# An Empirical Evaluation of the First and Second Order Mutation Testing Strategies

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

*Abstract*—**Various mutation approximation techniques have been proposed in the literature in order to reduce the expenses of mutation. This paper presents results from an empirical study conducted for first and second order mutation testing strategies. Its scope is to evaluate the relative application cost and effectiveness of the different mutation strategies. The application cost was based: on the number of mutants, the equivalent ones and on the number of test cases needed to expose them by each strategy. Each strategy's effectiveness was evaluated by its ability to expose a set of seeded faults. The results indicate that on the one hand the first order mutation testing strategies can be in general more effective than the second order ones. On the other hand, the second order strategies can drastically decrease the number of the introduced equivalent mutants, generally forming a valid cost effective alternative to mutation testing.**

***Keywords—mutation testing, higher order mutation***

## I. Introduction

Software testing has always been considered as the main method for revealing errors in an attempt to determine the level of confidence required before the software's prerelease time. This activity requires the generation of test data. The quality of these data sets is measured by their ability to examine certain software features. Measures of this kind identify the level of the occurrences of certain software features that are successfully exercised. Different features give rise to the definition of various criteria. Their evaluation requires the software execution with actual data in order to exercise specific elements, such as statements, branches, paths etc. Such being the case coverage criteria can be regarded as the vehicle for appropriate test data generation for increasing the software's level of confidence.

Mutation testing is a well known fault detection technique used for producing high quality test cases. Mutation has a widespread reputation of being one of the most effective but together most expensive software testing techniques. In order to reduce the application cost of mutation, various researchers have suggested the use of approximation strategies. These strategies promise to decimate the application cost while maintaining the widespread mutation effectiveness.

Empirical results of the various mutation alternatives have also appeared in the literature [1], [2] and [3]. These studies clearly show a remarkable reduction of the necessary elements required by the strategies. Nevertheless, because of the relatively small number of conducted studies, researchers have not gained enough experience about the application benefits of these strategies. This study presents extensive experimental data on mutation testing approximation strategies. It tries to answer the question of which strategy to use and when. The study and its results are concentrated and presented on a best effort basis in order to help testers choose and apply a specific strategy as appropriate.

Mutation testing approximation strategies mainly rely on the fact that most of the produced mutants are redundant in the sense that they are "almost" always killed when some others are killed. Thus, approximation strategies try to produce in a heuristic way non redundant mutants. Such strategies are commonly referred to in the literature as random selection or mutant sampling [3], [4] and selective mutation [2], [4], [5], [6]. The mutant sampling strategies randomly apply only a specific percentage of the whole set of introduced mutants. The selective strategies try to apply only a subset of the whole set of mutant operators.

Recently, new mutation testing strategies have been suggested based on the notion of higher order mutants [7], [8]. In this approach mutants are constructed based on the application of two or more mutants at a time. Preliminary investigation [7] seems to suggest their limited impact on the testing quality, while recording important savings in required test elements. Although promising, these strategies have not been adequately assessed empirically or compared to others. This forms the main issue of the present research.

The aim of the present study is to measure the impact of the fault detection ability and the main cost factors of the various mutation approximations, for both first and second order mutation methods, with respect to a given set of mutants. Additionally, to compare the mutation testing strategies in a cost-benefit fashion. The above intentions were studied in a conducted experiment involving a set of moderate size industrial programs written in the C programming language. The performed experiment tries to provide an insight of applying second order mutation testing strategies in practice.

The obtained results suggest that the second order strategies can achieve remarkable savings in relation to the introduced equivalent mutants when compared to the first order ones. Also noticeable is that they necessitate a small number of test cases without sacrificing their fault detection ability.

## II. Mutation Testing Strategies

Mutation testing is a powerful fault-based testing technique. It was initially introduced by Hamlet [9] and DeMillo, Lipton and Sayward [10] and forms the focus of this paper. Generally mutation analysis embeds faults into the testing objectives and assesses the performed test quality based on the exposition ratio of the embedded faults. Obviously, both the testing effort and quality are mainly influenced by the size and the quality of the introduced fault set. This section presents mutation testing strategies dealing with this issue.

## A. Mutation Testing Criterion

Usually, mutation testing induces syntactical alterations of the code under test producing mutant versions of the examined code. Each program version is called mutated version. These syntactic changes are performed based on a set of syntactic rules called mutation operators. Test cases are used to execute the candidate set of mutated versions with the goal of distinguishing them from the original one. A mutant is said to be killed if there is a test that distinguishes its output from the output of the original program whereas, it is said to be equivalent if there are not such distinguishing inputs. Assessing tests with the killing mutants' ratio is considered as a measure of the quality of the testing thoroughness. More details about mutation testing can be found in [4].

## B. First Order Mutation Testing Strategies

A significant portion of the mutation testing demands is influenced by the generation and execution of the candidate set of mutants. By considering a small sample (percentage say x%) of mutants, a significant cost reduction can be achieved [3]. Empirical studies [3] have shown that a selection of 10% of mutants results in a 16% loss of the fault detection ability of the produced test sets compared to full mutation testing. In the present study, the considered strategies referred to as first order strategies (Rand x%) select the x% portion of the initial mutant set, where x = 10, 20, 30, 40, 50 and 60. The reason behind the use of these sets of mutants in the present experiment is twofold. The first reason is to revalidate the findings of previous studies by their application to larger cases and different programming language constructs. The second is to attempt to answer the question of whether it is the number of mutants or the adopted strategies that influence the effectiveness of the first and second order mutation methods.

## C. Second Order Mutation Testing Strategies

Another approximation approach for reducing the mutation testing effort was proposed in [7], based on the notion of second order mutants. According to this approach, given a set of first order mutants, a reduced set of pairs constructed from them can be produced. Every mutant program is embedded with two faults at a time and every first order 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 suite [7].

Different strategies of how the first order mutants should be combined can then be deduced. In the study of Polo et al. [7] three strategies were proposed. These strategies are: "RandomMix", "DifferentOperators" and "LastToFirst". In the first strategy the combination is made by randomly selecting the pairs of first order mutants using each mutant once. In the second one, every combined pair uses mutants produced by different operators. In the "LastToFirst" strategy the mutants were combined according to the order they were handled by the underlying tool. The mutant pairs are constructed based on the combination of the first one with the ultimate one, the second with the penultimate, etc [7]. The experimental results obtained by the above strategies were very encouraging, as a high reduction on the number of equivalent mutants was recorded. Additionally, the obtained results and the conducted risk analysis [7] suggested that there should be a small loss on the quality of the second order test suites. This conclusion needs further investigation as it was conducted with a few and small sized programs. Thus, the determination of the impact on the testing quality when using second order mutation strategies in comparison to first order ones forms the main objective of this paper.

In the present experiment the first two of the above strategies were adopted as proposed in [7] and referred to as "RandMix" and "DiffOp" respectively. The third strategy ("LastToFirst") was application depended to the underlying tool and thus was not considered. Additionally, five new strategies were produced. These strategies were influenced by the previously proposed ones. The first of these named "FirstToLast" denoted as "First2Last", orders the first order mutants according to their respective statement appearance in the objective source code. The mutant pairs are then constructed based on the combination of the first with the ultimate mutant, the second with the penultimate one, etc. Of the remaining four, the "SameNode" combines the mutants by selecting them from the same basic block; the "SameUnit" selects the mutant pairs from the same program unit; the "Same Unit FirstToLast" denoted as "SU_F2Last" and "Same Unit Different Operators" denoted as "SU_DiffOp", select candidate mutants based on the use of the "First2Last" and "DiffOp" approaches by applying them individually to each program unit.

## III. Related Work

Higher order mutation was initially introduced and examined in the context of mutant coupling effect [11]. In this study, higher order mutants (HOMs) were considered in order to examine if the produced tests by first order mutants (FOMs) are capable enough to detect the majority of second or third order mutants. The results obtained were in favor of the above statement. The idea of using HOMs for testing purposes was introduced by Polo et al. [7] as described above. This idea was further developed by Jia and Harman [8] who introduced the concept of subsuming HOMs. They advocate that a subsuming HOM is harder to kill than the FOMs from which it is constructed of and thus, it should be preferable to replace the FOMs with a single HOM. "Harder to kill" signifies that tests able to kill a HOM are also capable of killing each one of the FOMs from which it is constructed too. Also in an additional research work [12] the HOMs are constructed with the aim of both producing harder to kill mutants and also syntactically similar to the original program. All studies [7], [8] and [12] focus on reducing the required effort by mutation. This is accomplished by reducing the number of the candidate and equivalent mutants.

Detecting equivalent mutants is a well known undecidable problem. Thus, only heuristic methods can be applied, such as those proposed by Offutt and Pan [13]. Gruen et al. [14] suggested an additional attempt to bypass this problem by focusing on specific (with high impact on program execution) likely to be non equivalent mutants.

A comparison between various mutation strategies has been attempted by Offutt and Lee [1] in order to compare weak mutation variants. The conducted experiment suggested that weak mutation forms a viable 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 randomly selecting mutants [3] and selective mutation [2], [4], [5], [6] as described in previous sections.

Experiments using mutation, such as [15] and [16], a comparison between various testing criteria is recorded. The comparison was performed by relating cost to the respective number of required tests and effectiveness to the number of faults revealed. In [15] the fault detection rate was shown to be similar to the mutants' killable rate and thus mutants were considered for modeling the criteria effectiveness instead. Whereas in [16] the comparison was based on the number of revealed manually seeded faults. However, these experiments do not consider second order mutants and therefore, not directly comparable with the present study.

## IV. EXPERIMENTATION

This section describes the details about the conducted experiments including experimental description, the test objectives used and the various artifacts of the test design, containing the mutation operators used, the test cases and the faults seeded in the test objectives.

In the experimental description that follows, the term strategy is used in order to refer to the various mutation approaches considered in the present experiment. The term fault is used to represent either manually seeded faults or real faults used to model the fault detection ability of the various strategies. The generated faults by the mutation operators are referred to as mutants.

### A. *Definition of the Experiment*

The objective of this experiment is to compare the application benefits of various first and second order mutation testing strategies. The comparison is made based on measuring the various cost factors introduced by the testing process, such as the number of the produced mutants, the number of equivalent ones and the number of test cases needed to satisfy each mutation variant criterion. Also, the number of the exposed faults by each strategy is used to evaluate its relative strength. The experiment uses eight programs, for which sets of faults and large pools of test cases are available. Program details are given in the "Subject Programs" section (IV.B). For each test objective used, sets of mutant programs were generated according to each mutation testing strategy. Mutation operators and strategy details are given in section "Mutation Tool and Operators Used" (IV.C). Test suites were then constructed by selecting tests at random from the test pools utilizing the policies of the mutation strategies. All tests sets were then executed against the available faulty versions of the test programs recording the exposed fault rates. Comparison was then made based on the various cost and effectiveness factors of the considered strategies. The full details of the experimental regime are given in the section "The Experimental Regime" (IV.D).

### B. *Selected Programs*

The selected objectives are composed of eight programs written in the C programming language. They have all been used in similar previous studies such as [8], [15], [17] and [18]. The first seven, form the well known Siemens suite while the eighth one is the Space program developed at the European Space Agency. All program details are presented in Table I.

The seven Siemens programs and their associate faulty versions and test cases have been initially used by Hutchins et al. [19] to compare structural criteria. These were modified, extended and used by Rothermel and Harrold [20] and Graves et al. [21] in later studies. The faulty program versions were introduced by researchers with the aim of producing realistic fault versions [19].

The last program (Space program) was initially used in the studies of Frankl et al. [22] and probably constitutes the most realistic object of the present study. This program is associated with 38 "real" faulty versions identified and corrected "during testing and the operative use of the program" as reported in [23]. The advantage of this program against to the other seven is that its faults are real ones. Thus, they should give a more realistic simulation on the fault detection ability of the produced test suites. All programs were chosen because of their extensive use in the literature and their availability along with their artifacts and can be found in the Subject Infrastructure Repository (SIR) at the University of Nebraska-Lincoln [18].

The test cases associated with each object program were constructed based on both black-box and white-box testing techniques such as: random, category-partition, all statements, all edges and all definition-use pairs.

TABLE I. TEST PROGRAMS DETAILS

| Program | Number of LOC | Number of Test Cases | Number of Faults |
|---|---|---|---|
| Schedule | 296 | 2650 | 9 |
| Schedule2 | 263 | 2710 | 10 |
| Tcas | 137 | 1608 | 41 |
| Totinfo | 281 | 1052 | 23 |
| Printtokens | 343 | 4130 | 7 |
| Printtokens2 | 355 | 4115 | 10 |
| Replace | 513 | 5542 | 32 |
| Space | 5905 | 13585 | 38 |

In Harder et al. [24], details of the construction of these test sets are given. The population of the available test cases consists of a relatively large number of test cases. These were not necessarily constructed for killing mutants and thus make the sampling population rather realistic.

### C. Mutation Tool and Operators Used

In order to apply the first and second order mutation testing strategies, the Proteum mutation testing system, by Delamaro and Maldonado [25], was used. Proteum uses 77 operators all implemented according to Agrawal et al. [26]. This set of operators requires huge computational resources in order to generate mutants and execute them. In order to complete the present experiment with reasonable resources two general reductions were used. First, a restriction on the considered mutant operators was made. All mutant operators provided under the general class of "operators" [26] were considered. This set of mutants is composed of 44 operators which either alter or insert programming language operators where appropriate. Second, for the space program, one additional restriction was imperative. This was due to the vast number of produced mutants (22,500 mutants). Therefore, a 10% of them were selected, based on their production order. Thus, every 10th produced mutant was considered. It is noted that a different set of first order mutants was constructed in every experiment repetition (all the experiments were repeated five times). This approach has also been undertaken by other studies such as [15] and [17].

The aim of the present paper is to measure the impact on the quality of the produced test cases when using approximation strategies over an initial mutant set. All the undertaken approximations rely only on an initial set of mutants (common to all used strategies) which form the basis of the experimental comparison. Thus, the use of the above restrictions should affect the test quality only of the initial set mutant sets.

The second order mutants were constructed based on the combined use of the first order ones. For the experimental needs, a prototype framework was built in order to collect the first order mutants and construct the various second order ones according to each strategy (see section "mutation testing strategies"-II). It is noted that all second order mutants were produced based on the considered set of first order ones and that every first order mutant is embodied in at least one pair of second order mutants for every undertaken strategy.

### D. The Experimental Regime

The results reported in the following sections are derived based on the application of the various first and second order mutation testing strategies. Generally, the experiment has two legs. In the first leg, the analysis procedure followed was initially to generate a set of mutants and their respective test cases according to each strategy. In the second leg, tests were chosen at random. Then for both legs a comparison based on various cost factors in combination to the fault exposing ability of the generated test sets was made.

Initially, all mutants were generated and compiled in order to determine the mutant candidate set for each strategy. In order to avoid any bias introduced by the random selection or combination of mutants all the experiments were repeated independently five times. All mutant sets were then executed against all available test cases. All live mutants were eliminated from the mutants' sets as being equivalent. The elimination is based on an approximation method described below. Although this approach does not guarantee their equivalence, it was chosen in order to complete the experiment with reasonable resources. It is believed that this approximation fulfils the general goals of the present study as there should be minimal chances for non-equivalent mutants to be left alive after their exercise with such a huge and high quality data test set. Additionally, such an approximation method has also been used in similar studies such as [17]. Nevertheless, all eliminated mutants form sets (one set per strategy) of likely to be equivalent mutant sets. It should be noted that each of these sets contains the maximum possible number of equivalent mutants and thus, in the results obtained their actual number should be even smaller than the one reported. In the rest of the present paper they will be referred to as equivalent mutants.

Initially, six sets of test cases per strategy were generated based on a random selection from the available test pools retaining only those that increase the coverage (kill additional mutant(s)) according to the strategy followed. All tests that do not contribute to the coverage increase were eliminated as redundant in respect of the followed strategy. In the second leg of the experiment, another six tests per program were randomly selected for each program. Then the achieved scores according to each strategy were recorded.

The produced test cases result from applying every strategy to the selected test objectives. As stated before, in order to avoid any side effects of the strategy or test selection method, a set of 30 test sets were generated according to each strategy (six tests due to each strategy and five times due to the repetition of the whole experiment). These tests were then executed against the various seeded program faults in order to determine their fault detection ability.

## V. Analysis of the Results

The experimental results derived from the application of the mutation strategies are presented and analysed in this section.

### A. Strength Comparison

Generally, the aim of testing criteria is to detect faults. The evaluation of a testing strategy should therefore, be based on the fault exposing ability of its produced tests. Thus, a comparison based on the fault detection ability exposes the individual strength of each of the considered strategies. The fault detection ability was measured based on the fault revealing ratio of the selected tests on a set of seeded faults.

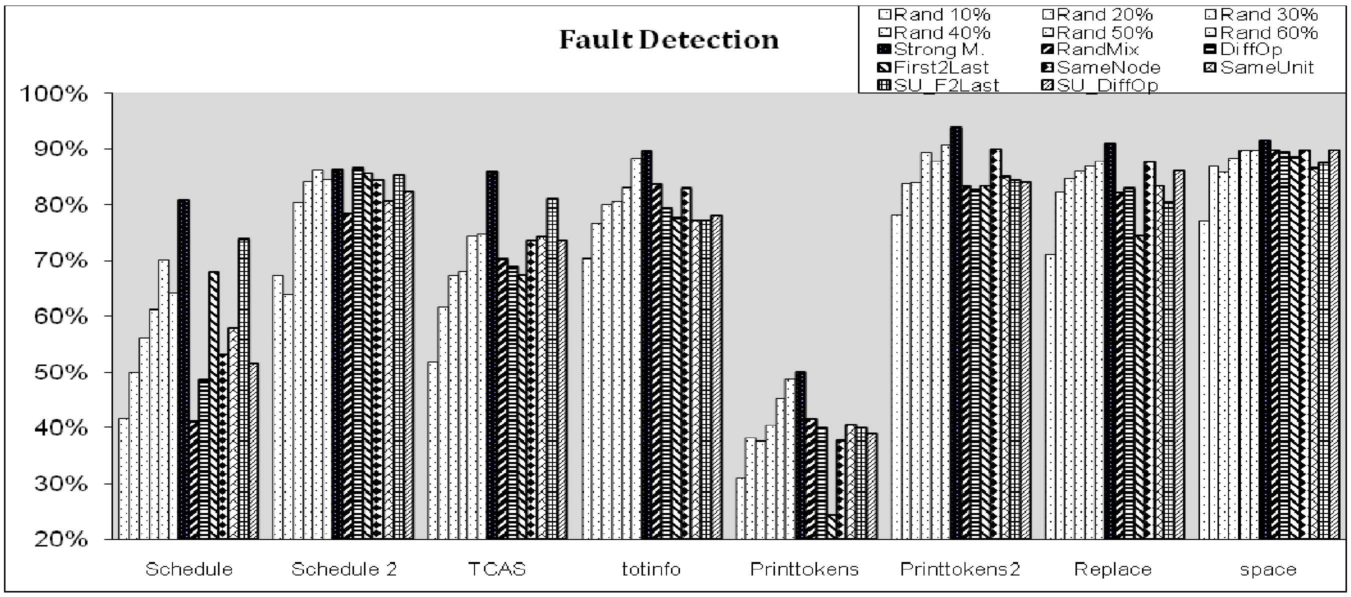

Figure 1. Fault detection ability (average values) of mutation testing strategies

Figure 1 presents the average percentage of detected faults per testing objective and undertaken strategy. Tables II and III in rows "Total Faults" and "Fault Rate" record the total number of detected faults and their respective fault detection rate per strategy.

The most interesting aspect of the above graph is that strong mutation always detects more faults than any other strategy and in the majority of the cases this situation is significant. Although second order strategies score high but not as high as strong mutation, indicating a high effectiveness, their behavior is similar to the one recorded for Rand 50% and 60%.

With respect to the second order strategies there appears to be a high variation on their fault detection effectiveness. Consequently, it can be argued that there is not any single strategy that could be clearly characterized as being the most effective. In general, more effective ones seem to be the "SameNode" and "SU_F2Last" strategies. The "SameNode" strategy scores best in two of the eight test objectives and scores almost best in three other cases. The "SU_F2Last" scores best in two cases and almost best in two others. On a total number of detected faults basis (tables II and III), "SameNode" and "SU_F2Last" detects 80.73% and 79.86% of the total number of introduced faults respectively. These scores are approximately higher than those achieved by the remainder of the strategies by approximately 1-5%. It must be noted that when second order strategies are applied to a unit level, they provide better results when compared to the ones obtained by their application to the entire program.

## B. Cost Comparison

One of the key issues for selecting a testing strategy, apart from its effectiveness, is its practicality. In view of this, in the present study a measurement of various cost factors was established in order to provide an insight of the application cost of the respective strategies. The overall application cost of a testing strategy may depend on many factors. Of these factors the usually regarded as more influencing are the number of produced mutants, the number of equivalent mutants and the number of the produced test cases per each strategy.

Figure 2 presents the average test size required by the considered strategies. Although this factor does not represent the true effort or cost of creating those tests, this measure can give a first indication about the strategies application cost. From the graph it can be seen that strong mutation requires by far more tests than all the alternative strategies. One additional point is that "SameNode" strategy also requires a considerable number of test cases compared to the other second order mutation testing ones.

When comparing strategies on the basis of producing approximately the same number of mutants, such as Rand 50% and 60% against second order mutation testing cases, it can observed that second order strategies require significantly fewer test cases. This fact suggests that second order strategies are in general less costly with respect to the phases of test generation and execution.

Another aspect that affects the overall application cost of mutation is the detection of equivalent mutants. This is an also well known undecidable problem and thus, the equivalent mutants' identification should be considered as a manual effort activity. Figure 3 presents the average number of possible equivalent mutants produced by the different strategies. It is noted that these sets of mutants are those left alive after their execution with all available test cases and hence their actual number should be even smaller. Nonetheless, the most surprising finding of the present study is the remarkable reduction of equivalent mutants attributed to the second order strategies. Randomly selecting two mutants from the set of first order ones, results in a chance of selecting two equivalent ones as approximately 5%≈$(22.5)^2$% (chance of selecting one equivalent – approximately 22.5%). This provides a random selection possibility of the produced second order equivalent mutants. As it can be observed, in table III, the top two rows ("Mutants" and "Equivalents"), that referred to strategies "RandMix", "DiffOp", "First2Last" and "SU_DiffOp" can be used to deduce on average 5-6% of equivalent mutants while "SameNode", "SameUnit" and "SU_F2Last" produce on average approximately 17%, 9.7%, and 8.6% respectively. Comparing these results with the random selection value of 5%, it becomes evident that the equivalent mutants are more strategy depended (with respect to the same unit strategies) than random selection depended. It must be noted that the second order strategies produce approximately 80-90% less equivalent mutants, compared to strong mutation, with the exception of "SameNode" strategy which produced 60% less equivalent mutants.

The high reduction on produced equivalent mutants can be considered as one of the major contributions of the second order mutation testing strategies. The ability of these strategies to heuristically reduce in an a priory basis the number of introduced equivalent mutants should result in considerable savings with respect to the required effort. The achieved savings should reduce not only the required manual effort but also the required computational resources for mutant execution. As pointed in [5] the equivalent mutants, contrary to non equivalent ones, should be executed against all produced test ceases. Thus, as computed in [5] the computational resources needed by mutation strategies are greatly influenced by the existence of equivalent mutants.

The application of second order strategies in general results in mutant sets approximately half the size of those produced by strong mutation. The only exception seems to be the two strategies based on "Different Operators", which produced a size approximately equal to 70% of the one for strong mutation. However, this handicap is counterbalanced by the ability of both above strategies, being able to create the least percentage of equivalent mutants compared to the remaining ones.

The presented results suggest that among the various second order mutation testing strategies, the "SameNode" strategy clearly achieves and by far the worst reduction. "First2Last" and "RandMix" strategies produce a 7-9% less equivalent mutants when applied to the same unit than to the whole program, whereas the behavior of the "Different Operators" strategies is indifferent.

### C. *Cost – Benefit Comparison*

The question that is raised here is: "which is the best strategy to be selected?". To answer this question, there is a need to combine all the cost and benefit factors related to each of the strategies. Such an attempt is made based on two cost-benefit measures. The first measure here refereed to as "Test Effectiveness", is defined as follows:

$$Test\ Effectiveness = \frac{No.of\ Test\ Cases}{No.of\ Exposed\ Faults}$$

The "Test Effectiveness" measure has been widely used in the literature [15], [16], [28]. This measure reflects the application cost of the testing process as being proportional to the test size which is dominated by the following cost factors: test production, test execution, test oracle generation and verification cost. However, a major burden of mutation testing is not accounted for. This is due to the presence of equivalent mutants. Consequently, there is a definite need to include their presence as also commented by Weyuker [28]. This measure assumes that equivalent mutants' identification requires approximately additional relative effort with test data generation. This effort is twofold: first as it is costly because of the human intervention for their identification and second as it has been proven by Offutt and Pan [13] that it forms an instance of the feasible path problem which is the basis of most test data generation techniques.

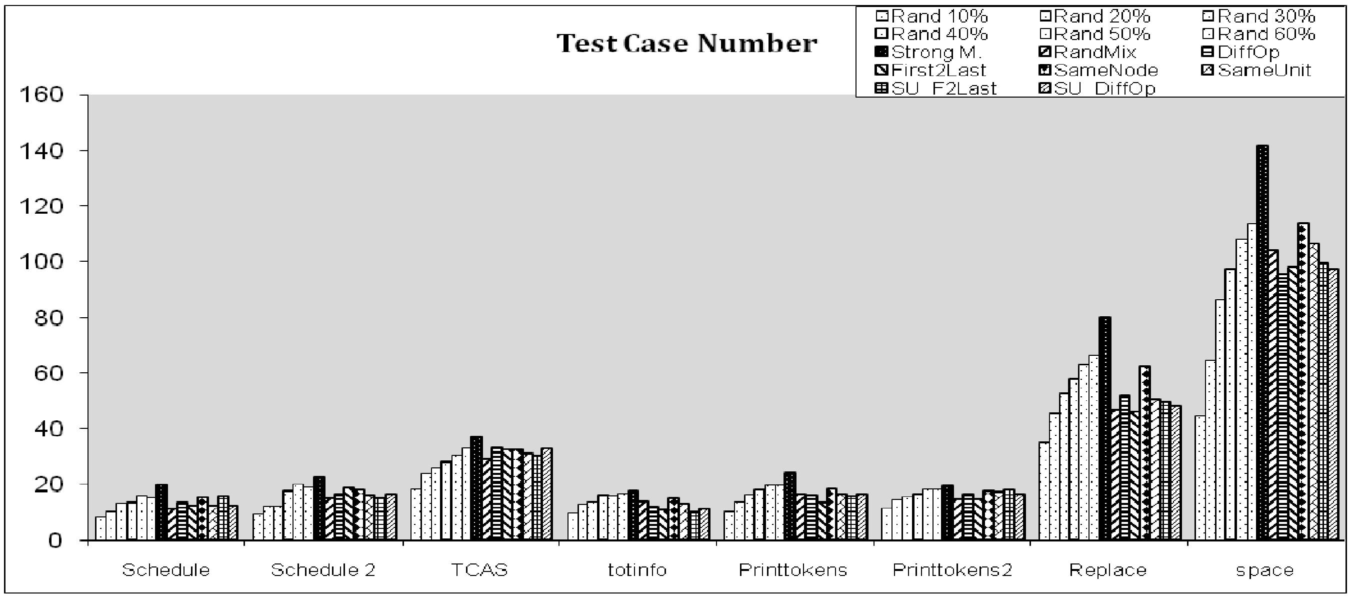

Figure 2. Number of Tests (average values) required by mutation testing strategies

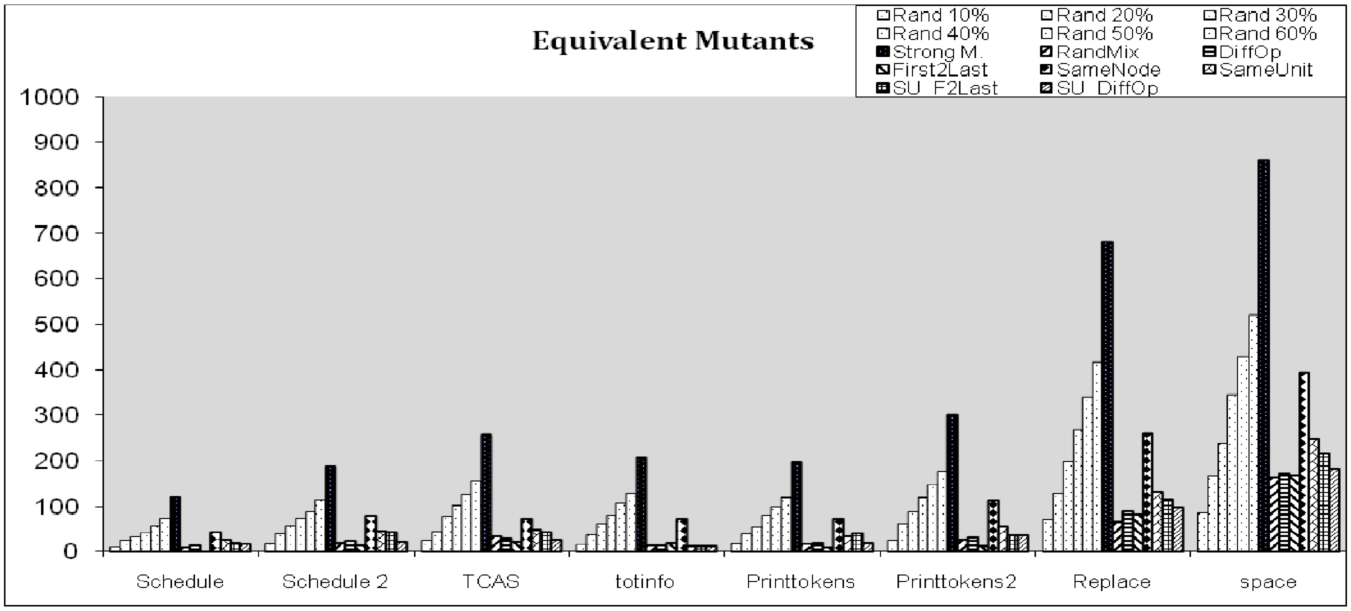

Figure 3. Number of possible (average values) of equivalent mutants produced by the mutation testing strategies

Additionally, in [29] a practical transformation of the killing mutants' problem to a covering branches one was suggested. As a consequence the identification of equivalent mutants forms a "harder" activity rather than the one of producing tests. Thus an alternative measure referred to as "CostEffectiveness" is defined as follows:

$$CostEffectiveness = \frac{No.of\ Test\ Cases + No.of\ Equivalent\ Mutants}{No.of\ Exposed\ Faults}$$

Tables II and III present the results of the considered strategies for first and second order respectively. Both tables record in their columns the measurements per each strategy while their rows represent details concerning the number of mutants, the number of equivalent mutants, the total number of required tests, the number of exposed faults, the fault detection rate and the two previously mentioned measures "Test-Effectiveness" and "Cost-Effectiveness". It is noted that the number of mutants (killable and equivalent ones) has been produced incrementally by considering all five experiment repetitions. Additionally, the total tests and faults were obtained by the addition of the 30 (6 tests per strategy x 5 experiment repetitions) produced test sets and their respective ability of detecting faults.

The most interesting aspects of these results are based on the cost benefit perspective. It can be noted that the two adopted measures provide different results. According to the "Test-Effectiveness" measure the overall most appropriate ones are the Rand 10%, 20%, 30% and "SU_F2Last" approaches. If the second order strategies were to be considered, the best choice appears to be the "SU_F2Last" whereas the worst the "SameNode". Another notable result is that strong mutation, detects by far the majority of faults but at a high cost (it requires a large number of test cases). Consequently, scoring for this reason by far worse than its rivals with respect to "Test-Effectiveness".

TABLE II. RESULTS OF FIRST ORDER MUTATION TESTING STRATEGIES

| | **Strong Mutation** | **Rand 10%** | **Rand 20%** | **Rand 30%** | **Rand 40%** | **Rand 50%** | **Rand 60%** |
|---|---|---|---|---|---|---|---|
| **Mutants** | *62714* | ***6280*** | *12552* | *18823* | *25093* | *31372* | *37649* |
| **Equivalents** | *14100* | ***1423*** | *2779* | *4104* | *5628* | *7048* | *8576* |
| **Tests Cases** | *10880* | ***4404*** | *5938* | *7103* | *7909* | *8732* | *9051* |
| **Total Faults** | ***4453*** | *3288* | *3736* | *3886* | *3986* | *4144* | *4193* |
| **Fault Rate** | ***87.31%*** | *64.47%* | *73.25%* | *76.20%* | *78.16%* | *81.25%* | *82.22%* |
| **Test-Effectiveness** | *2.4433* | ***1.3394*** | *1.5894* | *1.8278* | *1.9842* | *2.1071* | *2.1586* |
| **Cost-Effectiveness** | *5.6097* | ***1.7722*** | *2.3332* | *2.8839* | *3.3961* | *3.8079* | *4.2039* |

The "Cost-Effectiveness" measure produces a different picture. In this respect only the Rand 10% and 20% strategies score better than the second order ones. The Rand 30% now scores worse than five of the second order strategies. The best choice among the second order strategies is the "RandMix" approach, while the remaining of the strategies with the exception of "SameNode" score quite similarly.

Evaluating the above results with respect to their cost benefit contribution two points are clear. First, strong mutation falls considerably behind every other considered strategy. Perhaps the most appropriate choice in this respect seems to be the Rand 10%. Second, there is no second order strategy that is a clear winner among the other similar strategies, as they all score similarly. This implies that the application of the strategies should detect similar number of faults for a given amount of effort. As the application of the strategies may result in different amounts of effort, a more demanding strategy will detect more faults than a less demanding one. The above are direct results that are derived by adhering to the "Test-Effectiveness" and "Cost-Effectiveness" ratios.

TABLE III. RESULTS OF SECOND ORDER MUTATION TESTING STRATEGIES

| | Rand Mix | DiffOp | First 2Last | Same Node | Same Unit | SU_ F2Last | SU_ DiffOp |
|---|---|---|---|---|---|---|---|
| **Mutants** | *31362* | *45355* | ***31362*** | *32821* | *31620* | *31620* | *45407* |
| **Equivalents** | *1817* | *2024* | ***1724*** | *5605* | *3067* | *2710* | *2140* |
| **Tests Cases** | *7552* | *7589* | ***7335*** | *8828* | *7926* | *7604* | *7576* |
| **Total Faults** | *3943* | *3906* | *3804* | ***4117*** | *3987* | *4073* | *4027* |
| **Fault Rate** | *77.31%* | *76.59%* | *74.59%* | ***80.73%*** | *78.18%* | *79.86%* | *78.96%* |
| **Test-Effectiveness** | *1.9153* | *1.9429* | *1.9282* | *2.1443* | *1.9880* | ***1.8669*** | *1.8813* |
| **Cost-Effectiveness** | ***2.3761*** | *2.4611* | *2.3814* | *3.5057* | *2.7572* | *2.5323* | *2.4127* |

## D. Comparison Based on Strong Mutation

All the mutation testing approximations were proposed as alternatives to full strong mutation. Their construction was motivated by the practical need to reduce the strong mutation overheads. In this study, all obtained results were presented on a comparable basis among all considered strategies. The interest of the present section also focuses on the benefits recorded by the application of the strategies with respect to full strong mutation. Along these lines figure 4 depicts all four considered measures when compared against strong mutation.

Figure 4 is composed of four parts. The North West part corresponds to the loss on fault detection ability resulting by using the approximation alternatives. The North East part corresponds to the mutants' proportion reduction achieved with respect to the produced number of mutants. The South West part displays the required test set size reductions. The remaining part presents the achieved reductions of the introduced equivalent mutants. These graphs present all the application aspects of the approximation strategies compared to strong mutation.

Table IV records complementary details considering the fault detection loss of the examined strategies against strong mutation. Specifically, it records the fault loss in the best and worst cases of the experiment (higher and lower fault rate of the produced test sets according to the examined strategies). These values may be also considered as ranges of the expected fault loss of these strategies. Thus, it can be observed that the ranges of the first order strategies are reduced as the sampling percentage increases. High loss ranges are recorded for all second order strategies. The interesting point here is that the "SU_F2Last" strategy results in the smallest recorded loss for the worst case scenario while also resulting to a small range in comparison to the remaining strategies.

The graphs of figure 4 suggest that a considerable loss on the fault detection ability is observed in both first and second order strategies. This loss is more evident with the Rand 10% strategy. Remarkable savings are produced by the second order mutation testing strategies for all the cost measures, leading to the conclusion that they could be used as valid alternatives to strong mutation. The results also suggest that when using second order mutation testing strategies, the most appropriate choice appears be the "SU_F2Last" strategy as: a) it provides the best fault detection rate in the worst case, b) the smallest fault detection range (8.47%) and c) with reasonable test demands compared to the rest of the strategies.

TABLE IV. FAULT LOSS OF MUTATION TESTING STRATEGIES

| | | Rand 10% | Rand 20% | Rand 30% | Rand 40% | Rand 50% | Rand 60% |
|---|---|---|---|---|---|---|---|
| ***Worst*** | | *44.93%* | *26.81%* | *26.81%* | *17.39%* | *16.67%* | *17.39%* |
| ***Best*** | | *10.43%* | *7.98%* | *8.59%* | *4.29%* | *3.07%* | *1.84%* |
| | **Rand Mix** | **DiffOp** | **First 2Last** | **Same Node** | **Same Unit** | **SU_ F2Last** | **SU_ DiffOp** |
| ***Worst*** | *21.01%* | *21.74%* | *22.46%* | *20.29%* | *23.19%* | *15.22%* | *17.39%* |
| ***Best*** | *5.52%* | *6.75%* | *7.36%* | *2.45%* | *4.29%* | *6.75%* | *4.29%* |

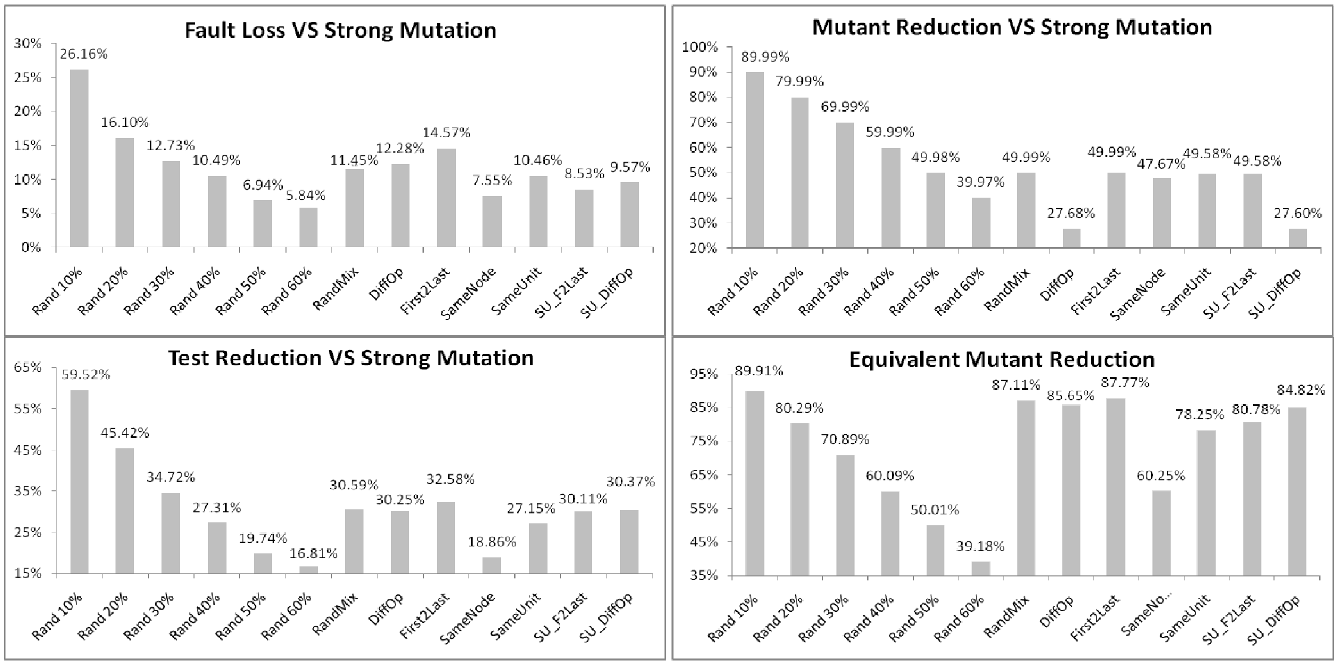

Figure 4. Achieved reductions by mutation testing strategies compared to Strong mutation

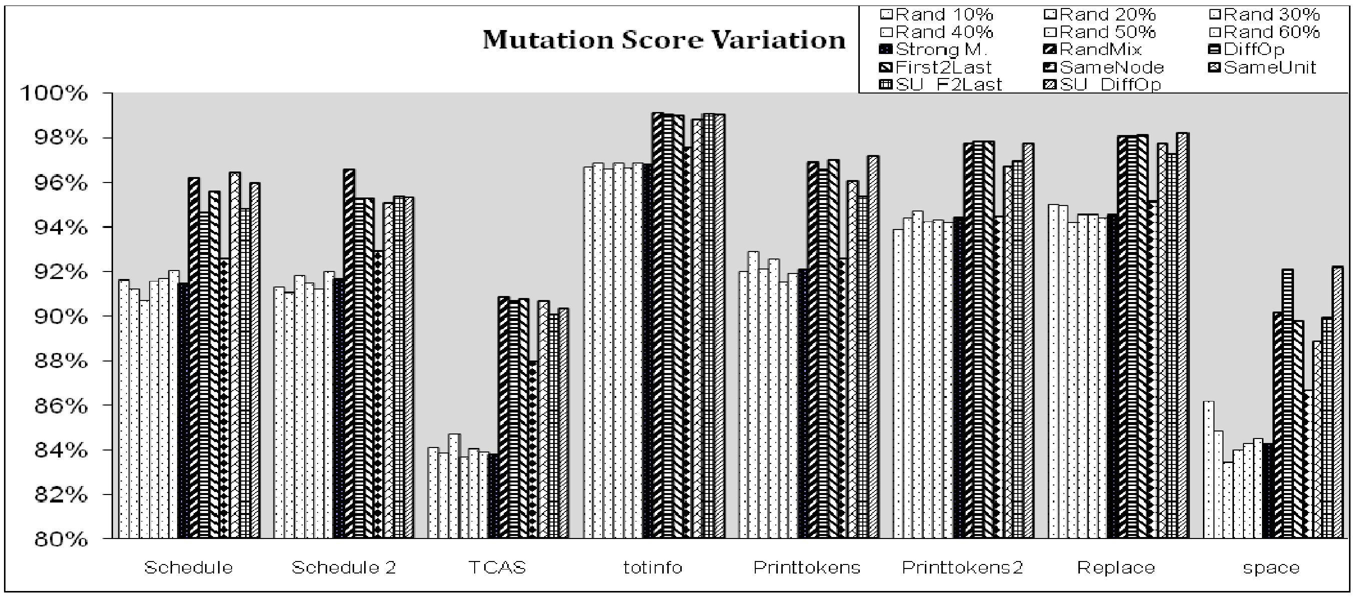

Figure 5. Mutation Score variation between mutation testing strategies

## E. *Mutation Score Variation Comparison*

Comparisons made so far, were based on cost and effective factors produced by the considered strategies. In order to achieve a straightforward comparison between the strategies, one additional experiment was also carried out. For this reason, same data sets were used. This differentiates this experiment from the one in the first leg, as it uses the same sets of data in an attempt to measure their behavior. Whereas, in the previous experiment the sets of data were intrinsic to the strategies. By doing so, all strategies' achieved scores were compared among them. Six different tests sets were constructed based on a random selection of tests from the test sample. Each set was sized according to each program's average size of strong mutation tests, as derived by the first leg experiment.

Figure 5 presents the average mutation scores achieved by applying the six test sets to each program. From the graph it can be observed that first order strategies provide lower scores than all the second order ones. This is somehow expected as the second order strategies provide weaker measurements over the first order ones as this was also demonstrated in the previous experiment. Among the second order ones, the "SameNode" strategy indicates a similar score rate as "Strong mutation". For the remaining of the second order strategies the coverage scores vary according to each program. The overall average recorded scores are 91.12%, 95.69%, 95.52%, 95.40%, 92.48%, 95.03%, 94.85%, and 95.75% for the "Strong mutation", "RandMix", "DiffOp", "First2Last", "SameNode", "SameUnit", "SU_F2Last", "SU_DiffOp" strategies respectively. The coverage behavior of the first order strategies is somehow similar to the coverage recorded for strong mutation as it was expected. It must be noted that the discrepancies between strong mutation and second order mutation coverage do not necessarily indicate the ability of the one strategy over the other. Conclusively, it can be argued that a lower coverage, as in the case of strong mutation, suggests that more tests are needed to increase the coverage, this contributing to the thoroughness of this method rather than its weakness.

## VI. THREATS TO VALIDITY

All empirical studies involving software lack of being representative and thus the external validity of the present study should also be uncertain. Nevertheless, all the test objectives used in the present study constitute industrial programs widely used in the literature.

One threat to the internal validity of this work can be related to the manual seeding of faults and generation of test cases. As the programs, test cases and faults were not purposely built for the present research, since being provided together with the programs, they should have a limited influence on the present results. Other issues affecting the reported results could be based on the choice of mutation operators. Different operators might produce different results. However, the selected set of operators forms a set of mutants embodying all language operators. Additionally, when employing techniques based on random facts as in the present work random selection of mutants or pairs, may contain a risk of low effectiveness. Thus, the tester should be aware of these risks. The present work gives some indication about these risks with the best and worst case scenarios presented in previous section. Nonetheless, this matter is left open for future research.

The aim of the study was to investigate the feasibility and the impact of the various strategies in comparison to strong mutation. For all these reasons the authors believe that the threats to the validity of the results obtained are negligible.

## VII. CONCLUSION AND FUTURE WORK

This paper presents an empirical study for using mutation testing and its first and second order mutation variants. The findings of this study suggest that these mutation variants can provide a significant reduction on various cost factors. Specifically, the results obtained indicate that first order strategies are generally more effective at detecting faults, than their second order rivals however, at a greater cost. Second order strategies can drastically decrease the number of equivalent mutants introduced and provide significant savings to both numbers of produced mutants and required test cases.

The results suggest that a reduction of approximately 80% to 90% of the equivalent mutants generated by second order strategies can be tackled. Moreover, second order strategies can accomplish reductions of roughly 30% of the required test cases with approximately 10% or less on the loss of their fault detection ability compared to strong mutation. Randomly selecting a percentage of first order mutants results in a fault loss ranging from 26% to 6% for the methods Rand 10% to 60%. Their test reductions range from 60% to 17%.

The experiment suggests that second order strategies succeed in significantly reducing the number of both produced and equivalent mutants. Additional savings are recorded according to the required test cases with little fault detection loss. Surprisingly, second order strategies can achieve approximately equal fault detection effectiveness as also do Rand 40% and 50% strategies (around 80%) with additional savings of over a 7% test case reduction and approximately a 25% reduction on the number of the produced equivalent mutants in comparison to the other two. This fact indicates that second order strategies can be in general more cost effective than first order ones. Nevertheless, the choice between second order strategies is not apparent. The one that appears here as having a slight advantage is the SU_F2Last strategy.

Future work is directed towards conducting more experiments in order to statistically validate the claims of the present findings. Currently, additional second order strategies are also examined.

## Acknowledgment

Thanks are due to Professor José Carlos Maldonado for providing us the Proteum mutation testing tool. The authors would like to thank the anonymous referees for their useful suggestions that helped improving the present paper. This work is supported by the Basic Research Funding (PEVE 2010) program of the Athens University of Economics and Business.

## References

[1] A. J. Offutt and D. S. Lee, “An Empirical Evaluation of Weak Mutation”, IEEE Trans. on Soft. Eng., vol. 20, 1994, pp. 337-344.

[2] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, C. Zapf, “An experimental Determination of Sufficient Mutation Operators”, ACM -TOSEM. vol. 5, 1996, pp. 99-118.

[3] W. E. Wong, “On Mutation and Data Flow,” PhD Thesis, Purdue University, West Lafayette, Indiana, 1993.

[4] J. Offutt and H. Untch, “Mutation 2000: Uniting the Orthogonal”, Mutation 2000: Mutation Testing in the Twentieth and the Twenty First Centuries, pp. 45-55, 2000.

[5] E. S. Mresa and L. Bottaci, “Efficiency of Mutation Operators and Selective Mutation Strategies: An Empirical Study,” Softw. Test., Verif. Reliab. , 1999, vol. 9, pp. 205-232.

[6] E. F. Barbosa, J. C. Maldonado, and A. M. R. Vincenzi, “Toward the determination of sufficient mutant operators for C,” Softw. Test., Verif. Reliab. , 2001, vol. 11, pp. 113–136.

[7] M. Polo, M. Piattini, I.G. Rodriguez, “Decreasing the cost of mutation testing with second-order mutants”, Softw. Test., Verif. Reliab., vol. 19, 2009, pp. 111-131.

[8] Y. Jia and M. Harman, “Higher Order Mutation Testing”, Information and Software Technology, vol.51, 2009, pp. 1379-1393.

[9] R. G. Hamlet, “Testing program with the aid of a compiler”, IEEE Trans. Softw. Eng., vol. 3, 1977, pp. 279-290.

[10] R. A. Demilo, R. J. Lipton, and F. D. Sayward, “Hints on test data selection: Help for the practicing programmer”, IEEE Computer, vol. 11, 1978, pp. 34-41.

[11] A. J. Offutt, “Investigations of the Software Testing Coupling Effect,” ACM -TOSEM, vol. 1, 1992, pp. 5–20.

[12] W. B. Langdon, M. Harman, Y. Jia, "Multi Objective Higher Order Mutation Testing with Genetic Programming”, Taic-part, 2009.

[13] J. Offutt and J. Pan, “Detecting equivalent mutants and the feasible path problem”, Softw. Test. Verif. Reliab., vol.7, 1997, pp.165-192.

[14] B. J. M. Gruen, D. Schuler, and A. Zeller, “The impact of equivalent mutants,” in Mutation ’09, 2009.

[15] J. H. Andrews, L. C. Briand, Y. Labiche, and A. Siami Namin. Using mutation analysis for assessing and comparing testing coverage criteria. IEEE Trans. Softw. Eng., vol. 32, 2006, pp.608-624.

[16] N. Li, U. Praphamontripong and J. Offutt, "An Experimental Comparison of Four Unit Test Criteria: Mutation, Edge-Pair, All-Uses and Prime Path Coverage," in Mutation ’09, 2009.

[17] J. H. Andrews, L. C. Briand, and Y. Labiche. Is Mutation an Appropriate Tool for Testing Exeperiments? Proc. ICSE, 2005.

[18] H. Do, G. Rothermel and S. Elbaum, “Infrastructure support for controlled experimentation with software testing and regression testing techniques,” Oregon State University, Corvallis, OR, USA, Technical report 04-06-01, January, 2004.

[19] M. Hutchins, H. Froster, T. Goradia and T. Ostrand, “Experiments on the Effectiveness of Dataflow and Controlflow-Based Test Adequacy Criteria,” Proc. ICSE, 1994.

[20] G. Rothermel and M. J. Harrold, “Empirical Studies of a Safe Regression Test Selection Technique,” IEEE Trans. on Soft. Eng., vol. 24, 1998, pp. 401-419.

[21] T. L. Graves, M. J. Harrold, J.-M. Kim, A. Porter and G. Rothermel, “An Empirical Study of Regression Test Selection Techniques,” ACM -TOSEM, vol. 10, 2001, pp. 184-208.

[22] P. G. Frankl and O. Iakounenko, “Further Empirical Studies of test Effectiveness,” Proc. FSE, 1998.

[23] F. I. Vokolos and P. G. Frankl, “Empirical evaluation of the textual differencing regression testing technique,” Proc. ICSM, 1998.

[24] M. Harder, J. Mellen and M. D. Ernst, “Improving Test Suites via Operational Abstraction”, Proc. ICSE, 2003.

[25] M. E. Delamaro and J. C. Maldonado. “Proteum – a tool for the assessment of test adequacy for C programs”. Proc. PCS, 1996.

[26] H. Agrawal, R. A. DeMillo, B. Hathaway, W. Hsu, W. Hsu, E. W. Krauser, R. J. Martin, A. P. Mathur, and E. Spafford, “Design of Mutant Operators for the C Programming Language,” Purdue University, West Lafayette, Indiana, Technique Report SERC-TR-41-P, March 1989.

[27] A. S. Namin, J. H. Andrews, and D. J. Murdoch, “Sufficient Mutation Operators for Measuring Test Effectiveness”, Proc. ICSE, 2008.

[28] E. J. Weyuker, More Experience with Data Flow Testing, IEEE Trans. on Soft. Eng., vol.19, 1993, pp.912-919.

[29] M. Papadakis, N. Malevris: An Effective Path Selection Strategy for Mutation Testing, Proc. APSEC, 2009.