Evaluating Mutation Testing Alternatives: A Collateral Experiment

Marinos Kintis, Mike Papadakis and Nicos Malevris
Department of Informatics, Athens University of Economics and Business
Athens, Greece
{kintism, mpapad, ngm}@aueb.gr

Abstract— **Mutation testing while being a successful fault revealing technique for unit testing, it is a rather expensive one for practical use. To bridge these two aspects there is a need to establish approximation techniques able to reduce its expenses while maintaining its effectiveness. In this paper several second order mutation testing strategies are introduced, assessed and compared along with weak mutation against strong. The experimental results suggest that they both constitute viable alternatives for mutation as they establish considerable effort reductions without greatly affecting the test effectiveness. The experimental assessment of weak mutation suggests that it reduces significantly the number of the produced equivalent mutants on the one hand and that the test criterion it provides is not as weak as is thought to be on the other. Finally, an approximation of the number of first order mutants needed to be killed in order to also kill the original mutant set is presented. The findings indicate that only a small portion of a set of mutants needs to be targeted in order to be killed while the rest can be killed collaterally.**

Keywords- mutation testing; higher order mutation; weak mutation; collateral coverage

I. INTRODUCTION

Software Testing is an essential activity of software engineering, as it provides the basic means of establishing some level of confidence in the correctness and reliability of the software under test. The "lowest" level of testing is unit testing, which is designed to assess the quality of individual units. Unit testing is becoming increasingly important in the development of software, as it promises to expose faults early in its development life cycle.

In general, software testing requires the generation and execution of the software under test with actual test data. The data generation process is often driven by formal coverage criteria, which impose specific requirements on the test values. The quality of a test set, therefore, is measured by its ability to exercise certain program features imposed by the adopted test criterion. The need to exercise different elements of a program gave rise to the definition of various test criteria, each of which imposed their own requirements on a test set.

Mutation testing is regarded as one of the most effective techniques for unit software testing. Mutation can be used to produce high quality test sets that are believed to achieve a high level of testing for the software under examination. This is supported by the recent experiments in [1] and [2] which present evidence that mutation induced faults resemble to real ones, thus providing a good estimation of the fault revealing ability of the corresponding test sets.

Despite its effectiveness, mutation is considered to be extremely expensive for practical use, primarily, due to the enormous number of candidate mutants it considers, which are estimated to be in the order of the square of the code lines of the program under test [3]. To intensify the problem, many of the generated mutants may be functionally equivalent, usually requiring additional manual effort. To resolve these issues, a number of techniques has been introduced aiming at reducing the application cost of mutation without affecting its effectiveness [4], [5].

The main idea behind these approaches is to select a small representative subset of all the candidate mutants based on which mutation could be applied. Various similar considerations have appeared in the literature [5], such as the random mutant sampling and the selective mutant operators methods [5], [6]. Experimentation on these suggestions shows that a remarkable reduction on the generated mutants can be achieved. Recently, a new mutation testing alternative approach has been suggested, based on the notion of higher order mutants [7], [8]. It suggests that mutants could be constructed based on the injection of two or more mutants at a time. Preliminary experiments have proved promising. Although benefits utilising these approaches have been identified, they have not been thouroughly investigated. This constitutes the main objective of the present paper.

In the present research work, an experimental study was conducted with the aim of suggesting new more effective and efficient second order mutation testing strategies and an attempt to combine first order and second order mutation approaches was undertaken. It must be noted that in the examined strategies weak mutation was also considered. Moreover, a study of their behaviour is attempted by evaluating their collateral coverage achieved with respect to strong mutation. The term collateral coverage is used when aiming at criterion, say *a*, one also calculates how much of criterion *b* can also be fulfilled. This idea has been investigated in depth by [9] where the collateral coverage of data flow criteria when aiming at branch testing was presented. This paper introduces a set of four second order strategies and examines them empirically based on fifteen selected test subjects. The experimental results suggest that both the second order mutation testing strategies and weak mutation are quite effective as they result in a relatively high strong mutation coverage score (96% - 98%) while introducing considerably less equivalent mutants compared to strong mutation.

Furthermore, the present experimental study also compared strong mutation against the examined alternatives in order to assess their strength. The most interesting aspect of the obtained results is that weak mutation is, in general, incomparable with strong as they both achieve a 97% collateral coverage on each other. This fact indicates that weak mutation is a quite effective testing criterion.

Finally, given a set of first order mutants, an approximation of the lower bound of the cardinality of a subset of mutants that need to be covered in order to also cover the original set, was attempted. To achieve this, the concept of a set of *disjoint* mutants is introduced. Two mutants are considered to be *disjoint* if the test sets that kill them are likely to be disjoint. Thus, the sets of test cases that kill each mutant of a set of *disjoint* mutants will be as disjoint as possible. The corresponding findings suggest that only a small number of mutants of a given set of first order ones is adequate to cover the whole set.

The rest of the paper is organized as follows. Section II and III provide details about the various studied mutation testing approaches. In Sections IV and V, the experimental regime and the obtained results of the conducted study to a set of programs are given respectively. Section VI discusses some threats to the validity of the conducted experiment while Section VII presents some related work. Finally, in Section VIII conclusions along with a discussion of the results obtained are analyzed.

II. Background

Mutation testing is a well known fault-based technique which was established and introduced by [10] and [11]. In general, this technique requires the injection of fabricated faults into the software's source code and measures the exposition ratio of those faults in order to assess the quality of the test cases. Naturally, the technique's effectiveness and application cost highly depends on the size and the quality of the injected fault set. This section introduces some underlying terminology, concepts and techniques used by the presented experimental study.

A. Mutation Testing Criterion

The injected faults of mutation are formed by syntactically altering the examined source code. This results in the production of different mutant program versions of the examined code, called mutated versions. These syntactic changes are performed based on a set of rules called mutation operators. Because of the potentially infinite number of such rules researchers have restricted them to include only simple ones [11]. Various strategies have also been proposed in an attempt to make the technique more practical. The performed experiment examines this issue.

The technique is applied by assessing the produced test cases based on their ability of exposing mutation based faults. This is done by executing all the candidate sets of mutated versions with the aim of distinguishing them from the original (non-mutated) one. Executing the mutants with test cases may result in distinguished outputs signifying a mutant killed or alive in the contrary situation. Furthermore, a mutant is called equivalent if the inexistence of such test cases holds. Measuring the testing quality based on the ratio of killed mutants results in a mutation based metric. This metric is usually called mutation score and it is defined as the ratio of killed mutants over the total number of non-equivalent ones.

B. Weak Mutation

In order to reduce the computational expenses of mutation, an approximation technique called weak mutation was introduced [12]. Weak mutation tries to reduce the computational resources needed to perform mutation by avoiding the complete execution of the mutants and the original program. To this end, weak mutation relaxes the condition that needs to hold for a mutant in order to be killed by comparing the internal states of the mutated and the original programs immediately after the execution of the mutated components. A mutated component can be an expression or a statement of the source code of the program under test or a basic block of its control flow graph [13].

Based on this approach, execution savings rely on the reduced execution traces. It has been found [13] that at least 50% of the execution time can be saved by using the weak mutation alternative.

Weak mutation can be implemented in various ways according to the adopted definition of a mutated component. The findings of [13] suggest the application of weak mutation using statement or basic block components rather than expression ones.

In the present study it was decided to use weak mutation as a common basis for comparison between the methods, since it is regarded as a practical alternative to strong.

C. Mutant Combination Strategies

Recently, a new mutation testing alternative approach has been proposed [7], based on the notion of second order mutants. According to this, given a set of first order mutants, a reduced set of second order ones is produced by combining the mutants of the original set into pairs. Thus, every mutated program is embedded with two mutants at a time and every mutant is contained in at least one second order mutated program. As a consequence, the number of mutants obtained is close to half the size of the original mutant set [7].

Various strategies of how and which mutant statements should be combined can be constructed. In the study of [7] three strategies were proposed, namely "RandomMix", "DifferentOperators" and "LastToFirst". The first one forms the required pairs by randomly selecting them. The second one constructs the pairs by using different mutant operators. In the third one, mutants are constructed according to the order of their production.

The experimentation with these strategies resulted in a high reduction on the number of candidate and equivalent mutants. Additionally, evidence was provided that there may be a small loss on the quality of the second order test suites [14]. It is this conclusion that is investigated in the present study by using the best of the already suggested strategies, as was concluded in [14] and the newly introduced ones here. This is performed in order to determine the impact on the testing quality of the second order mutation strategies in comparison to that of strong and weak mutation.

III. Second order mutation testing strategies

The primary purpose of this paper is to propose new approximation techniques that will reduce the cost of mutation while maintaining a reasonable level of test effectiveness. To this end, various second order mutation testing strategies were developed following the mutant combination approach.

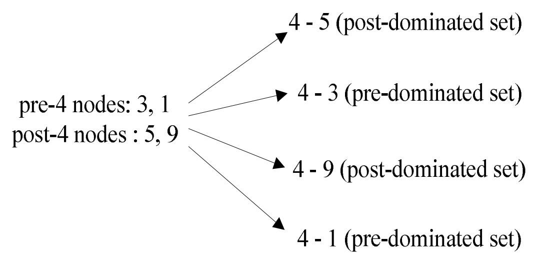
The introduced strategies are based on dominator analysis [15] of the programs under test. Dominator analysis is used to find related basic blocks, of which, if one is covered, then the rest are guaranteed to be covered too. Thus, if a basic block A dominates a basic block B, then a test execution cannot reach B without going through A and vice versa. The idea behind the use of dominator analysis was to increase the coupling chances between the combined mutants. To achieve this, the mutant pairs were forced to be selected from node pairs that were found to have a domination relation. The level of coupling between the combined first order mutants varies, from relaxed coupling to strict coupling, according to the domination relation being enforced. In the relaxed coupling case, the pre and post dominator relations are being used, so the combined mutants will only have a small chance of interacting with each other as there is no guarantee that they will lie on the same execution paths. In the strict coupling case, the dominator relation is being employed and as a consequence the combined mutants will always be executed together. This means that if a test execution reaches one mutant of the combined pair, then the other mutant is guaranteed to be reached too.

The proposed strategies fall into two categories, the *Second Order Strategies*, which generate only second order mutants and the *Hybrid Strategies*, which generate both first and second order mutants.

A. Second Order Strategies

The *Second Order Strategies* category includes strategies that generate a set of second order mutants from a specified set of first order ones. This category contains two strategies, the Relaxed Dominator strategy (*RDomF*) and the Strict Dominator strategy (*SDomF*). These strategies were chosen as they were found to be the most effective ones, originally proposed in [16] and [17].

The *RDomF* strategy employs a relaxed level of coupling, as it uses the pre and post domination relations. This strategy starts by constructing the sets of pre and post dominating nodes (*pre-n* and *post-n*) for each node (*n*) of the program's control flow graph. Then, the *RDomF* combines the first mutant of node *n* with the first unselected mutant of the first *post-n* node, the second with the first unselected of the first *pre-n* node etc. The process continues until all mutants of node *n* have been used at least once. It must be noted that the *RDomF* strategy uses each mutant as few times as possible. Fig. 1 illustrates the sequence of node pairs, generated for node 4, from which the second order mutants will be created.



pre-4 nodes: 3, 1
post-4 nodes : 5, 9
4 - 5 (post-dominated set)
4 - 3 (pre-dominated set)
4 - 9 (post-dominated set)
4 - 1 (pre-dominated set)

Figure 1. The sequence of node pairs, for node 4, from which the first order mutants will be combined.

The precondition for the generation of these node pairs is that both nodes must have unused mutants. In a different case the next available pair that meets this requirement will be returned. The sequence shown will be repeated until all mutants of node 4 are combined.

The *SDomF* strategy employs a strict level of coupling by restricting the selection of mutants among dominated node pairs. Consequently, it is expected that both or none of the mutants will be executed by the test cases. This strategy uses the same selection approach applied only to the dominated node sets as described above. It is apparent that by using such a technique mutants remain unused as they refer to nodes not being dominated. For these mutants the *RDomF* strategy is applied.

B. Hybrid Strategies

The *Hybrid Strategies* category owes its name to the fact that the generated sets of mutants, obtained by the application of its strategies, include both second order and first order mutants. By doing so, this category tries to combine the benefits of both mutation approaches, i.e. the equivalent mutant reduction promised by the use of second order mutants and the effectiveness of the first order ones.

During the evaluation of the *SDomF* strategy it was observed that killing the second order mutants that were created by combining the first order ones of the dominated basic blocks, resulted in killing most of these first order mutants. Thus, it was found interesting to investigate the effectiveness of a hybrid approach that would include in its generated set of mutants the second order mutants of the dominated basic blocks and a portion of the remaining first order mutants of the original set.

The strategies of this category share the same scheme for the generation of the second order mutants and differ on the number of the first order mutants included in the final set of mutants. Fig. 2 depicts the mutant generation process used by the *Hybrid Strategies* category. Initially, the first order mutants of the dominated basic blocks are combined into second order ones, as in the case of the *SDomF* strategy, and then a randomly selected subset of the remaining first order mutants, that is the mutants of the non-dominated basic blocks, is included in the generated set of mutants. The *Hybrid Strategies* category includes two strategies, the *HDom(20%)* and the *HDom(50%)*, which randomly select the 20% and the 50% of the remaining first order mutants.

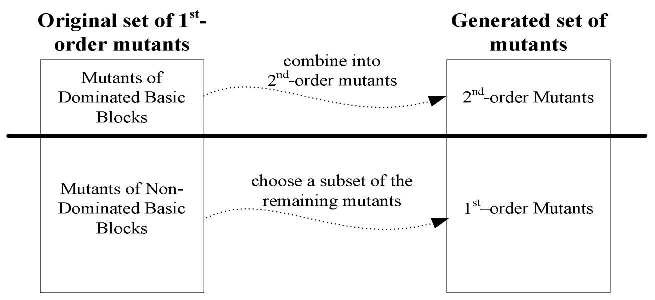

Original set of 1st-order mutants
Generated set of mutants
Mutants of Dominated Basic Blocks
combine into 2nd-order mutants
2nd-order Mutants
Mutants of Non-Dominated Basic Blocks
choose a subset of the remaining mutants
1st–order Mutants

Figure 2. The mutant generation process of the *Hybrid Strategies*.

IV. EXPERIMENTAL SET UP

Although the initial proposition of second order mutants was based on empirical observations and results, there is much to be done in order to gain useful insights about their application benefits. This is the subject of the present experimental study. For the purposes of this study three experiments were conducted. The first experiment was performed in order to investigate the effectiveness of the proposed second order mutation testing strategies as opposed to strong mutation. The second experiment compared strong mutation with the presented approximation techniques in order to evaluate their strength. Given a set of first order mutants, the third experiment tried to quantify the number of *disjoint* mutants. Two mutants are considered to be *disjoint* if the test sets that kill them are likely to be disjoint. In this sense, the sets of test cases that kill each mutant of a set of *disjoint* mutants will be as disjoint as possible. The mutants of this set, by definition, drive the data generation process as they need to be specifically targeted in order to be killed. This subset of mutants is very important because the test sets derived to kill them will also kill the original ones.

A. Subject Programs

The presented experimental study was based on a set of fifteen Java program test units, the majority of which were chosen from those used in [7] and [18]. As this study focuses on strategies mostly used for unit testing, the size of the programs is considered irrelevant and should not affect the generality of the results. However, this matter is left open for future research. Table I presents the number of executable lines of code and the number of the test cases used for each experimental subject. Sub-section D includes details about the construction of these test sets.

B. Supporting Tool

The three experiments were performed based on an automated framework for producing second order and weak mutation mutants for java [16]. The framework uses the mujava mutation testing tool [19] for the creation of first order mutants (employing all method-level operators) and generates weak and second order ones based on the strategies described in the previous section. Weak Mutation was implemented based on the *ST-WEAK/1* approach, suggested by [13]. According to this, the internal states of the original and the mutated programs are compared immediately after the first execution of the mutated statement.

TABLE I. SUBJECT PROGRAM UNITS

Test Program (Unit)	**Data of Experimental Subjects**	
	Lines of Code	***Test Cases***
Power(power)	13	128
printPrimes(printPrimes)	25	107
trashAndTakeOut(trash)	15	110
trashAndTakeOut(takeOut)	12	261
Trityp(Triang)	44	589
Find(partition)	44	273
Bisect(sqrt)	23	49
Cal(cal)	22	360
twoPred(twoPred)	14	415
InsertionSort(sort)	14	226
Max(max)	11	491
Euclid(gcd)	14	368
Tcas(alt_sep_test)	31	1560
Triangulo(getTipo)	47	589
TriType(Triangle)	43	601
Total	372	6127

C. Research questions

The aim of the present experimental study can be summarised in the following research questions:

- How effective are the selected second order mutation testing strategies and weak mutation?
- What are the benefits of the application of the second order mutation testing strategies and weak mutation compared to strong in terms of the produced equivalent mutants?
- Which is the most appropriate second order mutation testing strategy for general use?
- What is the collateral coverage level that strong mutation achieves when compared against its alternatives?
- What proportion of a given set of first order mutants is *disjoint*?

D. Experimental Procedure

The first experiment, which answers the first three research questions, was initiated by independently applying each one of the second order and weak mutation testing strategies on all test subjects and then a comparison was made based on strong mutation, which was done by recording the strong mutation score achieved by the constructed test sets per each strategy. The second experiment, which answers the fourth research question, follows the previous approach in the opposite order, i.e. first, strong mutation was applied to all test subjects and then a comparison was performed with each approximation technique.

In order to produce realistic results and avoid any bias involved in the test selection, the following procedure was applied. A relatively large and high quality set of test cases, namely “data pool”, was build for each test subject. This was based on three streams of data a: random test selection, b: the tests used in [7] and c: a mutation test generation framework suggested in [18] for generating test cases according to various automated structural testing tools. The data pool was used as the basis for simulating the population of all candidate test sets. All test cases contained in the data pool were executed against candidate mutant sets for each approach in order to identify possible equivalent mutants. The mutants that were not killed by any test case were eliminated by the candidate mutant sets as being equivalent. Because of the high quality test cases contained in the data pool it is anticipated that the identified mutants are more likely to be equivalent ones. However, determining the exact number of equivalent mutants is a tedious task and this is not the purpose of the present experiment. It must be noted that the produced equivalent mutants sets contain the maximum possible number of equivalent mutants and thus, in the results obtained their actual number should be even smaller than the one reported here.

In order to produce test sets specific to each mutation strategy, random selection of test cases from the data pool was performed until all candidate mutants were killed. All selected tests that failed to kill additional mutants according to a chosen strategy were eliminated from the test set. This was done because these tests while not contributing to the chosen strategy may though have a collateral impact on strong mutation. To eliminate any bias introduced by a particular test set, 10 separate test sets for each unit and for each variant were generated. For the *Hybrid Strategies* category, the experiment was repeated 5 times in order to obtain more reliable results.

Strong mutation was used as a collateral comparison reference to inter-evaluate the effectiveness of the different strategies. Additionally, the achieved cost reductions of the various mutation alternatives were measured by the number of produced equivalent mutants per each strategy.

The last conducted experiment tries to answer the ultimate research question, which refers to finding the *disjoint* proportion of a given set of first order mutants. Initially, all test cases contained in the data pool were executed against candidate first order mutant sets in order to identify the test set that killed each mutant. It must be mentioned that the smaller the size of this test set, the harder the corresponding mutant is to be killed. As in the previous experiments, mutants that were not killed by any test case were eliminated from their corresponding sets as being equivalent. Next, for each set of mutants, a subset was selected according to two requirements. The first is that the test sets that killed the selected mutants should be of minimum size, i.e. the included mutants should be the hardest to kill, and the second requirement states that these test sets should be as disjoint as possible. This process will produce a set of *disjoint* mutants for each test subject the coverage of which implies the coverage of the original set of mutants.

V. Experimental Results

This section presents the findings of the conducted experimental study. The analysis will be divided in three subsections according to the previously described experiments.

A. Approximation Techniques versus Strong Mutation

The results presented in this subsection refer to the first experiment, which tries to unveil details about the benefits of using second order mutation testing strategies or weak mutation instead of strong mutation. The benefits of utilizing alternative strategies rather than performing strong mutation directly need further investigation; however, it can be argued that an undisputable benefit is the effective handling of equivalent mutants by the second order strategies. This is the major contribution of second order mutation testing strategies as they alleviate the production of such mutants. Handling the equivalent mutants is not an easy task, as it forms an undecidable problem [20]. Thus, deciding mutant equivalence requires manual work which is quite significant as stated in [21]. In view of this, second order strategies are quite beneficial as they produce a minor number of such mutants, minimising the manual effort of the method.

Table II summarizes the produced equivalent mutant sets for all considered strategies (table columns) for all the test units. From these results it is obvious that second order mutation testing strategies produce by far less equivalent mutants than what strong mutation does. Weak mutation also produces significantly less equivalent mutants than its rival strong, but more than what the second order strategies produce, except from *HDom(50%)*. It must be mentioned that from the 102 equivalent mutants the *HDom(50%)* strategy produces, only 19 are second order ones.

Fig. 3 illustrates the achieved equivalent mutant reduction for all considered strategies and selected units with respect to strong mutation. This reduction varies from 65.5% for *HDom(50%)* to 86.8% for *SDomF*. In all cases, the reduction exceeds 50% indicating that the manual effort for the detection of the equivalent mutants is relatively shorter. To better understand the implications of this, given that an equivalent mutant requires on average fifteen minutes of manual work (performed by the tester) in order to be detected [21], the amount of time required to identify the equivalent mutants of the fifteen relatively small program units of the current experiment is approximately 75 working hours. By employing either weak mutation or the *SDomF* techniques to the same program units the identification process will require approximately 20 and 10 working hours respectively. Although the amount of time required by the alternative strategies is not negligible, the achieved savings are nonetheless substantial.

Besides the application benefits of the mutation testing strategies, the current experimental study tries to answer the question of how effective are the second order mutation testing strategies compared to strong mutation. In other words, the study tries to measure the impact on testing quality when employing a second order strategy. To this end, tests were purposely generated according to the selected strategies and executed in turn against the strong mutation mutants in order to determine the collateral level of coverage per each strategy.

In order to visualize and make a more direct comparison between the strategies and strong mutation, the obtained results are presented in a graph in Fig. 4. The figure shows the strong mutation scores achieved by the produced tests of the considered strategies. From this graph it is obvious that the *HDom(50%)* strategy, with an average mutation score of 98.25%, is the most effective in collaterally covering strong mutation, followed by the weak mutation approach with an average score of 97.03%.

One other contributing point that affects the application cost of testing strategies in general, usually considered in empirical studies such as [2], is the number of the required test cases by the used techniques. This is due to the fact that tests will be used for revealing the faults of the tested software. To consider this fact, Fig. 5 presents the average test set sizes produced by the examined strategies based on the conducted experiment. The graph suggests that a reduction of approximately 30% is achieved by the second order strategies. Surprisingly, weak mutation requires more tests than its rival strong, an issue that will be discussed in the next subsection.

Conclusively, this experiment suggests that considerable effort savings can be achieved by both second order and weak mutation strategies as opposed to strong mutation. The trade-off results in a 1.75% loss of test effectiveness to gain a 65.5% equivalent mutant reduction for the *HDom(50%)* strategy and an effectiveness loss of 3% with a gain of 73% in equivalent mutant reduction for weak mutation. These results suggest that it is possible to perform mutation with reasonable resources.

B. *Strong Mutation versus Approximation Techniques*

This subsection presents the results of the second experiment, which investigates the strength of the examined mutation approximation techniques. To this end, tests were purposely generated to satisfy strong mutation and executed in turn against the approximation techniques in order to determine the achieved level of coverage. It is expected that test sets generated to satisfy strong mutation will be very close to satisfying second order mutation [22] and consequently the proposed second order strategies, nevertheless, there is scant experience about the level of coverage that will be achieved for weak mutation.

Table III presents the findings of this experiment. As expected, test sets designed to satisfy strong mutation were very successful at satisfying second order strategies. The most interesting aspect of this table is that the achieved weak mutation score is 96.9%, which is equal to the strong mutation score achieved by weak mutation (see Fig. 4). This fact justifies the increased number of the required test cases (with respect to strong mutation) that are needed to cover weak mutation (see Fig.5) and indicates that weak mutation is not as weak as might thought to be.

To better investigate these findings, the mutants of weak mutation that remained alive after the execution of the strong mutation adequate test sets were examined. These mutants were found to refer to loop statements or to statements inside a loop block. By taking this fact into consideration, the inability of strong mutation to completely cover weak can be justified by the weak implementation approach adopted in the supporting tool. As mentioned in the previous section, the implementation of weak mutation was based on the *ST-WEAK/1* approach [13], which compares the internal states

TABLE II. PRODUCED EQUIVALENT MUTANTS PER STRATEGY

Test Units	**Mutation Testing Approaches**					
	Strong	***Weak***	***RDomF***	***SDomF***	***HDom(20%)***	***HDom(50%)***
Total	296	80	43	39	55	102

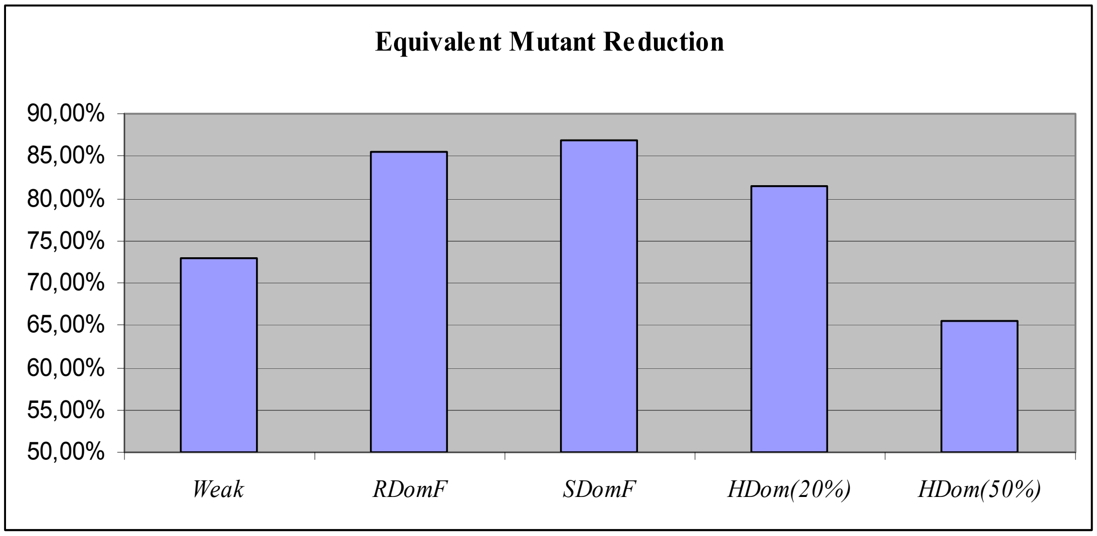


Figure 3. Achieved equivalent mutant reduction per strategy with respect to strong mutation.

of the original and the mutated programs immediately after the first execution of the mutated statement in order to decide if the corresponding mutant is killed or not. Thus, the condition that needs to hold in order to (weakly) kill a mutant that affects a loop statement or a statement inside a loop block can be difficult to satisfy; while these statements, by their definition, are expected to be executed more than once, this condition must strictly hold during their first execution.

C. Approximating the Disjoint Proportion of a Given Set of First Order Mutants

The results presented in this subsection refer to the third experiment, which tries to identify the proportion of a given set of first order mutants that are *disjoint*. The quantification of the number of the *disjoint* mutants contained in a specified set of first order ones is very important because it provides an approximation of the lower bound of the cardinality of the subset of mutants that need to be killed in order to reciprocally kill the original set. Table IV presents the results of this experiment for each test subject. The values of the second and third columns of the table represent the percentages of the hard-to-kill first order mutants of each test subject that are *disjoint* and that of all the first order mutants that are *disjoint*, respectively. In this experiment, a hard-to-kill mutant is considered to be a mutant that can only be killed by less than 15% of the total test cases of the test subject it refers to. It must be mentioned that the empty entries in the second column of the table are due to the inexistence of such mutants in the corresponding mutant sets of these test units.

By their definition and construction these *disjoint* sets of mutants satisfy two conditions. The first is that the test sets that kill the contained mutants must be as disjoint as possible and the second requires the size of the test set that kills each included mutant to be as small as possible. Thus, the included mutants will be the most difficult to kill, while the test sets that kill them are as disjoint as possible.

The results shown in the second column of Table IV suggest that from all the hard-to-kill mutants of a given set of first order ones an approximate 35% of them is adequate for also evaluating the rest of them. That is, the test cases that kill the 35% of the hard-to-kill mutants of a given set can also kill the remainder hard-to-kill mutants of that set.

According to Table IV, the percentage of all the first order mutants that are *disjoint* varies from 3.11% to 19.68%, with a mean value of 9.03%. These results indicate that only a 10% of the first order mutants of the examined test subjects needs to be covered in order to also evaluate the whole set. The remaining 90% can be killed collaterally, i.e. without being aimed at. In order to validate these results, test sets were designed to cover the *disjoint* subsets of the corresponding first order mutants of each test subject and executed in turn against the original sets of mutants. The findings of this experiment validated the aforementioned arguments, since the original sets of first order mutants were successfully killed.

The findings of this experiment justify the high collateral strong mutation coverage achieved by the second order mutation testing strategies (see Fig. 4), since most of the mutants of a given set of first order ones (65% of the hard-to-kill and 90% of the total) can be killed collaterally.

TABLE III. MUTATION SCORES ACHIEVED BY STRONG MUTATION ADEQUATE TEST SETS

Test Units	Mutation scores per approximation technique				
	RDomF	***SDomF***	***HDom(20%)***	***HDom(50%)***	***Weak***
Total	99.94%	99.99%	99.91%	99.91%	96.90%

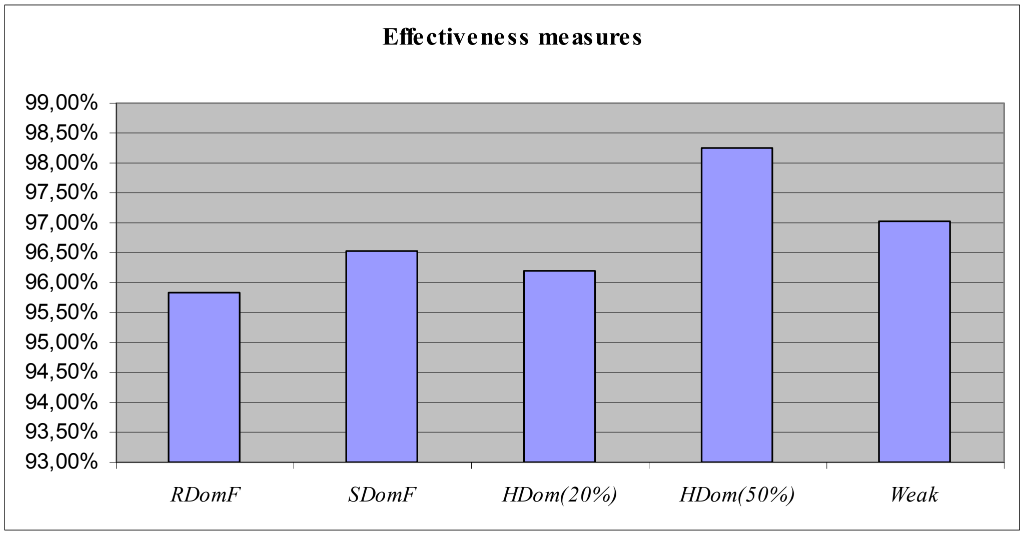


Figure 4. Achieved strong mutation scores per considered strategies.

VI. Threats to Validity

All empirical studies have threats to their validity. One possible threat to the present experiment's internal validity i.e. uncontrolled factors influencing the experiment, can be related to the use of the software test systems. To reduce this threat a well established and referenced mutation system such as mujava [19], was used. Additionally, a manual cross verification of the obtained results was undertaken. Another possible threat may be related to the selection of mutation as a fault detection method. As the considered strategies are alternatives to strong mutation and the comparison was based on it, this threat is limited within mutation itself. Thus, the threat is eliminated. Finally, the most obscure question to be answered is: do the obtained results represent a typical testing situation? The answer to this question can only be obtained by the extensive use of these strategies.

VII. Related Work

Higher order mutation (introducing more than one mutant at the same time) was initially introduced and examined in the context of mutant coupling effect [22]. Higher order mutants were considered in that study as representative of complex faults. In this study the ability of simple mutants to reveal complex ones, i.e. second or higher order mutants, was also measured. The results obtained showed that simple mutants can produce tests capable of killing the majority of complex ones.

The idea of using higher order mutants for reducing the effort of mutation was introduced by [7], as previously detailed. Based on the suggestion of [8] who introduced the concept of subsuming mutants (a more powerful higher order mutant than the ones that is composed of) it becomes clear that by replacing all simple ones with those less in number however more powerful, considerable savings can be recorded. These higher order mutants are constructed by formulating the mutant construction problem as a search based optimization one and using evolutionary approaches in order to solve it. Additionally, in [23] the mutants were constructed with the aim of both producing hard to kill mutants and also syntactically similar to the original program and hence simulate realistic faults. Both the above studies try to reduce the cost involved by the mutation testing technique. In [14] an empirical study investigating the fault detection effectiveness of the mutant sampling and some second order strategies was conducted. The results suggest that there should be on average a 10% loss on the fault detection ability of the various second order strategies while considerable savings are recorded.

In [13] an attempt to compare strong, weak and firm mutation testing strategies was undertaken. Their study suggests that weak mutation can form a cost effective alternative to mutation and also that it should be applied by using the statement or the basic block component strategy. Similar experiments were undertaken in the context of selective mutation [6], [24]. The results of these approaches suggest that it is possible to perform mutation with only a small subset of mutation operators while keeping over a 99% of the full mutation effectiveness. Recently, in [24] various selective mutation approaches were compared to each other along with the random mutant sampling technique. The experiment showed that random mutant selection has the same effectiveness as the selective approaches.

Testing expenses highly depend on their inherent level of automation. Following this observation, automated tools have been developed for performing mutation. One such tool that produces and executes mutated programs for java, is the mujava tool [19] which has been used in the present study. The generation of test cases has also been partially automated. A tool called Godzilla [25] was implemented for producing mutation tests for Fortran programs. Recently [26]

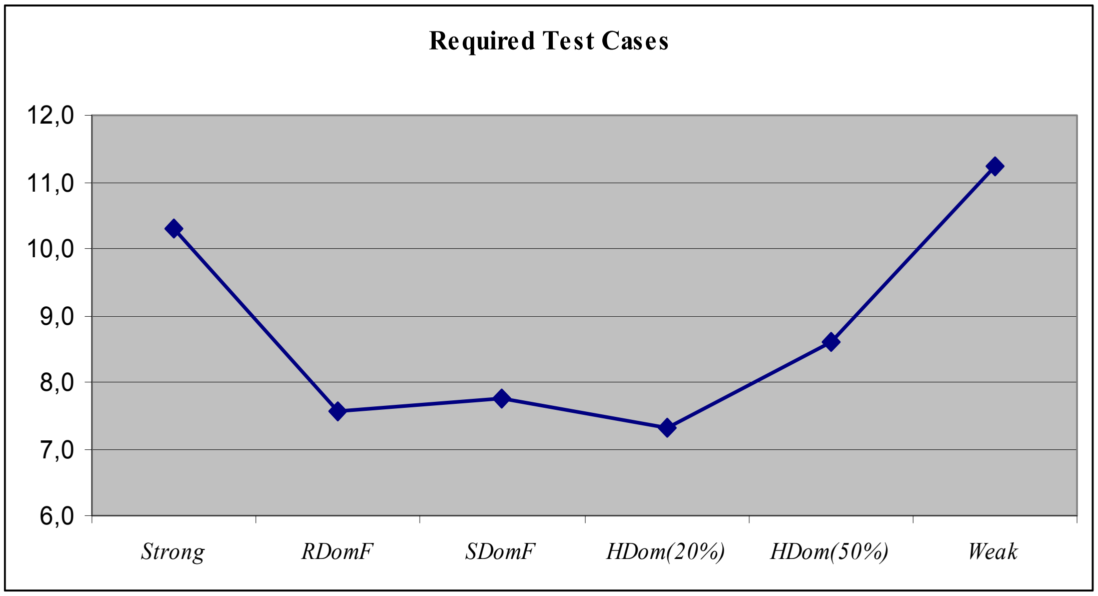


Figure 5. Required test sets sizes of the considered strategies.

TABLE IV. Disjoint Proportion of the First Order Mutants of each Test Subject

Test Units	Disjoint proportion of first order mutants	
	Hard to Kill Mutants	*All Mutants*
J-Unit1	75.00%	9.33%
J-Unit2	-	3.52%
J-Unit3	42.50%	14.16%
J-Unit4	40.00%	9.07%
J-Unit5	11.20%	11.30%
J-Unit6	45.65%	3.11%
J-Unit7	36.61%	4.87%
J-Unit8	9.83%	5.56%
J-Unit9	17.89%	19.68%
J-Unit10	-	3.24%
J-Unit11	-	9.23%
J-Unit12	98.53%	10.25%
J-Unit13	20.99%	13.99%
J-Unit14	15.17%	8.35%
J-Unit15	11.22%	9.80%
Total	35.38%	9.03%

an approach that transforms the mutant killing problem to a covering branches one and uses paths for killing the mutants has been proposed. Based on this suggestion, another approach [18] that uses mutant schemata and existing automated tools for structural testing has been shown to be effective in the context of mutation.

The major contribution of the higher order mutation testing approaches is the reduced number of introduced equivalent mutants. Equivalent mutants' identification constitutes a major overhead of mutation testing approach [21]. As proposed in [21] and [27] equivalent mutants can be heuristically identified by measuring their impact on the mutant programs. This impact can be quantified either by using dynamic invariants or recording the traversed paths. By targeting those mutants with a higher level of impact, a good measure of the adequacy of the test suite is established, while limiting the number of considered mutants and the equivalent ones. Another approach to tackle the equivalent mutants problem is due to [28]. In this approach program slices are used in order to heuristically indentify equivalent mutants or to assist the tester in detecting them.

VIII. Conclusions and Future Work

This paper presents an experimental study of mutation testing strategies. The motivation behind this work was to provide methods able to reduce the effort involved when performing mutation. One of the basic reasons that characterise mutation as an expensive technique is the presence of equivalent mutants. Second order mutation has the advantage of producing far less equivalent mutants [7], [14] compared to mutation. Thus, the employment of second order mutation may be beneficial on two counts: a) it may reduce the number of the generated equivalent mutants and b) it may provide the basis for achieving a high collateral coverage for strong mutation. Based on these arguments, a set of new second order mutation testing strategies were developed, some of which try to jointly combine second and first order mutation approaches. The originality of these strategies is founded on the internal structure of the program under test for constructing mutant pairs. Specifically, the proposed strategies make use of the dominance relation between program nodes in order to construct mutant pairs with a high chance of coupling.

For the purposes of the conducted study a series of three experiments was performed.

The first experiment tries to evaluate the use of second order and weak mutation testing strategies. The experiment suggests that they both constitute a viable alternative for mutation by establishing considerable effort reductions without greatly affecting the test effectiveness. The effort savings are based on the equivalent mutant reduction which varies from 65.5% for the *HDom(50%)* strategy to 86.8% for the *SDomF* strategy. The amount of the sacrificed effectiveness of the examined approaches varies from 4.2% for the *RDomF* strategy to 1.75% for the *HDom(50%)* strategy.

The second experiment tries to investigate the strength of the examined approaches by comparing strong mutation against them. The results suggest that test cases that were designed to satisfy strong mutation were also able to achieve a high level of coverage for second order strategies (nearly 100%); however, they achieved a weak mutation score of approximately 97%. This level of coverage is identical to the strong mutation score achieved when aiming at performing weak mutation. This renders the incomparability of the two criteria. Additionally, as collateral coverage can be seen as a measure of the testing criteria effectiveness [29], there is not a clear winner between weak and strong mutation.

Given a set of first order mutants, the last experiment tries to approximate the minimum size of a subset of mutants the coverage of which is sufficient in order to cover the original set. The findings of this experiment indicate that most of the first order mutants of a given set can be collaterally covered. This means that a smaller number of mutants need to be directly targeted in order to be killed. The proportion of these mutants is approximately 10% of the mutants of the original set. This measure for the hard to kill mutants is approximately on average 35%. The determination of such a set of mutants, without generating and executing the whole mutant set, forms a difficult and open to research problem. In the literature, this problem is usually tackled with the selective mutation testing approaches [5], [6]. It is noted that the present study is performed on a selective set of mutants [6]. Thus, the high level of collateral coverage between the mutants and the hard to kill ones, indicates that there is a lot of space for improvement for the cost reduction techniques, as they should target only a small portion of these mutants (35% of the hard to kill and 10% of the total).

From the conducted experiment it becomes evident that second order mutation testing strategies are in general quite effective in collaterally covering strong mutation. The best choice appears to be the *HDom(50%)* strategy as it was found to be the most effective, recording approximately only a 1.75% average loss of test effectiveness. It must be mentioned that the *HDom(50%)* strategy belongs to the *Hybrid Strategies* category, which combines second order and first order mutation approaches.

In future, series of experiments involving more and larger test programs and additional strategies will be considered. Furthermore, experimental comparisons between higher order and selective mutation testing approaches e.g. [6] are planned. Although the present study of the collateral behavior of second order strategies has proved promising, an investigation of which criterion should one aim first in order to collaterally cover the other and the benefits of this approach is needed.

ACKNOWLEDGMENT

This work is supported by the Basic Research Funding (PEVE 2010) program of the Athens University of Economics and Business.

REFERENCES

[1] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is Mutation an appropriate tool for testing experiments?," in Proceedings of the 27th international conference on Software Engineering (ICSE'05), St Louis, Missouri, 15-21 May 2005, pp. 402-411.

[2] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using Mutation Analysis for assessing and comparing testing coverage criteria," IEEE Transactions on Software Engineering, vol. 32, no. 8, pp. 608-624, August 2006.

[3] A. T. Acree, T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Mutation Analysis," School of Information and Computer Science, Georgia Institute of Technology Georgia Institute of Technology, Atlanta GA, 1979.

[4] A. J. Offutt and R. H. Untch, "Mutation 2000: uniting the orthogonal," in Proceedings of the 1st Workshop on Mutation Analysis (MUTATION'00), published in book form, as Mutation Testing for the New Century. San Jose, California, 6-7 October 2001, pp. 34-44.

[5] Y. Jia and M. Harman, "An analysis and survey of the development of Mutation Testing," IEEE Transactions of Software Engineering, vol. 99, 2010, in press.

[6] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf, "An experimental determination of sufficient mutant operators," ACM Trans. Softw. Eng. Methodol., vol. 5, no. 2, pp. 99-118, 1996.

[7] M. Polo, M. Piattini, and I. García-Rodríguez, "Decreasing the cost of Mutation Testing with second-order mutants," Software Testing, Verification and Reliability, vol. 19, no. 2, pp. 111-131, June 2008.

[8] Y. Jia and M. Harman, "Higher Order Mutation Testing," Journal of Information and Software Technology, vol. 51, no. 10, pp. 1379-1393, October 2009.

[9] N. Malevris and D. F. Yates, "The collateral coverage of data flow criteria when branch testing," Information and Software Technology, vol. 48, pp. 676-686, 2006.

[10] R. G. Hamlet, "Testing programs with the aid of a compiler," IEEE Transactions on Software Engineering, vol. 3, no. 4, pp. 279-290, July 1977.

[11] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on Test Data Selection: Help for the practicing programmer," IEEE Computer, vol. 11, no. 4, pp. 34-41, April 1978.

[12] W. E. Howden, "Weak Mutation Testing and completeness of test sets," IEEE Transactions on Software Engineering, vol. 8, no. 4, pp. 371-379, July 1982.

[13] A. J. Offutt and S. D. Lee, "An empirical evaluation of Weak Mutation," IEEE Transactions on Software Engineering, vol. 20, no. 5, pp. 337-344, May 1994.

[14] M. Papadakis and N. Malevris, "An empirical evaluation of the First and Second Order Mutation Testing strategies," in Proceedings of the 5th International Workshop on Mutation Analysis (MUTATION'10), France, Paris, April 2010.

[15] H. Agrawal, "Dominators, super blocks, and program coverage," in Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, Portland, Oregon, United States, January 1994, pp. 25-34.

[16] M. Kintis, "Mutation testing and its approximations," Master Thesis (in Greek), Department of Informatics, Athens University of Economics and Business, 2010.

[17] M. Papadakis, N. Malevris, and M. Kintis, "Mutation Testing Strategies: A collateral approach," in Proceedings of the 5th International Conference on Software and Data Technologies (ICSOFT 2010), vol.2, pp. 325-328, Athens, Greece, July 2010.

[18] M. Papadakis, N. Malevris, and M. Kallia, "Towards automating the generation of Mutation Tests," in Proceedings of the 5th Workshop on Automation of Software Test (AST'10), Cape Town, South Africa, May 2010, pp. 111-118.

[19] Y.-S. Ma, A. J. Offutt, and Y.-R. Kwon, "MuJava: An automated class mutation system," Software Testing, Verification & Reliability, vol. 15, no. 2, pp. 97-133, June 2005.

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

[21] D. Schuler and A. Zeller, "(Un-)Covering Equivalent Mutants," in Proceedings of the 3rd International Conference on Software Testing Verification and Validation (ICST'10), Paris, April 2010.

[22] A. J. Offutt, "Investigations of the software testing coupling effect," ACM Transactions on Software Engineering and Methodology, vol. 1, no. 1, pp. 5-20, January 1992.

[23] B. L. William, M. Harman, Y. Jia, "Multi Objective Higher Order Mutation Testing with Genetic Programming," in Proceedings of the 4th Testing: Academic and Industrial Conference - Practice and Research (TAIC PART'09), Windsor, UK, September 2009.

[24] L. Zhang, S.-S. Hou, J.-J. Hu, T. Xie, and H. Mei, "Is operator-based mutant selection superior to random mutant selection?," in Proc. of the 32nd ACM/IEEE International Conference on Software Engineering, vol. 1, pp. 435-444, South Africa, 2010.

[25] R. A. DeMillo and A. J. Offutt, "Constraint-based automatic test data generation," IEEE Transactions on Software Engineering, vol. 17, pp. 900-910, September 1991.

[26] M. Papadakis and N. Malevris, "An effective path selection strategy for Mutation Testing," in Proc. of the 16th Asia-Pacific Software Engineering Conf., Malaysia, Dec. 2009, pp. 422-429.

[27] D. Schuler and A. Zeller, "Javalanche: efficient Mutation Testing for Java," in Proc. of the 7th ESEC/FSE, Amsterdam, The Netherlands, August 2009, pp. 297-298.

[28] R. M. Hierons, M. Harman, and S. Danicic, "Using Program Slicing to assist in the detection of equivalent mutants," Software Testing, Verification and Reliability, vol. 9, no. 4, pp. 233-262, December 1999.

[29] D. F. Yates and N. Malevris, "An objective comparison of the cost effectiveness of three testing methods," Information and Software Technology, vol. 49, pp. 1045-1060, 2007.