An Experimental and Practical Study on the Equivalent Mutant Connection: An Evolutionary Approach.

Pedro Delgado-Pérez[a,*], Francisco Chicano[b]

[a] *Escuela Superior de Ingeniería,Universidad de Cádiz, Spain*
[b] *Escuela Técnica Superior de Ingeniería Informática, Universidad de Málaga, Spain*

Abstract

Context: Mutation testing is considered to be a powerful approach to assess and improve the quality of test suites. However, this technique is expensive mainly because some mutants are semantically equivalent to the original program; in general, equivalent mutants require manual revision to differentiate them from useful ones, which is known as the Equivalent Mutant Problem (EMP). **Objective:** In the past, several authors have proposed different techniques to individually identify certain equivalent mutants, with notable advances in the last years. In our work, by contrast, we address the EMP from a global perspective. Namely, we wonder the extent to which equivalent mutants are connected (i.e., whether they share mutation operators and code areas) as well as the extent to which the knowledge of that connection can benefit the mutant selection process. Such a study could allow going beyond the implicit limit in the traditional individual detection of equivalent mutants. **Method:** We use an evolutionary algorithm to select the mutants, an approach called Evolutionary Mutation Testing (EMT). We propose a new derived version, *Equivalence-Aware EMT* (EA-EMT), which penalizes the fitness of known equivalent mutants so that they do not transfer their features to the next generations of mutants. **Results:** In our experiments applying EMT to well-known C++ programs, we found that (i) equivalent mutants often originate from other equivalent mutants (over 60% on average); (ii) EA-EMT's approach of penalizing known equivalent mutants provides better results than the original EMT in most of the cases (notably, the more equivalent mutants are detected, the better); and (iii) we can combine EA-EMT with Trivial Compiler Equivalence as a way to automatically identify equivalent mutants in a real situation, reaching a more stable version of EMT. **Conclusions:** This novel approach opens the way for improvement in other related areas that deal with equivalent versions.

Keywords: mutation testing, equivalent mutant problem, search-based software engineering, evolutionary algorithm

*Corresponding author: pedro.delgado@uca.es

1. Introduction

Mutation testing [1, 2] has gradually spread over the last decades as a powerful method to assess and improve the ability of test suites to detect faults. This testing technique is based on the artificial injection of different faults in the system under test (SUT), modeled by *mutation operators*. *Mutant* is the term commonly used to refer to the new program resulting from the fault injection. The execution of a mutant and the unmodified version of our program against the test suite produces an output; when there is no difference in the outputs, we say that the mutant is *alive*. Otherwise, the mutant is *killed*. This process allows for the detection of deficiencies in the designed test suite, represented by the surviving mutants. Live mutants, in turn, give us the opportunity to enhance its fault detection capability —we can add new test cases to kill them.

There are two key drawbacks when applying mutation testing: the high number of mutants generated and the need to identify *equivalent mutants* (i.e., mutants that are semantically identical to the original program and for which the same output as the original program is obtained). Especially, the latter problem, known as the Equivalent Mutant Problem, represents the major stumbling block to the widespread adoption of mutation testing in industrial software projects. Fortunately, much work has been done in the past to automatically reveal a portion of the set of equivalent mutants, especially in the last years [3, 4, 5, 6]. However, the detection of equivalent mutants is an undecidable problem [7], so this task has to be ultimately done in a manual way. This fact suggests a theoretical limit to the number of equivalent mutants that can be automatically detected. Given that situation, the following question arises: *is there something else we could do to deal with equivalent mutants?*

As far as we know, there are no studies in the literature evaluating or leveraging a possible connection among equivalent mutants. With connection, we mean whether the set of equivalent mutants (or some subsets within that set) share some features. Two similarities among mutants to be considered are the areas of the code in which they appear and the group of mutation operators that generate them. Such a study is relevant because if the connection exists, we could seize that knowledge to reveal some new equivalent mutants that are similar to other equivalent mutants previously identified —or even better, we could avoid their generation.

In recent years, search-based techniques have been proposed to alleviate the problems of mutation testing [8]. A good example is the technique commonly known as *Evolutionary Mutation Testing* (EMT from now on) [9, 10]. The rationale behind EMT is that most of the useful mutants (i.e., those that can induce the enhancement of the test suite) have interrelated features. As a result, a subset of those useful mutants can guide an evolutionary algorithm towards the selection of other valuable mutants. Indeed, experimental results applying EMT support this assumption because the evolutionary technique shows ability to quickly find subgroups of valuable mutants. Notably, EMT required to select less than 50% of the mutants in most cases to reach up to 80% of the size of a *mutant-adequate test suite* (that is, one that kills all non-equivalent mutants in

a program) [10].

While experiments using EMT show that useful mutants can lead to the generation of new useful mutants, it is currently unclear whether the same holds for equivalent mutants. In this paper, we aim to shed light on a possible equivalent mutant connection following an evolutionary approach. The intention of this study making use of EMT is twofold: (1) we seek to clarify whether equivalent mutants selected by EMT often originate from other equivalent mutants, and (2) we want to know if we can improve the original version of EMT by means of that information, that is, if we can reach a more refined subset of mutants with a view to generating new test cases. We propose a new version of EMT, called *Equivalence-Aware EMT* (EA-EMT), to penalize mutants known to be equivalent when they are selected during the search. The idea of penalizing mutants identified as equivalent during the search was suggested in a previous paper [11], but neither an experimental nor practical analysis was done regarding that proposal. The unique contributions of this paper are:

1. **We give evidence, through the application of EMT to six widely-used C++ programs, that equivalent mutants are often connected in most of these SUTs**. This is a novel study that suggests that we could still go a step beyond a hypothetical limit to the automated detection of equivalent mutants.

2. **We perform an experimental study of Equivalence-Aware EMT that shows that we can take advantage of a plausible equivalent mutant connection**. The study reveals that the existing link helps guide the evolutionary search towards useful mutants when compared to the original version of EMT. We observe this in most of the programs with statistical significance, even with few mutants cataloged as equivalent. Noticeably, the best results for EA-EMT were achieved when the subsets of known equivalent mutants were large, which supports the analyzed connection.

3. **We also conduct a practical study by applying EMT in combination with Trivial Compiler Equivalence (TCE), called *TCE-Aware EMT* or simply *TCE-EMT***. The use of TCE [5], a well-established technique to determine a subset of equivalent mutants, confirms that the previous experimental analysis also translates into improved results in practice, especially in terms of stability —the new information narrows the search, preventing great differences among different executions of the evolutionary algorithm.

The remainder of this paper is structured as follows. Section 2 presents approaches that tackle the equivalence detection problem, and explains how EMT operates. Section 3 describes the goals of the study and presents the research questions. Section 4 details the techniques proposed to study the equivalent mutant connection, and Section 5 shows the configuration of the experiments. The results of the experimental and practical study are presented and discussed

in Section 6 and 7, respectively. Finally, the threats to validity and conclusions are shown in Section 8 and 9.

2. Background

2.1. Equivalence determination problem

Equivalent mutant detection is known to be an undecidable problem [7]. This means that there is an implicit limit in the detection power of techniques seeking to determine the equivalence of mutants. As such, deciding which mutants are equivalent requires human effort, which is costly for companies and tedious for testers. To ease that problem, several solutions have been proposed in the past, with a growing emphasis in the last years [6, 12].

According to the review by Madeyski et al. [6], existing approaches either *suggest* the likelihood that mutants will be equivalent, *avoid* their generation or *detect* equivalent mutants directly. Among the *suggest* methods, we can cite several studies analyzing the impact of mutations on the code coverage (the lower the change in coverage, the more likely is that a mutant is equivalent) [4, 13, 14] and, more recently, the use of automata language equivalence for models [15]. As for *avoid*, several authors have devised different methods, such as the application of higher-order mutation to isolate equivalent mutants [3, 6], selective mutation [16] or the use of a genetic algorithm to co-evolve both test cases and mutants [17, 18].

Most of the studies fall on the previous two categories. However, the number of studies on *detect* approaches has increased in the last years, such as those related to constraint solving [19] or data-flow analysis [20]. The most remarkable advance in this area is the technique called Trivial Compiler Equivalence (TCE) [5, 21]. Based on the transformations performed by compiler optimizations, TCE can detect equivalent mutants that produce the same binary files as those of the original version of the program. In those studies, TCE showed the ability to detect around 30% and 50% of all equivalent mutants respectively in C and Java programs. Apart from the application of TCE to traditional C [21] and Java mutants [5, 22], this technique has also been assessed with memory mutants [23] and class mutants for C++ [24], being able to discard 5.5% and 27% of the set of equivalent mutants, respectively.

All previous approaches aim at individually identifying equivalent mutants. In a recent related study, Ayad et al. [25] proposed some metrics to estimate the number of equivalent mutants in a program rather than identifying them one by one. By contrast, our approach focuses on the features that equivalent mutants share from a global perspective to benefit the whole process of mutant selection. Nonetheless, EA-EMT, the evolutionary approach presented in this paper, has to rely upon one of the aforementioned techniques for the identification of individual equivalent mutants. In this paper, TCE will be used as the underlying technique (i.e., TCE-Aware EMT) because of its character of direct indication of equivalent mutants.

2.2. Evolutionary Mutation Testing

Evolutionary Mutation Testing (EMT) is a cost reduction technique that transforms the problem of mutant selection into a search problem. EMT uses an evolutionary algorithm to search for a subset of mutants with great potential to assist the tester in improving the fault detection ability of an existing test suite. This can be achieved by favoring the generation of mutants that originate from other seemingly useful mutants: EMT promotes the application of the most effective mutation operators throughout the execution of a genetic algorithm (GA) [26]; similarly, the technique focuses on areas of code requiring more attention, such as those that have been barely exercised by the test suite or when significant combinations of input values are lacking, which results in mutants remaining alive. The information on useful operators and the worst-covered areas is known by executing the mutants selected by the algorithm so far; then, each mutant is assigned a fitness value proportional to the test cases that kill them. When none of the test cases in our current test suite is able to detect a mutant, that mutant becomes a candidate to help us enhance the test suite.

This technique was first proposed by Domínguez-Jiménez et al. [9] to test Web Services compositions in WS-BPEL with the tool `GAmera` [27]. They showed that EMT could find a greater number of strong mutants (i.e., mutants that can potentially help improve the initial test suite) than random selection (RS). In the last years, EMT has been extrapolated to other contexts, supporting the technique as a useful method to reduce the cost of mutation testing. Delgado-Pérez et al. conducted new experiments on C++ applications, and reported that EMT not only found a greater number of strong mutants [28] but also induced a greater improvement of the test suite than RS with the same number of mutants [10, 29]. Recently, Gutiérrez-Madroñal et al. applied EMT to EPL queries for IoT systems [30], and proposed a refinement over the original version of EMT following the guided mutation approach. Finally, a multi-objective approach based on EMT was proposed for the selection of mutants to improve performance tests [31].

In the following subsections, we summarize the main components of the underlying GA. In-depth information on the technical details of this algorithm can be found in a previous paper [10].

2.2.1. Individual's encoding

EMT addresses the problem of mutant selection; accordingly, mutants are the individuals or solutions for the GA. Those individuals require to be encoded to be uniquely identified. The GA solution encoding in EMT considers the two main features of a mutant: the mutation operator that generates it and the location where the mutation is injected:

- **Operator**: each mutation operator is assigned a unique code (an integer number).

- **Location**: the mutants produced by a mutation operator are listed in sequential order from beginning to end of the code, and are assigned a location identifier based on the position in that list.

As an example, consider the mutation operator *IOD* [32], which is assigned the code 05. If *IOD* injects two mutations in the code in lines 10 and 12, the GA will encode those two mutants as *mutant_ 05_ 1* and *mutant_ 05_ 2* respectively.

2.2.2. Fitness function

Broadly speaking, a mutant killed by many test cases is unlikely to help design a missing test case. On the contrary, a mutant surviving the current test suite execution represents a good opportunity to reveal a deficiency in that test suite and ultimately help to fix that weakness. On the basis of the above:

- **Live mutants**: they are assigned the maximum fitness.
- **Rest of the mutants**: their fitness decrease with an increase in the number of test cases that kill them. The fitness also decreases with the number of mutants killed by the test cases that kill the mutant evaluated: this serves the purpose of representing how difficult is to select other mutants that cover the same functionality as that mutant.

Equation (1) reflects the previously discussed reasoning, where I is the mutant under evaluation, M and T are the numbers of mutants and test cases respectively, and m_{ij} can take the value 1 or 0 depending on whether mutant i is killed by test case j or not, respectively (therefore, m_{Ij} is a special case focused on mutant I). Note that the fitness of a mutant can vary depending on the initial test suite (TS).

$$\text{Fitness}(I, TS) = M \times T - \sum_{j=1}^{T} \left(m_{Ij} \times \sum_{i=1}^{M} m_{ij} \right). \tag{1}$$

As expected, this formula attaches the maximum fitness to live mutants ($M \times T$) and the rest of the mutants achieve a lower value in the range $[0, M \times T - 1]$. The more test cases kill I (then m_{Ij} is 1), the lower the fitness; also, the more mutants those test cases kill ($\sum_{i=1}^{M} m_{ij}$), the lower the fitness. For instance, consider the following execution matrix containing which of the test cases ($T = 4$) in our test suite detect which of the mutants selected by EMT ($M = 4$). In this case, the highest fitness belongs to the live mutant $mutant_2$: $4 \times 4 = 16$; on the contrary, the lowest fitness is assigned to $mutant_1$: $(4 \times 4) - 1$ ($test_1$ kills $mutant_1$) -3 ($test_4$ kills mutants 1, 3 and 4) $= 12$. As such, $mutant_1$ and $mutant_2$ have the lowest and the highest chances of being selected for reproduction, respectively.

$$
\begin{array}{c} \\ \text{mutant}_1 \\ \text{mutant}_2 \\ \text{mutant}_3 \\ \text{mutant}_4 \end{array}
\begin{array}{c}
\begin{array}{cccc} \text{test}_1 & \text{test}_2 & \text{test}_3 & \text{test}_4 \end{array} \\
\left(\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{array} \right)
\end{array}
$$

0: alive, 1: killed

By using this fitness function, the GA runs in the hope that the strongest mutants are selected and give birth to new live mutants in successive generations. In this way, the algorithm will tend to focus on promising mutation operators and poorly-covered functionalities.

2.2.3. Population

The GA has to form a new population of individuals with size PS (population size) in each new generation. These individuals are selected from all the mutants that can potentially be generated in the SUT. We have to distinguish between the first and the rest of the generations:

- In the *first generation*, PS individuals are selected completely at random.
- In the *succeeding generations*, PS individuals are selected as follows:
  - N individuals are again randomly generated, allowing the GA to focus on other relatively unexplored areas.
  - $PS - N$ individuals are generated by means of variation operators.

The GA produces new generations of mutants until a *termination condition* is satisfied (e.g., when reaching a percentage of the total number of possible mutants).

2.2.4. Parent selection and variation operators

The process to generate new individuals from the previous generation of mutants involves two steps:

- **Parent selection**: This step seeks to select potentially useful mutants; the features of these mutants will be partially inherited by new mutants in the next generation. To this aim, the GA in EMT assigns a fitness to each mutant and makes use of *fitness proportionate selection*. Applying this selection method, the probability for a mutant to be selected is proportional to its fitness value.
- **Application of variation operators**: Once parents have been selected, they undergo a process of mutation and recombination. The mutation operator produces a change in one of the two features of the mutant encoding (operator or location), where the integer is increased or decreased

by a small value to produce new nearby mutants. Regarding the crossover operator, two parent mutants exchange their genes, resulting in two new mutants with half of the information from each parent. p_m and p_c represent the probability that mutation or crossover operators are applied respectively.

3. Goals of the study

3.1. Description

As far as the authors know, there are no studies in the literature that directly analyze the level of mutant connection. In particular, it remains unclear whether equivalent mutants share some features, making it possible to establish a connection among them. This study is interesting because such a link among equivalent mutants would allow moving from an individual perspective of equivalent mutant identification to a global perspective that could benefit the whole mutation testing process.

Figure 1 serves to illustrate the reasons why equivalent mutants can be often linked among them. The figure shows four equivalent mutants generated by two different mutation operators in two functions of the original code. In this paper, we argue that these mutants are connected (and form a subgroup of equivalent mutants) in the sense that they have been produced by a subset of operators and they are located in close lines of the code in two adjacent functions. Indeed, the mutation operator inserting the post-increment operator could generate two additional equivalent mutants in these functions ($b{+}{+}$ in line 3, $a{+}{+}$ in line 8). In this particular case, other mutation operators, such as the one replacing the arithmetic operators in lines 3 and 8, would not produce equivalent mutants; also, the operator swapping the parameters would not generate an equivalent mutant in similar functions like the one to divide two integers. The connection of these mutants has to do with both a syntactic and a semantic aspect; there is a syntactic relation because mutations 1, 3 and 2, 4 model a similar change within the structure of both functions, but there is also a semantic relation because these mutants affect two functions sharing similar characteristics (including the commutative property in this example), which is commonplace in most programs. The existence of equivalent mutants that present a syntactic and a semantic similarity has been assessed in the past [33].

In this work we propose the application of EMT as the method to gain insight into a plausible connection among equivalent mutants. As previously stated, EMT works on the basis that useful mutants can lead to new useful mutants. This strategy has empirically shown to reduce the cost of mutation testing [10]. In this paper we make use of EMT to better understand the extent to which equivalent mutants can also lead to other equivalent mutants. Moreover, this method allows us to put into practice and take advantage of a hypothetical connection among these mutants.

We have to note that the evolutionary algorithm has not directly addressed the Equivalent Mutant Problem up to now. Currently, its fitness function is

```
1  //Function sum              Mutant 1:                  Mutant 2:
2  int sum(int a, int b){     int sum(int a, int b){     int sum(int b, int a){
3     return a + b;              return a++ + b;            return a + b;
4  }                          }                          }
5
6  //Function multiply         Mutant 3:                  Mutant 4:
7  int mul(int a, int b){     int mul(int a, int b){     int mul(int b, int a){
8     return a * b;              return a * b++;            return a * b;
9  }                          }                          }
```

Figure 1: Example of "linked" equivalent mutants. The mutants are generated inserting the post-increment operator and swapping the parameters.

not able to distinguish between mutations not detected yet (live mutants) and mutations that cannot actually be detected (equivalent mutants). Indeed, the fitness function in EMT could have unintentionally been degrading its performance: the algorithm can misleadingly be breeding new mutants derived from equivalent mutants instead of useful mutants. If that is the case, EMT could be combined with state-of-the-art techniques to identify equivalent mutants, like *Trivial Compiler Equivalence* (TCE) [5, 21], thereby being able to distinguish live mutants from equivalent ones in a real situation.

3.2. Study on the equivalent mutant connection

In this paper we propose a study on the connection among equivalent mutants from two different perspectives: an experimental study, where we conduct experiments in a controlled environment using equivalence information that is not available in a real setting, and a practical study, where we simulate a real scenario using mechanisms that provide that information at runtime.

- **Experimental study:** The goal of this study is to analyze our approach in depth, evaluating different configurations to increase our knowledge of mutant connection. Such experiments are suitable on account of the uncertainty regarding the available information on equivalent mutants, their distribution in each subject under test and how the GA selects them. In connection with this, we will follow a twofold analysis:

  1. **Equivalence family tree**: This tree represents each equivalent mutant together with its ancestors, showing which mutants are derived from which mutants by means of the application of variation operators. Figure 2 shows an example of equivalence family tree. The analysis of the inheritance relationships between equivalent mutants is useful to know about the origin of these mutants.
  2. **Equivalence-Aware EMT (EA-EMT)**: This is a modified version of the original EMT in which information of known equivalent mutants is injected. Unlike the original EMT, the incorporation of this new information enables the selection method to distinguish between useful and non-useful individuals. To do so, this technique penalizes

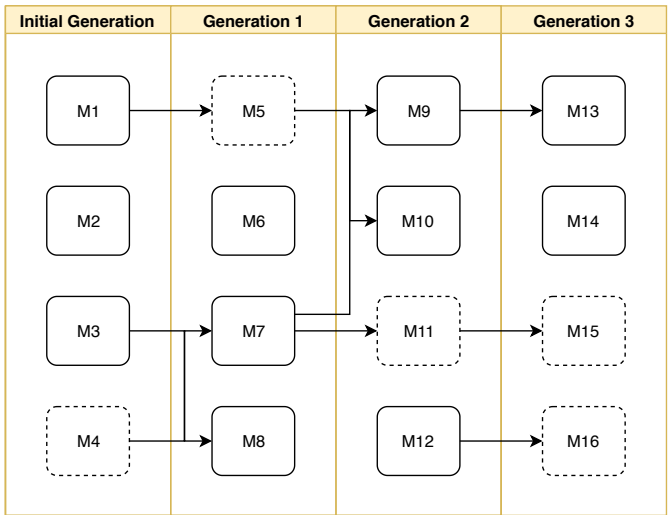

Figure 2: Example of equivalence family tree, where dotted boxes represent equivalent mutants. Among the four equivalent mutants generated with variation operators after the initial generation, only M15 and M11 have equivalent ancestors (M11 and M4 respectively).

the fitness of mutants identified as equivalent as they appear during the search. This modification could eventually bias the search in favor of the generation of valuable mutants.

- **Practical study:** This study seeks to assess the practical applicability of the approach through the combination of EMT with TCE [5]. TCE allows the automated detection of equivalent mutants by means of the transformations performed by compiler optimizations: if the application of these optimizations in a mutant and the original code results in identical binary files, the mutant can be determined as equivalent. Based on this idea we propose the following technique:

  1. **TCE-Aware EMT (TCE-EMT)**: This is a specialized version of EA-EMT where the source of information on equivalent mutants is provided by the application of TCE to the mutants in each generation. This is a convenient technique since, as claimed by the authors of the technique [21], mutants identified by TCE can be safely tagged as equivalent; moreover, its application comes 'almost for free' once mutants have been compiled, that is, its impact on the efficiency is marginal when compared to the need of manual review.

Details on the internals of the aforementioned methods are later provided in Section 4.

3.3. Research questions

The first research question seeks to know whether there exists a connection among equivalent mutants. To that end, we will apply EMT and will generate an *equivalence family tree*. An analysis of the equivalent mutants that have other equivalent mutants as descendants or ancestors can shed light on this.

> ***Research question 1:*** *Are the equivalent mutants in the equivalence family tree connected among them?*

In connection with the previous question, it is possible to analyze in more depth the reasons behind a possible link. More specifically, we want to know whether the linked equivalent mutants share their mutation operator.

> ***Research question 2:*** *Do the linked equivalent mutants originate from the same mutation operators?*

In case that such a connection among generations of mutants exists, we want to know whether the evolutionary approach can benefit from that information. This can be assessed by applying *EA-EMT* to punish equivalent mutants so that they are not selected to give birth new mutants. This is the aim of the third research question:

> ***Research question 3:*** *Can the proposed equivalence-aware approach further reduce the number of mutants selected by EMT?*

Related to the previous question, it is equally interesting to know if we can seize that connection to improve the effectiveness of the evolutionary approach in different contexts. For example, with different subsets of equivalent mutants of varying size, or as the level of demand of test suite improvement increases. To that end, we include the following research question:

> ***Research question 4:*** *Does the equivalent-aware approach adapt to different scenarios?*

Finally, we want to know if the proposed approach can be useful in practice. By combining EA-EMT with TCE as an automated method to identify equivalent mutants at execution time (i.e., *TCE-EMT*), we could assess if we can take advantage of that connection in real scenarios. To study the practical applicability of this approach, we add the following research question:

> ***Research question 5:*** *Can TCE-EMT be applied in practice to take advantage of the equivalent mutant connection?*

4. Methodology

In this section, we detail the working principles of the equivalence family tree and the new proposed version of EMT. These techniques first require a set of mutation operators and an initial test suite, which EMT will try to improve by means of the selected mutants. Then, both require information on the set of equivalent mutants in the SUT. In our experimental study, the set of equivalent mutants can be recorded thanks to a previous execution and inspection of mutants; in our practical study, TCE provides the equivalence information.

4.1. Equivalence family tree

This tree, which collects the ancestors of equivalent mutants, is generated while the GA is in progress. Below, we enumerate the steps to generate and analyze the tree (we use the tree in Figure 2 as a running example):

1. Starting from the second generation[1], those mutants derived from other mutants of the previous generation are processed.

2. Each processed mutant is then registered in the tree together with its parent mutant/s —one or two parents depending on the variation operator: mutation or recombination (see *Parent selection and variation operators* in Section 2.2). Therefore, the parent of M5 is M1 and the parents of M7 are M3 and M4.

3. After the execution, each mutant contained in the tree is analyzed. When a mutant belongs to the set of equivalent mutants, its ancestors are traversed recursively, seeking for equivalent ancestors. In this way, we can know that the equivalent mutant M11 has M4 as an equivalent ancestor.

4. Finally, the percentage of equivalent mutants selected by the GA descending from other equivalent mutants is calculated. In our example, 50% of the selected equivalent mutants have been derived (directly or indirectly) from other equivalent mutants.

We also repeat steps 3 and 4 but observing the number of equivalent and non-equivalent mutants with equivalent descendants.

4.2. Equivalence-Aware EMT

Making EMT aware of equivalence information involves adding a new step to the conventional application of EMT [10]. This step (marked in bold) distinguishes between equivalent and non-equivalent mutants, when possible:

[1]Note that mutants in the first generation and those selected at random to maintain the diversity in the rest of generations (N) are skipped as they have no ancestors (see *Population* in Section 2.2). Therefore, in our example, M1–M4 and M6, M12 and M14 respectively are not processed.

1. The first generation of mutants is created at random.
2. Mutants in the current generation are run against the test suite TS.
3. The fitness function is computed for each of the mutants in the generation, based on the execution results in the previous step (i.e., which test cases detect which mutants).
4. **For each live mutant I, then:**
   (a) **If the mutant is known to be equivalent, it is applied the death penalty [34].** This means that we attach the lowest fitness to the mutant:

$$\text{Fitness}(I, TS) = 0 \tag{2}$$

   (b) **Otherwise, the mutant maintains the maximum value:**

$$\text{Fitness}(I, TS) = M \times T \tag{3}$$

5. The *parent selection method* selects the mutants on the basis of the computed fitnesses in the two previous steps. The mutants for the next generation are then produced by means of the predefined *variation operators*.
6. If the stopping point is not reached, the algorithm goes back to step 2.
7. The algorithm returns the set of mutants generated.

As a result of the new step (step 4 above), mutants known to be equivalent are assigned the worst possible fitness. In this way, those mutants will not be selected for reproduction (unless all eligible mutants are equivalent). The aforementioned process resembles to constrained optimization problems, where only a subset of all solutions satisfy the constraints (feasible solutions) and the fitness function penalizes unfeasible solutions. In our particular case, non-equivalent mutants and equivalent mutants play the role of feasible and unfeasible solutions respectively —only the former are useful for the improvement of the test suite.

4.3. TCE-Aware EMT or TCE-EMT

The combination of EMT and TCE leads to a specialized version of EA-EMT, TCE-EMT, where the application of TCE becomes the source of equivalence information. To that end, the previous step *4.a* is modified as follows (marked in bold):

4. For each live mutant I, then:
   (a) **If the binary files of mutant I and the original program after applying compiler optimizations are identical**, it is applied the death penalty.

5. Experimental Setup

In this section, we first describe general aspects related to the experiments performed in our study, such as the stopping condition, case studies, mutation operators and tools. The two last subsections provide details on the particular configurations for the experimental and practical study.

5.1. Termination condition

As the first step for our evaluation, we need to define a proper termination condition for the GA. It is important to note that different criteria can be used depending on whether the algorithm is applied in practice or is under evaluation in a controlled experiment:

- **In a real situation**, we have no information on useful mutants or what is the level of improvement required by the current test suite. Our options to stop the algorithm narrow down to a number of epochs or mutants generated.
- **In an experimental procedure**, in contrast, we can count with broader information; we can perform an analysis of the mutants and the test suite prior to the GA execution. We can even enhance the test suite based on the inspection of live mutants, thereby reaching a mutant-adequate test suite. This test suite can serve as a reference to know how close the GA execution is to achieve certain improvement on the initial test suite. Therefore, as presented in previous papers [10, 28], it can be used to make an informed decision on the point to stop the algorithm.

Since our study is essentially experimental, we focus on the second case. As an example, imagine that our previously generated mutant-adequate test suite is composed of 20 test cases, and that we want to stop the GA execution when reaching 75% of the size of the mutant-adequate test suite (denoted with P), that is, 15 test cases; in this way, the GA will only stop when the mutants selected so far could induce the design of 15 out of those 20 test cases. This measurement is based on the execution results of the mutant-adequate test suite on the selected mutants. With that information, we can map which test cases in the mutant-adequate test suite are required to detect each of the mutants selected by the algorithm.

In our case, the computation of the *minimal* mutant-adequate test suite offers improved accuracy of the test suite enhancement that would be achieved; in a minimal test suite we cannot remove any test cases without reducing the number of mutants detected, and therefore we can be sure that all its test cases are really necessary.

Taking the level of test suite improvement as a reference is convenient in this study. This measurement allows determining whether the refinement in EA-EMT is accompanied by a more effective selection of mutants —therefore, reaching the expected level of test suite enhancement P more quickly than the original EMT.

Table 1: Features of the SUT: number of mutants, number of mutants remaining alive after the execution of the original test suite (TS_{Ori}), percentage of mutants manually identified as equivalent and size of the augmented mutant-adequate test suite (TS_{MA}).

	TCL	**DPH**	**TXM**	**RPC**	**SQL**	**DOM**
Mutants	137	219	614	191	683	1,146
Alive	45	103	159	76	446	348
Equivalent (%)	15%	32%	15%	14%	39%	21%
$\|TS_{Ori}\|$	17	61	57	26	271	46
$\|TS_{MA}\|$	24	70	62	34	294	56

5.2. Case studies, mutation operators and tools

In the experiments, we make use of six open source C++ programs. Namely, *Matrix TCL Pro* (TCL) [35], *Dolphin* (DPH) [36], *TinyXML2* (TXM) [37], *XmlRPC++* (RPC) [38], *MySQL Server* (SQL) [39] and *QtDOM* (DOM) [40]. Table 1 shows the number of mutants in those SUTs, live mutants with the initial test suite (the one distributed with the programs) and the percentage of equivalent mutants. We separated out useful and equivalent mutants manually, which act as ground truth in our experiments. This table also shows the size of the mutant-adequate test suite once we augmented the initial test suite with manually-generated test cases to kill non-equivalent mutants in these programs —as described in the previous section, this enables the application of the termination condition.

We apply the tool `MuCPP` to produce the mutants, focusing on class mutation operators for this language [32]. Table 2 details the mutation operators that generate some mutants for these programs in our study. A similar list of class mutation operators is implemented in other widely-used tools for object-oriented languages, like MuJava for Java [41], and class-level operators have been extensively evaluated in previous research papers [32, 42, 43, 44, 45].

The tool `GiGAn` [29] runs EMT, allowing the execution of the GA presented in Section 2.2 in connection with the mutation tool. For this work, `GiGAn` was parameterized to be able to select between the original EMT or the new EA-EMT. In the case of EA-EMT, we additionally modified the tool to:

- Implement step 4 presented in Section 4.2 and change the fitness of known equivalent mutants.
- Incorporate information on equivalence from an external source. In these experiments, we set the source as a text file where the names of manually-identified equivalent mutants are listed.

As for the configuration parameters of the GA (see Section 2.2), we use the set of optimized values applied in previous experiments [10, 29]:

- *Population size (PS)*: 5% of the total number of mutants.
- *Mutants generated randomly in each generation (N)*: 10% of the population size —the rest are generated with variation operators.

Table 2: List of mutation operators

Constructors/Destructors		**Polymorphism**	
CTD	*this* keyword deletion	PCD	Type cast operator deletion
CTI	*this* keyword insertion	PCI	Type cast operator insertion
CDC	Default constructor creation	PCC	Type cast operator change
CDD	Destructor method deletion	PNC	New method call with child class type
CID	Member variable initialization deletion	PPD	Parameter variable declaration with child class type
CCA	Copy constructor and assignment operator overloading deletion	PMD	Member variable declaration with parent class type
Inheritance		**Overloading**	
IHI	Hiding variable insertion	OMD	Overloading method deletion
ISD	Base keyword deletion	OMR	Overloading method contents replace
ISI	Base keyword insertion	OAN	Argument number change
IOD	Overriding method deletion	**Exceptions**	
IOP	Overriding method calling position change	EHC	Exception handling change
IOR	Overriding method rename	**Object replacement**	
IPC	Explicit call of a parent's constructor deletion	MCO	Member call from another object
		MCI	Member call from another inherited class

- *Mutation and crossover probability (p_m and p_c)*: 30% and 70% respectively.

Finally, this study counts with a number of independent and dependent variables. The SUTs, the initial and mutant-adequate test suites, the list of mutation operators, the configuration of the GA and the set of equivalent mutants, constitute the independent variables. Notice that the set (or sets) of equivalent mutants used in the experimental and the practical study will be detailed in the following subsections. The used cost-effectiveness measurements depend on the applied technique:

- *Family tree:* To answer RQ1, equivalent mutants in the tree with equivalent ancestors, and equivalent and non-equivalent mutants with equivalent descendants; to answer RQ2, pairs of ancestor-descendant equivalent mutants sharing the same mutation operator.
- *EA-EMT and TCE-EMT:* mutants generated until reaching the termination condition, i.e., a percentage of the size of the minimal mutant-adequate test suite (see the previous subsection). This measure will be used to answer RQ3, RQ4 and RQ5.

5.3. Configurations for the experimental study

In this subsection, we present specific configurations to properly analyze EA-EMT and the equivalence family tree:

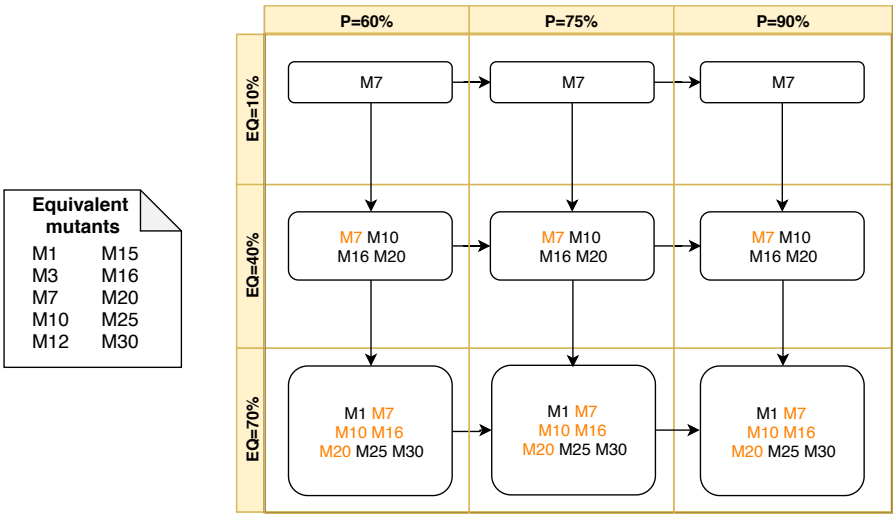

Figure 3: Example of subsets of equivalent mutants for the experiments with EA-EMT. As it can be seen, the subset with size $EQ = 70\%$ contains the equivalent mutants in the subset with $EQ = 40\%$, which in turn contains the subset with $EQ = 10\%$. These subsets are used to evaluate the behavior of the GA with the 3 stopping conditions ($P = 60\%$, 75% and 90%).

EA-EMT. We have to configure the following parameters for the experiments applying EA-EMT:

EQ $\rightarrow$ Size of the **subset of equivalent mutants** of which the GA is aware, measured as a percentage of the whole set of known equivalent mutants. Setting fixed instead of random sizes for EQ allows us to observe the evolution of the results with small and large subsets of equivalent mutants.

S $\rightarrow$ Number of **different subsets of equivalent mutants** for each size EQ. By preparing different subsets, we reduce the potential bias associated with just assessing a single random subset.

R $\rightarrow$ Number of **independent runs** with each subset of equivalent mutants.

P $\rightarrow$ Stopping condition or **level of test suite improvement** required. We take this parameter into account because the GA can behave differently with varying levels of demand.

Notice that the lower EQ (size of the subset of equivalent mutants), the more similar the behavior of EA-EMT will be to the one of the original EMT. In contrast, with $EQ = 100\%$, EA-EMT becomes aware of all available information on equivalence. Also note that, for each percentage set as EQ, S subsets of equivalent mutants are selected; for each of these subsets, R executions are performed. Therefore, we derive $|EQ| \times S \times R \times |P|$ results from each case study. As a consequence, we have selected a limited set of options to avoid an exponential explosion of executions. Namely:

- **EQ** = {10%, 40%, 70%}.
- **S** = 5 subsets.
- **R** = 30 executions.
- **P** = {60%, 75%, 90%}.

Therefore, on the basis of the previous formula, we have $3 \times 5 = 15$ available subsets and $3 \times 5 \times 30 \times 3 = 1,350$ results with each program. Figure 3 illustrates the selection mode of subsets of equivalent mutants in relation to the parameters EQ and P. In this example, there are 10 known equivalent mutants; therefore, a subset with size $EQ = 10\%$ contains only one mutant randomly selected, *M7* in this case. Note now that the process is incremental: the subset with size $EQ = 10\%$ is contained in the subset with $EQ = 40\%$ and again the subset with $EQ = 40\%$ is contained in the subset with $EQ = 70\%$. This allows us to observe the evolution of EA-EMT when new equivalent mutants are added to the subset. We should also remark that the same subsets are used to evaluate the different stopping points (P), which allows assessing EA-EMT's evolution over time. The same process depicted in Figure 3 is carried out S times.

Finally, the results of EA-EMT with these parameters are compared with the results of 30 executions of the original EMT in order to contrast the performance of both techniques, including a statistical test of the significance of the results.

Equivalence family tree. In the analysis of the tree, we also apply $R = 30$ independent runs and the same three values for P.

5.4. Configurations for the practical study

In this subsection, we present specific configurations for the experiments applying TCE-EMT. In order to detect some equivalent mutants, mutants selected by TCE-EMT in each generation undergo a TCE test. The application of TCE in our study relies on several software systems (further information can be found in [24]):

- First, the mutants and the original program are compiled with `g++` using the same optimization option. The output is a set of binary files.
- Second, `MuCPP` generates the mutants as branches of the version control system `git` [32]. As such, the option *diff* provided by `git` is used to compare the same binary files in the program and in the mutants. As an example, we compare *binaryfile* in *mutant1* with the one in the original program (represented by branch *master*) through the following command:

```
git diff --binary master mutant1 binaryfile
```

  When the output of that command is empty, there is no difference between both branches and, therefore, the mutant is flagged as equivalent.

Table 3: Number and percentage of equivalent mutants detected by TCE with *-O2* (*Mutants* and *% Equivalent*), and percentage of detected to the total number of mutants (*% Total*).

	TCL	**DPH**	**TXM**	**RPC**	**SQL**	**DOM**
Mutants	9	19	30	4	11	20
Equivalent (%)	45.0	27.5	33.0	14.8	4.1	8.5
Total (%)	6.6	8.7	4.9	2.1	1.6	1.7

Table 3 presents the results of the application of TCE in the programs using the popular optimization option *-O2*. As shown, there is a varied range of percentages of known equivalent mutants, from 4.1% in *SQL* to 45% in *TCL*. Depending on the total number of equivalent mutants in each program, this translates into subsets of equivalent mutants of a different size. Also, these subsets represent a different portion with respect to the total number of mutants, e.g., 19 out of 219 mutants (8.7%) are identified as equivalent with TCE in *DPH*.

As in the experimental study, results are based on 30 independent runs of TCE-EMT, and we measure the mean percentage of selected mutants and the standard deviation as indicators of its performance.

6. Experimental Study on the Equivalent Mutant Connection

This section is dedicated to the first part of our analysis (experimental study). Hereafter, we present and discuss the results of the equivalence family tree and EA-EMT.

6.1. Results of the equivalence family tree

Table 4 presents the results of the analysis of the equivalence family tree in the SUTs. This table shows average percentages of equivalent mutants — selected in 30 executions of the original EMT— that had another equivalent mutant as an ancestor.

Table 4: Average percentage of equivalent mutants with equivalence ancestors in the equivalence family tree, divided by program and stopping condition P. The average percentage of all programs is also shown.

Program	**P**		
	60%	**75%**	**90%**
TCL	44.3	46.4	47.3
DPH	51.7	52.6	54.9
TXM	69.8	77.3	76.3
RPC	36.8	42.8	46.7
SQL	78.4	75.0	68.8
DOM	83.0	80.0	77.7
Average	60.7	62.3	61.9

Based on these results, we can say that there is a notable percentage of equivalent mutants with equivalent ancestors (over 60% on average for all values

of P, 62.3% with $P = 75\%$). The percentage was quite relevant in the cases of *TXM*, *SQL* and especially *DOM* (83% with $P = 60\%$). In contrast, the percentage is not as high in *TCL* and *RPC*, the programs with the lowest number of mutants and a low percentage of equivalent mutants in total (see Table 1). Analyzing the variation among executions when $P = 90\%$, we observe that the standard deviation ranges between 9-10% (*TXM*, *SQL* and *DOM*) and 22-24% (*RPC* and *TCL*); this measure partially explains the results in the table, given that the executions in the two latter programs are not as stable as in the rest. Interestingly, in most cases there is an increase in that percentage from $P = 60\%$ to $P = 75\%$ (e.g., from 36.8% to 42.8% in *RPC*); probably, the GA required a longer execution time to be able to find more subgroups of equivalent mutants.

At this point, we also wondered whether the probability of an equivalent mutant to have an equivalent descendant was higher than the probability of a non-equivalent mutant to have an equivalent descendant. Table 5 shows the results of this analysis (percentage of equivalent and non-equivalent mutants with equivalent descendants) over the generated family trees. In all the cases, there were more pairs *<equivalent ancestor, equivalent descendant>* than *<non-equivalent ancestor, equivalent descendant>*. We also performed a hypothesis statistical test, where the null hypothesis (H_0) states that there is not a statistical difference between these pairs and the alternative hypothesis (H_1) states that the difference in favor of pairs of equivalent mutants is statistically significant. The test provides a *p-value* in the range [0,1]. We take as reference the widely-used level of significance 0.05; *p-values* under that level are considered to be sufficient to reject H_0. According to the *p-values* in this table, we can reject H_0 in 5 out of 6 programs. Although there are several factors influencing the outcome (e.g., how mutants are spread across the program and mutation operators), all these results support the assumption of the equivalent mutant connection.

Table 5: Percentage of equivalent (Equiv.) and non-equivalent (Non-Equiv.) mutants with an equivalent mutant as a descendant in the family tree. The table also shows the *p-value* of a Wilcoxon signed-rank test between the results of both variables in 30 executions.

Program	**P = 90%**		
	Equiv.	**Non-Equiv.**	***p-value***
TCL	24.26	17.14	0.0087
DPH	42.63	37.11	0.0003
TXM	25.26	21.01	0.0008
RPC	19.93	18.35	0.1267
SQL	40.86	38.33	0.0003
DOM	29.31	25.64	1.95e-05

***Answering RQ1**, the percentage of equivalent mutants descending from other equivalent mutants when running EMT was over 60% on average. The minimum and maximum percentage was 36.8% (*RPC*) and 83% (*DOM*) respectively. From the other perspective, we also found that it is more likely that an equivalent mutant gives rise to an equivalent mutant when compared to the probability of a non-equivalent one. Given these results, it is reasonable to think that equivalent mutants can lead to other equivalent mutants, that is, equivalent mutants (or some subsets within that set) are connected because they are produced in certain areas by certain mutation operators.*

Figure 4 analyzes in detail the mutation operators of the connected equivalent mutants in the tree. First, it shows the percentage of those pairs of ancestor-descendant equivalent mutants that share the same mutation operator in each program. As it can be seen, in four of the programs the percentage is between 39% and 47%. As in Table 4, the programs *TCL* and *RPC* do not adhere to this trend probably because of the few mutants generated by each operator. Second, the figure shows those shared mutation operators as a percentage of all the equivalent mutants sharing the mutation operator. Up to 17 out of 25 operators participate in the detected connection between equivalent mutants, ranging from 0.2% (ISI) to 27.3% (IHI). In general, these percentages are in line with the number of equivalent mutants produced by each mutation operator.

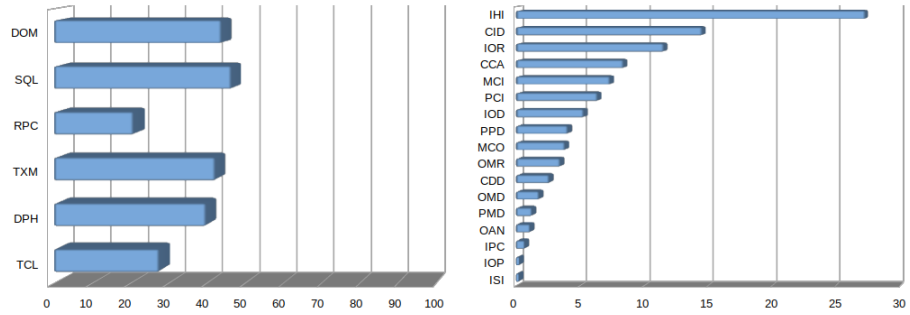

Figure 4: Percentage of all the equivalent mutants in the family tree with an equivalent ancestor that share the mutation operator in each of the six programs (left). Shared mutation operators in percentage (right).

***Answering RQ2**, a considerable number of the connected equivalent mutants (37% on average) originate from the same mutation operator, distributed among a broad set of operators (17 out of 25). While significant, the mutation operator is not the only reason behind the detected connection and the link is not directly related to a few particular transformations. These results suggest that the equivalent mutant connection deserves to be explored from a global perspective.*

6.2. Results of EA-EMT

Table 6 furnishes the comparative study between the application of the original EMT and the specialized version EA-EMT. The figures in this table represent the average percentage of mutants returned by the GA once the execution ends. In the case of EA-EMT, the table divides the results by the parameters P and EQ. As an example taking the program *TCL* and the stopping condition $P = 60\%$, the average percentage of mutants selected by the original EMT (*Ori.*) in 30 executions was 25.4%; in the same program and P, and being $EQ = 40\%$, the average percentage of EA-EMT in 150 executions ($S = 5$, $R = 30$) was 24.9%. Since the lower the percentage, the more effective the technique, in this case, EA-EMT surpassed the performance of the original version of EMT. The table also shows the best result obtained among the 15 different subsets of equivalent mutants (average of 30 executions with each subset). Thus, the best average percentage obtained with one of the formed subsets was 23.5%, whose size was $EQ = 70\%$.

Table 6: Results of the comparative study between the original EMT (*Ori.*) and EA-EMT. The table shows the average percentage of mutants generated by the GA (*Avg*) until reaching the three stopping points (P) per program. Results of EA-EMT are in turn divided by the three sizes for the subsets of known equivalent mutants (EQ). The average of *Ori.* is based on $R = 30$ executions; the average of EA-EMT is based on 30 runs of each of the five subsets generated (S), that is, 150 executions. It also shows the best average result (B) of EA-EMT with one of those subsets (the row in which the result appears represents its size EQ). The cases where EA-EMT outperformed *Ori.* are highlighted in bold.

P	EQ	TCL		DPH		TXM		RPC		SQL		DOM	
		Avg	B	Avg	B	Avg	B	Avg	B	Avg	B	Avg	B
	Ori.	25.4		36.1		13.2		31.0		19.6		8.8	
	10	**25.2**		36.2		13.6		**30.9**	**30.7**	19.9		**8.6**	
60	40	**24.9**		**35.0**	**34.2**	13.8	**12.9**	31.8		**19.6**	**18.7**	**8.4**	**8.0**
	70	**24.6**	**23.5**	**35.6**		13.8		31.7		**19.4**		**8.5**	
	Ori.	37.2		49.7		19.3		45.2		35.6		13.3	
	10	**36.4**		**49.0**		19.4		**44.5**		37.0		**12.8**	
75	40	**35.3**		**48.3**	**47.0**	**18.9**	**18.1**	**44.5**		36.3		**12.6**	
	70	**35.3**	**33.3**	**48.6**		**18.7**		**43.3**	**42.7**	**35.2**	**33.6**	**12.4**	**11.8**
	Ori.	49.2		66.3		31.9		59.4		58.2		21.4	
	10	50.4		66.5		33.0		**58.7**		**56.8**		**20.7**	
90	40	50.3	**47.6**	**65.7**		32.4		**58.6**		**55.9**		**20.7**	**20.2**
	70	50.0		**65.7**	**63.9**	32.0	**30.8**	**59.3**	**56.9**	**55.8**	**55.0**	**20.6**	

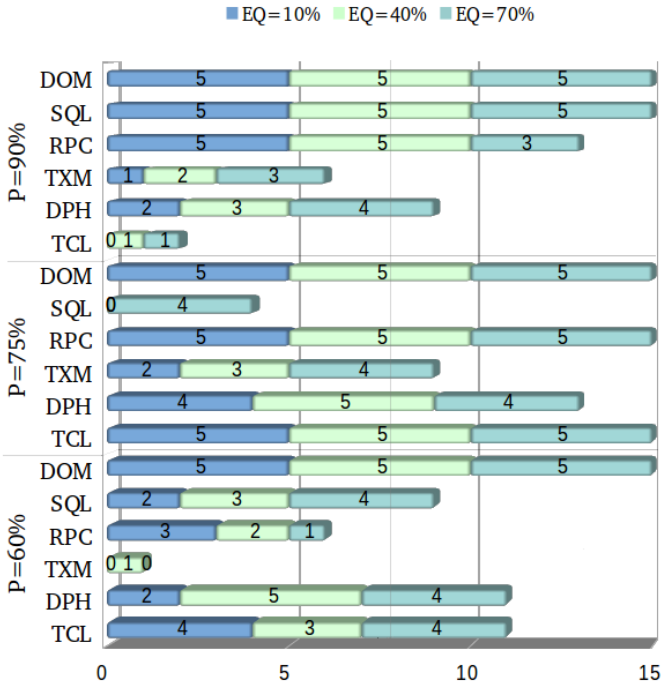

Figure 5: Number of subsets of equivalent mutants in which EA-EMT outperformed the original EMT on average. Results are divided by the 3 stopping points (P = 60%, 75% and 90%), the 6 programs and the 3 percentages of known equivalent mutants (EQ = 10%, 40% and 70%). Since the number of different subsets with each EQ is $S = 5$, each fragment of a bar is in the range [0,5].

Figure 5 complements the information provided by Table 6 by depicting how many of those 15 subsets allowed EA-EMT to achieve a better outcome when compared to that of the unmodified version of EMT. Results are again divided by P, EQ and program. Analyzing these data in depth, we can remark the following findings:

- **Original EMT vs EA-EMT**: In most cases, EA-EMT performed better than EMT in its original state. The best result was found in *DOM* (the average percentage was reduced for all values of P and EQ). Furthermore, all the 15 subsets provided a better result regardless of P (see Figure 5). On the contrary, the injection of equivalence information in *TXM* did not offer improvements in general. In quantitative terms, the largest difference between both techniques was reported in <SQL,P90,EQ70> (55.8% vs 58.2%). Regarding the best result, EA-EMT decreased the percentage of mutants generated by EMT in almost 4% in <TCL,P75,EQ70> (33.3% vs 37.2%). Still, note that the differences among SUTs and values of P are not completely comparable since the margin for improvement is not

the same in all cases.

- **Influence of EQ**: Overall, the percentage of mutants selected by EA-EMT decreased as the number of known equivalent mutants (EQ) increased, becoming this especially clear in the case of *TCL*. This situation is not particularly surprising, as counting with more equivalence information should have a greater impact on the result. However, this is not a rule of thumb; sometimes providing more information had detrimental effects on the search, such as in <RPC,P90> when moving from $EQ = 40\%$ to $EQ = 70\%$ (58.6% and 59.3%, respectively). An interesting observation is that the best result was always obtained with a subset of size 40% (8 times) or 70% (9 times), except in <RPC,P60>.

  To deepen in the influence of EQ, Table 7 shows the results of EA-EMT when $EQ = 100\%$, that is, when all equivalent mutants are known. In general, EA-EMT outperforms EMT and, in many cases, it keeps showing an improving tendency as the equivalence information increases. For instance, the performance of EA-EMT improves when moving from $EQ = 70\%$ to $EQ = 100\%$ in <TCL,P75> (35.3% and 34.2% respectively). This means that, even though we were able to detect all equivalent mutants, EA-EMT would further improve the mutant selection process in most cases.

- **Results in relation to P**: As it can be seen from Table 6 and Figure 5, the best results were obtained when $P = 75\%$. As an example, the executions of EA-EMT with all 15 subsets outperformed EMT in up to three of the programs (*TCL*, *RPC*, and *DOM*) with this stopping point. This seems to indicate that $P = 75\%$ is the point of the execution when the GA maximizes the contribution of the equivalence information. Again, this is not always true, as in the case of *SQL*, where EA-EMT required a longer execution time ($P = 90\%$) to reach its best performance.

Table 7: Results of the comparative study between the original EMT (*Ori.*) and EA-EMT when $EQ = 100\%$. The table shows the average percentage of mutants generated by the GA (30 executions) until reaching the 3 stopping points (P) per program.

P	TCL		DPH		TXY		RPC		SQL		DOM	
	Ori.	EQ100	Ori.	EQ100	Ori.	EQ100	Ori.	EQ100	Ori.	EQ100	Ori.	EQ100
60	25.39	**24.52**	36.14	**35.99**	13.21	14.23	30.97	32.07	19.62	**19.38**	8.79	**8.31**
75	37.24	**34.23**	49.75	**47.97**	19.26	**19.24**	45.29	**43.22**	35.61	35.89	13.33	**12.18**
90	49.24	49.73	66.33	**65.18**	31.93	**31.57**	59.42	61.74	58.19	**56.37**	21.41	**19.92**

Finally, we performed a statistical test to compare the executions of the original EMT and EA-EMT for $P = 75\%$. In this case, H_0 states that the difference between the percentage of mutants generated by EMT and EA-EMT is not significant in statistical terms; H_1 contrarily states that the percentage of mutants generated by EA-EMT is statistically lower compared with that of

the original EMT. Table 8 shows the results of the Wilcoxon signed-rank test executed in each combination of SUT and value of EQ. As can be seen from the *p-values* in this table, in most cases we can reject the null hypothesis that no variation exists between both techniques with two different levels of significance (0.05 and 0.1).

Table 8: Results of the Wilcoxon signed-rank test (*p-value*) based on the results of the original EMT and EA-EMT for $P = 75\%$ in each program and value of EQ. The cases where EMT obtained better results (see Table 6) are not computed (-). Favorable results for EA-EMT are highlighted in gray and boldface at 0.1 and 0.05 level of significance, respectively.

	TCL	**DPH**	**TXM**	**RPC**	**SQL**	**DOM**
EQ10	**0.019**	0.434	-	**0.002**	-	0.117
EQ40	**0.013**	0.144	**0.093**	**0.024**	-	**0.012**
EQ70	**0.023**	**0.083**	**0.059**	**<0.001**	0.491	**0.009**

***Answering RQ3**, EA-EMT did reduce the number of mutants generated by the original version of EMT in most of the programs. In all cases, we could find at least one subset of equivalent mutants which improved the execution of the evolutionary approach, with a reduction of 4% in the percentage of mutants selected in the best case.*

***Answering RQ4**, EA-EMT performed better than EMT in most of the combinations of EQ and P. Still, the results were more relevant with subsets of equivalent mutants that contained 40% or more of all equivalent mutants, and when 75% of the mutant-adequate test suite was required before stopping the GA (with statistical significance).*

6.3. Discussion

The search performed by EA-EMT is affected by several factors. We should take into account these factors to properly assess the full extent of the results offered by the experiments:

Selection frequency of known equivalent mutants. In the end, not all known equivalent mutants will have an influence on the search. Actually, that depends on how many of those equivalent mutants the GA selects throughout its execution. To illustrate the implications of this fact, imagine that our subset of known equivalent mutants contains 40% of all equivalent mutants (e.g., 10 out of 25), and that the GA stops when 30% of all mutants have been selected (e.g., 90 out of 300). In our example, we can expect the GA to generate around 30% of our subset of equivalent mutants: it turns out that, probably, only 3 of those 10 known equivalent mutants would affect the search. This means that the lower the percentage of mutants generated and the lower the percentage of known equivalent mutants, the less is the expected influence of the equivalence information on the search.

Side effects. The GA is only aware of a subset of equivalent mutants; the rest are unknown and remain indistinguishable from useful mutants. As a consequence, by punishing known equivalent mutants, we may be causing the GA to focus even more on other mutants that are not useful in the end. These include unknown equivalent mutants and mutants not leading to new test cases, which have no effect on the measure of test suite improvement for the termination condition. As a consequence, we cannot expect a proportional effect of the equivalence information on the performance of the evolutionary strategy.

Limited search space. EMT has shown to provide positive results in the past, being more effective than random selection and selective mutation in most cases [10]. However, as in the rest of cost reduction techniques, the space is limited by the available mutants in a program. This implies that we cannot avoid that the algorithm generates the mutants we want to avoid by other means. Namely:

- In EA-EMT, the number of selected equivalent mutants with other equivalent ancestors is reduced. However, those equivalent mutants can now descend from non-equivalent mutants instead.
- N mutants in a generation are randomly selected to maintain diversity.

To improve EA-EMT's performance in the future, we should investigate how to cope with those aspects over which we have no control to date.

7. Practical Study

This section focuses on the second part of the analysis (practical study). In the following subsections, we present and discuss the execution results of the application of TCE-EMT.

7.1. Results and discussion

Table 9 presents the results (average and standard deviation) of the performance of TCE-EMT in our SUTs. If we focus on the cases where EA-EMT performed better than EMT on average (those surrounded by a box), we can observe that the results of TCE-EMT are quite similar to those of EA-EMT (see Table 6), that is, in most of those cases TCE-EMT offers a better outcome than EMT. Remarkably, the results were also improved in *RPC*, even though TCE-EMT only counted with the knowledge of four equivalent mutants (see Table 3).

An important point to be considered is that the execution of TCE-EMT was accompanied by a decrease in the standard deviation in almost all cases. For instance, the standard deviation decreased from 5.00 to 3.89 in <DOM,P90> with the equivalence information. This means that the GA increases its stability among the 30 executions, which is a positive aspect given the stochastic nature of the technique. Interestingly, in all the cases where the original EMT gave a

(a) Average

P	TCL		DPH		TXY		RPC		SQL		DOM	
	Ori.	TCE	Ori.	TCE	Ori.	TCE	Ori.	TCE	Ori.	TCE	Ori.	TCE
60%	25.39	**24.98**	36.14	**35.79**	13.21	14.57	30.97	31.25	19.62	19.77	8.79	**8.67**
75%	37.24	**36.56**	49.75	**49.63**	19.26	19.69	45.25	**45.18**	35.60	36.24	13.33	**12.89**
90%	49.24	49.72	66.33	**65.22**	31.93	33.29	59.42	**59.07**	58.19	**56.60**	21.41	**21.36**

(b) Standard deviation

P	TCL		DPH		TXY		RPC		SQL		DOM	
	Ori.	TCE	Ori.	TCE	Ori.	TCE	Ori.	TCE	Ori.	TCE	Ori.	TCE
60%	6.63	**6.25**	6.72	**6.54**	2.89	**2.69**	7.40	**7.28**	2.92	**2.74**	1.61	1.89
75%	10.77	**10.00**	8.51	9.93	4.38	**4.25**	9.59	**8.24**	7.07	**5.26**	3.35	**3.15**
90%	13.41	**12.46**	8.61	12.30	7.13	**7.12**	9.11	**8.66**	8.49	**8.21**	5.00	**3.89**

Table 9: Comparison of results (average and standard deviation) between the original EMT (*Ori.*) and TCE-EMT (*TCE*), grouped by SUT and P. Favorable results for TCE-EMT are highlighted in boldface. Additionally, the cases where EA-EMT obtained a better average percentage than *Ori.* in the experimental study (see Table 6) are surrounded by a box.

better average result than TCE-EMT (like in <TCL,P90>), the standard deviation of TCE-EMT turned out to be lower. On the contrary, the standard deviation in *DPH* increased when $P = 75\%$ and $P = 90\%$ despite the improvement in the average percentage. This indicates that occasionally the equivalence information can notably guide but also misdirect the search depending on the execution.

In order to confirm these results, we run an automated configuration algorithm with the possibility to select between the original version of EMT and TCE-EMT. More specifically, we made use of the well-known *irace* package [46], which performs an iterated racing procedure to find the most appropriate parameter settings for optimization algorithms. In that procedure, the algorithm iteratively undertakes a race among different candidate configurations to determine which configurations show a good performance in a set of instances; otherwise, the rest of the configurations can be discarded. The sampled configurations are based on the results of the candidate configurations in the previous iterations; in this way, the algorithm pays more attention to those values that present a better performance over time. In our experiments, we applied *irace* to sample a number of configurations that select between EMT and TCE-EMT and a range of values for the rest of the configuration parameters of the GA (see Section 5.2); we used our set of programs as instances with $P = 75\%$ and a budget of 1,500 executions. As a result, 53 out of the 78 sampled candidate configurations enabled TCE-EMT, and, more importantly, the four best configurations found by the algorithm belonged to TCE-EMT.

Finally, we could not find any clear relation between TCE-EMT's effectiveness and the ratio of detected to the number of equivalent mutants, or the ratio of detected equivalent mutants to the total number of mutants. For instance, the improvements achieved in *DPH* and *DOM* are similar, but the aforementioned ratios are quite different in those programs (Equivalent: 27.5% vs 8.5%;

Total: 8.7% vs 1.7%). This suggests that the results depend on other aspects, such as how those mutants are distributed or the size of the GA population, which would merit further investigation.

In either case, TCE was able to detect a few of the equivalent mutants in these programs (between 4% in *SQL* and 45% in *TCL*). Had we complemented TCE with other equivalence detection techniques, it is likely that the impact of the equivalence information would have been greater, in line with the results of the experimental study when the percentage of known equivalent mutants is higher. A more effective working scenario would be the one replacing TCE by a tester who revises the mutants remaining alive in each generation. However, putting a human-in-the-loop is costly and can lead to errors in the classification. In contrast, the alternative proposed in this paper partially transfers this burden from the user to TCE, guaranteeing that the mutants marked as equivalent are actually equivalent ones.

> ***Answering RQ5****, the injection of equivalence information offered by TCE can benefit the execution of EMT, not only in terms of the average of mutants generated but also in terms of the variability among executions. The penalization of known equivalent mutants restricts the alternatives in the search, which results in a more stable version of EMT. The preference of TCE-EMT over the original version of EMT was supported by an automated configuration algorithm.*

8. Threats to validity

Below we present the main threats to the validity of the experimental procedures:

EA-EMT and TCE-EMT

The most evident threat in our experiments is the stochastic nature of the genetic algorithm. Even though the GA has been refined to boost the selection of useful mutants, it is possible that the outcomes are the result of a random phenomenon. To counter this threat, we have executed 30 independent runs with each of the configurations to minimize the probability that the observed performance is a matter of chance. For the same reason, we have used $S = 5$ different subsets of equivalent mutants for each of the three sizes for EQ. In total, we have executed EA-EMT 1,350 times per program.

In this study, we have applied the death penalty as the unique method to penalize known equivalent mutants so that they were not selected to produce new mutants in the next generation. In the same line, we have focused on TCE, as it allows identifying equivalent mutants with certainty. However, we could have opted for other more complex kinds of penalties [34, 47]. For instance, a static penalty depending on how likely is that each mutation operator will generate an equivalent mutant (based on past experience [16]), or even adaptive

penalties based on the feedback from the last generations [48]. Likewise, there are other methods that address the Equivalent Mutant Problem by avoiding or suggesting likely equivalent mutants [6] (see Section 2). For instance, we could have added penalties which were proportional to the changes in coverage produced by each mutation [4, 14].

The death penalty is quite simple to implement as it only implies assigning a zero to the fitness of mutants revealed as equivalent. We should note, however, that the extra step in EA-EMT adds an overhead to the GA, which depends on the method used to determine equivalence. However, given that all the mutants generated by the algorithm are returned in the end, this step avoids the manual review of equivalent mutants at a later stage. In the case of TCE, it increases the compilation time of mutants but it clearly compensates the effort required to manually identify equivalent mutants [5, 21].

Equivalence

We may have failed to classify equivalent mutants since this process was carried out entirely by hand. The fact that all the mutants determined as equivalent by TCE were contained in our set of manually-identified equivalent mutants gives us confidence in our classification. If we had missed some equivalent mutants, the results would probably have been even better: EA-EMT reported the best results with high values for *EQ*; contrarily, we do not think that killable mutants mistakenly tagged as equivalent would have had a strong influence on the experiments: the results were derived from several executions with different subsets, and it is unlikely that those mutants would have been selected in many of those executions —as discussed earlier in Section 6.3, only a portion of the known equivalent mutants will be selected and will affect the search in the end.

Generalization of results

In the hope that our experiments are representative, we have selected six well-known C++ programs and libraries of different domains, sizes, percentages of equivalence and initial mutant coverage of the test suites. In this work, we have focused on a particular domain: mutation operators at the class level for C++ applications. Thus, it is currently unknown how EA-EMT and TCE-EMT will behave in other contexts, such as with traditional operators for procedural programming languages like C. However, we have reasons to believe that these results will extrapolate to that domain. On the one hand, class-level operators are known to generate fewer mutants than traditional operators [32, 43], and previous experiments on EMT suggested that the technique performs better in programs with a high number of mutants [9, 10]. On the other hand, TCE was more effective detecting C traditional mutants than C++ class-level mutants, accounting for 30% of all existing equivalent mutants on average [5, 21] —as shown in this study, EA-EMT improves its effectiveness with large subsets of known equivalent mutants—. We selected *-O2* as the optimization option for the experiments with TCE-EMT. Previous experiments with these mutation

operators [24] revealed no differences with other popular optimization settings (*-O* and *-O3*). Regarding the initial mutant coverage, a previous study discussed that relatively weak test suites may affect EMT's performance due to the large number of mutants remaining alive [10]. Even though EMT still showed better results than random and selective mutation in those cases, the maturity of the initial test suite constitutes another threat to the generalization of the results of the proposed approach.

9. Conclusion and future work

This paper serves to present evidence on the connection between equivalent mutants and on the use that can be made of that connection for the first time. We have followed an evolutionary approach to benefit from equivalence information in the selection of mutants for test suite improvement. However, this study opens the way to look for new domains where this link can be exploited differently in the future. For instance, recent papers have proposed the use of equivalent mutations to produce more efficient versions of the program [49, 50] and remove static anomalies [51]. Hence, equivalent mutant connection can hint at operators and code areas that can help in code refactoring.

In the future we could opt for a midway solution between the original version of EMT (without penalties) and EA-EMT using the death penalty, especially in those domains where TCE could not be applied or was not particularly effective. If we could accurately measure how likely is that a live mutant will turn out to be equivalent, we could make the penalty proportional to that. A challenge for the future will be to add a penalty to the objective function that wisely combines past experience, the information provided by other techniques for the detection of equivalent mutants and feedback from the own execution. This would allow for fairer fitnesses, which could result in improved performance. Additionally, a recent study points out that a more challenging objective is selecting fault revealing mutants instead of just killable ones [52]. As such, knowing the fault revealing ability of EMT, or even devising a new fitness function to this end, should guide the future research direction. That study applies a machine learning approach to build a model based on the static analysis of the features of known fault revealing mutants, which is later used to drive the mutant selection. The interaction possibilities of machine learning and search methods have been scarcely investigated up to now in this area, so exploring the adoption of data-driven search-based approaches would be equally interesting.

10. Acknowledgment

This work is partially funded by the European Commission (FEDER), the University of Malaga (Exhauro project) and the Spanish Ministry of Innovation and Competitiveness under projects (RTI2018-093608-B-C33 and TIN2017-88213-R).

11. References

[1] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. L. Traon, M. Harman, Chapter six - mutation testing advances: An analysis and survey, Vol. 112 of Advances in Computers, Elsevier, 2019, pp. 275–378. doi:10.1016/bs.adcom.2018.03.015.

[2] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, Software Engineering, IEEE Transactions on 37 (5) (2011) 649–678. doi:10.1109/TSE.2010.62.

[3] M. Kintis, M. Papadakis, N. Malevris, Employing second-order mutation for isolating first-order equivalent mutants, Software Testing, Verification and Reliability 25 (5-7) (2015) 508–535. doi:10.1002/stvr.1529.

[4] M. Papadakis, M. Delamaro, Y. L. Traon, Mitigating the effects of equivalent mutants with mutant classification strategies, Science of Computer Programming 95, Part 3 (2014) 298–319, special Section: ACM SAC-SVT 2013 + Bytecode 2013. doi:10.1016/j.scico.2014.05.012.

[5] M. Kintis, M. Papadakis, Y. Jia, N. Malevris, Y. L. Traon, M. Harman, Detecting trivial mutant equivalences via compiler optimisations, IEEE Transactions on Software Engineering 44 (4) (2018) 308–333. doi:10.1109/TSE.2017.2684805.

[6] L. Madeyski, W. Orzeszyna, R. Torkar, M. Jozala, Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation, IEEE Transactions on Software Engineering 40 (1) (2014) 23–42. doi:10.1109/TSE.2013.44.

[7] T. A. Budd, D. Angluin, Two notions of correctness and their relation to testing, Acta Informatica 18 (1) (1982) 31–45. doi:10.1007/BF00625279.

[8] R. A. Silva, S. do Rocio Senger de Souza, P. S. L. de Souza, A systematic review on search based mutation testing, Information and Software Technology 81 (Supplement C) (2017) 19–35. doi:10.1016/j.infsof.2016.01.017.

[9] J. J. Domínguez-Jiménez, A. Estero-Botaro, A. García-Domínguez, I. Medina-Bulo, Evolutionary mutation testing, Information and Software Technology 53 (10) (2011) 1108–1123. doi:10.1016/j.infsof.2011.03.008.

[10] P. Delgado-Pérez, I. Medina-Bulo, Search-based mutant selection for efficient test suite improvement: Evaluation and results, Information and Software Technology 104 (2018) 130–143. doi:10.1016/j.infsof.2018.07.011.

[11] P. Delgado-Pérez, I. Medina-Bulo, M. G. Merayo, Using evolutionary computation to improve mutation testing, in: I. Rojas, G. Joya, A. Catala (Eds.), Advances in Computational Intelligence, Springer International Publishing, Cham, 2017, pp. 381–391. doi:10.1007/978-3-319-59147-6_33.

[12] A. V. Pizzoleto, F. C. Ferrari, J. Offutt, L. Fernandes, M. Ribeiro, A systematic literature review of techniques and metrics to reduce the cost of mutation testing, Journal of Systems and Software 157 (2019) 110388. doi:10.1016/j.jss.2019.07.100.

[13] B. J. M. Grün, D. Schuler, A. Zeller, The impact of equivalent mutants, in: Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 192–199. doi:10.1109/ICSTW.2009.37.

[14] D. Schuler, A. Zeller, Covering and uncovering equivalent mutants, Software Testing, Verification and Reliability 23 (5) (2013) 353–374. doi:10.1002/stvr.1473.

[15] X. Devroey, G. Perrouin, M. Papadakis, A. Legay, P.-Y. Schobbens, P. Heymans, Model-based mutant equivalence detection using automata language equivalence and simulations, Journal of Systems and Software 141 (2018) 1–15. doi:10.1016/j.jss.2018.03.010.

[16] E. S. Mresa, L. Bottaci, Efficiency of mutation operators and selective mutation strategies: an empirical study, Software Testing, Verification and Reliability 9 (4) (1999) 205–232. doi:10.1002/(SICI)1099-1689(199912)9:4<205::AID-STVR186>3.0.CO;2-X.

[17] K. Adamopoulos, M. Harman, R. M. Hierons, How to overcome the equivalent mutant problem and achieve tailored selective mutation using coevolution, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'04, 2004, pp. 1338–1349. doi:10.1007/978-3-540-24855-2_155.

[18] A. A. L. de Oliveira, C. G. Camilo-Junior, A. M. R. Vincenzi, A coevolutionary algorithm to automatic test case selection and mutant in mutation testing, in: Proceedings of the IEEE Congress on Evolutionary Computation, CEC'13, 2013, pp. 829–836. doi:10.1109/CEC.2013.6557654.

[19] S. Nica, F. Wotawa, Using constraints for equivalent mutant detection, in: Workshop on Formal Methods in the Development of Software, 2012.

[20] M. Kintis, N. Malevris, MEDIC: A static analysis framework for equivalent mutant identification, Information and Software Technology 68 (2015) 1–17. doi:10.1016/j.infsof.2015.07.009.

[21] M. Papadakis, Y. Jia, M. Harman, Y. Le Traon, Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique, in: Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE'15, IEEE Press, Piscataway, NJ, USA, 2015, pp. 936–946. doi:10.1109/ICSE.2015.103.

[22] M. Houshmand, S. Paydar, TCE+: An extension of the TCE method for detecting equivalent mutants in Java programs, in: M. Dastani, M. Sirjani (Eds.), Fundamentals of Software Engineering, Springer International Publishing, Cham, 2017, pp. 164–179. doi:10.1007/978-3-319-68972-2_11.

[23] F. Wu, J. Nanavati, M. Harman, Y. Jia, J. Krinke, Memory mutation testing, Information and Software Technology 81 (2017) 97–111. doi:10.1016/j.infsof.2016.03.002.

[24] P. Delgado-Pérez, S. Segura, Study of trivial compiler equivalence on C++ object-oriented mutation operators, in: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC '19, ACM, New York, NY, USA, 2019, pp. 2224–2230. doi:10.1145/3297280.3297499.

[25] A. Ayad, I. Marsit, J. Loh, M. N. Omri, A. Mili, Estimating the number of equivalent mutants, in: 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2019, pp. 112–121. doi:10.1109/ICSTW.2019.00039.

[26] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, 1st Edition, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[27] J. J. Domínguez-Jiménez, A. Estero-Botaro, A. García-Domínguez, I. Medina-Bulo, GAmera: an automatic mutant generation system for WS-BPEL compositions, in: R. Eshuis, P. Grefen, G. A. Papadopoulos (Eds.), Proceedings of the 7th IEEE European Conference on Web Services, IEEE Computer Society Press, Eindhoven, The Netherlands, 2009, pp. 97–106.

[28] P. Delgado-Pérez, I. Medina-Bulo, M. Núñez, Using evolutionary mutation testing to improve the quality of test suites, in: Proceedings of the IEEE Congress on Evolutionary Computation, CEC'17, 2017, pp. 596–603. doi:10.1109/CEC.2017.7969365.

[29] P. Delgado-Pérez, I. Medina-Bulo, S. Segura, A. García-Domínguez, J. J. Domínguez-Jiménez, GiGAn: Evolutionary mutation testing for C++ object-oriented systems, in: Proceedings of the Symposium on Applied Computing, SAC'17, ACM, New York, NY, USA, 2017, pp. 1387–1392. doi:10.1145/3019612.3019828.

[30] L. Gutiérrez-Madroñal, A. García-Domínguez, I. Medina-Bulo, Evolutionary mutation testing for IoT with recorded and generated events, Software: Practice and Experience 49 (4) (2019) 640–672. doi:10.1002/spe.2629.

[31] A. B. Sánchez, P. Delgado-Pérez, I. Medina-Bulo, S. Segura, Search-based mutation testing to improve performance tests, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO'18, ACM, New York, NY, USA, 2018, pp. 316–317. doi:10.1145/3205651.3205670.

[32] P. Delgado-Pérez, I. Medina-Bulo, F. Palomo-Lozano, A. García-Domínguez, J. J. Domínguez-Jiménez, Assessment of class mutation operators for C++ with the MuCPP mutation system, Information and Software Technology 81 (2017) 169–184. doi:10.1016/j.infsof.2016.07.002.

[33] M. Kintis, N. Malevris, Identifying more equivalent mutants via code similarity, in: 20th Asia-Pacific Software Engineering Conference (APSEC), Vol. 1, 2013, pp. 180–188. doi:10.1109/APSEC.2013.34.

[34] J. Periaux, F. Gonzalez, D. S. C. Lee, Evolutionary Methods, Springer Netherlands, Dordrecht, 2015, pp. 9–20. doi:10.1007/978-94-017-9520-3_2.

[35] Matrix TCL Pro, `http://www.techsoftpl.com/matrix/download.php`, [Online; accessed 04/07/2019].

[36] Dolphin, `https://www.kde.org/applications/system/dolphin`, [Online; last access: 04/07/2019].

[37] TinyXML2, `https://github.com/leethomason/tinyxml2`, [Online; last access: 04/07/2019].

[38] XmlRPC++, `http://xmlrpcpp.sourceforge.net/`, [Online; last access: 04/07/2019].

[39] MySQL Server, `https://github.com/mysql/mysql-server`, [Online; last access: 04/07/2019].

[40] QtDOM, `https://github.com/qtproject/qtbase/tree/dev/src/xml/dom`, [Online; last access: 04/07/2019].

[41] Y.-S. Ma, J. Offutt, Y.-R. Kwon, MuJava: A mutation system for Java, in: Proceedings of the 28th International Conference on Software Engineering, ICSE '06, Association for Computing Machinery, New York, NY, USA, 2006, pp. 827–830. doi:10.1145/1134285.1134425.

[42] Y.-S. Ma, J. Offutt, Y. R. Kwon, MuJava: An automated class mutation system: Research articles, Software Testing, Verification and Reliability 15 (2) (2005) 97–133.

[43] Y.-S. Ma, Y. R. Kwon, S.-W. Kim, Statistical investigation on class mutation operators, ETRI Journal 31 (2) (2009) 140–150. doi:10.4218/etrij.09.0108.0356.

[44] S. Segura, R. M. Hierons, D. Benavides, A. Ruiz-Cortés, Mutation testing on an object-oriented framework: An experience report, Information and Software Technology 53 (10) (2011) 1124–1136, special Section on Mutation Testing. doi:10.1016/j.infsof.2011.03.006.

[45] P. Delgado-Pérez, S. Segura, I. Medina-Bulo, Assessment of C++ object-oriented mutation operators: A selective mutation approach, Software Testing, Verification and Reliability 27 (4-5) (2017) e1630–n/a. doi:10.1002/stvr.1630.

[46] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58. doi:10.1016/j.orp.2016.09.002.

[47] J. A. Joines, C. R. Houck, On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's, in: Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence, Vol. 2, 1994, pp. 579–584. doi:10.1109/ICEC.1994.349995.

[48] H. J. C. Barbosa, A. C. C. Lemonge, An adaptive penalty scheme in genetic algorithms for constrained optimization problems, in: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, GECCO'02, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002, pp. 287–294.

[49] W. B. Langdon, M. Harman, Optimizing existing software with genetic programming, IEEE Transactions on Evolutionary Computation 19 (1) (2015) 118–135. doi:10.1109/TEVC.2013.2281544.

[50] J. López, N. Kushik, N. Yevtushenko, Source code optimization using equivalent mutants, Information and Software Technology 103 (2018) 138–141. doi:10.1016/j.infsof.2018.06.013.

[51] P. Arcaini, A. Gargantini, E. Riccobene, P. Vavassori, A novel use of equivalent mutants for static anomaly detection in software artifacts, Information and Software Technology 81 (2017) 52–64. doi:10.1016/j.infsof.2016.01.019.

[52] T. T. Chekam, M. Papadakis, T. F. Bissyandé, Y. Le Traon, K. Sen, Selecting fault revealing mutants, Empirical Software Engineering 25 (2020) 434–487. doi:10.1007/s10664-019-09778-7.