Mutation Testing Strategies using Mutant Classification

Mike Papadakis
Interdisciplinary Center for Security, Reliability and Trust (SnT),
University of Luxembourg
michail.papadakis@uni.lu

Yves Le Traon
Interdisciplinary Center for Security, Reliability and Trust (SnT),
University of Luxembourg
yves.letraon@uni.lu

ABSTRACT
Mutation testing has a widespread reputation of being a rather powerful testing technique. However, its practical application requires the detection of equivalent mutants. Detecting equivalent mutants is cumbersome since it requires manual analysis, resulting in unbearable testing cost. To overcome this difficulty, researchers have proposed the use of mutant classification, an approach that helps isolating equivalent mutants. From this perspective, the present paper establishes and assesses possible mutant classification strategies. The conducted study suggests that while mutant classification can be useful in isolating equivalent mutants, it fails to kill some mutants. Indeed, the experimental results show that the proposed strategies achieve to kill approximately 95% of the introduced killable mutants.

Categories and Subject Descriptors
D.2.5 [**Testing and Debugging**]: Testing tools

General Terms
Verification.

Keywords
Mutation Testing, Mutants' Impact, Mutant Classification.

1. INTRODUCTION
Mutation testing aims at detecting software defects by injecting artificial errors, called *mutants*, in the examined software [5], [17]. This 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 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 the test case power. However, while some mutants are killable, some mutants cannot be killed. In that case, such mutants are qualified as *equivalent*. As a result, an equivalent mutant forms a program version functionally equivalent to the original one since no test case is 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 [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 *impact*, introduced by mutants on the runtime program execution. It has been found that mutants with higher impact are more likely to be killable than those with less or no impact. Although mutant classification has been suggested [22], it is not evident how it could be applied in practice. Further, it is not evident what the practical benefit of its utilization is. Going a step further, the present paper defines strategies that take advantage of this 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 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 *how effective are these strategies compared to the "traditional" mutation testing approach* i.e. testing by using all mutants.

The above intentions were investigated based on a set of moderate size industrial programs written in C and using the Proteum [6] mutation testing system. It has been found, that the examined

SAC'13, March 18-22, 2013, Coimbra, Portugal.

strategies are 5-10% less effective than testing based on all mutants. Despite this weakness, employing these strategies can lead to substantial benefits with respect to the equivalent mutants' identification. Additionally, it has been found 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 reminder of this paper is organized as follows: Sections 2 and 3 present the concepts underlying the present work and the mutant classification process. Section 4 presents and details the conducted experiment and its respective results. In Section 5 some related work is discussed. Finally, Section 6 concludes the paper.

2. PROBLEM DEFINITION

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]. An equivalent mutant degrades the computation of the mutation score, since it cannot be killed. To this end, practitioners have two choices: a) to manually analyze the live mutants, which in practice is very hard, if not 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 on the testing process. Second, it does not give any guidance to the tester on 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 approximate the mutation score by identifying likely killable mutants. In other words, it allows deciding whether mutants can be killed or not. Therefore, the tester can concentrate only on these (likely killable) mutants. However, by doing so, *what level of the exact mutation score can be reached? How many mutants will be manually analyzed?* Answering the first question is important since it provides evidence about the strengths of these approaches. Answering the second question provides an insight regarding the required manual effort of the examined processes.

3. MUTANT CLASSIFICATION

3.1 Mutants' Impact

Schuler and Zeller [22] advocated that "if a mutant impacts internal program behavior, it is more likely to change external program behavior and thus impact the semantics of the program". In other words, their idea is to compare the internal program behavior differences 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 impact are more likely to be killable than mutants without impact [22]. Hence, mutants can be classified as likely killable, i.e. mutants with impact, and likely equivalent, i.e. mutants without 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 by 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. More details on how we calculate the coverage impact is given in the following section.

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 statement 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 method is computed. To this end, two variations of this metrics are used:

Approach A (CA): The coverage impact on all the program methods.

Approach B (CB): The coverage impact on all the program methods except the one containing the examined mutant.

Generally, the CB approach targets on non-local impact while the CA approach targets on both local and non-local impact. The CB approach is based on the lines suggested in [22] and it is expected to give more accurate results than the CA. 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 impact).

3.3 Mutant Classification Strategies

Figure 1 presents a generic mutation testing process based on mutant classifiers. This process 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. 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 1 in the step b) when no process iterations have been performed. The second issue is presented in the same figure in the step e). Setting these two parameters are important for the effectiveness of the examined strategies.

Considering the initially used test set, the present paper starts by using statement coverage test sets. The reason for such a choice 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 with 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.

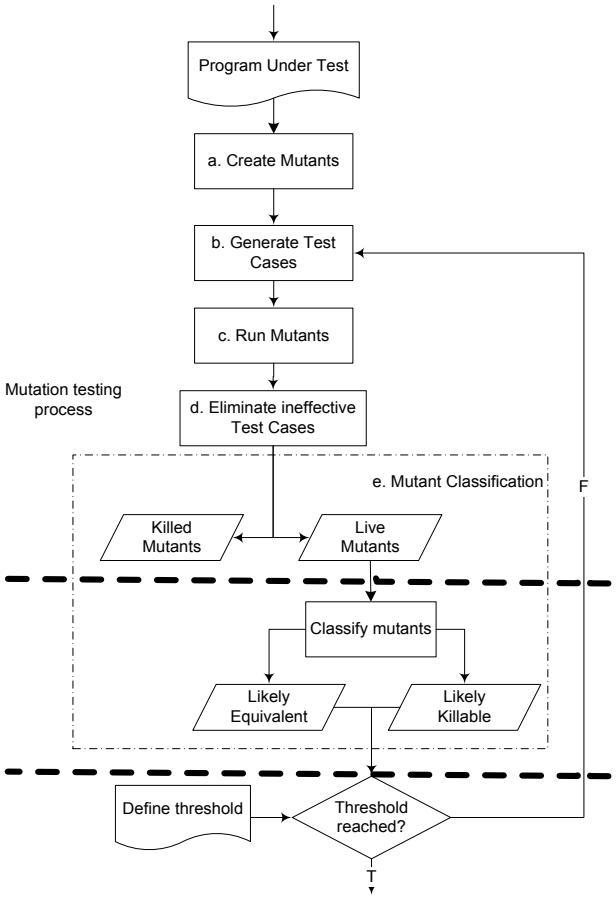

Figure 1. Mutation testing process using mutant classification

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, in each iteration of the process (Figure 1), all live mutants are executed with all the employed test cases. This process relies on mutant classification schemes (CA and CB) that are presented in section 3.2. Therefore, two possible strategies are considered and denoted as: CA, CB.

4. EMPIRICAL EVALUATION

4.1 Definition of the Experiment

The present study empirically investigates the use of mutant classifiers within the testing process. The main points of interest for this experiment are a) to study how the classification ability of the examined mutant schemes changes when the employed test suite evolves and b) to determine the relative effectiveness of the mutant classification strategies. Knowing these issues helps practitioners in choosing an appropriate strategy. Further, it answers the general question of whether mutant' classification can form a valid alternative to testing by using all mutants.

The above issues are based on the following research questions:

RQ1: How the classification ability of the mutant classifiers is affected by the increase in the actual mutation score?

RQ2: How effective are the examined strategies?

4.1.1 Subject Programs and Utilized tools

The present study uses six programs, Table 1, which come from the SIR repository [7] and include five programs of the Siemens suite [10]. It also includes an additional program (Space) developed by the European Space Agency. These programs were chosen because they were in C and are available 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 which was produced by several researchers using a combination of techniques, including random, category-partition, all statements, all edges and all definition-use pairs. More details about the construction of the test suites can be found in Harder et al. [8]. The associated test suite was produced independently of the present study and consists of a large number of high quality tests (since they were developed according to many testing criteria) and thus, they are well suited for the conducted experiment.

The Proteum [6] and gcov tools have been selected since they have been successfully used in many software testing experiments. Proteum is a mutation testing tool employing the Agrawal et al. [2] mutation operators while, gcov is a widely used GNU structural coverage tool. Regarding the implementation of the examined approaches, Proteum was used for producing 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 run the sought mutants. Therefore, to reduce the experimental cost, 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 approaches have also been undertaken in [3] and [18]. Additionally, since the space program involves a huge number of mutants (22,500 mutants) 10% of them were used. The selection of these mutants (10%) 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]. 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.

Table 1 record details about the utilized subjects, including the number of lines of code, the size of the accompanied test pool and the number of examined mutants per subject programs.

Table 1. Subject programs

Subject Program	Lines of Code	Test Pool Size	Number of Mutants
Schedule	296	2650	661
Schedule2	263	2710	887
Tcas	137	1608	940
Totinfo	281	1052	1645
Replace	513	5542	3526
Space	5905	13585	2250

4.1.2 Experimental procedure

To address the stated RQs, we report the results derived from the application of the proposed mutant classification strategies. Initially, gcov was employed to construct sets of test cases adequate with respect to basic block testing criterion per subject program. 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 experimental process: The experiment follows the mutation strategies described in Section 3 and presented in Figure 1. The statement based test suites were used as initial ones. After executing all mutants with the statement based 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 this mutant cannot be killed by any test of the whole test suite pool, 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 has impact (the likely killable mutants' set becomes empty). 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.

Generally, a mutant classifier is assessed based on its ability to categorize mutants as killable. This assessment is based on the following two measures:

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.*

Determining recall and precision values: These values were determined based on the mutants that were killed by the whole test suite pools. This practice fulfills the purpose of the present study which is to assess the effectiveness of the strategies when used as alternatives to testing by using all mutants. 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 like [3], [15] and [18].

To address **RQ1**, the respective recall and precision values were calculated every time the classification process was performed, (see Figure 1). Regarding **RQ2**, for each strategy and subject program we recorded the ratio of the killed mutants to those that can be killed by the whole test suite pool. When there are no mutants with impact anymore, the above ratio expresses the effectiveness measure of the process.

4.2 Results and discussion

4.2.1 Classification Ability (RQ1)

The ability of the proposed approach to classify mutants as likely killable and equivalent ones was assessed based on its respective recall and precision values. Figure 2 records these values according to the CB classification approach (see Section 3.2 for details) for all the subject programs. Due to lack of space, the results of the CA were not plotted. From these graphs, it can be

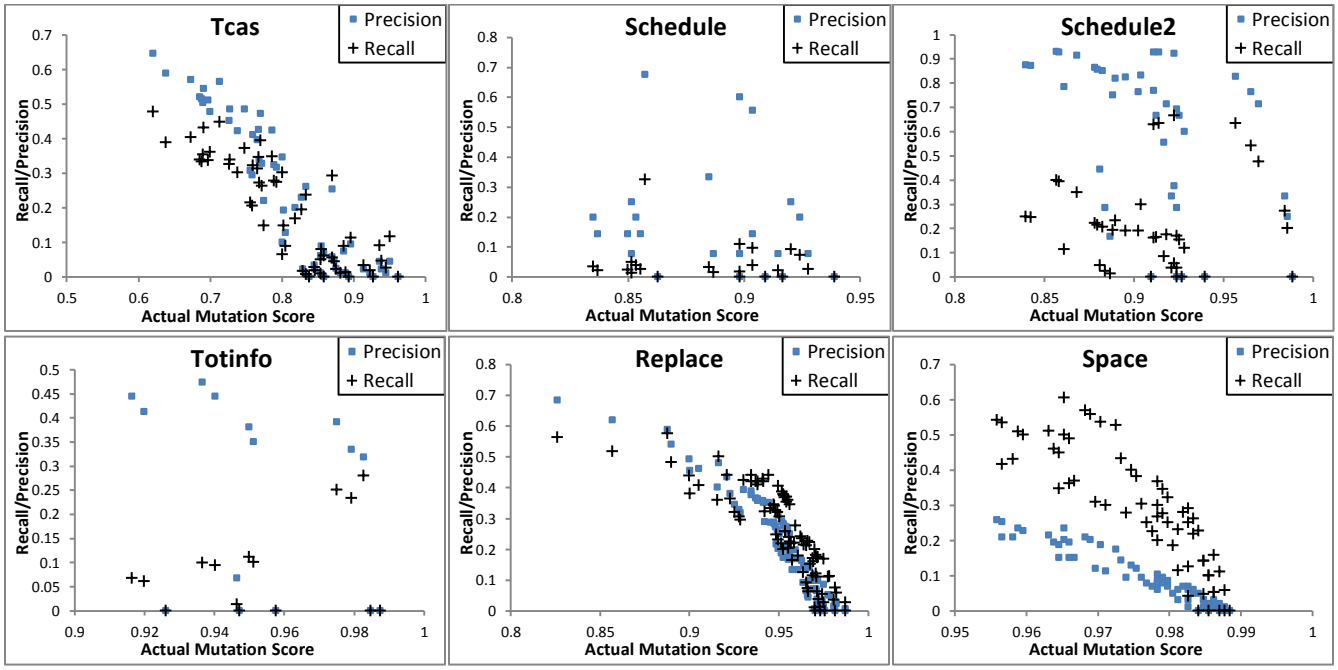

Figure 2. Recall and Precision values vs actual mutation score per utilized program for the classification approach B (CB)

Table 2. Empirical results

Subject Program	Method	#Tests	Actual MS	% killable mutants killed	Precision	Recall	Std. Deviation	#Tests	Actual MS	% killable mutants killed	Precision	Recall	Std. Deviation
		Approach A (CA)						Approach B (CB)					
Tcas	Statement	7.2	48.43%	66.74%	53.59%	40.27%	3.58%	7.2	48.43%	66.74%	55.21%	40.27%	3.58%
	Strategy	18.4	63.49%	87.51%	0.00%	0.00%	3.56%	18.2	64.96%	89.53%	0.00%	0.00%	5.05%
Schedule	Statement	2.6	70.44%	86.38%	32.41%	5.44%	2.65%	2.6	70.44%	86.38%	30.52%	4.99%	2.65%
	Strategy	6.4	73.86%	90.58%	0.00%	0.00%	1.98%	6.4	73.80%	90.50%	0.00%	0.00%	2.80%
Schedule2	Statement	7.4	68.41%	86.93%	75.07%	38.74%	2.71%	7.4	68.41%	86.93%	87.66%	32.32%	2.71%
	Strategy	14.4	76.35%	97.02%	0.00%	0.00%	2.65%	14.4	73.80%	93.78%	0.00%	0.00%	3.03%
Totinfo	Statement	6.8	83.09%	95.14%	49.09%	50.74%	2.63%	6.8	83.09%	95.14%	40.47%	15.20%	2.63%
	Strategy	10.8	86.68%	99.25%	0.00%	0.00%	0.53%	8.8	83.90%	96.06%	0.00%	0.00%	2.59%
Replace	Statement	16.6	71.28%	88.37%	49.49%	49.13%	4.17%	16.6	71.28%	88.37%	52.18%	45.79%	4.17%
	Strategy	36	79.04%	98.00%	0.00%	0.00%	0.35%	34	78.85%	97.76%	0.00%	0.00%	0.67%
Space	Statement	146.6	59.30%	96.16%	20.93%	47.67%	0.49%	146.6	59.30%	96.16%	20.93%	47.67%	0.49%
	Strategy	160.6	69.99%	98.89%	0.00%	0.00%	0.23%	158	60.85%	98.66%	0.00%	0.00%	0.17%
Average	Statement	**31.2**	**66.83%**	**86.62%**	**46.76%**	**38.67%**	**2.71%**	**31.20**	**66.83%**	**86.62%**	**47.83%**	**31.04%**	**2.71%**
	Strategy	**41.1**	**74.90%**	**95.21%**	**0.00%**	**0.00%**	**1.55%**	**39.97**	**72.69%**	**94.38%**	**0.00%**	**0.00%**	**2.39%**

deduced that both the recall and precision values are decreasing 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 is decreasing 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 impact [22]. This situation is depicted on Figure 3. Therefore, when the test suite evolves, mutants with 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 impact while leaving the number of equivalent mutants with impact constant. Perhaps some killable mutants will also have an impact due to the test suite evolution, but this is a minor number as the present experiment shows. Additionally, this is the case for equivalent mutants, which results in increasing the error rate. As a consequence, both recall and precision values are decreasing.

In practice, the decreasing trend explains why the effectiveness of the examined approaches is lower to 100% (subsection 4.2.2 presents the effectiveness results). 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.

In the columns "Precision" and "Recall" of Table 2, the respective values of these metrics are recorded per subject program and per employed method. The method "Statement" records the obtained results by the statement-based test suites. The "Approach A (CA)" and "Approach B (CB)" keep track of the results obtained by applying the classification process of Figure 1 by using the CA and CB classification schemes. Thus, it records the results of the test cases produced based on the studied strategies. These results indicate that the block test suites classify the live mutants on average with 47% precision value and 39% recall when using CA. In the case of CB, a 48% and 31% of precision and recall values is achieved. Here, it should be mentioned that recall and precision values of the examined strategies is 0 since all the killable mutants have been killed by applying the process of Figure 1.

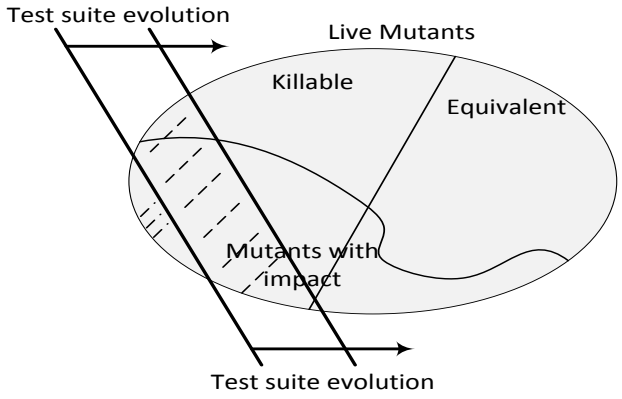

Figure 3. Classifying mutants (live) using the classification strategies. When tests are added, the killable mutants with impact are killed. Therefore, the percentage of killable mutants with impact is decreasing.

4.2.2 Approach Effectiveness (RQ2)

This section considers the relative effectiveness of the examined strategies. The results are summarized in Table 2. This table records the number of tests, the actual mutation scores, the percentage of killable mutants killed by the strategies, the precision and the recall values of the employed methods i.e. "Statement" and "Strategy", per classification scheme (Approach

A (CA) and approach B (CB)) per subject program. From the results of Table 2, 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 CA scheme achieves to kill a ratio from 87.51% to 99.25% of the mutants that can be killed by the test pool. The CB scheme achieve in the range of 89.53% to 98.66%. The effectiveness is lower than 100% due to mutants with no impact. Note that the proposed strategies aim only at mutants with impact.

In most of the cases CA scores better than CB. This can be explained by examining the methods recall and precision values. Generally, it can be deduced that between two classifiers the one that has higher recall values even with lower precision values, is generally more effective. However, in none of the studied subject a mutation score was 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.

5. DISCUSSION

The conducted study suggests that mutant classification strategies have a lower effectiveness when compared to testing by using all mutants. However, they have a decisive advantage which is their ability to isolate equivalent mutants [22] and [21]. Although the focus of the present study is to examine the effectiveness of these strategies, some conclusions regarding the equivalent mutants can be made. This is due to the employed tests which form a huge and comprehensive[1] test pool [8]. Thus, it is expected that only a small number of killable mutants should be live after their execution with this test suite. To this end, we measure the ratio, among the mutants that cannot be killed by any test of the test pool, of mutants with impact to the total number. These ratios represent an approximation of the percentage of equivalent mutants encountered by the strategies.

Generally, it is expected that a process based on a classifier with higher precision would be more efficient than one with lower precision. This is due to the more precise guidance that it provides towards killable mutants. The obtained results, confirm this argument on all the examined cases. Thus, CB is more precise than CA and hence it is expected to be more efficient. Indeed, the CB classification scheme encounters on average 14% such cases, while, CA encounters on average 17%. Recall that these percentages represent mutants with impact that cannot be killed. Hence, since mutants with impact are more likely to be killable than those without impact [22], it is expected that the exact ratios of the encountered equivalents mutants should be even smaller than the ones reported here. Overall, the conducted experiment suggests that, by using the proposed strategies, the manual work related to the identification of equivalent mutants can be greatly reduced (>20%).

5.1 THREATS TO VALIDITY

Generally, threats to the validity of the present experiment can be identified due to the use of the selected subjects. Thus, the representativeness of these subjects is questionable. Yet, all these subjects are benchmark programs which have been widely used in similar experiments [11]. The selected set of mutant operators introduces threats to the internal validity of this work. However, this set of mutants is a relatively large one since it is composed of 44 operators and involves all the language operators. Other threats could be attributed to the use of software systems. Additionally, the employed process for determining the killable mutants may influence the reported results. Though, the objectives of this experiment were to explore the relative effectiveness of the examined approaches compared to strong mutation based on the same test suites and tools. Therefore, it is believed that the indicated threats are not of such importance.

[1] The test pools were constructed based on a combination of techniques [8], including random, category-partition, all statements, all edges and all definition-use pairs.

6. 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 a small and killable at the same time, 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 can be identified on average. Other approaches aiming at identifying equivalent mutants employ program slicing [9] to assist the identification process. All the aforementioned techniques target on 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 as alternatives to the first order ones. Higher order mutation testing strategies such as [13] and [18] produce considerably less equivalent mutants, thus alleviating the problems that they are introducing. 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.

7. CONCLUSION AND FUTURE WORK

The present paper empirically investigates the use of dynamic strategies aiming at reducing the effects of equivalent mutants in mutation testing. 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 showed that the use of coverage impact can be beneficial towards the practical application of mutation. The proposed strategies were found to be effective in improving an existing test suite by reducing the effects of equivalent mutants. However, the strategies fail to kill the 5% of the killable mutants.

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) is decreasing. Hence, the guidance provided by the classification process is also decreasing.

Future work includes conducting additional experiments to statistically revalidate the findings of the present 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.

8. 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 Softw. Engg. 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. Softw. Eng. 37, 5 (September 2011), 649-678.

[12] Jia, Y. and Harman, M. 2009. Higher Order Mutation Testing. Inf. Softw. Technol. 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).