Resolving the Equivalent Mutant Problem in the Presence of Non-determinism and Coincidental Correctness

Krishna Patel ✉ and Robert M. Hierons

Department of Computer Science, Brunel University
Uxbridge, Middlesex, UB8 3PH, UK
krishna.patel@brunel.ac.uk, rob.hierons@brunel.ac.uk

Abstract. In this paper, we develop a new mutation testing technique called Interlocutory Mutation Testing (IMT) that mitigates the equivalent mutant problem in the presence of coincidental correctness and non-determinism. The accuracy of IMT was evaluated; it obtained a classification accuracy of 93.33% for non-equivalent mutants and 100% for equivalent mutants in a non-deterministic system with coincidental correctness.

Keywords: Mutation Testing · Coincidental Correctness · Non-determinism · Equivalent Mutant Problem

1 Introduction

Mutation Testing (MT) is a technique for generating artificial faults [15], which are reasonably accurate simulations of real faults [2]. MT operates by applying a minor augmentation (referred to as a mutation) to the system under test (SUT) S_o to produce a faulty version S_m [5] called a mutant. For example, a statement $X < 5$ in S_o might be transformed into $X > 5$ in S_m.

Unfortunately, a limitation of MT is that it can produce equivalent mutants [7] — this is known as the equivalent mutant problem. An equivalent mutant is an augmentation S_m that is observationally equivalent to the SUT S_o. For example, suppose that $Math.abs(5)$ and $Math.abs(-5)$ appear on Line 1 in S_o and S_m respectively. S_m is an equivalent mutant, because the augmentation is semantically equivalent and doesn't modify the behaviour of S_o.

A study conducted by Yao et al. [24] demonstrated that the equivalent mutant problem is pervasive. Despite the fact that deducing mutant equivalence is undecidable [4], this has motivated some research into how the problem can be circumvented [11]. For example, let $S_o(I)$ and $S_m(I)$ denote the respective outputs of S_o and S_m for a given input. Many researchers typically expose S_o and S_m to a test suite to obtain a set of pairs $\langle S_o(I), S_m(I)\rangle$ and assume that S_o and S_m are equivalent if the following condition holds for each pair: $S_o(I) = S_m(I)$.

For ease of reference, we refer to this as the Traditional Equivalent Mutant Detection Technique (TEMDT). An example of the use of TEMDT can be found in Sadi et al. [21].

However, this assumption doesn't always hold. For example, non-deterministic behaviours may be responsible for any observed discrepancies, and may be misinterpreted as having originated from the mutation [5]. Another example includes the presence of coincidental correctness; the SUT can misbehave but still produce the expected output, which can lead to non-equivalent mutants being mistakenly classified as equivalent. Alternative techniques have been proposed to address these problems, but have limitations (see Section 2). Manual inspection is typically used under such circumstances [1].

In our previous work, we developed Interlocutory Testing (IT), a testing technique that suppresses coincidental correctness and can operate effectively in the presence of non-determinism [19]. This paper explores how IT can be used to alleviate the Equivalent Mutant Problem in systems with non-determinism and/or coincidental correctness. We call the approach Interlocutory Mutation Testing (IMT).

The relationship between the input and output of the SUT, in conjunction with one's knowledge/expectations about the SUT, can be used to predict aspects of the execution trace. For example, consider the Bubble Sort algorithm; $Input$ and $Output$ are sequences of integers. If $Input \neq Output$, one can predict that the Swap Operator was invoked at least once. The correctness of this prediction is predicated on whether the SUT's behaviour mirrors the tester's expectations. Let f denote a fault in Bubble Sort that overwrites the value of the first element of $Input$ with a random value. f can lead to situations in which $Input \neq Output$ and the swap operator was not invoked; the failure to satisfy the prediction above in such situations shows that the behaviour of the SUT does not satisfy the tester's expectations[1]. IMT exploits this observation as follows. Let S denote the SUT and M denote a mutated version of S. Suppose that M is executed with an input $MInput$, and produces an execution trace MET and output $MOutput$. IMT uses the relationship between $MInput$ and $MOutput$, in conjunction with the tester's knowledge/expectations about S, to predict aspects of MET. If this prediction is incorrect then this suggests that M is not an equivalent mutant.

This paper makes the following main contributions:

1. A new technique called IMT that can classify mutants as equivalent and non-equivalent in programs with coincidental correctness and/or non-deterministic behaviours.
2. An evaluation of the accuracy of IMT.

The paper is structured as follows. We begin by presenting related work in Section 2. Section 3 describes our proposed technique and explains how the technique can be applied. Section 4 outlines our experimental set-up. The experiment

[1] Later we will see more complicated examples in which this process can help to overcome coincidental correctness.

results are presented and discussed in Section 5, along with threats to validity in Section 6. Conclusions are finally drawn in Section 7.

2 Related Work

2.1 The Equivalent Mutant Problem and Coincidental Correctness

Fault detection requires the execution of a faulty statement, that causes the subsequent infection of a state (to produce a failure), and propagation of an infected state to the output (so an oracle can assess it) [23]. According to Masri and Assi [14], strong coincidental correctness occurs when the first two conditions are satisfied, the third is not, and weak coincidental correctness occurs when the first condition is satisfied, but not the third; the second condition may or may not be satisfied. Weak coincidental correctness subsumes strong coincidental correctness. In this paper, "coincidental correctness" refers to weak coincidental correctness.

In the context of mutation testing, coincidental correctness can be described as follows: Let S_o be the SUT and S_m be a non-equivalent mutant. Also let s_m denote the state in S_m after the mutated statement executes and s_o be the corresponding state in S_o. Coincidental correctness occurs if s_m and s_o map to the same output, despite the differences in code.

Masri and Assi [13] define information flow strength as the percentage of information that propagates between two program points; a higher percentage indicates greater strength. This determines the probability that an infected state will propagate to the output, which is tantamount to the likelihood of observing coincidental correctness. Masri et al. [14] conducted a series of experiments that suggested that coincidental correctness is widespread. For example, 96.5% and 72% of the systems they investigated had strong and weak coincidental correctness respectively and between 63.76 - 97.58% of the weak information flows in six of these systems had a strength of 0.

Despite the prevalence of coincidental correctness, little research has been conducted on determining mutant equivalence in the context of coincidental correctness. To our knowledge, only one approach has been proposed. Offutt and Lee [16] extend TEMDT (see Section 1). They suggest additionally comparing s_o and s_m. While this can be useful in some situations it's not a universal solution e.g. its effectiveness may be limited in non-deterministic systems.

2.2 The Equivalent Mutant Problem and Non-Deterministic Systems

Non-deterministic systems are becoming increasingly prevalent e.g. concurrency can lead to alternative interleavings. For example, consider a variable X that is instantiated with a value of 3. Suppose we have two threads t_1 and t_2 and that t_1 applies the following operation to X: $X = X + 1$. Further, suppose that t_2 updates the value of X to $X = X \times 2$. The order of the interleavings affects the final state of X i.e. if t_1 executes first, then $X = 8$ and if t_2 executes first $X = 7$.

This complicates the mutant classification process. Several proposals have been made to address this. For example, Carver [5] identifies two methods - Multiple Execution Testing (MET) and Deterministic Execution Testing (DET). In MET, confidence is improved by executing the original S_o and mutant S_m versions multiple times and observing their output distributions. DET involves forcing the SUT to execute deterministically by manipulating conditions e.g. a Genetic Algorithms Mutation Rate can be set to 100% or 0% to force deterministic execution of the Mutation Operator.

Both strategies are viable, but have limitations. For example, MET is dictated by chance; thus there is scope for misclassification [5] and non-replicability [5]. It's also expensive because it uses multiple executions. On the other hand, DET limits test case selection; thus some mutation points may not be reachable with allowable test cases. Carver [5] attempted to reduce the impact of these weaknesses by combining MET and DET.

Gligoric et al. [7] suggest executing S_o with a test case t, and then establishing whether the mutant statement in S_m could have been reached by this execution. Non-reachability implies equivalence for t. This approach is limited to the identification of equivalent mutants in unexecuted code.

Finally, Papadakis et al. [17] propose comparing the S_m's object code to the object code of the S_o. If S_m's object code matches S_o's object code, then we can guarantee that S_o is equivalent to S_m. However, if the comparison reveals that there are discrepancies, S_m may either be equivalent or non-equivalent to S_o. Although the approach can't correctly classify all mutants, it is inexpensive and so can be a valuable complimentary equivalent mutant classification technique.

3 Interlocutory Mutation Testing

IMT was developed to enable the classification of equivalent and non-equivalent mutants in programs that are non-deterministic and/or are susceptible to coincidental correctness. Section 3.1 introduces the technique and demonstrates how it can classify mutants despite the presence of coincidental correctness, and Section 3.2 shows how the technique can be extended to cope with non-determinism.

3.1 Interlocutory Mutation Testing and Coincidental Correctness

This section draws on the following running example.

The SUT is a Genetic Algorithm, which is a search optimisation technique. The SUT consists of four major components: Initial Population Generator, Crossover, Mutation, and Selection. Let Sys denote the SUT.

Consider Sys's selection operator, denoted by Sys_{so}. Sys_{so}'s $Input$ consists of a population size parameter PS, which is the maximum population size, and a $Population$, such that $Population.size() \geq PS$. Let $Population_{SOI}$ be the state of $Population$ at this point in the execution trace. $Input$ is processed by the Sys_{so} as follows: random elements of $Population_{SOI}$ are iteratively removed until $Population.size() == PS$. Sys_{so}'s resultant $Output$ is a version of the

Population that has been subjected to this process; $Population_{SOO}$ denotes the state of *Population* at this point in the execution.

Suppose that a non-equivalent mutant, denoted by MUT, of the *Sys* was produced. The delta between MUT and *Sys* is that MUT performs an additional operation; it adds a random individual to $Population_{SOI}$ during Sys_{so}'s initialisation phase. Since Sys_{so} iteratively removes random individuals from $Population_{SOO}.size()$ until $Population_{SOO}.size() == PS$, all traces of an additional member being added to $Population_{SOI}$ might be lost by the time the execution reaches the $Population_{SOO}$ state. Thus, MUT is a coincidentally correct mutant.

Intuition Let's consider how MUT could be correctly classified. Suppose that MUT is executed and produces a log file that details the execution trace MET. Let MUT_{so} denote MUT's selection operator. The execution trace of MUT_{so} is a subsequence of MET. Let *MInput* and *MOutput* be MUT_{so}'s input and output respectively. Information about MET can be revealed by assessing the relationship between *MInput* and *MOutput*. For example, $Population_{SOI}.size() > Population_{SOO}.size()$ may be one relationship between *MInput* and *MOutput*, and from this, we can deduce that members of *Population* were removed during the execution.

If we assume that MUT is equivalent to *Sys*, we can use our knowledge about how *Sys* behaves in this context to predict aspects of MET. To illustrate, since we know that the Selection Operator iteratively removes random members of *Population* until $Population.size() == PS$, when $Population_{SOI}.size() > Population_{SOO}.size()$, we can deduce that the *Population* must have been expanded by $Population_{SOI}.size() - Population_{SOO}.size()$ individuals before the Selection Operator was executed. Since we also know that the only function that can add additional members to a *Population* of size PS is the Crossover Operator, the following prediction about MET can be made: the Crossover Operator generated $Population_{SOI}.size() - Population_{SOO}.size()$ individuals and added them to *Population*.

Finally, this prediction can be checked against MET. Let $CrossoverN$ be the total number of members that were actually generated by the Crossover Operator during the execution i.e. as reported in MET. In continuation of the example above, this involves checking $CrossoverN == Population_{SOI}.size() - Population_{SOO}.size()$. Since an additional member is added to $Population_{SOI}$ by MUT, this predicate would evaluate to false, which indicates that the prediction was incorrect. The behaviour of MUT deviated from how *Sys* would have behaved; thus we can conclude that MUT is not equivalent to *Sys*.

Had MUT been equivalent to *Sys* (i.e. had the additional member not been added to *Population* during the initialisation of the Selection Operator), the prediction would have been correct.

The example above demonstrates that one can use the relationship between *MInput* and *MOutput* to predict properties of MET. Discrepancies between this prediction and MET indicate that MUT is not equivalent to *Sys*. The ex-

ample also demonstrates that this approach works in the presence of coincidental correctness. This forms the intuition of our technique - Interlocutory Mutation Testing (IMT).

Technique Description This section outlines how IMT realises the intuition described above. IMT requires that the relationship between an input and output (Input-Output pair) is associated with a prediction about the mutants execution trace MET. Associating a prediction with every individual Input-Output pair would be impractical. Instead, IMT groups Input-Output pairs together using Input-Output Relationships (IORs). Certain predictions are applicable to all Input-Output pairs in such a group. Consider the earlier example; $Population_{SOI}.size() > Population_{SOO}.size()$ is an IOR (for ease of reference, we call this IOR_1), and it groups Input-Output pairs where the prediction is that the Crossover Operator produced $Population_{SOI}.size() - Population_{SOO}.size()$ members and added them to the $Population$.

The prediction that is associated with an IOR is referred to as an "Interlocutory Decision" (ID). An ID can be expressed using any method, on the proviso that it can unambiguously describe one's prediction about MET and be automatically compared with the execution trace MET. For example, as demonstrated above, IDs can be expressed as predicates e.g. $CrossoverN == Population_{SOI}.size() - Population_{SOO}.size()$ (this ID is associated with IOR_1). Alternative methods of expressing IDs are discussed in our previous work [19].

In IMT, the mutant is executed, which results in an execution trace MET. IMT checks whether an IOR is satisfied by an input $MInput$ and output $MOutput$, which are extracted from MET. In continuation of the example above, $MInput = Population_{SOI}$ and $MOutput = Population_{SOO}$. If $MInput.size() > MOutput.size()$, then IOR_1 is satisfied. If the IOR is satisfied, then IMT checks that MET satisfies the IOR's associated IDs (e.g. in the case of IOR_1, this would involve checking $CrossoverN == Population_{SOI}.size() - Population_{SOO}.size()$). Finally, if the prediction is correct (e.g. if $CrossoverN == Population_{SOI}.size() - Population_{SOO}.size()$), then IMT reports that the mutant is possibly equivalent, otherwise it reports that the mutant is non-equivalent.

An Input-Output pair I/O is said to be valid if the SUT can produce output O in response to input I. IOR_1 doesn't cater for all valid Input-Output pairs — it's possible to observe $Population_{SOI}.size() == Population_{SOO}.size()$ in Sys. IOR_1 must report that its classification was inconclusive in such cases. This can be remedied by creating more IORs that cover such pairs. For example, $Population_{SOI}.size() == Population_{SOO}.size()$ can be IOR_2 and $CrossoverExecuted == false$ can be its ID.

Interlocutory Relations (IRs) are the final construct used by IMT. An IR groups multiple IORs together to enable the definition of potentially complex relationships between IORs. Such relationships can enhance their classification accuracy. To illustrate, since all valid Input-Output pairs in Sys are collectively covered by IOR_1 and IOR_2, if a situation arises where neither IOR_1 nor IOR_2 is satisfied i.e. if $Population_{SOI}.size() < Population_{SOO}.size()$, then the IR can

guarantee that the Input-Output pair under consideration can not have been observed in Sys, and thus reports that the mutant is non-equivalent. We refer to this grouping of IOR_1 and IOR_2 as IR_1. Thus, an IR operates as follows: Each IOR that is associated with the IR is evaluated as described above to obtain a set of Possibly Equivalent/Non-Equivalent/Inconclusive classifications. These classifications are analysed by the IR to arrive at a final conclusion. If at least one classification is possibly equivalent and none are non-equivalent, then the final conclusion is that the mutant is equivalent, and if at least one is non-equivalent, then the final conclusion is non-equivalent. Assuming that the IR has IORs that collectively cover all valid Input-Output Pairs, the final conclusion can be non-equivalent if all classifications are inconclusive (as is the case for IR_1).

3.2 Interlocutory Mutation Testing and Non-determinism

Intuition Consider the Tournament Selection Operator (TSO) of a Genetic Algorithm. In particular, consider the logic that determines the winner of a tournament. A tournament consists of a set of competitors $tournament = \{Competitor_1, Competitor_2, ..., Competitor_n\}$, each of which is associated with a fitness value. One $Competitor_i \in tournament$ is randomly selected to be the winner of the tournament. A competitor's chance of winning is based to their fitness value, relative to the combined fitness values of all other competitors in the tournament. Thus, even though any competitor could win, the competitor with the highest fitness will have the greatest chance of being selected as the winner. Let $winner$ denote the selected competitor. On invocation of TSO, multiple tournaments are performed $tournaments{=}\{\langle tournament_1, winner_1\rangle, \langle tournament_2, winner_2\rangle, \langle tournament_3, winner_3\rangle, ...\}$.

An IR, which we will refer to as TournamentPIR, may be constructed for TSO. TournamentPIR may be associated with one IOR IOR_{TPIR} that is only satisfied under the following condition: For each $\langle tournament_i, winner_i\rangle$ in $tournaments$, $tournament_i$ contains at least two competitors $Competitor_j$ and $Competitor_k$, such that $Competitor_j.getFitnessValue() \neq Competitor_k.getFitnessValue()$.

Let $tournaments_{strong}$ be a subset of $tournaments$, such that for each $\langle tournament_i, winner_i\rangle \in tournaments_{strong}$, $winner_i$ was a solution with the highest fitness in $tournament_i$. Conversely, let $tournaments_{weak}$ be a subset of $tournaments$, where in each $\langle tournament_i, winner_i\rangle \in tournaments_{weak}$, $winner_i$ was a solution with the lowest fitness. IOR_{TPIR} may be associated with an ID that predicts that $tournaments_{strong}$ contains more members than $tournaments_{weak}$.

In summary, TournamentPIR predicts that $tournament_{strong}$ will contain more members than $tournaments_{weak}$ (this is the ID), when every tournament in $tournaments$ contains at least two competitors with different fitness values (this is the IOR). Although it's unlikely, it's possible that $tournament_{strong}$ may validly contains fewer members than $tournaments_{weak}$. This means that Tour-

namentPIR can misclassify an equivalent mutant as a non-equivalent mutant. We refer to such a misclassification error as a false positive.

This demonstrates that a revised evaluation method is necessary for IRs that deal with probabilistic behaviours, to reduce the incidence of false positives. We refer to IRs that use the revised evaluation method as Probabilistic IRs (PIRs). For the sake of clarity, we refer to IRs that use the evaluation method detailed above as Deterministic IRs.

The intuition behind the new evaluation method is as follows. As discussed above, certain behaviours can cause PIRs to report false positives e.g. when $tournament_{strong}$ contains fewer members than $tournament_{weak}$. The randomised properties of a system determine how frequently certain behaviours are observed. This means that all behaviours, including those that can lead to false positives will have a typical rate of occurrence. In other words, a PIR has a typical false positive rate. The proposed evaluation method is to use statistical techniques to compare a PIR's typical false positive rate to the proportion of non-equivalent classifications made by that PIR; if the proportion of non-equivalent mutant classifications is significantly higher than the false positive rate, then it's likely that the mutant is non-equivalent, otherwise, it's possible that the mutant is equivalent.

Technique Description This section introduces the evaluation method used by PIRs to reduce the impact of false positives.

The PIR evaluation method is two-fold. The first part of the evaluation method attempts to reduce the impact of false positives for a single test case *tc*. Let PIR be a PIR e.g. TournamentPIR and suppose that PIR has a typical false positive rate FPR_{tc} of 30%. FPR_{tc} can be determined by analysing the randomised properties of the SUT, extrapolated from empirical test data, or be based on the tester's expertise. PIR may be evaluated multiple times during an execution of *tc*. For example, TournamentPIR is evaluated each time TSO is executed, and TSO can execute multiple times if the Genetic Algorithm has been configured to perform more than one generation. Each evaluation of PIR will either yield an equivalent or non-equivalent classification. Let $R_{tc} = count(Non_Equivalent_{tc}) \div (count(Non_Equivalent_{tc}) + count(Equivalent_{tc}))$, where $count(Non_Equivalent_{tc})$ and $count(Equivalent_{tc})$ represent the number of times the mutant was classified as Non-Equivalent and Equivalent respectively. Thus, R_{tc} represents the proportion of times that PIR classified the mutant as Non-Equivalent in *tc*. In the first part of the evaluation method, R_{tc} is compared with FPR_{tc} using Pearsons χ^2. PIR's classification of the mutant based on tc is Non-equivalent if $R_{tc} > FPR_{tc}$ and the difference is statistically significant, otherwise the classification is equivalent. $PIR_C(tc)$ denotes this classification. To illustrate, suppose that $R_{tc} = 70\%$ and PIR was evaluated 100 times; since $70\% > 30\%$ and the difference between R_{tc} and FPR_{tc} is significant, $PIR_C(tc)$ would be Non-Equivalent. Conversely, if $R_{tc} = 33\%$, the difference between R_{tc}

and FPR_{tc} would not be statistically significant and $PIR_C(tc)$ would Equivalent.

As discussed above, the first part of the PIR evaluation method alleviates the impact of false positives for a single test case execution. However, because of non-determinism, it's also possible for $PIR_C(tc)$ to be a false positive. Typically, one has access to a test suite $ts = \{tc_1, tc_2, ...\}$. Each test case $tc_i \in ts$ would have been subjected to the first part of the PIR evaluation method to obtain an Equivalent or Non-Equivalent classification $TCClassifications = \{PIR_C(tc_1), PIR_C(tc_2), ...\}$. The second part of the PIR evaluation method compares the proportion of Non-Equivalent to Equivalent classifications in $TCClassifications$ to a known false positive rate for $TCClassifications$ for the PIR under consideration using Pearsons χ^2. This "known false positive rate" can be determined using the same methods as above. The results of this comparison is interpreted in the same way as in the first part of the evaluation method; the resulting classification is the PIR's final classification for the mutant.

3.3 Applying IMT

Multiple IRs In practice, one would typically leverage multiple IRs. Each IR may classify the mutant differently. This should be interpreted as follows: The mutant should be assumed to be non-equivalent if at least one IR classifies the mutant as non-equivalent, and should be considered to be equivalent if all IRs classify the mutant as equivalent.

Assumptions IMT assumes that an IR is encoded with accurate information about how Sys works. Unfortunately, this assumption may not hold if a real fault is in the system or IRs. To reduce the impact of this assumption, we recommend applying the IRs to Sys with a test suite. If any of the IRs indicate that the Sys is non-equivalent, then the assumption doesn't hold. In such cases, one can either modify the system and/or IRs, or remove IRs until all IRs report that Sys is equivalent. The same test suite should then be used for conducting IMT.

Constructing IRs s_i and s_o denote the program's input and output respectively. One must use one's domain knowledge to develop an intuition into how s_i and s_o are related. s_i, s_o and this intuition form an IOR. Tools that partially automate the exploration of relationships between inputs and outputs may simplify this task [6]. One must then leverage one's knowledge about the SUT's implementation details to identify execution trace behaviours that should manifest in executions in which this IOR is satisfied.

UCov is a test case coverage adequacy assessment tool for regression testing [3]. Like IMT, UCov leverages execution trace behaviours to achieve its objective. However, these execution trace behaviours are used to assess the intent of a test case i.e. program behaviours that should be executed by the test case, whilst such behaviours are used by IMT to assess the intent of the SUT i.e. program

behaviours that should manifest if the SUT has not been adversely affected by the mutation. Given their similarities, some of UCov's findings are relevant for IMT. For example, the aforementioned knowledge has been found to be available in the SUT's documentation [3].

Automated program analytic tools like Program Slicing [8] and Invariant Detection e.g. Daikon [9] can assist one in identifying useful execution trace behaviours. These behaviours are the IDs of IOR. This process is repeated to obtain multiple pairs $\langle IOR_i, IDs_i \rangle$, where IOR_i is an IOR and IDs_i is a set of IDs that are associated with IOR_i. Finally, one can group multiple pairs together, such that the IORs in these pairs have relationships. Identifying IORs that are amenable to such a grouping can be a natural task, because such IORs are typically highly related.

4 Experimental Set-up

4.1 Subject Program

The subject program is a Genetic Algorithm for the Bin Packing Problem that was developed by the author based on the design of Mladen Jankovic [10] with the JAGA Genetic Algorithm API toolbox [18]. The subject program consists of 1606 source lines of code (SLOC)[2], 29 classes and 244 methods (average 8 per class). The subject was partly selected to enhance the representativeness of the experiment and also minimise experimental bias. The former is achieved because it is non-deterministic and has weak information flow strength [14] and is thus susceptible to coincidental correctness. With regards to the latter, the implementation involves multiple developers, most of which were not aware of this research.

4.2 Interlocutory Relations

We used the same 48 IRs that were used in our previous work [19]. For a comprehensive list of these IRs, please see [19]. A real fault was present in the system, so we tested the assumption outlined in Section 3.3. We found that the assumption holds i.e. these IRs were not sensitive to the real fault. 42 IRs are Deterministic and 6 are Probabilistic.

4.3 Mutants

MuJava [12] was used to generate 30 non-equivalent mutants. It was applied to all classes that significantly contributed to the SUT's core functionality. 11 interface classes (MuJava couldn't produce mutants for these), 2 unused classes and the test case input class were excluded. We also excluded 3 simple data classes and 2 abstract classes that stored a single object and only implemented getter/setter

[2] We used the "Code Lines" metric in the Understand program [22] to compute SLOC. This metric ignores blank and comment lines.

methods and/or just exposed methods that this object already has. For example, the simple data class may have an ArrayList $ArrayObj$ and a method $remove(i)$, which simply calls $ArrayObj.remove(i)$. Finally, a comparator class was also excluded. Equivalent mutants and obvious mutants (i.e. mutants that resulted in system crashes or infinite loops) were also removed. We also rejected mutations of faulty code. These mutants were classified as either coincidentally correct or standard faults. Let S denote the system and M be a mutant of S. $ORACLE$ is an oracle that checks all of S's output properties (listed below). This was achieved by using $ORACLE$ on M's output. If $ORACLE$ fails, then the infected state didn't propagate to the output; thus M is coincidentally correct. We found that 15 were coincidentally correct and 15 were standard.

- Let $DataSet$ be the set of items to be sorted into bins. The output O should be a permutation of $DataSet$.
- O should contain at least one bin.
- O should not contain empty bins.
- O should not contain a bin that has more items than its capacity.
- O should not have a fitness that is greater than the maximum obtainable fitness (Fitness Function Constant).

Refactoring augments source code structure, while retaining behaviour; thus refactorings are effectively equivalent mutants. AutoRefactor [20] was used to generate 30 equivalent mutants.

In summary, this experiment leverages 60 mutants in total, 30 non-equivalent and 30 equivalent.

4.4 Test Cases

We use the same test suite that was used in our previous work [19]. The test suite consists of 100 test cases that were generated by Random Testing.

5 Results and Discussion

This section reports an empirical study that measures the accuracy of IMT for non-equivalent and equivalent mutants.

5.1 Non-Equivalent Mutants

IMT correctly classified 28/30 non-equivalent mutants. This suggests that IMT's classification accuracy can be high for non-equivalent mutants. Since the SUT is non-deterministic, this also demonstrates that the technique's classification accuracy for these mutants was not hampered by non-determinism. Specifically, 15/15 and 13/15 standard and coincidentally correct mutants were correctly classified. The difference in performance for these mutant types is not significant (Fisher's Exact Test: $p > 0.05$). This indicates that IMT can be effective for standard and coincidentally correct faults.

Recall that there are two types of IRs - Deterministic and Probabilistic IRs. These IRs are distinguished by the types of logic they are applied to — deterministic IRs are applied to aspects of the system that behave deterministically, whilst probabilistic IRs are applied to non-deterministic aspects of the system. To that end, each approach has different evaluation methods; the difference being, Probabilistic IRs leverage statistical techniques to factor out the effect of false positives that arise due to non-determinism. We therefore decided to further break down the analysis by these IR types.

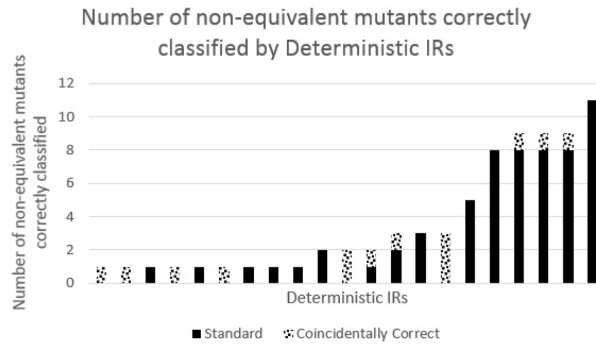

Fig. 1: Number of mutants that were correctly classified by Deterministic IRs, broken down by mutant type

Deterministic IRs correctly classified 23/30 (13/15 standard and 10/15 coincidentally correct) non-equivalent mutants. The difference in the Deterministic IR's performance for standard and coincidentally correct mutants is not statistically significant (Fisher's Exact Test: $p > 0.05$). This demonstrates that one can leverage these IRs in contexts where coincidental correctness is present, or absent. Each bar in Figure 1 represents a Deterministic IR that correctly classified a mutant. The height of the bar denotes the number of correctly classified non-equivalent mutants. Each bar also represents the proportion of mutants that were standard or coincidentally correct. Figure 1 demonstrates that some IRs are more accurate than others for different mutants. For example, the IR represented by the third bar correctly classifies standard mutants, but not coincidentally correct mutants, and the converse is true for the IR represented by the second bar.

19/30 (14/15 standard and 5/15 coincidentally correct) non-equivalent mutants were correctly classified by Probabilistic IRs. A comparison of the performance of Deterministic and Probabilistic IRs for standard faults revealed that the difference was not statistically significant (Fisher's Exact Test: $p > 0.05$), but was for coincidentally correct faults (Fisher's Exact Test: $p < 0.05$). This suggests that Probabilistic IRs may be less effective in situations where coincidental correctness is present. However, we observed that 3 of the coincidentally

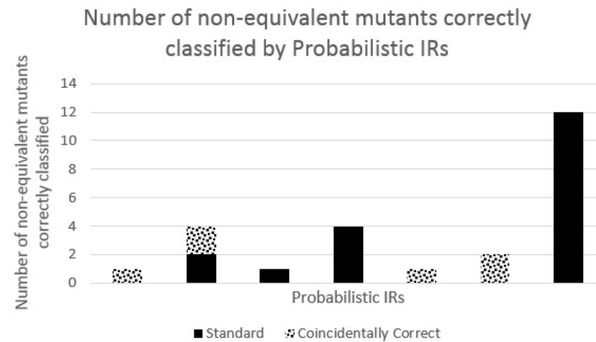

Fig. 2: Number of mutants that were correctly classified by Probabilistic IRs, broken down by mutant type

correct faults found by IMT were uniquely identified by Probabilistic IRs, which means that they can add value in situations where coincidental correctness is present. Figure 2 presents the same information as in Figure 1, but for Probabilistic IRs; it shows the breakdown of the results; similar observations can be made to those in Figure 1.

As discussed above, all of the IRs collectively, correctly classified 28/30 non-equivalent mutants. Deterministic IRs and Probabilistic IRs correctly classified 23 and 19 mutants respectively, which means that neither IR type correctly classified all of the mutants on their own. This demonstrates that both IR types can add value.

Interestingly, these results also suggest that there was a substantial degree of overlap in terms of the number of mutants that were correctly classified by the IRs. We therefore decided to perform a subsumption analysis to determine the smallest number of IRs that would be required to obtain the same results. We found that only 12 were necessary: AverageFitnessGeneration, ChoosingPairsOfParentsComposition, CreateRandomIndividualNewBins, CrossoverRate, DecidingWhoShouldMutateFineGrained, GAController, MutateIndividual, PartitionChild, ReplacementOperationIntegrity, ShouldUseNewIndividual, TerminateGA, TournamentComposition. This shows that the technique can be effective with relatively few IRs.

5.2 Equivalent Mutants

Promisingly, IMT correctly classified 30/30 equivalent mutants. Since Deterministic IRs don't check non-deterministic aspects of the system, they aren't susceptible to false positives, assuming that the assumption detailed in Section 3.3 holds. It's therefore not surprising that they did not misclassify any equivalent mutants. Since Probabilistic IRs do check such behaviours, false positives may be possible. To that end, we extended the evaluation method used by Probabilistic IRs, as described in Section 3.2, to curtail the incidence of false positives.

These results illustrate that this evaluation method was successful in achieving this goal.

6 Threats to Validity

There are several threats to validity. We attempted to address these where possible e.g. randomisation was used throughout the experiment to reduce experimental bias.

Firstly, the presence of real faults may confound the results i.e. an IR may assume that misbehaviour emanating from a real fault actually originated from the mutant process. To mitigated the impact of real faults on the experiment, we only used IRs that were not sensitive to the real fault and excluded mutations of the real fault.

Each IR is associated with a logging function. These logging functions capture data about the execution trace, during the execution of the SUT. Some mutants can alter the SUT's control flow. These alterations can cause the logging functions to crash. In such situations, the IR has effectively recognised that the SUT's control flow is incorrect and has thus correctly classified the non-equivalent mutant. Our experiment did not distinguish between these crashes and system crashes, and so they were conservatively removed. Therefore, the experimental results presented in this paper for non-equivalent mutants underestimate the technique's effectiveness. However, we do not believe that this had a significant impact on the results, since the technique already correctly classifies most of the mutants.

There is also a threat to generalisability; we only used one subject program. However, the subject program had the operating environment that we were studying i.e. non-determinism and a high propensity for coincidental correctness, and was therefore suitable for assessing our research objectives. As a part of ongoing research, we are currently applying IMT to four other subject programs; the preliminary results are promising, see Section 7.

Finally, the results demonstrated that different IRs obtained different levels of effectiveness. Thus, the effectiveness of the technique may vary considerably, depending on one's choice of IRs. This may be a threat to repeatability.

7 Conclusion

In this paper, we proposed Interlocutory Mutation Testing, the first mutant classification technique that can be applied in the presence of coincidental correctness and/or non-determinism. The technique correctly classified 93.33% of the non-equivalent mutants and 100% of the equivalent mutants, which suggests that the technique is capable of producing highly accurate results. We also observed that different IRs are more effective than others for classifying different faults, which suggests that using a diverse range of IRs can be valuable.

As mentioned in Section 6, one of the limitations of our study is that we only considered one subject program. As a part of on going research, we are currently

conducting IMT on four other subject programs. A brief summary of the preliminary results are as follows. We applied IMT to Dijkstra's Algorithm. IMT obtained a non-equivalent mutant classification accuracy of 93.33%, and 100% mutant classification accuracy for equivalent mutants; 30 non-equivalent and 30 equivalent mutants were used. 34 mutants, which include a mixture of equivalent and non-equivalent mutants, were also generated across Bubble Sort, Binary Search and Knuth-Morris-Pratt. All of these mutants were correctly classified. It is our hope that these experiments will reduce the impact of this limitation.

Another limitation of our work is the effort required to apply the technique. Our experiment leveraged 48 IRs, which may be unacceptable in some cases. In Section 5.1, we observed that a small proportion (12) of the IRs subsumed all of the other IRs. This demonstrates that the technique can be applied with relatively few IRs, which may be more acceptable in the aforementioned cases, if one restricts their development efforts to such IRs. Unfortunately, the results did not indicate how one might do this. We would therefore like to investigate this in future work.

In Section 3.3, we detailed the partially automated process that is used to develop IRs. Increasing the degrees of automation further will also reduce the effort required to use the technique and so can reduce the impact of the limitation above. Thus, for future work, we would like to explore methods of automating the development of IRs further.

In the future, we would also like to assess the impact that IMT has on one's mutant classification productivity. This would involve determining the costs that are associated with developing IRs, and the cost savings that can be obtained from leveraging the technique. As a part of ongoing work, we are currently investigating the latter.

References

1. Aichernig, B.K., Jobstl, E.: Efficient Refinement Checking for Model-Based Mutation Testing. In: International Conference on Quality Software (QSIC). pp. 21–30. IEEE, Xi'an, China (2012)
2. Androutsopoulos, K., Clark, D., Dan, H., Hierons, R.M., Harman, M.: An Analysis of the Relationship Between Conditional Entropy and Failed Error Propagation in Software Testing. In: Proceedings of the 36th International Conference on Software Engineering. pp. 573–583. ACM, NY, USA (2014)
3. Assi, R.A., Masri, W., Zaraket, F.: UCov: a user-defined coverage criterion for test case intent verification. Software Testing, Verification and Reliability pp. 1–32 (2016)
4. Budd, T.A., Angluin, D.: Two notions of correctness and their relation to testing. Acta Informatica 18(1), 3145 (1982)
5. Carver, R.: Mutation-based testing of concurrent programs. In: Proceedings of IEEE International Test Conference. pp. 845–853. IEEE, USA (1993)
6. Chen, T.Y., Poon, P.L., Xie, X.: METRIC: METamorphic relation identification based on the category-choice framework. Journal of Systems and Software 116, 177–190 (2016)

7. Gligoric, M., Jagannath, V., Marinov, D.: MuTMuT: Efficient Exploration for Mutation Testing of Multithreaded Code. In: Proceedings of the Third International Conference on Software Testing, Verification and Validation. pp. 55–64. IEEE Computer Society, USA (2010)
8. Harman, M., Hierons, R.M.: An overview of program slicing. Software Focus 2(3), 85–92 (2001)
9. Harman, M., McMinn, P., Shahbaz, M., Yoo, S.: A comprehensive survey of trends in oracles for software testing. Tech. Rep. Tech. Rep. TR-09-03, University of Sheffield (2013)
10. Jankovic, M.: Genetic Algorithm for Bin Packing Problem (2013)
11. Jia, Y., Harman, M.: An Analysis and Survey of the Development of Mutation Testing. IEEE Transactions on Software Engineering 37(5), 649–678 (2011)
12. Ma, Y.S., Kwon, Y.R., Offutt, J., Li, N.: Mujava. `http://cs.gmu.edu/~offutt/mujava/` (2013)
13. Masri, W., Assi, R.: Cleansing Test Suites from Coincidental Correctness to Enhance Fault-Localization. In: Third International Conference on Software Testing, Verification and Validation (ICST). pp. 165–174. IEEE, Paris, France (2010)
14. Masri, W., Assi, R.A.: Prevalence of coincidental correctness and mitigation of its impact on fault localization. ACM Transactions on Software Engineering and Methodology 23(1), 1–28 (2014)
15. Offutt, A.J.: Investigations of the software testing coupling effect. ACM Transactions on Software Engineering and Methodology 1(1), 5–20 (1992)
16. Offutt, A.J., Lee, S.D.: How Strong is Weak Mutation? In: Proceedings of the Symposium on Testing, Analysis, and Verification. pp. 200–213. ACM, NY, USA (1991)
17. Papadakis, M., Jia, Y., Harman, M., Traon, Y.L.: 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. pp. 936–946. IEEE, USA (2015)
18. Paperin, G.: JAGA - Java API for Genetic Algorithms. `http://www.jaga.org/index.html` (2004)
19. Patel, K., Hierons, R.M.: Interlocutory Testing: Combating Coincidental Correctness in Testing. `people.brunel.ac.uk/~csstrmh/Intt/IT.pdf` (2015)
20. Rouvignac, J.N.: AutoRefactor. `https://marketplace.eclipse.org/content/autorefactor` (2015)
21. Sadi, M.S., Kuo, F.C., Ho, J.W.K., Charleston, M.A., Chen, T.Y.: Verification of phylogenetic inference programs using metamorphic testing. Journal of Bioinformatics and Computational Biology 9(6), 729–747 (2011)
22. Scitools: Understand static code analysis tool. `https://scitools.com/` (2016)
23. Voas, J.: PIE: a dynamic failure-based technique. IEEE Transactions on Software Engineering 18(8), 717–727 (1992)
24. Yao, X., Harman, M., Jia, Y.: A study of equivalent and stubborn mutation operators using human analysis of equivalence. In: Proceedings of the 36th International Conference on Software Engineering. pp. 919–930. ACM, NY, USA (2014)