A Novel Mask-Coding Representation for Set Cover Problems with Applications in Test Suite Minimisation

Shin Yoo
Centre for Research on Search, Evolution and Testing
King's College London
London, UK
Email: shin.yoo@kcl.ac.uk

***Abstract*—Multi-Objective Set Cover problem forms the basis of many optimisation problems in software testing because the concept of code coverage is based on the set theory. This paper presents Mask-Coding, a novel representation of solutions for set cover optimisation problems that explores the problem space rather than the solution space. The new representation is empirically evaluated with set cover problems formulated from real code coverage data. The results show that Mask-Coding representation can improve both the convergence and diversity of the Pareto-efficient solution set of the multi-objective set cover optimisation.**

***Keywords*-set-cover representation; search-based software engineering; test suite minimisation**

I. Introduction

Multi-Objective Set cover Problem forms an important basis for optimisation problems in software testing. The set cover problem itself is essential to software testing because the concept of code coverage is based on the set theory. For example, the problem of reducing redundancy in test suite can be formulated as a set cover problem [1]–[3]. Multi-Objective approach to set cover problem highlights the trade-offs between the coverage and the cost (often execution time of test cases). Instead of obtaining the maximum set cover, the Multi-Objective Set Cover aims to identify all the maximum coverage possible values for any given budget [4], [5]. While the traditional greedy heuristic is very effective to single-objective set cover problems, its inability to cope with additional objectives facilitated the use of Multi-Objective Evolutionary Algorithms (MOEAs) [5], [6].

Two important factors when applying a meta-heuristic algorithm to a software engineering problem is the fitness function and the representation of a candidate solution [7]. It is known that the choice of the representation of a solution can have a significant impact on the performance of a meta-heuristic algorithm [8], [9]. For example, Rothlauf et al. report that the use of Random NetKey representation reduced the distance to the optimal solution by almost 13% compared to the conventional Characteristic Vector (CV) representation for network design problem [9].

The *de-facto* standard representation for set cover problem is to encode the selection of subsets as a binary string: the ith digit of the binary string is 1 if the test case t_i is included in the solution and 0 otherwise. Following this, the neighbouring solutions for local search algorithms are often defined as solutions with a single digit different from the original solution. Genetic operators work similarly on the binary string representation; cross-over mixes the choice of test cases, while a single bit-flip mutation adds or subtracts a test case. While the bit-string representation is innocuous in itself, it may not be the ideal representation for the set cover problems that deal with code coverage. The empirical study presents some evidence that the bit-string representation may actually be sub-optimal for Multi-Objective Set Cover problem based on code coverage data.

This paper presents a novel representation for set cover problems called Mask-Coding representation. The main idea behind the new representation is to replace the *solution* space with the *problem* space, following the approach of Storer et al. to Job-Shop Scheduling Problem [10]. Mask-Coding representation still uses binary strings, but an instance of Mask-Coding representation would denote an alternative set cover problem in the problem space. The evaluation of this instance would require an efficient and effective domain specific construction heuristic: in case of set cover problem, this would be the greedy algorithm. The solution to the alternative problem, obtained by the construction heuristic, is then evaluated against the original problem.

One potential strength of problem space exploration is that it can be free of the challenges that arise from the features of the solution search landscape, such as a large plateau. The empirical evaluation of the new representation on widely studied code coverage data shows that it can indeed improve the performance of multi-objective meta-heuristic algorithms both in terms of convergence and diversity of the resulting Pareto-front.

The contributions of this paper are as follows:

1) This paper introduces a novel representation for set cover problems, Mask-Coding representation, based on the idea of problem space exploration.
2) The paper presents empirical evidence that the widely used binary string representation may not be ideal for set cover problems based on code coverage data.

3) The paper presents an empirical evaluation of Mask-Coding representation using multi-objective test suite minimisation problems with real-world coverage data. The results show that the new representation can improve both the convergence and diversity of the results. In the context of software testing, this means more efficient regression testing with more insightful cost-benefit analysis of regression test suites.

The rest of the paper is organised as follows: Section II describes the problems in traditional binary string representation with empirical evidence. Section III introduces Mask-Coding representation and the idea of problem space exploration. Section IV presents the research questions and describes the settings for the empirical study based on test suite minimisation problems, the result of which is discussed in Section V. Section VI presents related work and Section VII concludes.

II. Problems in Traditional Binary String Representation

A. Set Cover Problem

Single-objective set cover problem is NP-hard [11] and can be described as follows: given several sets that share some common elements, the goal is to select the minimum number of these sets so that the selected sets contain all the elements that are contained in any of the input sets. More formally,

Set Cover Optimisation Problem
Given a universe $\mathcal{U}$ of n elements and a family $\mathcal{S}$ of m subsets of $\mathcal{U}$, a *cover* is a subfamily $\mathcal{C} \subseteq \mathcal{U}$ whose union is equal to $\mathcal{U}$. The problem is to find a cover of $\mathcal{U}$ that uses the fewest sets.

It may not be possible to cover $\mathcal{U}$ completely. For example, assume that $\mathcal{U}$ is the set of all program statements in SUT (System Under Test) and $\mathcal{S}$ is the collection of execution traces of test cases: any unreachable code in SUT will not be covered by any combination of traces in $\mathcal{S}$. In this case, the goal of set cover optimisation becomes to achieve the highest coverage possible with the fewest sets. Coverage is defined as the ratio between the size of the cover and the size of $\mathcal{U}$, i.e.:

$$\text{coverage}(\mathcal{C}) = \frac{|\bigcup_{S_i \in C} S_i|}{|\mathcal{U}|}$$

Multi-Objective Set Cover optimisation assigns cost to each set in $\mathcal{S}$ and adopts an additional objective to actively minimise the cost. More formally,

Multi-Objective Set Cover Optimisation Problem
Given a universe $\mathcal{U}$, a family $\mathcal{S}$ of subsets of $\mathcal{U}$ and a cost function $cost : \mathcal{S} \rightarrow \mathbb{R}$, the problem is to find a cover $\mathcal{C}$ that maximises coverage($\mathcal{C}$) and minimises $\sum_{S_i \in C} cost(S_i)$.

While the definition of the multi-objective formulation appears similar to that of the single-objective set cover problem, the Pareto-optimisation [12] of both objectives shows the trade-off between coverage and cost of the set cover, which has application to software testing [5].

B. Binary String Representation and Dimensional Plateau

When applying meta-heuristic optimisation to set cover problem, the most commonly used representation of an individual solution is the binary string representation [4]–[6]. The length of the binary string is equal to the number of subsets in family $\mathcal{S}$. If the member S_i of $\mathcal{S}$ is included in the solution, the ith digit of the binary string is 1; if it is not included, 0.

While the definition of the binary string representation is innocuous in nature, there is a specific problem that arises when it is used for set cover problems based on code coverage data. It is known that different paths in a program get executed with different frequency. For example, the initialisation code will be executed with every execution, while a procedure that deals with a very rare situation, e.g. exception handling code, will be executed less frequently. Since code coverage data represent recorded execution traces, this difference in execution frequency will be reflected in the data. More formally, this means that a significantly large number of members in family $\mathcal{S}$ (i.e. test cases) may cover similar sets of elements in $\mathcal{U}$ that take up the majority of $\mathcal{U}$.

This redundancy in coverage has an important implication for the traditional binary string representation, namely, a large plateau in the coverage dimension. Intuitively, if a large number of members in $\mathcal{S}$ covers largely similar sets of the majority of elements in $\mathcal{U}$, the chance for any mutation on an arbitrary digit i of the binary string representation to make any impact on coverage significantly decreases. This is because there is a high probability that S_j such that $i \neq j, S_j \in \mathcal{S}$ will cover the same or very similar set of elements in $\mathcal{U}$. This would result in a large plateau in the coverage dimension of the search space. This problem will be referred to as the *dimensional plateau problem*:

Dimensional Plateau: in multi-objective search landscape, if one of the objective value remains the same while the other objectives changes, this creates a dimensional plateau for the dimension of the unchanging objective.

When applied to the multi-objective test suite minimisation problem, the existence of a dimensional plateau would mean that the search algorithm may fail to find any solutions with low-coverage and low-cost. If the search algorithm fails to escape the coverage dimensional plateau, it will only optimise the cost of the subset of test cases that will achieve the maximum coverage. Reaching the part of the search

landscape where solutions with lower coverage/lower cost exist may be very challenging.

III. Mask-Coding Representation

A. Problem Space Exploration

The idea of problem space exploration was first introduced by Storer et al. in order to design a new neighbourhood definition for stochastic local search for sequencing problems [10]. A similar idea was also introduced under the name of 'noising method' by Charon et al [13]. The key idea lies in the observation that often there exists a fast and deterministic heuristic for many combinatorial optimisation problems. Given such a heuristic h, and a problem instance p, it is possible to calculate a solution s very efficiently and, therefore, the pair (h, p) can be read as an encoding for a solution $s = h(p)$. By *perturbing* the heuristic h, the problem p, or both, a subset of solutions can be generated, which forms the neighbourhood for the local search. The *representation* used by the local search is not an encoding of the solution for the original problem p, but an encoding of the *perturbation* d. The problem p can be perturbed by changing the original problem data, whereas the heuristic h can be perturbed by changing its configuration. In the context of the paper, let us focus on the problem perturbation.

The problem space exploration can be powerful when the search landscape in the solution space for the original problem p presents challenges such as a large plateau. Problem space exploration can be used for any class of problems for which there exists a fast and deterministic construction heuristic. Since Storer et al. demonstrated the idea with Job Shop Scheduling Problem, it has been successfully applied to various combinatorial optimisation problems including 0-1 Multiple Knapsack Problem [14], Graph Partitioning Problem [15], Routing Problem [16] and Travelling Salesman Problem [17].

B. Mask-Coding Representation

The representation for problem perturbation is sometimes called 'Weight-Coding' because the perturbation is represented by a vector of weights that is applied to the original problem data [14], [16]. For example, the perturbation vector for a 0-1 knapsack problem would be a collection of weights that will be multiplied to the value of each item in the 0-1 Knapsack Problem.

For set cover problems, a vector of real numbers is not suitable for perturbation as the data consist of sets. Mask-Coding representation introduced in the paper uses bit-masking to perturb either the universe, $\mathcal{U}$, or the family of subsets, $\mathcal{S}$, or both. A genotype representation of a solution encoded with Mask-Coding would still be a binary string, but it does not depict the selection of members in $\mathcal{S}$ as in the traditional binary string representation. Depending on where the masking is applied, there are three different ways to apply Mask-Coding representation to the genotype representation for multi-objective set cover problem: $\mathcal{U}$-mask, $\mathcal{S}$-mask and $\mathcal{US}$-mask.

1) $\mathcal{U}$-Mask Representation: An $\mathcal{U}$-mask perturbs the original problem p by masking a subset of elements in $\mathcal{U}$. An instance of $\mathcal{U}$-mask representation is a binary string, $d = d_1 d_2 \dots d_n$, whose length equals the size of the original universe, $\mathcal{U}$. Without losing generality, let $\mathcal{U}$ be an ordered set with n elements, $\{e_1, \dots, e_n\}$. The perturbed (i.e. masked) universe, $\mathcal{U}^d$ only contains elements e_i such that $d_i = 1$. More formally,

$$\mathcal{U}^d = \mathcal{U} - \{e_i | d_i = 0\}$$

Similarly, the subsets of $\mathcal{U}$ in $\mathcal{S}$ are also masked using d:

$$\mathcal{S}^d = \{S_j^d | \forall S_j \in \mathcal{S}, S_j^d = S_j - \{e_i | d_i = 0\}\}$$

The pair of $(\mathcal{U}^d, \mathcal{S}^d)$ denotes the perturbed problem. Let x be the traditional binary string representation of the solution to the perturbed problem $(\mathcal{U}^d, \mathcal{S}^d)$, which is obtained using a construction heuristic h. It follows that x can also be a solution to the original problem $(\mathcal{U}, \mathcal{S})$ because the length of x remains equal to $|\mathcal{S}|$ and is irrelevant to neither the length nor the cardinality of d. Therefore, it is possible to measure coverage or cost of the set cover expressed with x using the original problem data, $(\mathcal{U}, \mathcal{S})$. Algorithm 1 illustrates the process of measuring coverage and cost of a solution encoded with $\mathcal{U}$-mask representation using the greedy algorithm as the construction heuristic (see Section IV-D for details of the construction heuristic).

Algorithm 1: Fitness evaluation for Multi-Objective Set Cover optimisation using $\mathcal{U}$ Mask-Coding representation and greedy heuristic

Input: the original universe, $\mathcal{U}$, the original family of subsets, $\mathcal{S}$, a solution encoded with $\mathcal{S}$-mask, d

Output: a coverage of d, $coverage_d$, and a cost of d, $cost_d$

FitnessEvaluationForUMask($\mathcal{U}$, $\mathcal{S}$, d)

(1) $\mathcal{U}^d = \mathcal{U} - \{e_i | d_i = 0\}$
(2) $\mathcal{S}^d = \{S_j^d | \forall S_j \in \mathcal{S}, S_j^d = S_j - \{e_i | d_i = 0\}\}$
(3) $x \leftarrow greedy(\mathcal{U}^d, \mathcal{S}^d)$
(4) $coverage_d \leftarrow coverage(x, \mathcal{U}, \mathcal{S})$
(5) $cost_d \leftarrow cost(x, \mathcal{U}, \mathcal{S})$
(6) **return** $coverage_d$, $cost_d$

2) $\mathcal{S}$-Mask Representation: An $\mathcal{S}$-mask perturbs the original problem by masking a subset of family members in $\mathcal{S}$. An instance of $\mathcal{S}$-mask is a binary string, $d = d_1 d_2 \dots d_m$, whose length equals the size of the original $\mathcal{S}$. Without losing generality, let $\mathcal{S}$ be an ordered set with m subsets of $\mathcal{U}$, $\{S_1, \dots, S_m\}$. After perturbation, the original $\mathcal{U}$ remains the same. However, the perturbed $\mathcal{S}$ is defined as follows:

$$\mathcal{S}^d = \mathcal{S} - \{S_j | d_j = 1\}$$

The pair of $(\mathcal{U}, \mathcal{S}^d)$ forms the perturbed problem. Unlike $\mathcal{U}$-mask, the solution x of the perturbed problem cannot be

accepted as a solution to the original problem as the length of x would be equal to the size of $\mathcal{S}^d$ rather than $\mathcal{S}$. Therefore, the solution x to $(\mathcal{U}, \mathcal{S}^d)$ needs to be decoded into a solution $x' = x'_1 \dots x'_m$ for the original problem $(\mathcal{U}, \mathcal{S})$. Each digit of x' is defined as follows w.r.t. the mask d:

$$x'_i = \begin{cases} 1 \text{ if } d_j = 1 \text{ and } S_j \text{ is selected by } x \\ 0 \text{ otherwise} \end{cases}$$

Algorithm 2 shows the process of measuring coverage and cost of a solution encoded with $\mathcal{S}$-mask representation using greedy algorithm as the construction heuristic.

Algorithm 2: Fitness evaluation for Multi-Objective Set Cover optimisation using $\mathcal{S}$ Mask-Coding representation and greedy heuristic

Input: the original universe, $\mathcal{U}$, the original family of subsets, $\mathcal{S}$, a solution encoded with $\mathcal{S}$-mask, d

Output: a coverage of d, $coverage_d$, and a cost of d, $cost_d$

FITNESSEVALUATIONFORSMASK($\mathcal{U}$, $\mathcal{S}$, d)

(1) $\mathcal{S}^d = \mathcal{S} - \{S_j | d_j = 1\}$
(2) $x \leftarrow greedy(\mathcal{U}, \mathcal{S}^d)$
(3) $x' \leftarrow decode(x, d)$
(4) $coverage_d \leftarrow coverage(x', \mathcal{U}, \mathcal{S})$
(5) $cost_d \leftarrow cost(x', \mathcal{U}, \mathcal{S})$
(6) **return** $coverage_d$, $cost_d$

3) $\mathcal{US}$-Mask Representation: It is also possible to perturb both $\mathcal{U}$ and $\mathcal{S}$ simultaneously. An instance of $\mathcal{US}$-mask is a binary string of length $(n+m)$, $d = d_1 d_2 \dots d_{n+m}$. The first n digits of d form the $\mathcal{U}$ mask, du, whereas the following m digits form the $\mathcal{S}$ mask, ds. The perturbation of $\mathcal{U}$ remains the same as in the case of $\mathcal{U}$ masking:

$$\mathcal{U}^{du} = \mathcal{U} - \{e_i | du_i = 0\}$$

However, the perturbation of $\mathcal{S}$ requires a different approach. First, the members of $\mathcal{S}$ that are masked by ds should be removed from $\mathcal{S}^{ds}$. Second, the masked elements of $\mathcal{U}$ should be also masked in each member of $\mathcal{S}^{ds}$. More formally,

$$\mathcal{S}^{ds} = \{S_j^{ds} | \forall S_j \in \mathcal{S} - \{S_j | ds_j = 1\}, \\ S_j^{ds} = S_j - \{e_i | du_i = 0\}\}$$

Since $\mathcal{S}$ has also been perturbed, the solution x from greedy algorithm needs to be decoded following the description in Section III-B2.

IV. EXPERIMENTAL SET-UP

A. Research Questions

The aim of the empirical study is to evaluate the impact of Mask-Coding representation on the optimisation of set cover problems based on code coverage data. The empirical study compares 4 different representations for Multi-Objective Set Cover: the traditional binary string representation, the $\mathcal{U}$-mask representation, the $\mathcal{S}$-mask representation and the $\mathcal{US}$-mask representation. In comparing these representations, the paper asks the following research questions:

- **RQ1. Convergence:** how well do the solutions from each representation converge to the optimal Pareto-frontier?
- **RQ2. Diversity:** how diverse are the solutions from each representation?
- **RQ3. Efficiency:** what is the impact of using Mask-Coding representation on the running time of the algorithm?

RQ1 and **RQ2** are answered by analysing the Pareto-fronts produced by different representations. Ideal measurement of convergence and diversity would require the knowledge of the true Pareto-fronts. Since it is not available, reference Pareto-fronts are formed by combining all the available results. **RQ3** concerns the additional computation resource required when using Mask-Coding representation, i.e. that of greedy algorithm. It is answered by measuring the execution time of each representation.

B. Subjects

Table I shows the subject test suites used in the empirical study. The test suites are obtained from Software Infrastructure Repository [18]. The set cover problem is instantiated with statement coverage data. That is, the universe $\mathcal{U}$ corresponds to the set of all statements in programs. In turn, the family of subsets $\mathcal{S}$ corresponds to the set of all execution traces of all the test cases in test suites. Two different types of test suites were deliberately chosen: ones with a small $\mathcal{U}$ and a large $\mathcal{S}$ (`printtokens` and `tcas`) and ones with a large $\mathcal{U}$ and a small $\mathcal{S}$ (`flex` and `gzip`). The level of redundancy in $\mathcal{S}$ (i.e. test suites) is much higher in the test suites that belong to the first class than the second class. Note that, while the test suites in the first class have already been studied for multi-objective test suite minimisation [5], only smaller subsets of the entire test suite have been considered. This paper deliberately uses the entire pool of test cases in order to force the high level of redundancy.

Table I
SIZES OF FOUR SUBJECT TEST SUITES OBTAINED FROM SIR

Subject	No. of statements	Test Suite Size
`printtokens`	189	4,115
`tcas`	65	1,608
`flex`	3,965	103
`gzip`	2,007	213

The coverage for each test has been measured using `gcov`, a widely used code profiling tool from the `gcc` compiler suite. The cost of executing each test has been measured using `valgrind` profiling tool [19]; for the

execution of each test, the number of CPU instructions has been measured and used as the execution cost.

While the empirical study is based on code coverage data, its aim is to analyse the impact of Mask-Coding representation rather than to show their benefits in the context of software testing. Therefore, the impact of Multi-Objective Test Suite Minimisation on fault detection capability lies beyond the interest of this paper and will not be considered.

C. MOEA Algorithm

The four representations are evaluated using a Multi-Objective Evolutionary Algorithm (MOEA) called Two-Archive Algorithm [20]. Two Archive Algorithm maintains two separate memorisation archive for convergence and diversity respectively. Algorithm 3 shows the high-level outline of Two-Archive Algorithm.

The key idea behind Two-Archive Algorithm lies in the algorithm that collects non-dominated solutions to two separate archives. A non-dominated solution from the population is first compared to both archives. If the new solution is dominated by a solution from archives, it is discarded. If the new solution is not dominated, there are two possibilities: 1) the new solution dominates a solution in archives, in which case the dominated solution is removed from the archive and the new solution is added to the convergence archive, and 2) the new solution is not dominated by and does not dominate any solution in archives, in which case the new solution is added to the diversity archive. In order to control the size of the archive, solutions are removed from the diversity archive when the size goes over a predefined limit: the solution in diversity archive that has the shortest distance to any solution in the convergence archive is removed. For more details, readers are encouraged to refer to Praditwong and Yao [20].

Algorithm 3: Outline of Two-Archive Algorithm

(1) Initialise the population
(2) Initialise archives to the empty set
(3) Evaluate initial population
(4) **while** stopping criterion is not met
(5) Collect non-dominated individuals to archives
(6) Select parents from archives
(7) Generate a new population from parents
(8) Evaluate the new population

When using Mask-Coding, the individual solutions in the population represent the masking, i.e. the input d of Algorithm 1-**??**, rather than the actual solution. The selection operator for Two-Archive Algorithm selects two parents from both archives with uniform probability distribution. It also uses the standard single-point crossover operator with the crossover rate of 0.9 and the single bit-flip mutation. The population size was set to 100. The stopping criterion was set to the maximum of 25,000 fitness evaluations.

D. Construction Heuristic

Fitness evaluation using problem space exploration requires an efficient and effective construction heuristic. For set cover, the greedy algorithm is known to produce results that are within $\ln n$ of the optimal cost [21]. Algorithm 4 describes the additional greedy algorithm used as the construction heuristic in the empirical study.

Algorithm 4: Outline of additional greedy algorithm ADDITIONALGREEDY($\mathcal{U}$, $\mathcal{S}$)

(1) $C \leftarrow \phi$ // covered elements in $\mathcal{U}$
(2) **repeat**
(3) $k \leftarrow min_k(cost_k/|S_k - C|)$
(4) add S_k to solution
(5) $C = C \bigcup S_k$
(6) **until** $C = \mathcal{U}$

E. Evaluation

In order to cater for the inherent randomness in population-based evolutionary algorithm, each experiment was repeated 30 times. The reference Pareto-fronts for convergence and diversity research questions were formed by combining solutions from all four representations and identifying a Pareto-front from the combined set of solutions, i.e. the results from 120 individual runs (4 representations, 30 runs per representation).

RQ1 and **RQ2** are answered by statistically analysing the number of solutions contributed to the reference Pareto-front by each representation. The hypothesis test is performed using t-test; while the distribution of the sample is not known, the central limit theorem dictates that the distribution approximates the normal distribution with a large enough sample size [22]. Additionally, Wilcoxon's rank-sum test, the non-parametric alternative, would not produce the precise p-value under the existence of ties, which have been frequently observed in the results.

V. RESULTS AND ANALYSIS

A. Convergence

Figure 1 shows the boxplots of the number of unique solutions contributed to the reference Pareto-fronts by each representation. The plot $u0s0$ represents neither $\mathcal{U}$ nor $\mathcal{S}$ mask, i.e. the traditional binary string representation. Respectively, $u1$ and $s1$ represent $\mathcal{U}$- and $\mathcal{S}$-mask being used.

One surprising finding is that the traditional binary string representation failed to produce any solution on the reference Pareto-front in the cases of `printtokens` and `tcas`. Additionally, even though the redundancy is not so severe in the test suite of `gzip`, the traditional binary string representation contributes very little. This shows that the traditional binary string representation may not be effective if the search landscape contains a large plateau (which, in the case of `printtokens` and `tcas`, is incurred by the high level of redundancy in the test suites).

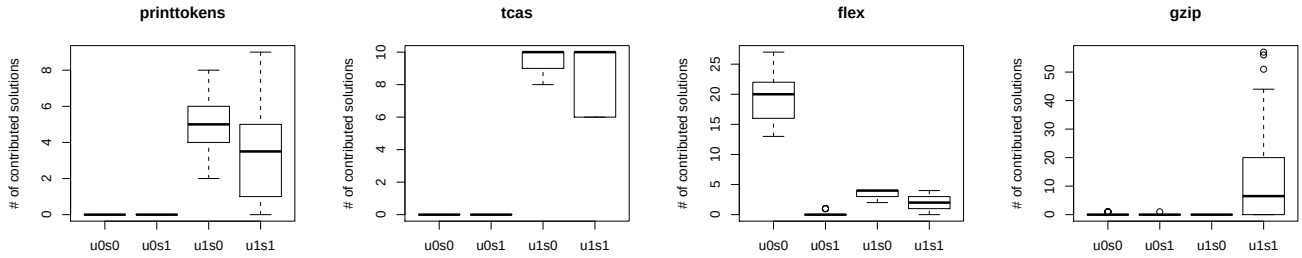

Figure 1. Number of unique solutions that are contributed to the reference Parefo-front by different representations. Boxplot $u0s0$ represents both $\mathcal{U}$ and $\mathcal{S}$ masking turned off, i.e. the traditional binary string representation, whereas $u1$ and $s1$ represent $\mathcal{U}$ and $\mathcal{S}$ masking turned on respectively. While different combinations of Mask-Coding work best for different problems, $\mathcal{US}$- and $\mathcal{U}$-mask tend to produce a larger number of non-dominated solutions for 3 out of 4 datasets.

Table II
MEAN AND STANDARD DEVIATION OF THE NUMBER OF UNIQUE SOLUTIONS CONTRIBUTED TO THE REFERENCE PARETO-FRONT

Subject	Traditional		$\mathcal{U}$-Mask		$\mathcal{S}$-Mask		$\mathcal{US}$-Mask	
	$\bar{n}$	σ	$\bar{n}$	σ	$\bar{n}$	σ	$\bar{n}$	σ
printtokens	0.0	0.0	4.83	1.55	0.0	0.0	3.57	2.63
tcas	0.0	0.0	9.60	0.55	0.0	0.0	8.80	1.83
flex	19.47	1.26	3.56	0.56	0.13	0.34	2.23	1.26
gzip	0.20	0.40	0.0	0.0	0.03	0.18	14.83	18.49

Table III
THE p-VALUES OF THE STATISTICAL HYPOTHESIS TEST BETWEEN $\bar{n}_{\mathcal{U}}$ AND $\bar{n}_{\mathcal{US}}$. THE HYPOTHESIS IS TESTED WITH ONE-TAILED t-TEST WITH THE SIGNIFICANCE LEVEL OF 95%.

Subject	p-value ($\bar{n}_{\mathcal{U}} > \bar{n}_{\mathcal{US}}$)
printtokens	0.015
tcas	0.015
flex	< 0.001
gzip	1.0

In printtokens, tcas and gzip, $\mathcal{U}$-mask tends to contribute the most to the number of solutions contributed to the reference Pareto-front. Interestingly, $\mathcal{U}$-mask seems to be more effective than $\mathcal{US}$-mask for printtokens and tcas. In all programs, $\mathcal{S}$-mask alone did not perform very well. Table II shows the statistical details of the results presented in Figure 1.

Table III shows the results of statistical hypothesis test of the data presented in Figure 1. The comparison between the traditional binary string representation and Mask-Coding representations does not require any statistical analysis: the results from the traditional representation is either almost always 0 (printtokens, tcas and gzip) or completely surpasses other representations (flex). Rather, the statistical analysis was performed to see whether $\mathcal{U}$-mask is more effective than $\mathcal{US}$-mask with statistical significance. The *null* hypothesis is that $\bar{n}_{\mathcal{U}}$ and $\bar{n}_{\mathcal{US}}$ are the same. The *alternative* hypothesis is that $\bar{n}_{\mathcal{U}}$ is greater than $\bar{n}_{\mathcal{US}}$. The hypothesis is tested with one-tailed t-test with 95% significance level.

In printtokens, tcas and flex, the null hypothesis is rejected with statistical significance, meaning that $\mathcal{U}$-mask produces more unique solutions on the reference Pareto-front than $\mathcal{US}$-mask. From Table II and Table III, **RQ1** is answered as follows: Mask-Coding representation results in higher convergence compared to the traditional binary string representation if the search landscape contains a large dimensional plateau. This claim is backed by the observation of larger number of unique solutions contributed to the reference Pareto-fronts for subjects printtokens and tcas.

B. Diversity

In order to answer **RQ2**, the number of unique solutions produced by each representation is compared statistically. Unlike Section V-A, all solutions produced by each representation are considered, regardless of whether they are on the reference Pareto-fronts or not.

Table IV
MEAN AND STANDARD DEVIATION OF THE NUMBER OF UNIQUE SOLUTIONS PRODUCED BY EACH REPRESENTATION.

Subject	Traditional		$\mathcal{U}$-Mask		$\mathcal{S}$-Mask		$\mathcal{US}$-Mask	
	$\bar{n}$	σ	$\bar{n}$	σ	$\bar{n}$	σ	$\bar{n}$	σ
prttkn	1.03	0.18	10.7	1.83	1.0	0.0	11.03	3.40
tcas	1.00	0.0	9.60	0.55	1.0	0.0	8.80	1.83
flex	26.73	2.34	8.66	1.81	26.77	6.13	41.07	8.57
gzip	1.10	0.30	16.77	3.87	47.30	8.47	126.77	21.07

Figure 2 shows the boxplots of the number of unique solutions produced by each representation. When compared to Figure 1, it can be observed that $\mathcal{US}$-mask had produced some solutions for printtokens that were dominated by the solutions produced by $\mathcal{U}$-mask. Another interesting observation is that $\mathcal{US}$-mask has produced a much larger number of solutions for flex compared to the traditional binary string representation, but most of those additional

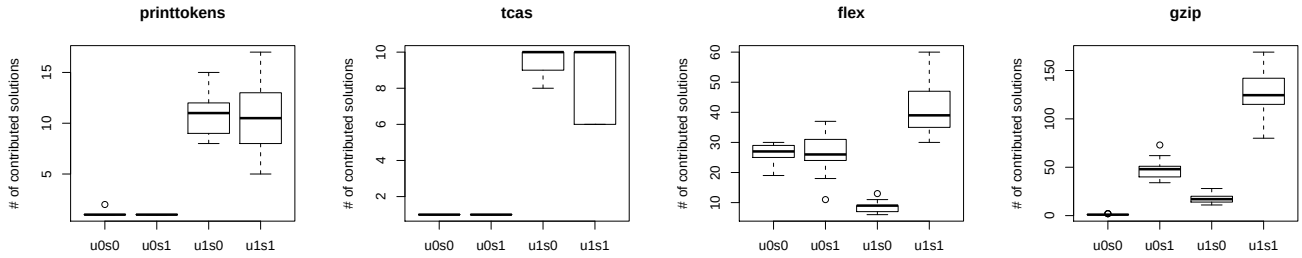

Figure 2. Number of unique solutions that are produced by different representations. Boxplot $u0s0$ represents both $\mathcal{U}$ and $\mathcal{S}$ masking turned off, i.e. the traditional binary string representation, whereas $u1$ and $s1$ represent $\mathcal{U}$ and $\mathcal{S}$ masking turned on respectively. Overall, $\mathcal{US}$-mask tends to produce a larger number solutions compared to other representations.

Table V
THE p-VALUES OF THE STATISTICAL HYPOTHESIS TEST BETWEEN $\bar{n}_{\mathcal{US}}$, $\bar{n}_{\mathcal{U}}$, $\bar{n}_{\mathcal{S}}$ AND $\bar{n}_0$ (TRADITIONAL BINARY STRING REPRESENTATION) FOR **RQ2**. THE HYPOTHESIS IS TESTED WITH ONE-TAILED t-TEST WITH THE SIGNIFICANCE LEVEL OF 95%.

Subject	p-value		
	$\bar{n}_{\mathcal{US}} > \bar{n}_0$	$\bar{n}_{\mathcal{U}} > \bar{n}_0$	$\bar{n}_{\mathcal{S}} > \bar{n}_0$
`flex`	$< 10^{-9}$	1.0	0.48
`gzip`	$< 10^{-19}$	$< 10^{-19}$	$< 10^{-19}$

solutions were at the same time dominated by the solutions produced by the binary string representation.

Since the boxplots for `printtokens` and `tcas` largely reproduce the results in Figure 1 for which the traditional binary string representation and $\mathcal{S}$-mask do not produce almost any solution at all, the statistical analysis of diversity results focuses on the cases of `flex` and `gzip`. For each representation, the *null* hypothesis is that there is no difference in the number of unique solutions produced by the traditional binary string representation and the corresponding Mask-Coding representation. The alternative hypothesis is that the corresponding Mask-Coding representation produces a larger number of unique solutions. The results from the traditional binary string representation are denoted with $\bar{n}_0$.

Table V shows the result of the statistical hypothesis test. For `flex`, the alternative hypothesis is only accepted for $\mathcal{US}$-mask at the significance level of 95%. For `gzip`, all three Mask-Coding representations produce a larger number of unique solutions compared to the traditional binary string representation.

Figure 3 shows the shape of Pareto-fronts produced by different representations in order to facilitate more qualitative analysis of the results. The plot for each representation consists of non-dominated solutions collected from the combined results of the 30 repeated runs. For `printtokens` and `tcas`, the Pareto-fronts from both $\mathcal{US}$- and $\mathcal{U}$-mask covers the widest range of solutions. In contrast, the traditional binary string representation fails to escape the dimensional plateau and produces only one solution.

With `flex` and `gzip`, it can be observed that all three types of Mask-Coding largely fail to produce solutions with low cost and low coverage. This may be explained by the differences in redundancy in test suites. Solutions with low cost and low coverage will in turn require the inclusion of test cases with extremely low cost and coverage. These test cases are more likely to represent less frequent usage pattern of the program, e.g. error handling routines, and, therefore, the proportion of such test cases in the entire test suite is likely to be small. If the test suite has a very high level of redundancy that can lead to a dimensional plateau, the probability for the masking to hide these low cost/low coverage test cases is relatively low. On the other hand, if the level of redundancy is low, the probability for the masking to hide these test cases increase. This in turn may prevent the optimisation algorithm to produce solutions with low cost and low coverage.

Overall, **RQ2** is answered as follows: if the set cover problem contains a high level of redundancy in $\mathcal{S}$ that can lead to a dimensional plateau, Mask-Coding representation can help escaping the dimensional plateau to produce a Pareto-front with high diversity.

C. Impact on Performance

Since the fitness evaluation for Mask-Coding representation involves using a separate construction heuristic, it requires additional computation resources. Figure 4 shows the boxplots of the wall-clock execution time measured for the runs of different representations. While all three Mask-Coding representations require more computational resources than the traditional binary string representation, the amount of resources additionally required differs depending on the type of masking.

For `printtokens` and `tcas`, $\mathcal{U}$- and $\mathcal{S}$-mask requires similarly large amounts of additional computational cost whereas $\mathcal{US}$-mask requires significantly less. The combined use of both types of masking has reduced the size of problem instances for the construction heuristic significantly enough

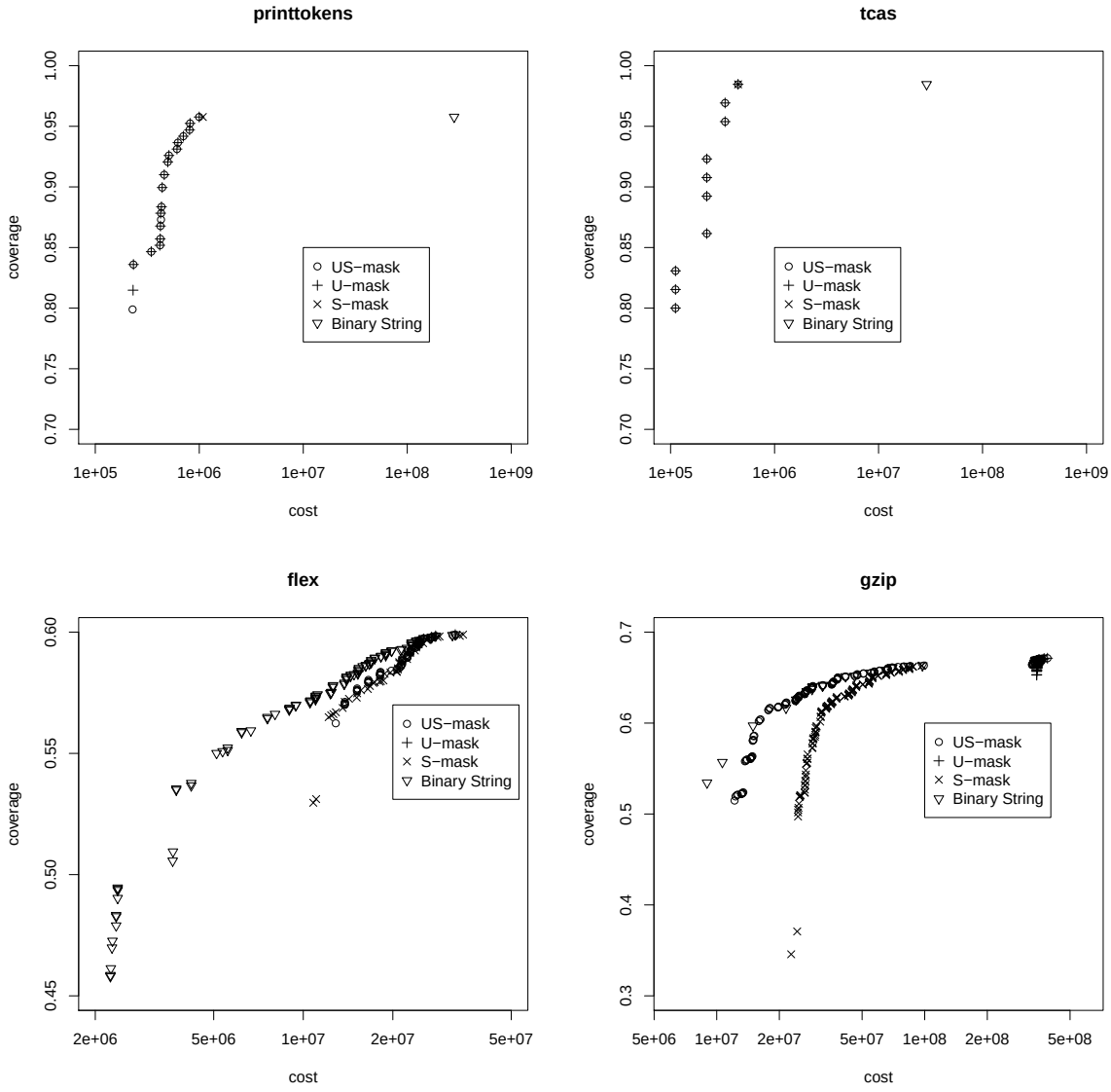

Figure 3. The shape of Pareto-fronts produced by different representations. Each Pareto-front consists of non-dominated solutions collected from the combined results of the 30 repeated runs. The x-axis is in a logarithmic scale.

to have an impact on the overall execution time. However, for `flex` and `gzip`, only $\mathcal{S}$-mask seems to make any difference in the amount of additionally required computational resources. This is probably because $|\mathcal{U}|$ is much larger than $|\mathcal{S}|$. Masking one element in $\mathcal{U}$ saves $|\mathcal{S}|$ steps for the construction heuristic, and vice versa. Therefore, if $|\mathcal{U}| \gg |\mathcal{S}|$, the impact of $\mathcal{S}$-mask is much bigger than that of $\mathcal{U}$-mask.

For all subjects, all three Mask-Coding representation require a significantly large amount of additional computation power. This partially answers **RQ4**. However, it should be noted that the data presented in Figure 4 are the measurements of execution time for fixed number of fitness evaluations. That is, the algorithms may have continued to run even after they have converged. Therefore, these data should not be read as the true cost of the use of Mask-Coding, which can be only measured by the time it took to converge to the Pareto-front. However, since the use of convergence as a stopping criterion requires the knowledge of the true Pareto-front *a priori*, here only the additional overhead of using Mask-Coding is studied and presented.

D. Threats to Validity

There are a few threats to validity regarding the generalisation of the results presented in this paper. First, most

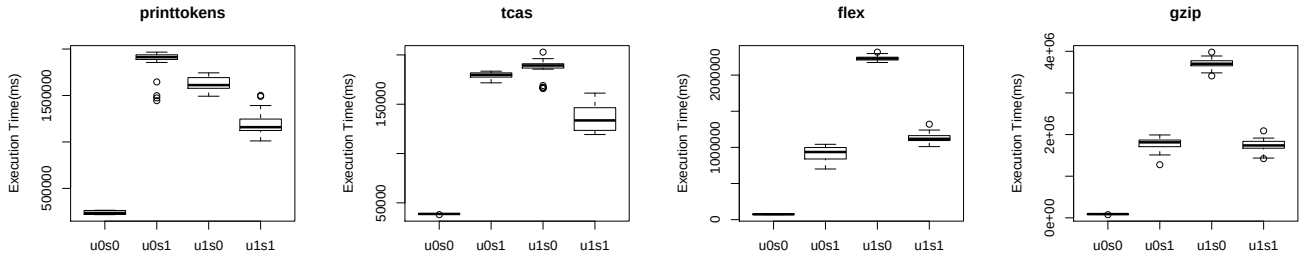

Figure 4. Execution time required by the use of different representations. Boxplot $u0s0$ represents both $\mathcal{U}$ and $\mathcal{S}$ masking turned off, i.e. the traditional binary string representation, whereas $u1$ and $s1$ represent $\mathcal{U}$ and $\mathcal{S}$ masking turned on respectively. For `printtokens` and `tcas`, $\mathcal{US}$-mask requires the least amount of time as the use of both types of masking reduces the computational cost for the construction heuristic by reducing the problem size. For `flex` and `gzip`, since $|\mathcal{U}|$ is much larger than $|\mathcal{S}|$, $\mathcal{U}$-masking requires less computational cost for the construction heuristic than $\mathcal{S}$-masking.

of existing work on problem space exploration has been done with single objective optimisation. The implications of applying the same idea to multi-objective optimisation problems are not clear and Mask-Coding representation may perform differently when applied to single-objective problems. This can only be answered with further empirical evaluation of the new representation. However, this paper chooses to evaluate the new representation with respect to the multi-objective optimisation because, in the context of Search-Based Software Engineering, the multi-objective version of set cover problem provides much more value to practitioners compared to the single-objective version of the same problem [5]. Second, there is no evidence that the additional greedy algorithm is the ideal choice of construction heuristic for the approach presented here. However, the additional greedy algorithm was selected due to its known effectiveness for set cover problem and it fits the profile of an ideal construction heuristic.

Threats to construct validity arises when the measurement used in the study does not reflect the concepts they represent. It should be noted that the research question on performance only evaluates the additional computational resource required by the masking. It does not reflect the savings in fitness evaluation that could have been gained if the stopping criterion was set differently. For example, if the true reference Pareto-front had been known, the stopping criterion could have been set with respect to the distance to the reference Pareto-front. If the new representation converges faster than the traditional representation, it would require less fitness evaluations. However, without the knowledge of the true Pareto-fronts, it was not possible to set the stopping criterion with respect to convergence.

VI. Related Work

Problem space exploration was first suggested by Storer et al. in an attempt to improve the optimisation for Job-Shop Scheduling problem [10]. A similar approach was also introduced as *noising* by Charon and Hudry [13]. Storer et al. discussed two different approaches of exploring the problem space: by perturbing the problem and by perturbing the construction heuristic (e.g. changing parameters of the construction heuristic). Since the additional greedy algorithm does not require any parameter tuning, the heuristic perturbation has not been considered in this paper.

The idea was applied to various combinatorial optimisation problems including 0-1 Multiple Knapsack Problem [14], Graph Partitioning Problem [15], Routing Problem [16] and Travelling Salesman Problem [17]. For all of these problems, the problem perturbation is represented as a vector of real numbers, which are usually weights that are multiplied to the numbers in the original problem. Therefore, these representations are often called *weight-coding*. However, no existing work uses bit-masking to perturb problem data expressed as sets.

Set-cover problem formed the basis of the widely studied test suite minimisation problem [1], [2], [23], [24]. Recently, formulating the test suite minimisation problem as a multi-objective set cover optimisation is an emerging trend found in search-based software testing [4]–[6]. This is because shorter development cycle often require the precise knowledge of how much testing is feasible given a budget on time. All of the existing work rely on the traditional binary string representation and, therefore, potentially suffer from the existence of dimensional plateau.

VII. Conclusions and Future Work

The paper introduces Mask-Coding, a novel representation for solutions of multi-objective set cover problem based on the concept of problem space exploration and problem perturbation. Mask-Coding uses bit-masks to perturb instances of set-cover problems. The empirical evaluation of the novel representation has shown that it can outperform the traditional binary string representation, especially under the existence of the dimensional plateau. Future work will consider evaluation of the representation with wider problem instances.

ACKNOWLEDGEMENT

The author is grateful to Xin Yao for an insightful discussion on the significance of the right choice of genotype representation.

REFERENCES

[1] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Transactions on Software Engineering and Methodology*, vol. 2, no. 3, pp. 270–285, 1993.

[2] J. Offutt, J. Pan, and J. Voas, "Procedures for reducing the size of coverage-based test sets," in *Proceedings of the 12th International Conference on Testing Computer Software*. ACM Press, June 1995, pp. 111–123.

[3] T. Y. Chen and M. F. Lau, "Dividing strategies for the optimization of a test suite," *Information Processing Letters*, vol. 60, no. 3, pp. 135–141, 1996.

[4] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, "Time aware test suite prioritization," in *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2006)*. ACM Press, July 2006, pp. 1–12.

[5] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in *Proceedings of International Symposium on Software Testing and Analysis (ISSTA 2007)*. ACM Press, July 2007, pp. 140–150.

[6] C. L. B. Maia, R. A. F. do Carmo, F. G. de Freitas, G. A. L. de Campos, and J. T. de Souza, "A multi-objective approach for the regression test case selection problem," in *Proceedings of Anais do XLI Simpòsio Brasileiro de Pesquisa Operacional (SBPO 2009)*, 2009, pp. 1824–1835.

[7] M. Harman and B. F. Jones, "Search based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, Dec. 2001.

[8] J. Gottlieb, B. A. Julstrom, F. Rothlauf, and G. R. Raidl, "Prüfer numbers: A poor representation of spanning trees for evolutionary search." in *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 2001, pp. 343–350.

[9] F. Rothlauf, D. E. Goldberg, and A. Heinzl, "Network random keys—a tree representation scheme for genetic and evolutionary algorithms," *Evolutionary Computation*, vol. 10, no. 1, pp. 75–97, 2002.

[10] R. H. Storer, S. D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Journal of Management Science*, vol. 38, no. 10, pp. 1495–1509, 1992.

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

[12] D. Fudenberg and J. Tirole, *Game Theory*. MIT Press, 1983.

[13] I. Charon and O. Hudry, "The noising method: a new method for combinatorial optimization," *Operations Research Letters*, vol. 14, no. 3, pp. 133–137, 1993.

[14] C. Cotta and J. Troya, "A hybrid genetic algorithm for the 0-1 multiple knapsack problem," in *Artificial Neural Networks and Genetic Algorithms*. Springer-Verlag, 1997, pp. 251–255.

[15] V. Sudhakar and C. Siva Ram Murthy, "A modified noising algorithm for the graph partitioning problem," *Integration, the VLSI Journal*, vol. 22, no. 1-2, pp. 101–113, 1997.

[16] M. J. Morgan and C. L. Mumford, "A weight-coded genetic algorithm for the capacitated arc routing problem," in *Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation (GECCO 2009)*. New York, NY, USA: ACM, 2009, pp. 325–332.

[17] B. Codenotti, G. Manzini, L. Margara, and G. Resta, "Perturbation: An efficient technique for the solution of very large instances of the euclidean tsp," *INFORMS JOURNAL ON COMPUTING*, vol. 8, no. 2, pp. 125–133, 1996.

[18] H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact." *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.

[19] N. Nethercote and J. Seward, "Valgrind: A program supervision framework," in *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2007)*. ACM Press, June 2007, pp. 89–100.

[20] K. Praditwong and X. Yao, "A new multi-objective evolutionary optimisation algorithm: The two-archive algorithm," in *Proceedings of Computational Intelligence and Security, International Conference*, ser. Lecture Notes in Computer Science, vol. 4456, November 2006, pp. 95–104.

[21] D. S. Johnson, "Approximation algorithms for combinatorial problems," in *Proceedings of the 5th annual ACM Symposium on Theory of Computing (STOC 1973)*. ACM Press, May 1973, pp. 38–49.

[22] J. Rice, *Mathematical Statistics and Data Analysis*. Duxbury Press, 2nd Ed. 1995.

[23] J. Black, E. Melachrinoudis, and D. Kaeli, "Bi-criteria models for all-uses test suite reduction," in *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*. ACM Press, May 2004, pp. 106–115.

[24] G. Rothermel, M. Harrold, J. Ronne, and C. Hong, "Empirical studies of test suite reduction," *Software Testing, Verification, and Reliability*, vol. 4, no. 2, pp. 219–249, December 2002.