D.J. 2008. Sufficient mutation operators for measuring test effectiveness. In Proceedings of the 30th international conference on Software engineering (ICSE '08), 351-360.

[16] Offutt, A.J. and Pan, J. 1997. Automatically Detecting Equivalent Mutants and Infeasible Paths. Software Testing, Verification and Reliability. 7, 3 (1997), 165-192.

[17] Offutt, A.J. and Untch, R.H. 2001. Mutation 2000: uniting the orthogonal. In Mutation testing for the new century. Kluwer Academic Publishers, 34-44.

[18] Papadakis, M. and Malevris, N. 2010. An Empirical Evaluation of the First and Second Order Mutation Testing Strategies. In Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW '10), 90-99.

[19] Papadakis, M. and Malevris, N. 2010. Automatic Mutation Test Case Generation via Dynamic Symbolic Execution. In Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE '10), 121-130.

[20] Papadakis, M. and Malevris, N. 2011. Automatically performing weak mutation with the aid of symbolic execution, concolic testing and search-based testing. Software Quality Journal. 19, 4 (2011), 691-723.

[21] Schuler, D., Dallmeier, V. and Zeller, A. 2009. Efficient mutation testing by checking invariant violations. In Proceedings of the eighteenth international symposium on Software testing and analysis. (ISSTA '09), 69-80.

[22] Schuler, D. and Zeller, A. 2012. Covering and Uncovering Equivalent Mutants. Software Testing, Verification and Reliability. (2012).

[23] Papadakis, M. and Malevris, N. 2012. Mutation based test case generation via a path selection strategy. Information and Software Technology journal. 54, 9 (September 2012), 915-932.

[24] Papadakis, M. and Le Traon, Yves. 2013. Mutation testing strategies using mutant classification. In Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC '13). ACM, New York, NY, USA, 1223-1229.

[25] Yoo, S., Harman, M. and Clark, D. 2013. Fault Localization Prioritization: Comparing Information Theoretic and Coverage Based Approaches. ACM Transactions on Software Engineering Methodology, in press. 22, 3 (July 2013), 1-29.