Third International Conference on Software Testing, Verification, and Validation Workshops

Metamorphic Testing of Stochastic Optimisation

Shin Yoo
Centre for Research on Evolution, Search & Testing
King's College London
London, UK
Shin.Yoo@kcl.ac.uk

***Abstract*—Testing stochastic optimisation algorithms presents an unique challenge because of two reasons. First, these algorithms are *non-testable* programs, i.e. if the test oracle was known, there wouldn't have been the need for those algorithms in the first place. Second, their performance can vary depending on the problem instances they are used to solve. This paper applies the statistical metamorphic testing approach to stochastic optimisation algorithms and investigates the impact that different problem instances have on testing optimisation algorithms. The paper presents an empirical evaluation of the approach using instances of Next Release Problem (NRP). The effectiveness of the testing method is evaluated using mutation testing. The result shows that, despite the challenges from the stochastic nature of the optimisation algorithm, metamorphic testing can be effective in testing them.**

***Keywords*-Metamorphic Testing, Optimisation, Search-Based Software Engineering**

I. Introduction

The use of meta-heuristic optimisation techniques in software engineering, often referred to as Search-Based Software Engineering (SBSE), has been of growing interest to researchers [1–4]. The main idea behind SBSE is that software engineering problems can be reformulated as optimisation problems and 'good enough' solutions can be obtained by the use of existing optimisation algorithms. This approach has been successfully applied to problems across the software life-cycle, from Requirements Engineering to regression testing [4].

As the SBSE approach matures, various optimisation techniques find their place in software tools [5–10]. These practical applications of optimisation techniques raise an interesting software testing question: how do we *test* these optimisation techniques? Or, more precisely, how do we test the implementations of these optimisation algorithms?

The answer to this question is not as trivial as to compare the implementation to the reference algorithm design, often because there is no such reference. Many well-studied optimisation algorithms come with multiple variants of themselves. Genetic Algorithms (GAs), for example, allow the user to select genetic operators from a huge pool of existing selection, cross-over and mutation operators; each unique combination of these operators can be thought of as a new variant of GA. Furthermore, it is common that these optimisation algorithms need to go through slight modifications in order to be applied to a specific software engineering problem.

However, testing the optimisation algorithms involves solving a much more fundamental testing problem; that is, the problem of testing 'non-testable' programs [11]. Weyuker described 'non-testable' programs as "programs which were written *in order to determine the answer in the first place*". Testing the optimisation algorithms would require a test oracle to compare the test results to. However, if the correct answer to the optimisation problem was known in advance, there would not have been the need for the optimisation algorithm.

Metamorphic testing is a testing approach that seeks to overcome the lack of test oracle [12]. The idea behind metamorphic testing is that certain properties of the program under test can allow the tester to create, from an existing test case, a new 'follow-up' test case whose outcome can be predicted according to the *metamorphic relations*. The outcome of the follow-up test case can act as a pseudo-oracle for the original test case, or vice versa. It has been applied to testing heuristics such as machine learning algorithms [13].

However, there are a few challenges in applying metamorphic testing to stochastic optimisation algorithms. First, the stochastic nature of these algorithms prevents the direct comparison of test results to the test oracle. For metamorphic testing of nondeterministic programs, Guderlei and Mayer proposed the Statistical Metamorphic Testing (SMT), combining statistical hypothesis test with metamorphic testing, in order to overcome the randomness in output [14].

The second issue is perhaps more uniquely problematic for meta-heuristic optimisation algorithms; the observed performance of an optimisation algorithms may depend not only on the correctness of the implementation of the algorithm, but also the relative *difficulty* of the instance of the problem the algorithm is applied to. Guderlei and Mayer only considered a program with hard-coded randomness; an implementation of the inverse cumulative distribution function of the normal distribution Φ^{-1}. For stochastic optimisation algorithms, it is possible that an implementation passes the metamorphic test with a problem instance p_A, but fails to perform correctly for another problem instance, p_B. Therefore, it is crucial to measure and evaluate the

978-0-7695-4050-4/10 $26.00 © 2010 IEEE
DOI 10.1109/ICSTW.2010.26

impact of different problem instances on the effectiveness of metamorphic testing of meta-heuristic optimisation.

This paper presents a metamorphic testing approach for stochastic optimisation algorithms based on statistical analysis and aims to investigate the impact that different problem instances can have on the effectiveness of metamorphic testing of stochastic optimisation algorithms. Using injected faults, a well studied stochastic optimisation algorithm (Simulated Annealing) has been tested for multiple instances of Next Release Problem (also known as *Release Planning*) formulated from both real-world and synthetic datasets. The effectiveness of SMT approach was measured using mutation testing. The results showed that metamorphic testing combined with statistical analysis of its results can be effective for testing stochastic optimisation algorithms against certain classes of faults. While the paper considers NRP and Simulated Annealing as a test subject, the outlined SMT approach is applicable to search-based software testing with any stochastic optimisation algorithm.

The contributions of this paper are as follows:

1) The paper presents the first attempt to provide a systematic testing framework for stochastic optimisation algorithms using the statistical metamorphic testing methodology.

2) The paper presents a controlled empirical evaluation to investigate how different problem instances can affect the effectiveness of metamorphic testing of optimisation algorithms, using multiple datasets including both real-world and synthetic data.

3) The paper considers the impact of different statistical confidence level on the accuracy of the statistical metamorphic testing approach.

The rest of the paper is organised as follows. Section II introduces metamorphic testing approach. Section III discusses both the simulated annealing optimisation algorithm and the NRP from Requirements Engineering that will be studied in the paper. Section IV describes the set-up of the empirical study, the results of which are discussed in Section V. Section VI discusses the related work and Section VII concludes.

II. Metamorphic Testing

A. Concepts

Metamorphic testing is a process of generating *follow-up* test cases from correctly executed test cases in order to check important properties of the target function [12]. A successful test case is a one on which the program computes the correct output. Chen at al. argued that, although successful test cases are conventionally considered to be useless with respect to fault finding, they do carry valuable information that contribute towards a solution to the oracle problem [12]. This is made possible by exploiting specific relations between the System Under Test (SUT) and test cases.

A *metamorphic relation* (MR) is an expected relation between different pairs of inputs and outputs to SUT. Intuitively, if a metamorphic relation holds between a pair of test inputs, it follows that there also holds a specific relation between the pair of corresponding outputs, which allows the tester to infer what the test oracle should be. The following relations are some of the examples listed by Chen et al. [12] for the metamorphic testing of a function that calculates sine function:

- $R_1 : \sin x = \sin(2\pi + x)$
- $R_2 : \sin x = -\sin(\pi + x)$

Suppose that the Function Under Test (FUT) f, which is an implementation of sine function, returned a value α for a specific input value x. Using R_1, it is possible to obtain a follow-up test case with a metamorphic test oracle: $2\pi - x$ and α. That is, based on R_1, it follows that $f(x) = \alpha \rightarrow f(2\pi - x) = \alpha$. Similarly, it can be stated that, based on R_2, $f(x) = \alpha \rightarrow f(\pi + x) = -\alpha$. Using this metamorphic test oracles, fault can be detected when $\sin(x) \neq \sin(2\pi - x)$ or when $\sin x \neq -\sin(\pi + x)$. While R_1 and R_2 represent the identity relation (i.e. the outputs should be identical), there can be other types of metamorphic relations. For example, if the FUT is linear, a rational relation would be expressed as following: $f(x) = y \rightarrow f(\alpha x) = \alpha f(x) = \alpha y$

B. Statistical Metamorphic Testing

When the FUT is deterministic, applying a test oracle is essentially an equality check between a pair of outputs. However, when the FUT is stochastic, a simple equality check is not possible due to the inherent randomness. This, in turn, requires statistical analysis of multiple pairs of outputs from the original test case and the metamorphic follow-up test case. Statistical metamorphic testing is a sub-branch of metamorphic testing introduced by Guderlei and Mayer [14]. It overcomes where the metamorphic test oracle can only be checked by means of statistical analysis.

The exact type of statistical analysis to be performed should be decided based on the nature of the specific metamorphic relation in question. For example, if the metamorphic relation is identity relations similar to R_1 and R_2 in Section II-A, hypothesis testing for equality would be the appropriate choice. However, if exists, further domain knowledge such as the expected distribution of the test output can be used to select more precise hypothesis testing method.

C. Applying SMT to Optimisation Algorithms

Algorithm 1 outlines how SMT can be applied to stochastic optimisation algorithms. Given an implementation of an optimisation algorithm to test, A, a problem instance to test A with, P and a metamorphic relation, M, SMT approach

first obtains P' which is derived from P using M. It means that A should converge to the same solution most of times, if not always. SMT executes A with both P and P' for a given number of times (N) and then applies a given statistical hypothesis test, S, to the hypothesis H that two samples obtained respectively from $A(P)$ and $A(P')$ share the same mean value.

Algorithm 1: Outline of SMT for Optimisation Algorithms

Input: an implementation of an optimisation algorithm, A, a problem instance, P, a metamorphic relation, M, a statistical hypothesis test, S, a sample size, N
Output: *pass* if no fault is suspected, *fail* if a fault is likely

(1) $P' \leftarrow M(P)$
(2) $R \leftarrow \emptyset$, $R' \leftarrow \emptyset$
(3) **repeat**
(4) $\quad R \leftarrow R \cup \{A(P)\}$
(5) $\quad R' \leftarrow R' \cup \{A(P')\}$
(6) **until** N times
(7) $H \leftarrow$ hypothesis that $mean(R) = mean(R')$
(8) **if** H holds w.r.t. S **then return** pass
(9) **else return** fail

Since the outcome is based on a statistical analysis, it is not possible to decide whether A is definitely faulty or not. If H is statistically confirmed, it is likely that A is not faulty. However, if H is rejected, then A did not converge to the same solution for P and P' and, therefore, is more likely to be faulty. This means that the choice of S can affect the accuracy of SMT.

D. Research Questions

This paper aims to answer the following research questions.

- **RQ1. Accuracy:** How accurately can statistical metamorphic testing approach detect faults in Simulated Annealing algorithm?
- **RQ2. Insight:** What are the difficulties for applying statistical metamorphic testing approach?
- **RQ3. Impact of Problem Instance:** Does the accuracy of SMT approach differ between different problem instances?
- **RQ4. Impact of Strength of Hypothesis Testing:** Does the accuracy of SMT approach differ between different statistical significance level for hypothesis testing required by SMT approach?

RQ1 will be answered by evaluating SMT approach using mutation faults. **RQ2** will require more qualitative analysis of the results for **RQ1**. **RQ3** and **RQ4** will be answered by comparing the fault detection capability of SMT approach using different problem instances (for **RQ3**) and different statistical significance levels (for **RQ4**).

III. Next Release Problem & Stochastic Optimisation

A. Next Release Problem (NRP)

This paper uses instances of Next Release Problem (NRP) to illustrate the application of metamorphic testing to stochastic optimisation algorithms. NRP is a software engineering problem that aims to select the ideal subset of software features that should be implemented and included in the next version of the software system [15, 16]

More formally, let $R = \{r_1, \ldots, r_n\}$ be the set of all candidate software features that are available for the next version of the system. Let $C = \{c_1, \ldots, c_n\}$ be a set of costs for each features in R. A boolean decision vector, $x =< x_1, \ldots, x_n >$ denotes the subset of features that would be implemented in the next version; r_i is selected if $x_i = 1$, otherwise not.

Let $U = \{u_1, \ldots, u_m\}$ be the set of m customers who use the software system; each customer is assigned with a relative degree of importance to the company that releases the software system. Let $W = \{w_1, \ldots, w_m\}$ be the set of weights that reflect the relative importance of each customer to the company. Similarly, not all features in R carry the same value to customers. Each customer $u_j (1 \leq j \leq m)$ assigns the value of a specific requirement $r_i (1 \leq j \leq n)$, which is denoted by $v(r_i, u_j)$. Each feature r_i is assigned a score, s_i, based on these information: $S = \{s_i | s_i = \sum_{j=i}^{m} w_j \cdot v(r_i, u_j), 1 \leq i \leq n\}$. Finally, let B be the budget allowed for the selection of features. The basic NRP is defined by a tuple $< R, C, S, B >$, from which the aim is to obtain a decision vector x such that:

$$\text{maximise} \sum_{i=1}^{n} x_i \cdot s_i \text{ while subject to } \sum_{i=1}^{n} x_i \cdot c_i \leq B$$

This is the well-known 0/1 knapsack problem, which is known to be NP-complete [17] and, therefore, an appropriate candidate for meta-heuristic optimisation techniques. Bagnal et al. called this the *basic* NRP because this formulation does not consider the dependency relations between features [15]. However, Bagnal et al. also noted that an instance of NRP with dependency relations can be approximated by basic NRP by combining features in dependency chains into one large feature.

B. Simulated Annealing

Simulated Annealing is a type of stochastic local search that is specially designed to overcome one of the major drawbacks of local search techniques, i.e., to escape local optimum. When the algorithm starts, it will accept a move towards an inferior neighbouring solution according to a pre-defined probability. This probability, however, decreases as

the search progresses, converging to 0 near the end of the search. The decrease of the probability is modelled from the annealing process in metallurgy, which the algorithm its name. It has been successfully applied to NRP [18]. Algorithm 2 contains the outline of Simulated Annealing.

More formally, the algorithm starts with temperature T set to t_0. In each iteration, the probability of accepting an inferior neighbouring solution in an attempt to escape a local optimum is defined as $e^{\frac{-\Delta E}{T}}$, where ΔE denotes the difference in fitness value between the current solution and the candidate solution. At the end of each iteration, the value of T is decreased according to a specific *cooling scheme*. The algorithm finishes when the temperature reaches a predefined point.

Algorithm 2: Pseudocode for Simulated Annealing

```
(1)  s ← a random initial solution
(2)  T ← t_0
(3)  while T ≥ t_f
(4)      for i = 1 to L
(5)          n ← a neighbouring solution of s
(6)          if n is fitter than s
(7)              s ← n
(8)          else
(9)              r ← a random probability
(10)             if r < e^(−(fit(n)−fit(s))/T) then s ← n
(11)     T ← coolingScheme(T)
(12) return s
```

Following Baker et al. [18], the implementation used in the empirical study has been modified to what is outlined in Algorithm 2 in order to cater for the budget constraint that is an inherent part of NRP. It considers a neighbouring solution only when the budget constraint is satisfied in Line (5). The implementation of Simulated Annealing has been written in Java; the algorithm is contained in a method with 25 Non-Comment Source Statement (NCSS) and its cyclomatic complexity equals 25. Source code metrics have been obtained using `JavaNCSS` [19].

One particular strength of Simulated Annealing in the context of this paper is that the algorithm allows a systematic parameter tuning for a specific instance of a given optimisation problem [20]. This makes Simulated Annealing an ideal candidate for the empirical study in this paper, as the impact of inappropriate parameter setting on the performance and accuracy of the optimisation can be significantly reduced.

IV. Experiment Setup

A. Mutation Faults

This paper uses injected mutation faults in order to evaluate, with respect to **RQ1**, the effectiveness of SMT approach. Mutation faults are based on the idea of mutation testing, where the testing adequacy is measured based on simple syntactic modifications artificially made to the SUT [21].

Table I: Mutation operators used in the paper

Operator	Description
AOIS	Insert short-cut arithmetic operators
AIOU	Insert basic arithmetic operators
AORB	Replace basic binary arithmetic operators
COI	Insert unary conditional operators
LOI	Insert unary logic operators
ROR	Replace relational operators

A well known mutation tool, `muJava` [22], was applied to the implementation of Simulated Annealing algorithm outlined in Algorithm 2. Table I lists the mutation operators available from `muJava` that resulted in executable mutants of the implementation of Simulated Annealing; other operators did not produce executable variants. Since the algorithm was contained in a single Java method, mutation operators that concern Object-Oriented aspects of Java have not been applied. A total of 96 executable mutant programs have been generated.

One of the difficulties of injecting faults using mutation is the possibility of *equivalent* mutants, i.e. program variants that differ from the original program syntactically but remain the same semantically. In the context of this paper, injected faults that correspond to equivalent mutants are not real faults and, therefore, cannot be detected by any testing methodology. To answer **RQ2**, all 96 mutants have been manually analysed and inspected to classify them according to their semantic impact to Simulated Annealing. Out of the 96 mutants obtained by `muJava`, 10 were equivalent. Some of the non-equivalent mutants also resulted in program variants with infinite loops, of which there were 22. These were detected by the time-out routine in the testing harness and regarded as being killed.

B. Simulated Annealing Configuration

Simulated Annealing require a set of configuration parameters: a initial temperature, t_0, a final temperature, t_f, a cooling scheme which is a function of the temperature variable and a length for the Markov chain, L. Steinhöfel et al. suggested the following setup for t_0 and L [20]:

$$t_0 = \frac{\Delta_{max} fitness}{log(1 - p_1)} \text{ for a small positive } p_1$$

$$L = h \cdot [\text{num. of neighbouring solutions}] \ (h > 0)$$

Initial temperature t_0 is configured in such a way that, at the beginning, the chance of accepting any transition would be close to 1; the empirical study uses $p_1 = 0.8$. Length of Markov chain, L, is configured so that all neighbouring solutions can have enough chance of being considered; L was set to 15,000. A simple linear cooling scheme, $t_{i+1} = t_i * (1 - p_2)$ has been adopted, with $p_2 = 0.2$. The final temperature, t_f, is set to 0.005. These parameters

reproduces the configuration used in previous work that applied Simulated Annealing to NRP [18].

C. Problem Instances

To answer **RQ3**, three different feature datasets have been used to formulate problem instances studied in this paper. The first dataset, D_1, is based on a real-world feature requirement dataset from a large telecommunication company, which previously has been used for the analysis of multi-objective optimisation for NRP [16]. It contains 35 distinctive features. The second and the third sets, D_2 and D_3 respectively, have been generated randomly. Both contain 100 features. For D_2, the values of cost and score for each feature have been sampled from a discrete uniform distribution whereas, for D_3, these values have been sampled from a normal distribution.

Table II: Instances of Next Release Problem

Dataset	B_{low}	B_{mid}	B_{high}
Real-world (D_1)	674	3,370	6,066
Random/Uniform (D_2)	792	3,962	7,131
Random/Normal (D_3)	807	4,039	7,072

An instance of NRP requires not only a feature dataset with costs and scores but also a budget value. For each dataset, three different budget values have been defined: B_{low}, B_{mid} and B_{high}. Table II presents the budget values for different datasets. For each dataset, B_{low} was set to be about 10% of the total cost of all features. Similarly, B_{mid} and B_{high} were set to be about 50% and 90% of the total cost of the corresponding dataset respectively.

Unlike different datasets, the effect of which is difficult to anticipate, different budget sizes allows some prediction of the results. It may be anticipated that different budget values will affect the *difficulty* associated with the optimisation for the specific problem instance. For each dataset, the lower budget, B_{low}, has been set sufficiently low so that the majority candidate solutions will be unacceptable due to the budget constraint. This effectively reduces the search space, helping the algorithm to find the global optimum. Similarly, the higher budget, B_{high}, is set sufficiently high so that the majority of the features can be safely included in the final solution. This effectively lessens the effect of the budget constraint, helping the algorithm to find the global optimum. However, the mid-level budget, B_{mid}, is set so that there are many different combinations of features that satisfy the given budget constraint, thereby presenting a much *harder* optimisation problem.

D. Metamorphic Relations and Evaluation

Identifying useful metamorphic relations for SBSE problems can be a challenging task on its own. The discovery of more sophisticated metamorphic relations would present an interesting future work; this paper begins the investigation with a basic and intuitive convergence relation for NRP based on permutation of problem instances. Theoretically, the set of candidate features R is a *set* and, therefore, not ordered. Practically, the same information is almost always contained in an *ordered* data structure such as arrays or vectors and each feature is accessed using iterators. However, different ordering of features should not affect the final solution that Simulated Annealing converges to.

More formally, let P be a permutation operator for ordered sets; if an implementation of Simulated Annealing, SA, is *correct*, it follows that $SA(< R, C, S, B >)$ and $SA(< P(R), P(C), P(S), B >)$ should converge to the same solution within the margin of error allowed by the inherent randomness of Simulated Annealing.

To cater for the inherent randomness and the subsequent need for the margin of error for comparing solutions, statistical hypothesis testing is applied. The implementation of Simulated Annealing guarantees that every solution produced satisfies the budget constraint; therefore, only the scores of the solution are compared using Wilcoxon's rank-sum test. Wilcoxon's rank-sum test is a non-parametric statistical hypothesis test that does not require the population to be normally distributed.

For each instance of NRP, two samples consist of solutions obtained by $SA(< R, C, S, B >)$ and $SA(< P(R), P(C), P(S), B >)$ respectively; the algorithm was executed for 30 times for each instance. The random number generator has been seeded with the system clock. The null hypothesis is that there is no difference between two samples, while the alternative hypothesis is that two samples are different. Accepting the null hypothesis means that the metamorphic relation holds and, therefore, the implementation is correct. Accepting the alternative hypothesis means that the metamorphic relation does not hold and, therefore, the implementation is faulty. For mutated implementations of Simulated Annealing, the desired outcome is to reject the null hypothesis and to accept the alternative hypothesis (i.e. the mutant is *killed*). To answer **RQ4**, 95% and 99% significance levels have been used.

V. Results & Analysis

A. Effectiveness

Figure 1 represents the overall results from applying SMT approach to different problem instances described in Table II. Recall from Section IV-A that 10 of the 96 mutant programs were equivalent mutants; these mutants cannot be killed. Based on this, the results have been classified into the following four categories:

- **TP (True Positive):** the mutant program is not equivalent and SMT approach statistically revealed differences in output between the original and the mutant program (i.e. killed the mutant). This would equal to the mutation score.

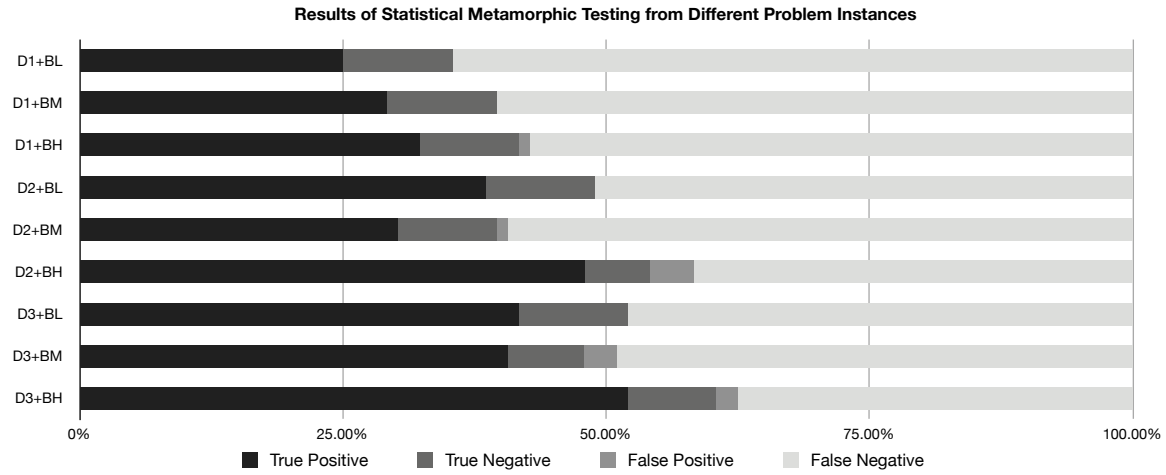

Figure 1: Results of applying statistical metamorphic testing to different problem instances. True Positive (TP) represents non-equivalent mutant programs killed by SMT approach, whereas True Negative (TN) represents non-equivalent mutant programs that are *not* killed. Similarly, False Positive (FP) represents equivalent mutant programs *wrongly* killed by SMT approach, whereas False Negative (FN) represents non-equivalent programs that are *not* killed. In all cases, False Positive (FP) accounts for less than 5%. Correct diagnosis (TP + TN) ranges from 25.42% to 60.41%.

- **TN (True Negative):** the mutant program is equivalent and SMT approach did not statistically reveal differences in output between the original and the mutant program (i.e. did not kill the mutant).
- **FP (False Positive):** the mutant program is equivalent and, therefore, impossible to kill, but SMT approach statistically revealed differences in output between the original and the mutant program.
- **FN (False Negative):** the mutant program is not equivalent but SMT approach did not statistically reveal the differences in output between the original and the mutant program.

Table III presents the results in more details. Overall, correct behaviours (TP + TN) account for from 35.42% ($D_1 + B_{low}$) to 60.41% ($D_3 + B_{high}$) of all the mutants. In most cases, TN values are 8 or higher, showing that equivalent mutants were properly detected as not–kill*able*. Naturally, FP is the category with the lowest ratio across all problem instances. To answer **RQ1**, statistical metamorphic testing of Simulated Annealing for NRP achieved mutation scores (i.e. TN) ranging from 24 to 50 out of 86 non-equivalent mutants.

B. Insights

To answer **RQ2**, it is necessary to understand the nature of each mutation. 96 mutation faults studied here have been manually classified according to their semantic impact on Simulated Annealing. The type *equivalent* corresponds to equivalent mutants and, therefore, not really faults. *Configurational* faults are those whose effect can be recreated by configuring the parameters of Simulated Annealing in a wrong manner; for example, mutation faults that result in an initial temperature that is too low. *Control-flow* faults are those that directly affect the control flow of the algorithm; for example, mutation faults that result in infinite loops by increasing the temperature rather than decreasing. Finally, *miscalculation* faults are those that affect the core mathematical expressions in the algorithm; for example, mutation faults that affect the calculation of the probability to accept a move to a suboptimal solution.

Table IV: Classification of Mutation Faults

Type	Size	Never Detected
Equivalent	10	N/A
Configurational	22	4
Control-flow	40	5
Miscalculation	24	0

Table IV shows the result of manual classification of the 96 mutation faults studied. The largest group is the control-flow faults, which includes the 22 mutation faults that lead to infinite loops. However, what is interesting is that significant portion of the mutation faults can be also thought of as configurational faults, e.g. wrong initial temperatures or inappropriate cooling scheme. Whether these configurational faults will be propagated to the outcome of the optimisation largely depends on the characteristics of the specific problem instance and the algorithm. If the inappropriate configuration for the algorithm is still *good enough* for the problem instance, it would not be propagated and the metamorphic testing would not work. There were 9 mutation faults that were not killed in any of the problem instances. Out of those

Table III: Details of the results shown in Figure 1

Type	$D_1 + B_{low}$	$D_1 + B_{mid}$	$D_1 + B_{high}$	$D_2 + B_{low}$	$D_2 + B_{mid}$	$D_2 + B_{high}$	$D_3 + B_{low}$	$D_3 + B_{mid}$	$D_3 + B_{high}$
TP	24 (25.00%)	28 (29.17%)	31 (32.29%)	37 (38.54%)	29 (30.21%)	46 (47.92%)	40 (41.67%)	39 (40.63%)	50 (52.08%)
TN	10 (10.42%)	10 (10.42%)	9 (9.38%)	10 (10.42%)	9 (9.38%)	6 (6.25%)	10 (10.42%)	7 (7.29%)	8 (8.33%)
FP	0 (0.00%)	0 (0.00%)	1 (1.04%)	0 (0.00%)	1 (1.04%)	4 (4.17%)	0 (0.00%)	3 (3.13%)	2 (2.08%)
FN	62 (64.58%)	58 (60.42%)	55 (57.29%)	49 (51.04%)	57 (59.38%)	40 (41.67%)	46 (47.92%)	47 (48.96%)	36 (37.50%)

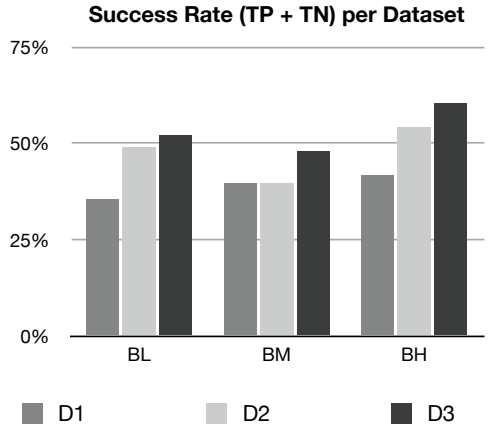

Figure 2: Comparison of success rates (TP + TN) between different datasets for the same budget setting. The real-world data (D_1) shows the lowest success rate for all budget settings. The success rates for D_3 are higher than those of D_2.

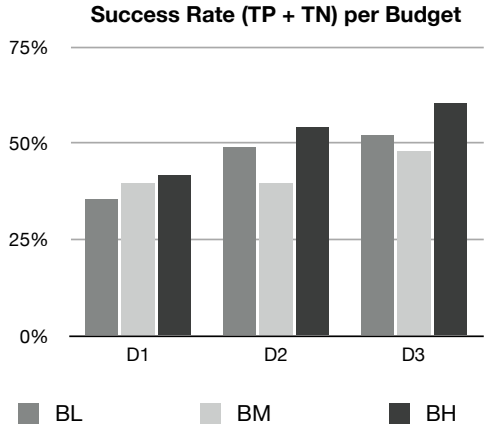

Figure 3: Comparison of success rates (TP + TN) between different budget settings for the same dataset. The results from D_2 and D_3 confirms **H2** that SMT approach will perform least effectively against the problem instances based on B_{mid} budget setting.

9, there were 4 configurational faults and 5 control-flow faults.

Another interesting observation is that, for certain problem instances, the use of statistical hypothesis test was not effective. For example, some mutant programs changed the control-flow structure of the algorithm so that the main loop of Simulated Annealing was never executed. The algorithm, therefore, returns the random initial solution that only satisfies the budget constraint. These initial solutions are randomly sampled from the search space, which may sometimes allow the statistical hypothesis test to accept the null hypothesis (i.e. no shift between mean values) even when the algorithm did not actually converge at all.

C. Problem Instances

Section IV-C described two controlled properties that led to the generation of 9 different problem instances: distribution of scores and costs (D_1, D_2 and D_3) and budget (B_{low}, B_{mid} and B_{high}). Figure 2 and 3 shows the impact these factors have on the success rate, i.e. the combined ratio of TP and TN. Figure 2 shows that D_3 shows higher success rates than D_2 across all budget settings. Similarly, Figure 3 shows that B_{mid} budget setting produced the lowest success rates for all datasets, confirming the conjecture on the difficulty of each problem made in Section IV-C. Overall, the results show that the choice of problem instances can affect the effectiveness of the statistical metamorphic testing. This answers **RQ3**; different problem instances bring about non-negligible changes in the performance of the statistical metamorphic testing.

One particularly interesting observation is that problem instances based on D_1, i.e. the real-world requirement dataset, shows different behaviour from other, randomly generated datasets in both Figure 2 and 3. The success rate for D_1 is consistently lower than that of other problem instances, suggesting a level of complexity that has not been reproduced by the random generation of datasets. Analysing the source of the complexity in the real-world dataset lies beyond the scope of this paper. However, one potential explanation is that, in the real-world dataset, there may exist not only a certain level of cost-score correlation between the cost and the score but also some outliers to the correlation, which in turn may create additional subtlety and complexity to the search landscape.

While the effect that different problem instances have on the performance of SMT approach has been shown in the result, providing a generalised explanation on why and how they affect the performance would not be an easy task. One general principle could be that the problem instances for

metamorphic testing should be neither too easy nor too hard. In metamorphic testing approach, a potential fault is defined by the difference observed between the outcomes of two test cases that are connected by a metamorphic relation. If the problem instance is too easy, it may not fail to discern the difficulties because even a slightly faulty (and, therefore, sub-optimal) optimisation algorithm may still succeed in solving the problem. However, if the problem instance is too hard, the differences between a correct optimisation algorithm and the faulty one will be again blurred because neither of them will succeed in solving the problem.

It may not be clear how to define the relative *difficulty* of a class of optimisation problem in general, let alone precisely control it. The issue of the appropriate level of difficulty would require both theoretical and empirical further studies. The practical message here is that it would be more advisable to use multiple problem instances in order to avoid over-fitting the metamorphic testing to a single problem instance.

D. Significance Level

To answer **RQ4**, the results from the statistical metamorphic testing have been analysed using different statistical significance level. Table V shows the change in the number of True Positive diagnoses as different confidence levels ranging from 91% interval ($\alpha = 0.09$) to 99% interval ($\alpha = 0.01$) are applied. A higher confidence interval would mean that it becomes harder to accept the alternative hypothesis, i.e. harder to kill the mutant program. Naturally, TP values monotonically decrease in all problem instances.

Normally, higher confidence levels mean more precise statistical hypothesis test. However, a too strict hypothesis test can actually have a negative impact on the performance of the statistical metamorphic testing. For example, 99% confidence level for $D_1 + B_{low}$ results in only 22 True Positive cases, all of which are due to timeout (refer to Section IV-A). On the other hand, a too low confidence level may introduce more False Positives. From the results, it shows that the impact of different confidence levels vary between problem instances. In practice, it would be advisable to try different confidence levels and observe the change in the effectiveness of SMT approach.

Table V: Impact of Confidence Level on Effectiveness (TP)

Problem Instance	Significance Level(α)				
	0.09	0.7	0.5	0.3	0.1
$D_1 + B_{low}$	28	25	24	23	22
$D_1 + B_{mid}$	28	28	28	27	25
$D_1 + B_{high}$	33	31	31	27	26
$D_2 + B_{low}$	42	40	40	40	40
$D_2 + B_{mid}$	42	40	39	37	35
$D_2 + B_{high}$	52	50	50	48	46
$D_3 + B_{low}$	39	38	37	37	36
$D_3 + B_{mid}$	32	32	29	27	24
$D_3 + B_{high}$	48	47	46	44	43

E. Threats to Validity

Threats to internal validity concern the factors that could have affected the empirical study in the paper. The accuracy of the proposed testing approach depends heavily on the accuracy of the statistical hypothesis test used. Since the sample consists of results of optimisation algorithms (that are trying to converge to an optimal solution, assuming that the implementation is correct), it may be said that its probabilistic distribution would be contrived, which in turn can make it difficult to choose the most suitable statistical hypothesis test for the task of comparing these samples. While there may exist other sophisticated statistical analyses that may be more suitable for the proposed approach, Wilcoxon's rank-sum test is still the most widely used non-parametric statistical testing method when the distribution of samples is not known. Another potential source of errors is the floating point precision error; it can also add up to incorrect outcomes. Interestingly, the better the algorithm performs and converges to a very small range of values, the more the precision errors may matter. In the empirical study, all values were handled with double precision and statistical analysis was performed using `JavaStatSoft`, an open-source statistical library for Java programming language [23].

Threats to external validity concern the factors that may prevent the generalisation of the empirical study in the paper. The empirical study only considered one class of optimisation problem (0-1 knapsack problem) using one optimisation algorithm (Simulated Annealing). However, Simulated Annealing was an ideal candidate because the systematic parameter tuning allowed the exclusion of the impact from incorrect parameters. While the permutation-based metamorphic relation is a sound one, it may not be *strong enough* for Simulated Annealing let alone be appropriate for other stochastic optimisation algorithms. Generalising the impact of different problem instances would require a further study involving wider range of problem instances.

Threats to construct validity arise when the measurements used in the experiment do not capture the concepts they represent. In the empirical work, the comparison of solutions from problem instances with metamorphic relation was only performed using their fitness values and not the actual solution they represent, i.e. the choice of software features. While the decision whether to treat different sets of features with the same fitness value as being identical lies beyond the scope of this paper, the discrepancy between the fitness values and the actual solutions should be noted.

VI. Related Work

Chen et al. introduced metamorphic testing as a way of generating a *follow-up* test case without a-priori knowledge of the test oracle [12, 24]. Numerical computation is an area to which the idea of metamorphic relation applies most naturally [14, 25]; however, it also has been applied to Service-

oriented Architecture [26] and machine learning [13]. Guderlei and Mayer extended the metamorphic testing approach to testing nondeterministic programs, resulting in statistical metamorphic testing [14]. In the context of SBSE, McMinn presented an approach towards the generation of pseudo-oracle using program transformation [27]. However, this paper presents the first attempt to apply the statistical metamorphic testing to stochastic optimisation algorithms and consider the impact of different problem instances.

The empirical study uses mutation testing to create artificial faults. Mutation testing is a systematic way of injecting faults into a program by making small syntactic changes [21]. The idea is that the tester should aim to detect as many mutation faults as possible, thereby enhancing the quality of the test data. While it serves as a testing methodology on its own, injecting mutation fault has been widely used to evaluate other testing methodologies by simulating faults [28–30].

Next Release Problem (NRP) is a problem in Requirements Engineering, also known as Release Planning, that aims to determine the optimal subset of software features for the release of the next version [15]. Search-based approach have been applied to the problem in different contexts including multi-objective approaches [16, 31], scheduling [32], analysis of fairness in resource distribution [33] and data sensitivity [34]. Simulated Annealing, the algorithm studied in this paper, has been proven to be very effective for solving NRP [18]. While this paper specifically consider the pair of Simulated Annealing and NRP as the subject of SMT, the same approach can be applied to any search-based software testing technique based on optimisation algorithms.

VII. Conclusion and Future Work

This paper introduces a testing approach for stochastic optimisation algorithms based on the idea of metamorphic testing and the statistical hypothesis testing. It allows the tester to come up with metamorphic test cases for optimisation algorithms, which are essentially *non-testable* programs, i.e. programs the aim of which is to determine the solution in the first place and, therefore, for which it is not trivial to derive test oracles. Given a test case, metamorphic relations can provide a follow-up test case that would function as a pseudo-oracle. The inherent stochastic nature of optimisation algorithms require statistical hypothesis test in order to confirm the metamorphic relation. The result from the empirical study showed that SMT approach can be effective for certain class of faults in optimisation algorithms. It also showed that the effectiveness of SMT approach not only depends on the algorithm and the fault but also on the problem instance that was used for the test.

Future work will include the application of more sophisticated statistical analysis for the confirmation of metamorphic relations and the identification of stronger metamorphic relations for a specific optimisation algorithm.

References

[1] J. Clark, J. J. Dolado, M. Harman, R. M. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating software engineering as a search problem," *IEE Proceedings — Software*, vol. 150, no. 3, pp. 161–175, 2003.

[2] P. McMinn, "Search-based software test data generation: A survey," *Software Testing, Verification and Reliability*, vol. 14, no. 2, pp. 105–156, Jun. 2004.

[3] O. Räihä, "A survey on search-based software design," Department of Computer Science, University of Tampere, Tech. Rep. D-2009-1, 2009. [Online]. Available: http://www.cs.uta.fi/reports/dsarja/D-2009-1.pdf

[4] M. Harman, S. A. Mansouri, and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications," Department of Computer Science, King's College London, Tech. Rep. TR-09-03, April 2009.

[5] T. Bodhuin, G. Canfora, and L. Troiano, "SORMASA: A tool for suggesting model refactoring actions by metrics-led genetic algorithm," in *Proceedings of the 1st Workshop on Refactoring Tools (WRT 2007)*. TU Berlin, 2007, pp. 23–24.

[6] V. Cortellessa, F. Marinelli, and P. Potena, "An optimization framework for "build-or-buy" decisions in software architecture," *Computers & Operations Research*, vol. 35, no. 10, pp. 3090–3106, 2008.

[7] M. C. F. P. Emer and S. R. Vergilio, "GPTesT: A testing tool based on genetic programming," in *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO '02)*. Morgan Kaufmann Publishers, 2002, pp. 1343–1350.

[8] Y. Jia and M. Harman, "Milu : A customizable, runtime-optimized higher order mutation testing tool for the full c language," in *Proceedings of the 3rd Testing: Academic and Industrial Conference - Practice and Research Techniques (TAIC PART '08)*. IEEE, 2008, pp. 94–98.

[9] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner, "Bunch: A clustering tool for the recovery and maintenance of software system structures," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM '99)*. IEEE, 1999, pp. 50–59.

[10] J. C. B. Ribeiro, M. Zenha-Rela, and F. F. de Vega, "eCrash: a Framework for Performing Evolutionary Testing on Third-Party Java Components," in *Proceedings of Jornadas sobre Algoritmos Evolutivosy Metaheuristicas (JAEM '07)*, 2007, pp. 137–144.

[11] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, Nov.

1982.

[12] T. Y. Chen, F.-C. Kuo, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing and beyond," in *Proceedings of the International Workshop on Software Technology and Engineering Practice (STEP 2003)*, September 2004, pp. 94–100.

[13] C. Murphy, K. Shen, and G. Kaiser, "Automatic system testing of programs without test oracles," in *Proceedings of the eighteenth International Symposium on Software Testing and Analysis (ISSTA 2009)*. New York, NY, USA: ACM Press, 2009, pp. 189–200.

[14] R. Guderlei and J. Mayer, "Statistical metamorphic testing testing programs with random output by means of statistical hypothesis tests and metamorphic testing," in *Proceedings of the 7th International Conference on Quality Software (QSIC 2007)*, October 2007, pp. 404–409.

[15] A. Bagnall, V. Rayward-Smith, and I. Whittley, "The next release problem," *Information and Software Technology*, vol. 43, no. 14, pp. 883–890, Dec. 2001.

[16] Y. Zhang, M. Harman, and S. A. Mansouri, "The Multi-Objective Next Release Problem," in *GECCO '07: Proceedings of the 2007 Genetic and Evolutionary Computation Conference*. ACM Press, 2007, pp. 1129–1136.

[17] R. Karp, "Reducibility among combinatorial problems," *Complexity of Computer Computations*, 1972.

[18] P. Baker, M. Harman, K. Steinhöfel, and A. Skaliotis, "Software component ranking as feature subset selection," in *Genetic and Evolutionary Computation – GECCO-2006*, Seattle, Washington, USA, Jul. 2006.

[19] JavaNCSS: A source measurement suite for java (http://www.kclee.de/clemens/java/javancss).

[20] K. Steinhöfel, A. Albrecht, and C. K. Wong, "Two simulated annealing-based heuristics for the job shop scheduling problem," *European Journal of Operational Research*, vol. 118, no. 3, pp. 524 – 548, 1999.

[21] T. A. Budd, "Mutation analysis of program test data," Ph.D. dissertation, Yale University, New Haven, CT, USA, 1980.

[22] Y. S. Ma, J. Offutt, and Y. R. Kwon, "muJava: An Automated Class Mutation System," *Software Testing, Verification, and Reliability*, vol. 15, no. 2, pp. 97–133, 2005.

[23] Javastatsoft: an open source statistics library for Java (`http://www2.thu.edu.tw/~wenwei/`).

[24] T. Chen, S. Cheung, and S. Yiu, "Metamorphic testing: A new approach for generating next test cases," Department of Computer Science, Hong Kong University of Science and Technology, Tech. Rep. HKUST-CS98-01, 1998.

[25] T. Chen, J. Feng, and T. Tse, "Metamorphic testing of programs on partial differential equations: a case study," in *Proceedings of the 26th International Computer Software and Applications Conference (COMPSAC 2002)*, 2002, pp. 327–333.

[26] W. Chan, S. Cheung, and K. Leung, "Towards a metamorphic testing methodology for service-oriented software applications," in *Proceedings of the 5th International Conference on Quality Software (QSIC 2005)*, September 2005, pp. 470–476.

[27] P. McMinn, "Search-based failure discovery using testability transformations to generate pseudo-oracles," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2009)*. ACM Press, 2009, pp. 1689–1696.

[28] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" in *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*. ACM Press, May 2005, pp. 402–411.

[29] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," *IEEE Transactions on Software Engineering*, vol. 32, no. 9, pp. 733–752, 2006.

[30] S.-S. Hou, L. Zhang, T. Xie, H. Mei, and J.-S. Sun, "Applying interface-contract mutation in regression testing of component-based software," in *Proc. 23rd IEEE International Conference on Software Maintenance (ICSM 2007)*. IEEE Computer Society Press, October 2007, pp. 174–183.

[31] J. J. Durillo, Y. Zhang, E. Alba, and A. J. Nebro, "A study of the multi-objective next release problem," in *Proceedings of the 1st International Symposium on Search Based Software Engineering (SSBSE '09)*. Cumberland Lodge, Windsor, UK: IEEE Computer Society, 13-15 May 2009, pp. 49–58.

[32] C. Li, M. van den Akker, S. Brinkkemper, and G. Diepen, "Integrated requirement selection and scheduling for the release planning of a software product," in *Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality (RefsQ '07)*, vol. 4542. Trondheim, Norway: Springer, 11-12 June 2007, pp. 93–108.

[33] A. Finkelstein, M. Harman, S. A. Mansouri, J. Ren, and Y. Zhang, "A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making," *Requirements Engineering Journal (RE '08 Special Issue)*, vol. 14, no. 4, pp. 231–245, December 2009.

[34] M. Harman, J. Krinke, J. Ren, and S. Yoo, "Search based data sensitivity analysis applied to requirement engineering," in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09)*. Montreal, Canada: ACM, 8-12 July 2009, pp. 1681–1688.