Multi-objective Integer Programming Approaches for Solving the Multi-criteria Test-suite Minimization Problem: Towards Sound and Complete Solutions of a Particular Search-based Software-engineering Problem

YINXING XUE, School of Computer Science and Technology, University of Science and Technology of China, China
YAN-FU LI, Department of Industrial Engineering, Tsinghua University, China

Test-suite minimization is one key technique for optimizing the software testing process. Due to the need to balance multiple factors, multi-criteria test-suite minimization (MCTSM) becomes a popular research topic in the recent decade. The MCTSM problem is typically modeled as integer linear programming (ILP) problem and solved with weighted-sum single objective approach. However, there is no existing approach that can generate sound (i.e., being Pareto-optimal) and complete (i.e., covering the entire Pareto front) Pareto-optimal solution set, to the knowledge of the authors. In this work, we first prove that the ILP formulation can accurately model the MCTSM problem and then propose the multi-objective integer programming (MOIP) approaches to solve it. We apply our MOIP approaches on three specific MCTSM problems and compare the results with those of the cutting-edge methods, namely, *NonlinearFormulation_LinearSolver* (NF_LS) and two Multi-Objective Evolutionary Algorithms (MOEAs). The results show that our MOIP approaches can always find sound and complete solutions on five subject programs, using similar or significantly less time than NF_LS and two MOEAs do. The current experimental results are quite promising, and our approaches have the potential to be applied for other similar search-based software engineering problems.

CCS Concepts: • **Software and its engineering → Search-based software engineering**; *Software maintenance tools*;

Additional Key Words and Phrases: Regression testing, search-based software engineering, test-suite minimization, multi-objective integer programming, big-M method, ϵ-constraint method, CWMOIP

ACM Reference format:
Yinxing Xue and Yan-Fu Li. 2020. Multi-objective Integer Programming Approaches for Solving the Multi-criteria Test-suite Minimization Problem: Towards Sound and Complete Solutions of a Particular Search-based Software-engineering Problem. *ACM Trans. Softw. Eng. Methodol.* 29, 3, Article 20 (May 2020), 50 pages.
https://doi.org/10.1145/3392031

This work is supported by the National Natural Science Foundation of China Key Program (Grant No. 71731008) and General Program (Grant No. 61972373). The research of Dr. Xue is also supported by CAS Pioneer Hundred Talents Program.
Authors' addresses: Y. Xue, School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, 230026, China; email: yxxue@ustc.edu.cn; Y.-F. Li (corresponding author), Department of Industrial Engineering, Tsinghua University, Beijing, 100084, China; email: liyanfu@tsinghua.edu.cn.

1049-331X/2020/05-ART20 $15.00
https://doi.org/10.1145/3392031

1 INTRODUCTION

Software testing is a process that assesses or evaluates the quality of the software product or service under test throughout the Software Development Life Cycle. In particular, the software testing process can include several types of testing tasks: namely, functional testing, non-functional testing, and maintenance testing (a.k.a., regression testing) [96]. Among these, functional testing is a process that evaluates the compliance of a system or component with specified functional requirements; non-functional testing evaluates the quality of the program at runtime, such as reliability, robustness, and scalability; regression testing reruns functional and non-functional test cases to ensure that the newly modified software still performs the same as it previously did before a change. Software testing, no matter what type of testing, is critical to the final success of a software project. According to Reference [33], debugging, testing, and verification activities can easily range from 50% to 75% of the total development cost. Recently, estimated by Grechanik et al. [47], only the costs associated with maintaining and evolving test scripts could range from $50M to $120M per year. Hence, optimizing the testing process means minimizing the development or maintenance costs.

In general, during the testing process, test cases need to be crafted and executed to expose the potential program faults. In particular, during the process of regression testing, new test cases are added for the newly introduced functionality and those old ones are still executed for the original functionality—that is, the *test-suite* (i.e., the whole set of test cases) keeps expanding. As a program might evolve for several years, the size of the test-suite will keep increasing until it becomes too large to be executed in whole at once [50]. For example, an earlier paper reports that the regression testing of a Microsoft product required several days [16], and usually a product may require at least one build per day. Hence, the strategy of rebuilding with all test cases is not a feasible solution for industrial products, given the limited time and compilation resources. To address this issue, researchers have proposed three categories of techniques [101], namely, *Test-Suite Minimization* (TSM), *Test-Case Selection* (TCS), and *Test-Case Prioritization* (TCP).

TSM is to find a *minimal* set of test cases that satisfies the constraints of given requirements [36], which essentially is the dual problem of the minimal set covering problem [28]. In contrast, TCS is to retain the test cases that still could be used to test a *modified* version of the program [78], while TCP aims to find the optimal execution order of *all* test cases [25]. In this article, we consider the testing scenarios, where neither a modified version is considered nor all test cases will be executed. Thus, we will focus only on the approaches relevant to TSM.

In the earlier studies on TSM [21, 36], the reduction process is guided on the basis of a certain *testing criterion*, which refers to a collection of requirements (or conditions) to be fulfilled for the minimized test-suite [60]. Specifically, the single criterion is normally the conditions on a metric of the minimized test-suite (e.g., statement coverage, test-suite size, or total execution time) that is used to guide the reduction process. Although these existing techniques could perform well for single-criterion test-suite minimization, their applications in the real testing scenarios are still restricted. The rationale is twofold: (1) The metrics of testing coverage cannot guarantee the fault detection capability [42]—in practice, a higher overall statement coverage does not always assure that more *unique* faults are detected. (2) Software testing engineers often need to balance between multiple factors, including the number of detected faults, the tight time schedule, and limited compilation resources. Therefore, multi-criteria test-suite minimization (MCTSM) becomes a popular research topic in the recent decade, as predicted by Yoo et al. [101].

Before introducing the existing approaches to MCTSM, we need to define the two terms used in the article, namely, *soundness* and *completeness*. The former refers to optimality of the solution for single-objective optimization or non-dominance of the solution for multiple-objective optimization. The latter refers to the fullness of the Pareto-optimal front that a set of solutions

can cover—the solution set is considered complete if it can cover the whole Pareto front. Recent studies on MCTSM are mainly in two lines of research: using Integer Linear Programming (ILP) to solve for a weighted solution (scalarizing functions for multiple criteria) [9, 39, 55] and using heuristic search (e.g., Multi-objective Evolutionary Algorithm, MOEA for short) to find nondominated solutions (different objectives for multiple criteria) [91, 102, 109]. In reality, methods using ILP are more popular and considered to be more effective than those based on heuristic search. The reasons lie in two aspects: first, for the MCTSM problem, usually less than five criteria are considered in practice, which might not need the heavy shotguns (e.g., MOEAs) that work better for five or even more criteria. Second, it was believed that ILP methods can find the sound (namely, optimal for single objective or nondominated for multiple objectives) solutions [9, 39], while heuristic search can guarantee neither soundness nor completeness (meaning *all* potential nondominated) of the solutions. Hence, IP (including ILP and INP) methods are suitable for instances with a small-to-medium size (less than 10K test cases and 100K lines of code) and less than five types of criteria.

However, the recent study [55] insightfully pinpoints that ILP methods in References [9, 39] actually sometimes find sub-optimal (unsound) solutions for MCTSM. Due to the test case dependency relation (i.e., multiple cases may reveal the same fault), Lin et al. argue that *the MCTSM problem is essentially an Integer Nonlinear Programming (INP) problem* [55]. To facilitate the solving process, Lin et al. convert this INP program into an ILP program via introducing a larger number of decisive variables (particularly, $|test\ case\ size| \times |fault\ size|$) and call a linear solver for optimal solutions. From our observation, the problem formulation and solving in Reference [55] is sound and interesting, but it still inspires several unresolved research problems (RPs):

RP1. Towards the sound solutions, is it necessary for the MCTSM problem to be formulated as an INP problem [55]? Specifically, can we use the number of decisive variables that is less than $|test\ case\ size| \times |fault\ size|$ to accurately model the problem?

RP2. Towards the complete solutions, can the weighted-sum approach (i.e., scalarizing multiple criteria to one objective) be replaced by a general approach that aims at finding all nondominated solutions? Specifically, can multi-objective integer programming (MOIP) methods used in Reference [98]—namely, ***ϵ*-constraint** [34] and Constraint Weighted Multi-objective Integer Programming (**CWMOIP**)[68]—be applicable to the MCTSM instances?

RP3. For the tri-criteria TSM problem (namely, the TSM problem that considers the coverage, the size of minimized test-suite, and also the number of revealed faults—see Definition 7), the method proposed in Reference [55] (namely, NF_LS) takes a considerably long solving time. Can MOIP methods perform comparatively better? Besides, how do the existing state-of-the-art MOEAs (e.g., those used in Reference [109]) perform on this problem?

In this study, we aim to address the above three RPs via MOIP methods. We start with the potential limitations of the existing ILP methods towards the MCTSM program in Section 2.3. Then, we show that the MCTSM problem can be accurately formulated as an ILP problem with a number of $|test\ case\ size| + |fault\ size|$ decision variables. To overcome the extra complexity due to using INP (see Section 3), the technical novelty lies in the adoption of two ILP formulation methods for the MCTSM problem, namely, **the Big-M formulation** and **the Or-relation formulation** (see Section 4). Here, in operational research, Big-M is a simple but effective method that can convert a logical constraint into a set of constraints that describe the same feasible region such that the original optimization problem containing logical constraints can be eventually solved by mathematical programming technique, e.g., Simplex method [40]. Other than the Big-M formulation method, we also utilize the concept of Or-relation between features in Software Product Line Engineering (SPLE) [6, 98] and propose another formulation for this problem (hereinafter, we call it Or-relation

formulation). Besides applying these two formulation methods on the motivating example, *we further theoretically prove the equivalence of the two formulation methods* (see Section 4.4).

The existing methods based on Integer programming (IP) actually combine multiple criteria into a weighted-sum single objective and have certain drawbacks such as the weights to be predetermined and the ineffectiveness on non-convex solution space. Consequently, these methods cannot have a full picture brought by the Pareto-optimal front and hence fail to provide more trade-off solutions for software engineers to optimize the testing plan—for example, in our evaluation (Figure 7 of Section 6.3), it is observed that the state-of-the-art method (NF_LS) always returns the nondominated solution that maximizes the number of faults to reveal, but it totally ignores other trade-off solutions that maximize the statement coverage or the balance between both objectives. To address the issue of using weighted-sum, in this work, we treat multiple criteria equally and model the MCTSM problem as a MOIP problem via the Big-M or Or-relation formulation. For the MOIP problem, we apply ***ϵ*-constraint** [34] and Constraint Weighted Multi-objective Integer Programming (**CWMOIP**) [68] to solve the problem (see Section 5). As proved in Reference [68] and evaluated in Reference [98], these two MOIP solving methods will yield exactly the same nondominated solutions, given enough time for them. Following the successful application of ϵ-constraint and CWMOIP on the small instances of the optimal feature selection problem in SPLE [98], in this study, we further testify the generality of these two MOIP solving methods on the instances of the MCTSM problem.

Technically, three concrete MCTSM problems are modeled and solved in this study—that is, classic bi-criteria problem, variant bi-criteria problem, and tri-criteria problem. As these three concrete MCTSM problems are handled by the NF_LS method proposed by Lin et al. [55], we can compare our MOIP methods with NF_LS on the five subject programs that are commonly used for the MCTSM research (see Section 6). After all, our study aims at answering the above three RPs.

For RP1, our experiments prove that the four combinations of two formulations and two solving methods will yield exactly the same nondominated solutions, only with some differences in efficiency. The experiments also show that Big-M+CWMOIP is the most efficient method to solve the MCTSM problem, with the linear complexity of introduced decisive variables. For RP2, we compare our IP methods (**Big-M+CWMOIP** as the *fastest* method and **Or-relation+*ϵ*-constraint** as the *slowest* method) with NF_LS. We find that, on the classic bi-criteria problem and the variant bi-criteria problem, NF_LS is efficient and sound, but not complete. The weighted-sum objective makes NF_LS always find the nondominated solution that optimizes the size of test-suite, while our MOIP methods can find the whole set of nondominated solutions in terms of the size of test-suite and the number of revealed defects, using comparable or less time. On the tri-criteria problem, NF_LS is significantly less efficient (10 to 100 times slower) than our MOIP-methods, as NF_LS takes thousands to tens of thousands of seconds to find one nondominated solution, and our MOIP methods can find the whole set of nondominated solutions in dozens to hundreds of seconds. For RP3, on the tri-criteria problem, after witnessing the inefficiency of NF_LS, we apply MOEAs methods and compare with our MOIP methods. The experimental results show that our fastest method (Big-M+CWMOIP) could build up the complete Pareto front, being at least 10 times faster than two adopted MOEAs (i.e., NSGA-II [23] and MOEA/D [105]). Even for the slowest method, it might be 2 to 3 times slower than the two MOEAs, but it can find the complete nondominated solutions in all cases. However, the two MOEAs fail to find the complete Pareto front in many cases on the large subject programs Flex and Grep.

To sum up, we have made the following major contributions in this study:

(1) We adopt two integer linear formulations (Big-M and Or-relation) for the MCTSM problem and then prove their equivalence. To the best of our knowledge, the Big-M

formulation has not been applied for the MCTSM problem or similar SBSE problems. Besides, we model the MCTSM problem in a MOIP way, not with a weighted-sum single objective.

(2) ϵ-constraint and CWMOIP are previously applied for optimal feature selection problem in SPLE [98], not originally for the MCTSM problem. Via our study, we prove the generality of these two MOIP solving methods for similar SBSE problems that have linear constraints and objectives.

(3) Compared with the state-of-the-art methods (NF_LS and MOEAs), our MOIP methods can significantly improve the solving efficiency and guarantee the completeness of the resulted nondominated solutions, supported by the excellent experimental results on commonly used subject programs. Besides, we have published our MOIP solver and experimental raw data on the GitHub repository: https://github.com/yinxingxue/Testcase_MOIP.

2 BACKGROUND AND MOTIVATING EXAMPLE

In this section, we formally define the MCTSM problem and introduce its three concrete problems that are addressed in Reference [55]. Last, we show a motivating example to illustrate these problems.

2.1 Multi-criteria in Regression Testing

According to the recent survey on the TSM problem [45], among the 103 most relevant research papers, only 15 papers model the problem as a multiple-objective optimization problem with less than five objectives, while the majority of other papers model this problem as single-objective. Besides, according to the surveys [45, 91], testing criteria for this problem can be categorized intofour types: coverage-based (e.g., statement coverage, branch coverage), cost-based (e.g., execution time, the size of test-suite), effect-based (e.g., fault revealing capability), and others (e.g., the similarity between the test and commit logs of test cases [62, 76]).

2.2 Definition of the MCTSM Problem

The MCTSM problem can be defined in two ways, by using a weighted-sum single objective (WSO) for different criteria [39, 55] or adopting multiple objectives (MO) [109] for them.

Definition 1 (MCTSM with WSO). Given a test-suite T with n test cases from t_1 to t_n, a set of constraint criteria $C = \{C_i | i = 1 \dots m\}$ to be satisfied by T, a set of optimization criteria $O = \{O_i | i = 1 \dots l\}$ to be optimized by T, MCTSM with WSO is to find $T' \subseteq T$, a subset of T, which is to

$$\begin{array}{ll} \text{Max} & B_{wso}(T') = w_1 O_1(T') + \cdots + w_l O_l(T'), \\ \text{s.t.} & \forall i \in \{1 \dots m\} \cdot C_i(T') = C_i(T), \\ & \sum_{i=1}^{l} w_i = 1. \end{array} \tag{1}$$

Note that $B_{wso}(T')$ denotes the function that aggregates the weighted benefits of using multiple optimization criteria. Let T'_o be the optimal solution (subset of T), we have: $\nexists T'' \subseteq T \cdot B_{wso}(T'') > B_{wso}(T'_o)$. In practice, there might exist several optimal solutions that achieve the same value: $\exists S_{T'} \cdot \forall T'' \in S_{T'} \cdot B_{wso}(T'') = B_{wso}(T'_o)$, where $S_{T'}$ denotes the set of optimal solutions, the symbol "$\cdot$" is used to connect a quantifier and the predicates on it.

Before defining the MCTSM with MO, we first need to define the non-dominance relation that is used to compare the solutions of MOIP or MOEAs. Here, T' still denotes a solution, so $T' \in \mathbb{P}(T)$, and $\mathbb{P}(T)$ denotes the power set of T.

Definition 2. In the case of maximization, given two solutions T_1', T_2' and a set of optimization criteria O, T_1' **dominates** T_2' ($T_1' \succ T_2'$) if

$$\forall i \in \{1, \ldots, l\} \cdot O_i(T_1') \geq O_i(T_2'), \tag{2}$$

$$\exists j \in \{1, \ldots, l\} \cdot O_j(T_1') > O_j(T_2'), \tag{3}$$

otherwise $T_1' \not\succ T_2'$. In the case of minimization, we can similarly define that T_1' **dominates** T_2', if all optimization criteria for T_1' are not larger than those for T_2' and also at least one optimization criterion for T_1' is smaller than that for T_2'.

Definition 3. Given a solution T' and a set of solutions S_T, T' is **nondominated** iff

$$\forall T'' \in S_T \cdot T'' \not\succ T'. \tag{4}$$

T' is also called a *Pareto optimal* solution if T' satisfies all constraint criteria C and is *not dominated* by any other solutions. All Pareto-optimal solutions constitute the true Pareto front.

Definition 4 (MCTSM with MO). Given a test-suite T with n test cases from t_1 to t_n, a set of constraint criteria $C = \{C_i | i = 1 \ldots m\}$ to be satisfied by T, a set of optimization criteria $O = \{O_i | i = 1 \ldots l\}$ to be optimized by T, MCTSM with MO is to find $T' \subseteq T$, which is to

$$\begin{array}{ll} \text{Max} & B_{mo}(T') = (O_1(T'), O_2(T'), \ldots, O_l(T')) \\ \text{s.t.} & \forall i \in \{1 \ldots m\} \cdot C_i(T') = C_i(T). \end{array} \tag{5}$$

Here, $B_{mo}(T')$ denotes the function to search for the Pareto-optimal (shortly, optimal) solutions. In most multi-objective optimization (MOO) problems, there exist a number of nondominated solutions, as usually there exists no single solution globally dominating all the others—normally, the optimal value for each objective cannot be achieved at the same time [23].

To summarize, MCTSM with WSO is simple to solve with ILP but not generally effective, as the weight scheme may differ from project to project and the determination of the weights for different objectives is often susceptible to various factors that could be uncertain and subjective. In addition, using ILP for MCTSM with WSO guarantees to find only one optimal solution as long as the formulation is correct. In contrast, MCTSM with MO can provide more nondominated solutions for user choice [101], but it is in general considered to be computationally costly. Hence, MOEAs are usually adopted to build the Pareto front for efficiency, but MOEAs, as heuristic methods, can guarantee neither the soundness nor the completeness of solutions. In this study, to remedy the above issues, we aim to apply MOIP to build the true Pareto front.

2.3 Motivating Example and Three Specific MCTSM Problems

In some scenarios (e.g., deployment requirements), software projects need to be rebuilt and retested, even without code change. In such a regression testing scenario, the information on statement coverage and fault revealing of each test case is usually available for the task of TSM. The above Table 1 demonstrates a small example for the MCTSM problem, in which there are four test cases, four statements, and four faults. Each test case covers certain statements and exposes certain faults. For example, in Table 1, t_1 covers s_1 and s_2 and exposes f_1, f_2, and f_3. As the MCTCM problem is a minimal set covering problem [28], the total size or the total execution time of the reduced test-suite is a criterion that must be considered by engineers. In reality, it might be difficult to accurately measure the total execution time, since the program itself might be non-deterministic [65] or contain random or multi-thread operations [24]. Hence, in our example, the total size of reduced test-suite is the factor motivating the minimization process.

Table 1. An Example Test-suite with Coverage and Fault Detection Data, Which Is Based on the Example from Reference [55]

Criteria / Test cases	Statement				Fault			
	s_1	s_2	s_3	s_4	f_1	f_2	f_3	f_4
t_1	1	0	1	0	1	1	1	0
t_2	0	1	1	0	1	1	1	0
t_3	1	0	0	1	0	0	0	1
t_4	0	1	0	1	1	0	1	0

Based on the above example, we can observe several common testing criteria in specific MCTSM problems [93]: the statement coverage, the fault detection ratio, and the size of the reduced test-suite. Regarding which criteria are used as constraint criteria and which as optimization criteria, it varies in different specific problems. We will define these specific problems as follows:

Definition 5 (The Classical Bi-criteria Problem). Given a test-suite T with n test cases from t_1 to t_n, the statement coverage C_1 (for the statement set $S = \{s_1 \dots s_p\}$) as the only constraint criterion, the size of reduced test-suite O_1 and the number of revealed faults O_2 (for the fault set $F = \{f_1 \dots f_q\}$) as the two optimization criteria, the classical bi-criteria problem is to find $T' \subseteq T$, which is to

$$\begin{aligned} \text{Min} \quad & O_1(T') = \quad |T'| = |\bigcup_{t_i \in T'} \{t_i\}|, \\ \text{Max} \quad & O_2(T') = \quad |\bigcup_{t_i \in T'} findFault(t_i, F)|, \\ \text{s.t.} \quad & \forall k \in \{1 \dots p\} \cdot isCovered(s_k, T') = isCovered(s_k, T). \end{aligned} \tag{6}$$

Note that in Definition 5, $|X|$ returns the size of the set X (X can be a set of test cases, faults, or statements). Function $findFault(t_i, F)$ returns the set of faults in F found by the execution of t_i, so O_2 is the size of the union of the sets of faults found by the executions of the reduced test-suite T'. Besides, $isCovered(s_k, T')$ returns a Boolean value (1 or 0) indicating whether the given statement s_k is covered by the test case T'. For example, in Table 1, if $T' = \{t_2, t_3\}$, then $O_1(T') = |\{t_2, t_3\}| = 2$, $O_2(T') = |findFault(t_2, F) \cup findFault(t_3, F)| = |\{f_1, f_2, f_3\} \cup \{f_4\}| = |\{f_1, f_2, f_3, f_4|\} = 4$. Hence, $\{t_2, t_3\}$ is the optimal solution, which utilizes only two test cases but covers all the four statements and reveals all the four defects.

Definition 6 (The Variant Bi-criteria Problem). Given a test-suite T with n test cases from t_1 to t_n, the statement coverage C_1 as one constraint criterion, the number of times for which the set of important statements S_I ($S_I \subseteq S$) are covered, C_2, as the extra constraint criterion, the size of reduced test-suite O_1 and the number of revealed faults O_2 as the two optimization criteria, the variant bi-criteria problem is to find $T' \subseteq T$, which is to

$$\begin{aligned} \text{Min} \quad & O_1(T') = \quad |T'| = |\bigcup_{t_i \in T'} \{t_i\}|, \\ \text{Max} \quad & O_2(T') = \quad |\bigcup_{t_i \in T'} findFault(t_i, F)|, \\ \text{s.t.} \quad & \forall k \in \{1 \dots p\} \cdot isCovered(s_k, T') = isCovered(s_k, T), \\ & \forall s' \in S_I \cdot coveredTimes(s', T') \geq \mu. \end{aligned} \tag{7}$$

Comparing Definition 5 and Definition 6, we can find that the variant bi-objective problem is more strict than the classic one—they share the same two optimization criteria and one constraint criterion, but the variant problem has one additional constraint criterion on covering the important statements for at least μ times. For example, suppose $S_t = \{s_4\}$ and $\mu = 2$ in Table 1, $T' = \{t_2, t_3\}$, which is an optimal solution to the classical problem, is no longer a valid solution now. As s_4

requires to be covered at least twice, valid solutions must have t_3 and t_4 to satisfy the constraint criterion C_2. Hence, $\{t_1, t_3, t_4\}$ and $\{t_2, t_3, t_4\}$ are optimal solutions that constitute the complete true Pareto front.

Definition 7 (The Tri-criteria Problem). Given a test-suite T with n test cases from t_1 to t_n, the size of reduced test-suite C_1 as the only constraint criterion, the number of covered statements O_1 (for the statement set $S = \{s_1 \dots s_p\}$) and the number of revealed faults O_2 (for the fault set $F = \{f_1 \dots f_q\}$) as the two optimization criteria, the tri-criteria problem is to find $T' \subseteq T$, which is to

$$\begin{aligned} &\text{Max} \quad O_1(T') = \quad |\bigcup_{t_i \in T'} coverStmt(t_i, S)|, \\ &\text{Max} \quad O_2(T') = \quad |\bigcup_{t_i \in T'} findFault(t_i, F)|, \\ &\text{s.t.} \quad |T'| = \eta \times |T| = \eta n. \end{aligned} \tag{8}$$

Note that in Definition 7, given a test case t_i, $coverStmt(t_i, S)$ returns the set of statements covered by t_i. So, O_1 now becomes the size of the union of the sets of statements covered by the executions of the reduced test-suite T'. The constant η denotes the maximally allowed shrinkage percentage and $|T|$ denotes the size of the original test-suite. In comparison with Definition 5 and Definition 6, the tri-criteria problem does not require the original test-suite T and the reduced one T' to have the exact same coverage. Instead, the statement coverage is more relaxed, converting from a constraint criterion (i.e., the first constraint in Definition 5 and Definition 6) to an optimization criterion (i.e., O_1 in Definition 7). Hence, this criteria problem is actually still a bi-objective optimization problem—using the number of covered statements and the number of revealed faults as objectives, and adjusting the value of shrinkage percentage η for the reduced test-suite size. Taking Table 1, for instance, if we set $\eta = 50\%$, $T' = \{t_2, t_3\}$ will be the optimal solution, using half size of T and covering all faults and statements; if we set $\eta = 25\%$, $T' = \{t_1\}$ and $\{t_2\}$ will be the optimal solutions, using one-fourth of T and covering two statements and three faults.

The two objectives (number of revealed faults and statement coverage) in Definition 7 are also competing objectives. In common sense, unknown bugs to detect and statement coverage are usually relevant (high statement coverage may lead to more bugs to detect)—that is the reason why software fuzzing adopts coverage as feedback to detect more bugs. However, faults to reveal are different from unknown bugs in fuzzing—faults in TSM refer to the old known bugs in regression testing, and hence revealing faults does not necessarily lead to detecting unknown bugs. To summarize, the two objectives in Definition 7 are considered competing—number of revealed faults is to re-expose the known bugs in regression testing, while statement coverage is to help detect unknown bugs.

2.4 The Challenges of These Problems

As can be observed from the above definitions, the constraint criteria seem to be straightforward and easy to model. The challenges lie in how to properly model those optimization criteria. In Section 3, we show how the existing IP methods handle the optimization criteria in the WSO way and their limitations. In Sections 4 and 5, we study how to accurately formulate this problem via ILP and solve it for Pareto-optimal solutions in the MO way.

3 LIMITATIONS OF EXISTING INTEGER PROGRAMMING METHODS

In this section, we will elaborate on the existing IP formulations for the MCTSM problem. For ease of understanding, we illustrate the examples for the classic bi-criteria problem. Finally, we

summarize the potential weakness of these formulations and propose a new perspective of ILP formulation for the MCTSM problem.

3.1 Existing Integer Programming Methods

There mainly exist three IP formulations for the MCTSM problem [55], namely, LF_LS, NF_NS, and NF_LS.

3.1.1 The MINTS Method. LF_LS (*LinearFormulation_LinearSolver*), as the name implies, refers to the earlier method MINTS [39], which models an MCTSM problem with a linear formulation and solves it with a linear solver. LF_LS used in [55] is actually the reimplementation of MINTS using Python. MINTS adopts an arithmetic summation for the criterion O_2, the number of revealed faults, in Definition 5 to Definition 7. Specifically, we let a binary decision variable t_i represent whether the ith test case is included in the reduced suite ($t_i = 1$ if it is selected, and 0 otherwise), and the optimization criterion on fault detection is defined as below [39, 55]:

$$\text{Min} \quad \sum_{i=1}^{|T|} \Big(1 - w(t_i)\Big)\, t_i \tag{9}$$

where $w(t_i)$ denotes the capability in detecting faults for the given test case t_i, and we can formally define $w(t_i) = (\sum_{j=1}^{|F|} v_{ij})/|F|$. Here, $|F|$ denotes the number of faults, and v_{ij} is a binary variable indicating whether test case t_i can expose fault f_j.

We can further deduct from the above formula: Min $\sum_{i=1}^{|T|}(1 - w(t_i))\, t_i$ = Min $(\sum_{i=1}^{|T|} t_i + \sum_{i=1}^{|T|} -t_i w(t_i))$ = Min $\sum_{i=1}^{|T|} t_i$ + Min $\sum_{i=1}^{|T|} -t_i w(t_i)$. Actually, Min $\sum_{i=1}^{|T|} t_i$ represents the criterion to minimize the size of reduced test-suite; Min $\sum_{i=1}^{|T|} -t_i w(t_i)$ represents the criterion to maximize the capability to detect faults, as Max $\sum_{i=1}^{|T|} t_i w(t_i)$ and Min $\sum_{i=1}^{|T|} -t_i w(t_i)$ are equivalent problems. In logic, Equation (9) combines the criteria of the size and the detection capability of the reduced test-suite.

Using MINTS, we will have the solution $\{t_1, t_4\}$ for the motivating example. After scrutiny, this solution is unsound (non-optimal), as $\{t_1, t_4\}$ reveals not five faults, but three *distinct* faults: $\{f_1, f_2, f_3\}$. Actually, the optimal solution is $\{t_2, t_3\}$, which covers four statements and reveal four faults. Hence, the MINTS method is not an accurate formulation of the MCTSM-WSO problem, which will only produce an approximate solution.

As pointed out by Lin et al. [55], the arithmetic summation of the fault detection capability of each test case (i.e., $\sum_{i=1}^{|T|} t_i w(t_i)$) is not accurate, as the relation among a fault and multiple test cases that reveal this fault is not considered. To model such dependency in the optimization criterion, a nonlinear objective function is required—integer nonlinear programming (INP) has been adopted by Lin et al. to formulate this problem.

3.1.2 The NF_NS Method. NF_NS (*NonlinearFormulation_NonlinearSolver*) refers to the method that models an MCTSM problem with a nonlinear formulation and solves it with a nonlinear solver. Before the definition, we need to recall the binary variable v_{ij} that indicates whether test case t_i can reveal the fault f_j (true for 1, false for 0), and introduce a binary variable d_{ij} that accounts for dependencies among test cases in terms of each f_j. For the criterion of the revealed fault number O_2, given a set of test cases T and a set of distinct faults F, the optimization criterion on fault detection is defined as below:

$$\text{Min} \quad \sum_{i=1}^{|T|} (1 - \tilde{w}_o(t_i))\, t_i, \tag{10}$$

where, $\tilde{w}_o(t_i)$ denotes the capability of detecting distinct faults for test case t_i, taking model dependencies into consideration. To count the number of faults actually revealed by a test, it will be considered whether the faults are already revealed by other selected tests. $\tilde{w}_o(t_i)$ is formally defined as follows:

$$\tilde{w}_o(t_i) = \frac{1}{|F|}\left(\sum_{j=1}^{|F|} v_{ij}d_{ij}\right), \tag{11}$$

$$d_{ij} = \prod_{t' \in T_j} (1 - t'), t' \neq t_i. \tag{12}$$

Note that $|F|$ denotes the size of the fault set and T_j denotes the set of test cases that reveal the jth fault (f_j). If any test case $t' \in T_j$ is included in the reduced test-suite T', then according to Equation (12), d_{ij} will evaluate to be 0—logically, if f_j is already revealed by other test cases, then this fault will not be counted in $\tilde{w}_o(t_i)$, the distinct fault detection capability of t_i. Hence, via using d_{ij}, a distinct fault will be counted only once for the optimization criteria O_2. Theoretically, the introduction of d_{ij} lowers the value of $\tilde{w}_o(t_i)$ for avoiding the repeated counting of a distinct fault, and hence decreases the likelihood of t_i being selected if it reveals those commonly found faults. However, the issue of introducing d_{ij} is that the Equation (10) becomes a non-linear formulation, as the multiplication of d_{ij} and t_i will lead to a polynomial expression.

However, the NF_NS method fails to guarantee finding the optimal solutions—as the experiments in Reference [55] show, NF_NS sometimes finds sub-optimal solutions. The nonlinear formulation is sound, and the issue rises from the solving step to the nonlinear problem. As nonlinear problem solving is much more computationally costly than linear problem solving [66, 95], a sound solving method is desired to rapidly find the optimal solutions. To mitigate such issue, Lin et al. make the endeavor on addressing this non-linear formulation with a linear solver.

3.1.3 The NF_LS Method. NF_LS (*NonlinearFormulation_LinearSolver*) refers to the method that models an MCTSM problem with a nonlinear formulation and solves it with a linear solver. The basic idea is that even with a non-linear formulation, the MCTSM problem can be converted into an equivalent problem that could be addressed via linear problem solving. To facilitate such conversion, Lin et al. follow the solution from Reference [12] and introduce new *auxiliary* decisive variables to simplify this problem. Before defining the new formulation for the criterion O_2, i.e., the number of revealed faults, it is necessary to recall several predefined variables: Given a test case t_i, we have the binary variable v_{ij} denoting whether t_i can reveal the fault f_j, and d_{ij} denoting the dependencies among test cases t_i and other test cases that reveal f_j. The optimization criterion on the number of detected faults is defined as below:

$$\text{Min} \quad \sum_{i=1}^{|T|} (1 - \tilde{w}_o(t_i))\ t_i, \tag{13}$$

$$= \quad \sum_{i=1}^{|T|}\left(1 - \frac{1}{|F|}\left(\sum_{j=1}^{|F|} v_{ij}d_{ij}\right)\right) t_i = \quad \sum_{i=1}^{|T|}\left(t_i - \frac{t_i}{|F|}\left(\sum_{j=1}^{|F|} v_{ij}d_{ij}\right)\right), \tag{14}$$

$$= \quad \sum_{i=1}^{|T|}\left(t_i - \frac{1}{|F|}\left(\sum_{j=1}^{|F|} v_{ij}d_{ij}t_i\right)\right) = \quad \sum_{i=1}^{|T|}\left(t_i - \frac{1}{|F|}\left(\sum_{j=1}^{|F|} \bar{v}_{ij}\right)\right). \tag{15}$$

In Equation (15), Lin et al. convert the objective function into a linear one by introducing the new auxiliary decisive variables $\bar{v}_{ij} = v_{ij}d_{ij}t_i$. Logically, the binary variable $\bar{v}_{ij}$ indicates whether the fault f_j is *first* revealed by t_i or any other previously selected test case—that is, $\bar{v}_{ij} = 1$ if t_i is revealed *first* by f_j, and $\bar{v}_{ij} = 0$ otherwise. Note that $\bar{v}_{ij}$ is used to logically model the first time of

revealing the fault f_j, not the test case execution order to reveal f_j. As a fault f_j can be discovered (revealed for the first time) by at most one test case, the sum of $\bar{v}_{ij}$ for a given f_j is bound to be not greater than 1 (see Equation (17)). In using these auxiliary variables, there are some constraints to be added into the original formulation. There are two types of constraints on each $\bar{v}_{ij}$:

$$\forall f_j \in F \cdot \text{if } t_i \text{ reveals } f_j, \text{then } \bar{v}_{ij} \leq t_i, \tag{16}$$

$$\forall f_j \in F \cdot \sum_{i=1}^{|T|} v_{ij}\bar{v}_{ij} \leq 1. \tag{17}$$

The first type of constraint is easy to infer: as $\bar{v}_{ij} = v_{ij}d_{ij}t_i$, $0 \leq v_{ij} \leq 1$ and $0 \leq d \leq 1$, we have $\bar{v}_{ij} \leq v_{ij}$. So, the first type is on the valid value range of each $\bar{v}_{ij}$. The second type of constraint is on the alternative relation between multiple test cases on the same fault f_j—the intuition is that the first reveal of the fault f_j can only be counted once for a certain test case $t_i, i \in \{1, \ldots, |T|\}$. After the conversion, the optimization criterion in Equation (15) becomes a linear formulation, and there are no polynomial expressions.

As being admitted in Reference [55], in the worst case, the NF_LS method may introduce a number of new decisive variables up to $|F| \times |T|$. Essentially, NF_LS is still formulating the problem by INP, and the conversion is to exchange space (i.e., using more auxiliary decisive variables) for time (i.e., avoiding the time-consuming nonlinear problem solving). Although the MCTSM is generally NP-complete, after the conversion, many MCTSM instances can be efficiently solved by modern solvers. Owing to the algorithm advancement and implementation optimization of modern ILP solvers such as Cplex [41], usually the problems with up to dozens of thousands of decisive variables can be solved within a reasonable time limit (e.g., several hours or one day). Based on linear problem solving, if solutions are found by NF_LS, they are guaranteed to be optimal (i.e., sound). However, as NF_LS addresses the MCTSM problem with WSO, other potential Pareto-optimal solutions for multiple optimization criteria are not explored.

3.2 Comparison Summary and New Perspective

For ease of understanding, the above three IP methods are illustrated with the motivating example in Table 2. As introduced in the previous section, LF_LS is straightforward for both the constraint criterion and the optimization criteria, and it only requires $|T|$ decisive variables. Although the solving part of LF_LS is very efficient, the solutions are not optimal—the solution cannot find the minimized test-suite that leads to most distinct faults. Prior to the study of Lin et al., LF_LS is widely used in the relevant publications [9, 35, 39]. In contrast, NF_NS addresses the issue of repeated fault counting in LF_LS and improves the optimization criteria with INP. However, the INP formulation may produce a polynomial of higher degree—in Table 2, it will lead to a polynomial of degree 3 such as $d_{11}t_1 = (1 - t_2)(1 - t_4)t_1$. In the worst case, if a fault is revealed by all n test cases, a polynomial expression of degree n (i.e., $t_1t_2 \ldots t_n$) needs to be introduced. Hence, the solving part of NF_NS is not efficient and cannot scale up to medium-to-large MCTSM instances (namely, those with $n = |T| \geq 400$). Even if a solution is found by NF_NS, it is not guaranteed to be optimal.

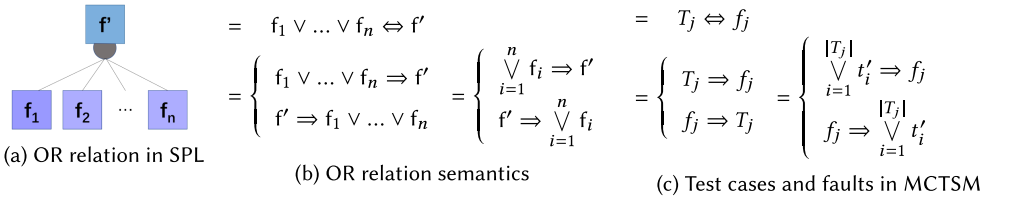
Witnessing the weakness of LF_LS and NF_NS, Lin et al. proposed NF_LS, which introduces auxiliary variables $\bar{v}_{ij}$ to replace those non-linear terms $v_{ij}d_{ij}t_i$. The idea is feasible and it helps to achieve the soundness for both the formulation and the solving parts. The only drawback of this method is that it requires a quadratic number of auxiliary decisive variables. For example, in Table 2, it requires totally nine ($\sum_{i=1}^{|T|}\sum_{j=1}^{|F|} v_{ij} = 9$) auxiliary decisive variables. In worst case, where all the faults are revealed by all the test cases, a maximum of $|T| \times |F|$ auxiliary decisive variables are required.

Table 2. Comparison of Three Existing IP Methods on the Motivating Example

Method	LF_LS	NF_NS	NF_LS
Formulations for classic bi-criteria.	Min: $(1-\frac{3}{4})t_1+(1-\frac{3}{4})t_2$ $+(1-\frac{1}{4})t_3+(1-\frac{2}{4})t_4$ st: $t_1+t_3\geq 1$ $t_2+t_4\geq 1$ $t_1+t_2\geq 1$ $t_3+t_4\geq 1$	Min: $(1-\frac{1}{4}((1-t_2)(1-t_4)+$ $(1-t_2)+(1-t_2)(1-t_4)))t_1+$ $(1-\frac{1}{4}((1-t_1)(1-t_4)+(1-$ $t_1)+(1-t_1)(1-t_4)))t_2+(1-$ $\frac{1}{4})t_3+(1-\frac{1}{4}((1-t_1)(1-t_2)+$ $(1-t_1)(1-t_2)))t_4$ st: $t_1+t_3\geq 1$ $t_2+t_4\geq 1$ $t_1+t_2\geq 1$ $t_3+t_4\geq 1$	Min: $(t_1-\frac{1}{4}(\bar{v}_{11}+\bar{v}_{12}+$ $\bar{v}_{13}))+(t_2-\frac{1}{4}(\bar{v}_{21}+\bar{v}_{22}+$ $\bar{v}_{23}))+(t_3-\frac{1}{4}(\bar{v}_{34}))+(t_4-$ $\frac{1}{4}(\bar{v}_{41}+\bar{v}_{43}))$ st: $t_1+t_3\geq 1$ $t_2+t_4\geq 1$ $t_1+t_2\geq 1$ $t_3+t_4\geq 1$ $\bar{v}_{11}\leq t_1, \bar{v}_{12}\leq t_1, \bar{v}_{13}\leq t_1$ $\bar{v}_{21}\leq t_2, \bar{v}_{22}\leq t_2, \bar{v}_{23}\leq t_2$ $\bar{v}_{34}\leq t_3$ $\bar{v}_{41}\leq t_4, \bar{v}_{43}\leq t_4$ $\bar{v}_{11}+\bar{v}_{21}+\bar{v}_{41}\leq 1$ $\bar{v}_{12}+\bar{v}_{22}\leq 1$ $\bar{v}_{13}+\bar{v}_{23}+\bar{v}_{43}\leq 1$ $\bar{v}_{34}\leq 1$
Number of Decisive Variables	$\lvert T\rvert$	$\lvert T\rvert$	up to $\lvert T\rvert+\lvert T\rvert\times\lvert F\rvert$
Formulation Soundness	No	Yes	Yes
Solving Soundness	Yes	No	Yes
Related Literature	[9] [39] [35]	[55]	[55]
WSO or MO	WSO	WSO	WSO

NF_LS relies on ILP for efficient solving and seems to be the ultimate approach for the MCTSM problem, which is essentially an NP-complete problem. Due to the inherent complexity of this problem and internal dependencies of the MCTSM model, as claimed by Lin et al. [55], an INP formulation for this problem seems to be unavoidable. However, according to the experience and observations in the optimal feature selection problem from SPL [98], we think applying linear formulation and linear solving for the MCTSM is feasible, as long as the inner-model dependencies (the relation that multiple test cases can only reveal the same fault once) can be modeled with constraints in first-order logic. For the MCTSM problem, actually it is not necessary to explicitly model which fault is first revealed by which test case—the set of auxiliary decisive variables $\bar{v}_{ij}$ are not necessary. Instead, the key idea is to utilize logical constraint to model whether fault f_j is *actually* revealed by the test case set T_j (T_j, the set of test cases that reveal fault f_j).

To summarize, a linear formulation and linear solving method for this MCTSM problem is feasible and desired. Compared with MINTS (a.k.a., LF_LS) and NF_NS, the desired method should guarantee the soundness of both formulation and solving. Besides, compared with NF_LS, the desired method should have a linear number of decisive variables for speeding up solving. Last but not the least, the existing IP methods are all for MCTSM-WSO. In general, they do not obtain alternative solutions. If users want to seek for alternative solutions, users have to adjust the weights for each optimization criterion. In addition, since MCTSM is non-convex, adjusting the weights might not find all Pareto-optimal solutions for the multi-objective problem [53]. Hence, the desired method is also expected to support MCTSM-MO, building a complete true Pareto front.



(a) OR relation in SPL (b) OR relation semantics (c) Test cases and faults in MCTSM

Fig. 1. The OR relation in SPL domain, its semantics, and its similarity to the relation between a fault and the test cases revealing it.

4 TWO MOIP FORMULATION METHODS

In this section, we propose two different linear formulations for modeling the MCTSM problem with MO. First, we adopt the formulation that is used for OR-relation features [6] in the optimal feature selection problem in SPL. Second, we introduce the Big-M method and apply for the classic bi-criteria problem. Subsequently, we prove the equivalence of the two formulation methods. Last, we show the generality of our formulation methods by applying for the two other specific MCTSM problems: namely, the variant bi-criteria and tri-criteria problems.

4.1 The Or-relation Formulation

In SPL domain, Apel and Kästner define software *feature* as "a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option" [2]. To show software functionality and the relations among different functional modules, features are organized in the tree-like hierarchy, called feature diagram [83]. Inside the feature diagram, there exists one type of *OR* relation between a feature and its several subfeatures [6]. Here, subfeatures refer to the submodules or sub-functions of a feature. For example, in the SPL of the *Web Portal* system [61], under the root feature, there is a feature (module) called *Security* and its three *OR*-relation subfeatures, namely, *Data storage*, *Data transfer*, and *User Auth*. This relation requires the following condition: The parent feature *Security* is to be selected when at least one of its subfeatures is selected, and vice versa.

In Figure 1(a), we show a general case of an *OR*-relation among features in a system of SPL—feature f' is the parent feature and features f_1 to f_n are the *OR*-relation subfeatures. In Figure 1(b), we use proposition logic to model the semantics behind this *OR*-relation, according to References [6, 98]. In Figure 1(c), we also use proposition logic to model the semantics behind a fault as well as the test cases that can reveal this fault. Analogically, the fault is semantically like the parent feature in SPL, and the test cases are semantically like those *OR*-relation subfeatures. So, we can conclude that a fault is to be revealed when at least one of its associated test cases is executed, and vice versa.

Lemma 4.1. *According to the lemma of converting the logical formula to linear inequalities in Reference [98], given a fault f_j and the test case $T_j = \{t'_1, \ldots, t'_{|T_j|}\}$, we can have the following linear inequalities:*

$$\forall i \in \{1, \ldots, |T_j|\} \cdot t'_i - f_j \leq 0, \tag{18}$$

$$\sum_{i=1}^{|T_j|} t'_i - f_j \geq 0. \tag{19}$$

Proof. $\bigvee_{i=1}^{|T_j|} t'_i \Leftrightarrow f_j$ is *true* according to the semantics in Figure 1(c). Here, $\bigvee_{i=1}^{|T_j|} t'_i$ notates $t'_1 \vee t'_2 \vee \cdots \vee t'_{|T_j|}$. Thus, the formulae $\bigvee_{i=1}^{|T_j|} t'_i \Rightarrow f_j$ and $f_j \Rightarrow \bigvee_{i=1}^{|T_j|} t'_i$ need to be *true*. For the first formula, we can represent it as the CNF and get the resulting formula $\bigwedge_{i=1}^{|T_j|}(\neg t'_i \vee f_j)$ to be *true*. Note that for an indexed set of propositions $P = \{p_1, \ldots, p_n\}$, $\bigwedge_{i=1}^{n}(p_i)$ means that each proposition in P needs to be *true*. If the formula $\neg t'_i \vee f_j$ is *true*, then we can infer the equality $(1 - t'_i) + f_j \geq 1$. Thus, we can get $|T_j|$ derived linear inequalities that are listed in Equation (18). For the second formula $f_j \Rightarrow \bigvee_{i=1}^{|T_j|} t'_i$, we get $\neg f' \vee \bigvee_{i=1}^{|T_j|} t'_i$ to be *true*, that is, $\neg f_j \vee t'_1 \vee \cdots \vee t'_{|T_j|}$ to be true—accordingly, we infer the inequality $(1 - f_j) + t'_1 + \cdots + t'_{|T_j|} \geq 1$. So, Equation (19) can be deduced from the second formula. □

For the classic bi-criteria problem in Definition 5, based on the *OR*-relation modeling, it can be formulated as follows:

$$\begin{aligned}
&\text{Max} \quad O_1(T) = \quad -\sum_{i=1}^{n} t_i, \\
&\text{Max} \quad O_2(T) = \quad \sum_{j=1}^{q} f_j, \\
&\text{s.t.} \quad \forall k \in \{1 \ldots p\} \cdot \sum_{i=1}^{n} \sigma_{ik} t_i \geq 1, \\
&\qquad \forall j \in \{1 \ldots q\} \cdot \begin{cases} \forall i \in \{1 \ldots n\} \cdot v_{ij} t_i - f_j \leq 0 \\ \sum_{i=1}^{n} v_{ij} t_i - f_j \geq 0 \end{cases},
\end{aligned} \tag{20}$$

where n denotes the number of original test cases, p denotes the number of the statements to cover, q denotes the number of faults to reveal, $\sigma_{ik} t_i$ denotes a binary variable indicating whether t_i covers the kth statement, v_{ij} denotes a binary variable indicating whether t_i reveals the jth fault. Note that $T_j = \{t'_1, \ldots, t'_{|T_j|}\}$ in Equations (18) and (19) is replaced with $T_j = \{v_{1j} t_1, \ldots, v_{nj} t_n\}$. In such a way, the first part of constraints in Equation (20) is on the statement coverage constraint (i.e., the constraint criterion C_1 in Definition 5), while the second part of constraints is on the *OR*-relation between a fault and its associated test cases.

4.2 The Big-M Formulation

Big-M is an effective and simple method frequently applied in operations research studies to convert a logical constraint into a set of constraints describing the same feasible region [40]. One simple example of a logical constraint is whether the other constraint is satisfied or not. Given a general constraint,

$$g(x_1, x_2, \ldots, x_n) \geq b, \tag{21}$$

then the logical constraint can be expressed as a binary variable with the following expression:

$$y = \begin{cases} 1, & g(x_1, x_2, \ldots, x_n) \geq b \\ 0, & \text{otherwise} \end{cases}. \tag{22}$$

Let M denote a very large positive number, then the above logical constraint can be converted into the following two arithmetic constraints:

$$\begin{aligned}
&g(x_1, x_2, \ldots, x_n) \geq b - M(1 - y), \\
&g(x_1, x_2, \ldots, x_n) < My + b.
\end{aligned} \tag{23}$$

Lemma 4.2. *The logical constraint in Equation (23) is equal to the Big-M constraints in Equation (22).*

Proof. In Equation (23), if we have $g(x_1, x_2, \ldots, x_n) \geq b$, then y can not be 0. Otherwise, according to the second constraint, we will have $g(x_1, x_2, \ldots, x_n) < b$. This is a contradiction. Thus, y must equal to 1. However, if $y = 1$, then we obtain the following constraints:

$$\begin{aligned} &g(x_1, x_2, \ldots, x_n) \geq b, \\ &g(x_1, x_2, \ldots, x_n) < M + b. \end{aligned} \tag{24}$$

The first one is just the condition itself. The second constraint is always satisfied, when $M + b$ is larger than the upper limit of $g(x_1, x_2, \ldots, x_n)$, which can be achieved if M is sufficiently large. Similarly, based on Equation (23), we can prove the following: (1) if $g(x_1, x_2, \ldots, x_n) < b$, then y must be 0; (2) if $y = 0$, then $g(x_1, x_2, \ldots, x_n) < b$. □

With respect to the classic bi-criteria problem in Definition 5, it can be reformulated into the following optimization problem with logical constraints, using the symbols defined in Section 4.1:

$$\begin{aligned} \text{Max} \quad & O_1(T) = -\sum_{i=1}^{n} t_i, \\ \text{Max} \quad & O_2(T) = \sum_{j=1}^{q} f_j, \\ \text{s.t.} \quad & \forall k \in \{1 \ldots p\} \cdot \sum_{i=1}^{n} \sigma_{ik} t_i \geq 1, \\ & \forall j \in \{1 \ldots q\} \cdot f_j = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} v_{ij} t_i \geq 1 \\ 0, & \text{otherwise} \end{cases}. \end{aligned} \tag{25}$$

After implementing the Big-M method onto Equation (22), we have the following linear formulation:

$$\begin{aligned} \text{Max} \quad & O_1(T) = -\sum_{i=1}^{n} t_i, \\ \text{Max} \quad & O_2(T) = \sum_{j=1}^{q} f_j, \\ \text{s.t.} \quad & \forall k \in \{1 \ldots p\} \cdot \sum_{i=1}^{n} \sigma_{ik} t_i \geq 1, \\ & \forall j \in \{1 \ldots q\} \cdot \begin{cases} \sum_{i=1}^{n} v_{ij} t_i \geq 1 - M(1 - f_j) \\ \sum_{i=1}^{n} v_{ij} t_i < M f_j + 1 \Leftrightarrow \sum_{i=1}^{n} v_{ij} t_i \leq M f_j \end{cases}. \end{aligned} \tag{26}$$

4.3 Application on Motivating Example

Based on the previously inferred Equations (20) and (26), we apply both of the formulation methods for the motivating example. As observed in Tables 2 and 3, the existing IP methods from Table 2 cannot accurately model the relation among a fault and multiple test cases that reveal this fault. Particularly, LF_LS ignores such a relation and counts the faults repetitively. NF_NS and NF_LS aim to model such a relation, but they adopt the INP formulation by defining and introducing a complicated objective function. In contrast, we aim to explicitly define the variables $\{f_1, \ldots, f_q\}$ for faults and model their relation with test cases $\{t_1, \ldots, t_n\}$ via the Big-M and Or-relation formulations, where the constraints on $\{f_1, \ldots, f_q\}$ and $\{t_1, \ldots, t_n\}$ are all linear.

Table 3. Application of Two New ILP Methods on the Motivating Example

Method	Big-M	OR-relation
Formulations for classic bi-criteria.	Max: $-(t_1 + t_2 + t_3 + t_4)$ Max: $f_1 + f_2 + f_3 + f_4$ st: $t_1 + t_3 \geq 1$ $t_2 + t_4 \geq 1$ $t_1 + t_2 \geq 1$ $t_3 + t_4 \geq 1$ $t_1 + t_2 + t_4 \geq 1 - M(1 - f_1)$ $t_1 + t_2 + t_4 \leq Mf_1$ $t_1 + t_2 \geq 1 - M(1 - f_2)$ $t_1 + t_2 \leq Mf_2$ $t_1 + t_2 + t_4 \geq 1 - M(1 - f_3)$ $t_1 + t_2 + t_4 \leq Mf_3$ $t_3 \geq 1 - M(1 - f_4)$ $t_3 \leq Mf_4$ $M = 5$	Max: $-(t_1 + t_2 + t_3 + t_4)$ Max: $f_1 + f_2 + f_3 + f_4$ st: $t_1 + t_3 \geq 1$ $t_2 + t_4 \geq 1$ $t_1 + t_2 \geq 1$ $t_3 + t_4 \geq 1$ $t_1 \leq f_1, t_2 \leq f_1, t_4 \leq f_1$ $t_1 + t_2 + t_4 - f_1 \geq 0$ $t_1 \leq f_2, t_2 \leq f_2$ $t_1 + t_2 - f_2 \geq 0$ $t_1 \leq f_3, t_2 \leq f_3, t_4 \leq f_3$ $t_1 + t_2 + t_4 - f_3 \geq 0$ $t_3 \leq f_4$ $t_3 - f_4 \geq 0$
Number of Decisive Variables	$\|T\| + \|F\|$ for bi-criteria problem $\|T\| + \|F\| + \|S\|$ for tri-criteria problem	$\|T\| + \|F\|$ for bi-criteria problem $\|T\| + \|F\| + \|S\|$ for tri-criteria problem
Formulation Complexity	Linear	Linear
Formulation Soundness	Yes	Yes
Solving Soundness	Yes	Yes
Related Literature	[40]	[98]
WSO or MO	MO	MO

As shown in Table 3, the numbers of decisive variables adopted in both methods are the same, with a linear size as $|T| + |F|$ for the bi-criteria problems and $|T| + |F| + |S|$ for the tri-criteria problem. The two optimization criteria on minimization of the test-suite size and maximization of the revealed faults have the linear formulation, and sound solutions will be achieved with linear solvers. Hence, it seems that the MCTSM problem could be elegantly formulated without using nonlinear formulation. However, two issues need to be addressed before we can assure the soundness and completeness of our methods: (1) it needs to be proved that the Big-M formulation and *OR*-relation formulation must be equivalent; (2) as we address this MCTSM problem in the MO way, instead of using a WSO method via gradually adjusting the weight scheme, we need a solving approach that can guarantee to find all the solutions for this MOIP problem when the problem size (i.e., $|T| + |F| + |S|$) is not too big.

4.4 Proof of Equivalence

From Section 4.1 and Section 4.2, we have obtained two different formulations, as the two conversion methods come from different research areas. We can see that Big-M method is applicable to a generic logical constraint where certain terms can have general forms, e.g., polynomial, exponential, and so on. Whereas, OR-relation is for purely logical terms and logical operators.

But for MCTSM problem, we can prove that these two methods are actually equal. The key is to prove that the generated Big-M constraints and the generated OR-relation constraints are equivalent.

Theorem 4.3. *The Big-M constraints in Equation (26) are equal to the OR-relation constraints in Equation (20).*

Proof. For any fault f_j, according to Lemma 4.2, we know that

$$\begin{aligned} \sum_{i=1}^{n} v_{ij} t_i &\geq 1 - M(1 - f_j), \\ \sum_{i=1}^{n} v_{ij} t_i &\leq M f_j, \end{aligned} \tag{27}$$

is equal to

$$f_j = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} v_{ij} t_i \geq 1 \\ 0, & \text{if } \sum_{i=1}^{n} v_{ij} t_i \leq 0 \end{cases}, \tag{28}$$

and according to Lemma 4.1, we know that

$$\begin{aligned} &\forall i \in \{1 \ldots n\} \cdot v_{ij} t_i - f_j \leq 0, \\ &\sum_{i=1}^{n} v_{ij} t_i - f_j = 0, \end{aligned} \tag{29}$$

is equal to the logical relation $\bigvee_{i=1}^{|T_j|} t'_i \Leftrightarrow f_j$. Because $t'_i = v_{ij} t_i$, we have $\sum_{i=1}^{n} v_{ij} t_i \geq 1 \Leftrightarrow \bigvee_{i=1}^{|T_j|} t'_i = 1$ and $\sum_{i=1}^{n} v_{ij} t_i = 0 \Leftrightarrow \bigvee_{i=1}^{|T_j|} t'_i = 0$. Thus, the two sets of constraints are equal. □

To clarify, the reasons of exploring both methods (Big-M and Or-relation) are twofold: first, the two methods can mutually validate the results; second, they are from different research backgrounds—Big-M is a standard method in operational research, while Or-relation is a practical method that we previously applied for the optimal feature selection problem [98].

4.5 Application for Variant Problems

In Section 4.3, we show how both formulation methods are used for the classic bi-criteria problem on the motivating example. As the above two formulation methods are proved to be equivalent, we will only show how to apply the *OR*-relation formulation to model the other two variant problems: variant bi-criteria and tri-criteria problem.

For the variant bi-criteria problem in Definition 6, based on the *OR*-relation modeling, the formulated equation is almost the same as Equation (20), except on the right sides of those statement coverage constraints. In the classic bi-criteria problem, the right sides of the statement coverage constraints are all equivalent to 1, while their right sides may not be 1 in the variant bi-criteria problem—the important statements need to be covered at least μ times ($\forall s_{k'} \in S_I \cdot \sum_{i=1}^{n} \sigma_{ik'} t_i \geq \mu$), as defined in Equation (7). For example, in Table 1, if s_1 needs to be executed twice, then the corresponding inequality is: $t_1 + t_3 \geq 2$. We can conclude that the variant bi-criteria problem is more strict than the classic one, and hence the solution space (or Pareto front size) should be smaller than that of the classic one.

For the tri-criteria problem in Definition 7, based on the *OR*-relation modeling, it can be formulated as follows:

$$\begin{aligned}
\text{Max} \quad & O_1(T) = \sum_{k=1}^{p} s_k, \\
\text{Max} \quad & O_2(T) = \sum_{j=1}^{q} f_j, \\
\text{s.t.} \quad & \forall k \in \{1 \dots p\} \cdot \begin{cases} \forall i \in \{1 \dots n\} \cdot \sigma_{ik} t_i - s_k \leq 0 \\ \sum_{i=1}^{n} \sigma_{ik} t_i - s_k \geq 0 \end{cases}, \\
& \forall j \in \{1 \dots q\} \cdot \begin{cases} \forall i \in \{1 \dots n\} \cdot v_{ij} t_i - f_j \leq 0 \\ \sum_{i=1}^{n} v_{ij} t_i - f_j \geq 0 \end{cases}, \\
& \sum_{i=1}^{n} t_i = \eta n,
\end{aligned} \tag{30}$$

where n denotes the number of original test cases, p denotes the total number of the statements to cover, q denotes the total number of faults to reveal, $\sigma_{ik} t_i$ denotes a binary variable indicating whether t_i covers the kth statement, v_{ij} denotes a binary variable indicating whether t_i reveals the jth fault, and η denotes the target percentage of the test-suite. The idea of the formulation is to model the relation between a statement and the associated test cases via linear formulation, similar to the *OR*-relation between a fault and its associated test cases. Besides, as the statement coverage becomes a constraint criterion, not an optimization criterion, there are totally two objectives to be maximized by using a total number of $|T| + |F| + |S|$ decisive variables.

5 SOLVING FOR MOIP AND IMPLEMENTATION

As defined in Definitions 5, 6, and 7, the three specific MCTSM problems solved in this study are all with two optimization objectives. As the size of true Pareto front (a.k.a., the complete set of nondominated solutions) increases with the number of objectives and the number of decisive variables, these MCTSM problems with two objectives and less than 10 thousands of decisive variables highly likely yield the true Pareto front of a comparatively small size. Under such circumstance, the two MOIP solving methods, ϵ-constraint method and CWMOIP, that are successfully applied for small instances of the optimal feature selection problem in SPL domain [98] are adopted to build the true Pareto front.

5.1 ϵ-constraint Method

As ILP solvers can only handle one objective at once, the idea is to make $k - 1$ objectives (k = 2 in the MCTSM problems) as the range constraints and use the first one as the objective function in binary integer problem (BIP) [34]. Each range constraint's upper bound will be iterated from 0 to the upper bound of the corresponding objective, by step size 1. In our MCTSM problems, since we have two optimization criteria, O_2 can be converted to range constraints and O_1 still serves as the objective function. The range constraint converted from O_2 will have its upper bound iterated from 0 to the maximal value of O_2.

The detailed procedures are shown in Algorithm 1. Note that $getObjUpperBound(O_2)$ at line 2 finds the theoretic upper bound B_2^{TUB} for O_2—for the example in Table 1, it is 4 when all faults are found by the reduced test-suite. Similarly, $getObjLowerBound(O_2)$ at the same line finds the theoretic lower bound B_2^{TLB} for O_2, and it is 0 when no fault is found for the motivating

$$\begin{array}{llll} \text{Max} & O_1(T) = & -\sum_{i=1}^{n} t_i & \\ \text{Max} & O_2(T) = & \sum_{j=1}^{q} f_j & \Rightarrow \\ \text{s.t.} & \text{the constraints } C \text{ in Eq. (20) hold} & & \end{array} \qquad \begin{array}{llll} \text{Max} & O_1(T) = & -\sum_{i=1}^{n} t_i & \\ \text{s.t.} & O_2(T) = & \sum_{j=1}^{q} f_j \le \upsilon, \upsilon \in \{0, \ldots, q\} & \\ & \text{the constraints in } C \text{ Eq. (20) hold} & & \end{array} \tag{31}$$

Fig. 2. An example of applying ϵ-constraint for the classic bi-criteria problem.

ALGORITHM 1: Function *EpsilonCont*() for an MCTSM problem

Input: O_1, O_2: the two optimization criteria
Input: C: the set of constraint criteria
Output: E: a non-dominated solution set for an MCTSM problem

```
1  E ← ∅;
2  B_2^TUB ← getObjUpperBound(O_2), B_2^TLB ← getObjLowerBound(O_2);
3  for υ ← B_2^TUB; υ ≥ B_2^TLB; υ ← υ−1 do
4  |   allCons ← C ∪ {O_2 ≤ υ};
5  |   ME ← bintprog(allCons, O_1);
6  |   for e ∈ ME do
7  |   |   E ← E\{e' ∈ E|e ≻ e'};
8  |   |   if e ∉ E∧ E ⊁ e then
9  |   |   |   E ← E ∪ {e} ;
10 |   |   end
11 |   end
12 end
13 return E;
```

example. The new constraint converted from O_1 ranges from $[0, 4]$, and the loop variable will iterate from 4 to 0 at line 3. Inside the loop, at line 4, *allCons* is the union of the original inequalities of Equation (20) and one new inequality converted from the objective O_2 in Equation (31) in Figure 2. At line 5, $bintprog(allCons, O_1)$ calls the BIP function to solve for the objective O_1 such that *allCons* is satisfied and the solution *ME* is returned by the solver. At line 7, ϵ-constraint method removes the very few weakly dominated solutions from E, where $\{e' \in E | e \succ e'\}$ refers to these removable solutions that are dominated by the new nondominated solution e in *ME*. At line 8, it is checked whether the solution *ME* is not contained or dominated by the current Pareto front E. If so, *ME* is nondominated and it should be added to the Pareto front at line 9.

bintprog() function represents the key ILP model solving function in Cplex. To the best of our knowledge, the theoretical time complexity of *bintprog*() function is essentially NP-hard. Hence, the worst-case time-complexity of Algorithm 1 is non-polynomial—this complies with the common sense, as the TSM is essentially an NP-hard minimal set covering problem. If we want to find the true Pareto front for a TSM instance of any scale, the time complexity of an exact approach must be non-polynomial. However, owing to the advances of modern IP solvers, Cplex is very efficient for the small-to-medium size solving models (e.g., the TSM instances addressed in this article) with less than 100K decisive variables. Usually, setting a solving time limit for Cplex is a practical routine of using ILP solvers. In our study, one second is more than enough for any single solving attempt calling *bintprog*().

$$\begin{array}{lll} \text{Max} & O_1(T) = & -\sum_{i=1}^{n} t_i \\ \text{Max} & O_2(T) = & \sum_{j=1}^{q} f_j \\ \text{s.t.} & \text{the constraints in } C \text{ Eq. (20) hold} & \end{array} \Rightarrow \begin{array}{lll} \text{Max} & O_1'(T) = & -\sum_{i=1}^{n} t_i + w_2(\sum_{j=1}^{q} f_j) \\ \text{s.t.} & O_2(T) = & \sum_{j=1}^{q} f_j \le l_2 \\ & w_2 = \frac{1}{(f_2^{UB} - f_2^{LB}+1)} & \\ & \text{the constraints in } C \text{ Eq. (20) hold} & \end{array} \quad (32)$$

Fig. 3. An example of applying CWMOIP for the classic bi-criteria problem.

If considering BIP solving function *bintprog*() takes constant time,[1] Algorithm 1 is of the time complexity of $O(n)$, where n represents the feasible range $(B_2^{TUB} - B_2^{TLB})$ of the second objective O_2. Solving the MOBIP problem in Equation (20) is reduced to solving the BIP problem in Equation (31) many times—a number of $(q + 1)$ times, and q is the size of the fault set (i.e.,$|F|$). Note that this algorithm could be applied to multiple objectives (more than two), and in this case the time complexity will increase exponentially with the number of converted constraints. Interested readers can refer to the relevant literature [34, 98] for details.

5.2 CWMOIP

ϵ-constraint method can be inefficient when the range of an objective is very big and there are several objectives' ranges to be iterated. To address these issues, Özlen et al. [68] propose CWMOIP, which is an objective-reduction technique for MOIP to generate *all* nondominated solutions. The improvements lie in two aspects: (1) for each objective, the lower bound and the upper bound of an objective is based on not only its own formula, but also the constraints of the problem. Precisely, BIP is applied to get the *true* lower and upper bounds for each objective (i.e., O_2 to O_k) separately, subject to the set of constraint criteria C. (2) As named, it applies the constraint weight method for objective reduction and elegantly avoids the production of weakly dominated solutions [68]. The k-objective problem is first reduced to a problem of $k - 1$ objectives, then $k - 2$, iteratively, until the last one (i.e., the first objective O_1) is left.

Example for (1). In calculating the true upper and lower bounds, for the example in Table 1, the true bounds B_2^{LB} and B_2^{UB} for O_2 are 3 (choosing $\{t_1, t_4\}$ to cover all statements but find least faults) and 4 (choosing $\{t_2, t_3\}$ to cover all statements and find most faults), respectively. Hence, the range of O_2 in CWMOIP is $[3, 4]$, not $[0, 4]$ in the ϵ-constraint method—only at most two iterations are needed for CWMOIP while five iterations are needed for the loop at line 3 in Algorithm 1 for the ϵ-constraint method.

Example for (2). Figure 3 shows the example of objective-reduction for the *bi*-objective problem in Equation (20). After reduction, the bi-objective problem in Equation (20) is reduced to a single-objective problem in Equation (32) by l_k times. It is named "constraint weighted" because of the weight w_k and the constraint $O_k \le l_k$. Variable l_k iterates from the lower bound B_k^{LB} to the upper bound B_k^{UB} of O_k.

In Algorithm 2, the general steps of finding all nondominated solutions are shown for any given k-objective BIP problem [68]. Algorithm 2 is a recursive function, whose initial invocation

[1]The official document of Cplex suggests setting a time limit in practical usage: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.cplex.help/CPLEX/UsrMan/topics/progr_consid/tuning/09_eg_time_limits.html. The reason is that the time complexity of *bintprog*() function is essentially NP-hard. According to our experience, the ILP solver (e.g., Cplex) may take a very long solving time (even hang) for some large problems with more than a million of decisive variables. However, on small-to-medium models addressed in this study, all the solving attempts are finished in one second.

ALGORITHM 2: Function *CWMOIP()* for an MCTSM problem

Input: k: the initial number of objectives (k starts with 2 in our example)
Input: l_k: constrained value for the weighted kth *obj*
Input: C: the set of constraint criteria
Output: E: a non-dominated solution set for an MCTSM problem

1 $E \leftarrow \emptyset$;
2 **if** $k = 1$ **then**
3 $\quad E \leftarrow E \cup bintprog(C, O'_1)$;
4 $\quad$ **return** E;
5 **end**
6 $B_k^{UB}, B_k^{LB} \leftarrow getObjTrueBound(C, O_k)$;
7 $w_k \leftarrow \frac{1}{(f_k^{UB} - f_k^{LB} + 1)}$;
8 $O'_1 \leftarrow addObjFuncSuffix(O_1, w_k O_k)$; /* the addition of suffix $w_k O_k$ for O_1 is done once */
9 $C' \leftarrow C \cup \{O_k \leq l_k\}$;
10 **while** *true* **do**
11 $\quad ME \leftarrow CWMOIP(k-1, l_k, C')$;
12 $\quad$ **if** $ME = \emptyset$ **then**
13 $\quad\quad$ *break*;
14 $\quad$ **end**
15 $\quad E \leftarrow E \cup \{e \in ME | E \not\succ e\}$;
16 $\quad l_k \leftarrow Max(O_k(e), e \in ME) - 1$; /* update the right side of constraint $O_k \leq l_k$ */
17 **end**
18 **return** E;

is $CWMOIP(k, B_k^{UB}, C)$, and then $CWMOIP(k-1, l_k, C')$ at line 11, recursively, until calling $CWMOIP(1, l_2, C')$. Each time of recursion, line 6 calculates the true upper and lower bounds of the current kth objective, subject to the constraint set C. Then w_k is calculated for the kth objective at line 7. Inside the loop at line 10, the k-objective problem is reduced to a new (k-1)-objective problem (line 11), which has the new suffix $w_k O_k$ for the objective function (line 8) and the new constraint $O_k \leq l_k$ for the constraint set C' (line 9). If no results are found (lines 12–14), the *while* loop terminates. If found, at lines 15 and 16, the constraint l_k is tightened to the value just smaller than the largest value of $O_k(e)$ for $e \in ME$. Last, lines 2 to 5 represent one iteration of BIP solving function invocation for O'_1, which is with many suffixes of other weighted objectives.

According to Reference [68], the maximum number of recursions is $ceiling(\frac{|E|(|E|+1)\ldots(|E|+k-2)}{2\times3\times\cdots\times(k-1)})$, where $|E|$ denotes the size of true Pareto front (a.k.a., the number of nondominated solutions) of this problem instance. However, for the MCTSM problems in this study, they are bi-objective problems ($k = 2$). The maximum number will be $ceiling(\frac{|E|}{2})$. For the example in Table 1, $w_2 = \frac{1}{2}$, as $B_2^{UB} = 4$ and $B_2^{LB} = 3$. As l_2 is by default using the actual upper bound of O_2 (B_2^{UB}), we have $l_2 = 4$. Notably, there exists only one global nondominated solution $\{t_2, t_3\}$ when $l_2 = 4$. Hence, after l_2 is tightened to 3 at line 16, no other nondominated solutions would be found—in other words, there does not exist other nondominated solution that uses only one test case ($< |\{t_2, t_3\}|$) to reveal three faults, meanwhile covering all statements. To sum up, if considering BIP solving function *bintprog*() takes constant time, the time complexity of Algorithm 2 is $O(|E|)$, where $|E|$ is the total number of nondominated solutions. Normally, we have $|E| \leq n$, because there are no more number of efficient solutions than the feasible range.

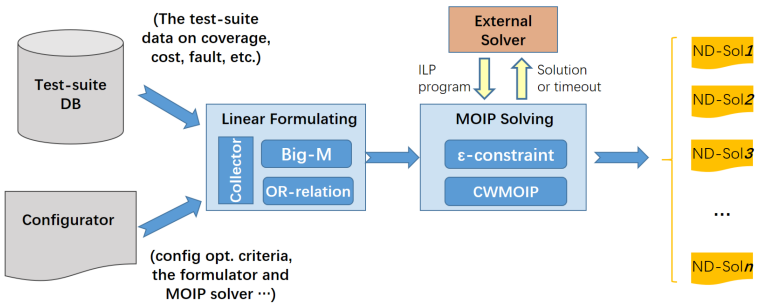


Fig. 4. System overview.

5.3 Implementation

5.3.1 System Overview. In Figure 4, the overview and major components of our overall approach are illustrated. The input of our approach includes a test-suite and its associated coverage data, cost data (e.g., execution time for each test case), fault detection data, and so on. Since the MCTSM may have the specific variant problems and we have the aforementioned alternative ILP-methods (two ways of linear formulation together with two ways of linear solving), we need a configuration file to configure which variant problem is to be addressed and decide which formulation and solving method to be used. In the first step of processing, before calling Big-M or OR-relation formulation submodules, the module of *Linear Formulating* has a submodule, called *Collector*, to compile the information from different criteria of the test-suite. After linear formulating, our approach will call one of the two MOIP solving submodules that rely on an external solver for ILP solving. Last, instead of returning a single solution for test-suite minimization, the output of our approach offers a notable abundance of choices (various nondominated solutions for trade-off decisions) for the users with respect to multiple optimization criteria via constructing a complete Pareto front. Each nondominated solution could result into a different minimized test-suite.

5.3.2 MOIP Solver. The implementation of our approach contains more than 5K lines of *Python* code, inside which around 3.5K lines of code belong to the module of *MOIP Solving*. Different from the implementation of ϵ-constraint and CWMOIP in Java for the optimal feature selection problem in SPL [98, 99], *MOIP Solving* is purely developed in Python and applicable to any general MOIP problem with linear constraints and linear objectives. This module takes the formulation of Equation (20) as input and produces the converted standard (single-objective) ILP formulations in Equation (31) and Equation (32). Then, the mainstreaming ILP solver (e.g., Cplex [41] at the user-end) can be externally called to solve this problem. Note that the timeout for external ILP solving can be customized, and a solution is returned if the input ILP problem is solvable within the timeout. Last, the optimal solutions will be collected, verified, and sorted with non-dominance relation with the module *Post Processing* for the purpose of building the sound and complete Pareto front. Last, for reproduction, our tool and experimental data are published at https://github.com/yinxingxue/Testcase_MOIP.

6 EVALUATION

In evaluation, we conduct experiments with our two linear formulation methods together with two MOIP solving techniques and compare them with the state-of-the-art techniques (i.e., the existing

IP methods in Reference [55] and the MOEAs techniques in Reference [109]) for this problem. Specifically, we aim at answering the following research questions (RQs):

RQ1. Though we have theoretically proved the equivalence of the Big-M and *OR*-relation formulations, can we further confirm their *equivalence* on basis of the empirical results from real-world test-suites and compare their *performances* when combined with the two MOIP solving techniques (i.e., ϵ-constraint and CWMOIP)?

RQ2. To prove the *soundness* and *completeness* of our IP-methods, we need to compare them with three state-of-the-art IP methods (i.e., LF_LS, NF_NS, and NF_LS) proposed by Lin et al. [55]. Can the results found by the two sound approaches (i.e., NF_NS and NF_LS) be always covered by the results of our methods?

RQ3. Apart from IP methods, it is interesting to compare our methods with MOEAs (e.g., NSGA-II [23] and MOEA/D [105]) that are widely used for the regression testing optimization problem [109]. Is the Pareto front built by our exact methods (epsilon-constraints and CWMOIP) the true Pareto front that could be approximated by MOEAs? And how are the results in terms of *performance*?

Since we have several types of methods for comparison on the three specific TCTSM problems, it is difficult to include all the results for each method on three specific problems due to page limit. Instead, to answer RQ1, we conduct the experiments on five real-world test-suites for the two small problems (classic and variant bi-criteria problems)—the rationale is that the confirmation of the equivalence on small problems should be convincing, considering the validation of the theoretic proof in Section 4.4. To address RQ2, we compare our methods with the existing IP approaches on all the experimented test-suites for all the three specific problems to avoid any bias and show the full picture of the comparison. Last, to address RQ3, the comparison between our MOIP methods and the existing MOEAs is only conducted on the relatively big problem (i.e., the tri-criteria problem) that requires a sophisticated optimization searching algorithm, while the two bi-criteria problems are efficiently solved within one second by existing IP methods [55].

6.1 Experiments Setup

6.1.1 Target System. The experiments are conducted on the following five open-source C projects, namely, Grep, Flex, Sed, Make, and Gzip. The reasons that we adopt them for evaluation are twofold. First, they are part of the publicly available dataset [37], most of which are widely used in various testing tasks. For example, Make and Grep are included by Marcel Böhme and Roychoudhury in the regression error benchmark, CoREBench [10]; Make, Grep, and Flex are used by Cotroneo et al. [19] to test the offline learning method that combines testing techniques; Sed, Gzip, and Grep are adopted in the implementation of BugRedux [43] for the reproduction of failures in debugging task. Second, using the same five projects as done by Nemo [55] enables a direct comparison between our methods and their methods from Nemo. Table 4 shows the detailed information of the subject programs that are used in our experiments. Apart from its own information of each program, we also append the last three columns to illustrate the complexity of each program for different formulation methods. Related discussions on these three columns can be found in Sections 3.2 and 4.5.

6.1.2 Test Suites & Faults. As pointed out by Lin et al. [55], the original test-suites coming with each program are not sufficient to achieve a high testing coverage for its core functions. Hence, Lin et al. extend the original test suites with additional tests and use KLEE [14] to ensure a coverage higher than 60% on core functions and also gcov [29] to measure the statement coverage. For the newly added test cases, Lin et al. follow the same technique as the original test suites do to generate

Table 4. Subject Programs Used in the Empirical Evaluation

Program	Version	Description	LOC	#Tests($\|T\|$)	#Faults($\|F\|$)	$\|T\|^*\|F\|$	$\|T\|+\|F\|$	$\|T\|+\|F\|+\|S\|$
Make	3.8	Executables builder and generator	23,400	158	15	2,370	173	23,573
Sed	4.2	Command-line text editor	26,466	324	25	8,100	349	26,815
Gzip	1.3	Data compressor	5,682	397	56	22,232	453	6,135
Flex	2.5.4	Lexical analyzer	12,366	605	37	22,385	642	13,008
Grep	2.7	Pattern search and matching utility	58,344	746	54	40,284	800	59,144

faults, which is to inject mutants and minimize them via Trivial Compiler Equivalence [72] (TCE). In this article, to have the fair comparison, we also use the same test suites and generate faults as those in Nemo [55]. Specifically, the test case information file (*rtime.info*), the statement coverage file (*cov.info*), and the fault reveal file (*fault.info*) used as the inputs of Nemo are used as the same inputs of our methods.

6.1.3 Quality Indicators. For RQ1 and RQ2, since we need to compare the results of different IP methods, we mainly adopt two indicators to measure the solution quality: execution time and Pareto front size (a.k.a., the number of the found nondominated solutions). As all the IP methods to be evaluated—no matter ours or those in Nemo—are all deterministic, we only execute each method for five times. For RQ3, as we need to compare our IP methods with the state-of-the-art MOEAs, we will employ these performance indicators: Hypervolume (HV) [111] and the inclusion relation between two Pareto fronts. Hypervolume (HV) [111] is a metric to measure the size of the objective space covered by the nondominated solutions. Given a solution set $S = (s_1, ..., s_n)$, HV is the volume of the region that is dominated by S in the objective space. In the implantation of most MOEA libraries, the front and reference point will be normalized and translated, and hence the preferred Pareto front would be with the highest HV. According to Reference [94], HV is considered as a good quality indicator, since it combines coverage with diversity, and meanwhile is easy to calculate. Note that MOEAs were executed for 30 times for mitigating the possible randomness.

6.2 Answer to RQ1

6.2.1 Four Configurations. As we propose two linear formulation methods in Section 4 and present two solving methods in Section 5, there are four different combinations of formulations and solving techniques: **Big-M+ϵ-constraint** (that is, Big-M for formulation and ϵ-constraint for solving, referred to as **Config.1**), **Big-M+CWMOIP** (**Config.2**), **OR-relation+ϵ-constraint** (**Config.3**), **OR-relation+CWMOIP** (**Config.4**). We will experiment with these four configurations to check whether they yield the exactly identical results and what the performance gaps are.

6.2.2 Comparison on the Bi-criteria Problem. We apply the four configurations on the bi-criteria problem for the five selected programs. Table 5 shows the results for each configuration, in which **t(s)** denotes the time used for solving the MOIP problem, $|\mathcal{P}|$ denotes the size of the returned Pareto front, **#A** denotes the times of attempts to call an ILP solving. Basically, we can observe the following facts from Table 5: (1) All the four configurations yield the Pareto fronts of the same size for the same instance. Through careful scrutiny, we confirm that the Pareto fronts yielded by the four configurations are *completely* identical for the same instance—this proves the equivalence of different combinations of formulation and solving techniques in practice. (2) However, the different

Table 5. Effectiveness of Different Formulation and Solving Techniques for the **Classic Bi-criteria** Problem

Setup \ Program	Make			Sed			Gzip			Flex			Grep		
	t(s)	$\|\mathcal{P}\|$	#A	t(s)	$\|\mathcal{P}\|$	#A	t(s)	$\|\mathcal{P}\|$	#A	t(s)	$\|\mathcal{P}\|$	#A	t(s)	$\|\mathcal{P}\|$	#A
Config. 1	1.31	3	16	0.89	1	26	1.41	5	57	6.40	5	38	6.59	14	55
Config. 2	0.87	3	8	0.34	1	6	0.39	5	10	2.36	5	10	2.85	14	21
Config. 3	1.40	3	16	1.08	1	26	1.76	5	57	6.93	5	38	7.27	14	55
Config. 4	0.91	3	8	0.40	1	6	0.57	5	10	2.53	5	10	3.31	14	21

Table 6. Effectiveness of Different Formulation and Solving Techniques for the **Variant Bi-criteria** Problem

Setup \ Program	Make			Sed			Gzip			Flex			Grep		
	t(s)	$\|\mathcal{P}\|$	#A	t(s)	$\|\mathcal{P}\|$	#A	t(s)	$\|\mathcal{P}\|$	#A	t(s)	$\|\mathcal{P}\|$	#A	t(s)	$\|\mathcal{P}\|$	#A
Config. 1	1.43	1	16	0.98	1	26	1.87	3	57	7.04	1	38	7.87	1	55
Config. 2	0.76	1	6	0.36	1	6	0.39	3	8	1.90	1	6	1.20	1	6
Config. 3	1.52	1	16	1.10	1	26	2.22	3	57	7.88	1	38	8.67	1	55
Config. 4	0.84	1	6	0.42	1	6	0.58	3	8	2.04	1	6	1.57	1	6

configurations may yield noticeable performance gaps. After examining the data in Table 5, we find out that there are some existing rules that can explain the performance gaps among them, and these rules are generally valid on each of these programs.

Here, we elaborate on these rules explaining the performance gap. The **first** rule is that the solving attempts chiefly decide the total time to solve a MOIP problem. Specifically, each solving attempt is to call $bintprog(allCons, O_1)$ at line 5 in Algorithm 1 or $bintprog(C, O_1')$ at line 3 in Algorithm 2. Each operation of $bintprog()$ is to solve an ILP problem via a standard solver (i.e., Cplex in our implementation), which may take at least dozens of milliseconds for a small ILP problem instance, normally a half-second for most instances in our experiments, or even hours for some big instances. For example, for Config.2 in Table 5, it averagely takes 39 milliseconds (0.39s/10) for one solving on Gzip, while it averagely takes 143 milliseconds (2.85s/20) for one solving on Grep. Hence, the bigger the problem is, the longer solving time it may take.

The **second** rule is that, for an operation of $bintprog()$ with deterministic parameters, the formulation with less constraints takes a shorter solving time if the decisive variables and optimization objective are the same. As shown in Table 5, on the program of Sed, Config.3 takes accumulatively 190 more milliseconds than Config.1 in a total of 26 solving attempts (both of them). Since the formulations behind Config.1 and Config.3 are equivalent, the difference is that the constraints of Config.3 are more redundant and those of Config.1 are more compact. Details of the performance gaps are shown in Table 7. On each program, Config.3 takes more time than Config.1 does, as Config.3 produces more constraints ($|C|$ in Table 7 denotes the number of constraints of the problem instance). As indicated by Δ (the differences between the Big-M and Or-Relation methods) in Table 7, a big performance gap (i.e., a large Δt) is attributed to a large $\Delta|C|$ and the number of decisive variables of the problem. For example, Make has the smallest Δt, as it has the smallest $\Delta|C|$ and the least number of decisive variables; Flex has the second biggest Δt, as it has the biggest $\Delta|C|$; Grep owns most decisive variables and many solving attempts—even if its $\Delta|C|$ is only 1,536, it still has the biggest Δt. Similarly, we can clearly observe such differences between Config.2 and Config.4 on the five programs. However, some performance gaps are somehow mitigated owing to the effectiveness of the solving method CWMOIP, which skips some unnecessary solving attempts.

Table 7. The Performance Gaps between the Big-M Method and the Or-relation Method for the **Classic Bi-criteria** Problem

Program / Setup	Make		Sed		Gzip		Flex		Grep	
	t(s)	$\|C\|$	t(s)	$\|C\|$	t(s)	$\|C\|$	t(s)	$\|C\|$	t(s)	$\|C\|$
Config.1	1.31	3,833	0.89	995	1.41	1,521	6.4	3,217	6.59	1,803
Config.3	1.4	4,226	1.08	3,044	1.76	5,150	6.93	6,873	7.27	3,339
Δ	0.09	393	0.19	2,049	0.35	3,629	0.53	3,656	0.68	1,536
Config.2	0.87	3,833	0.34	995	0.39	1,521	2.36	3,217	2.85	1,803
Config.4	0.91	4,226	0.4	3,044	0.57	5,150	2.53	6,873	3.31	3,339
Δ	0.04	393	0.06	2,049	0.18	3,629	0.17	3,656	0.46	1,536

Table 8. The Performance Gaps between the Big-M Method and the Or-relation Method for the **Variant Bi-criteria** Problem

Program / Setup	Make		Sed		Gzip		Flex		Grep	
	t(s)	$\|C\|$	t(s)	$\|C\|$	t(s)	$\|C\|$	t(s)	$\|C\|$	t(s)	$\|C\|$
Config.1	1.43	3,833	0.98	995	1.87	1,521	7.04	3,217	7.87	1,803
Config.3	1.52	4,226	1.10	3,044	2.22	5,150	7.88	6,873	8.67	3,339
Δ	0.09	393	0.12	2,049	0.35	3,629	0.84	3,656	0.8	1,536
Config.2	0.76	3,833	0.36	995	0.39	1,521	1.9	3,217	1.2	1,803
Config.4	0.84	4,226	0.42	3,044	0.58	5,150	2.04	6,873	1.57	3,339
Δ	0.08	393	0.06	2,049	0.19	3,629	0.14	3,656	0.37	1,536

6.2.3 Comparison on the Variant Bi-criteria Problem. We also apply these configurations on the variant bi-criteria problem for the five selected programs. As shown in Table 6, we find that: (1) all four configurations yield exactly the same Pareto front for each program; (2) there still exists the performance gap among these configurations. After investigating the Pareto fronts, we find that four of the five programs own a globally optimal solution dominating all other solutions, except the program Gzip. The rationale behind the reduction of the nondominated solutions is that the constraints of the variant bi-objective problem are more strict than those of classic bi-objective problem. As can be seen from Tables 7 and 8, for the same configuration, the number of constraints $|C|$ does not change in these two problems. The only changed part is the right-hand side value of the constraints—*for the top 10% of the statements executed most frequently, each of them needs to be executed at least 10% of the number of times that they were executed by the entire test-suite* [55]. In a classic problem, all statements are required to be executed just *once*, while in a variant problem, the top 10% frequently executed statements are required to be executed at least 10% of the total execution times of the test-suite, which is larger than once in most cases. Hence, many valid nondominated solutions of the classic problem are no longer valid for the variant problem—the size of Pareto front is greatly reduced till one last valid nondominated solution is left in many cases.

When comparing the data in Tables 5 and 6, we can have the following interesting findings:

(1) For Config.1 and Config.3 (the ϵ-constraint-based methods), the solving time for the variant problem is *noticeably longer than* that for the classic problem. For Config.2 and Config.4 (the CWMOIP-based methods), the solving time for the variant problem is *similar to or shorter than* that for the classic problem.
(2) The solving attempts (**#A**) for the ϵ-constraint-based methods are identical on the same program in Table 5 and 6—we can infer $\#\mathbf{A} = |F| + 1$, as the solving attempts is equivalent

to the range of the second objective of the problem (i.e., the size of the fault set plus 1—see Section 5.1). However, for the CWMOIP-based methods, the solving attempts for the variant problem (in Table 6) are generally less than those for the classic problem (in Table 5)—the reason is that the size of Pareto front is reduced (many invalid or dominated solutions could be skipped) and the CWMOIP method becomes more efficient.

The above second finding actually helps to explain the first finding: For the CWMOIP-based methods, the less solving attempts for the variant problem will lead to the less total solving time. For example, for the classic problem on the instance Grep, Config.2 and Config.4 take 20 solving attempts, using 2.85 s and 3.31 s, respectively. However, for the variant problem on Grep, Config.2 and Config.4 take just 6 solving attempts owing to the boost of CWMOIP, using only 1.2 s and 1.57 s, respectively. The above finding also explains that in Tables 7 and 8, the Δt between Config.1 and Config.3 is generally bigger on the variant problem than that on the classic problem, while the Δt between Config.2 and Config.4 is generally smaller on the variant problem than that on the classic problem. Therefore, CWMOIP helps to significantly mitigate the performance gaps between the two formulations (i.e., Big-M and Or-Relation) via skipping unnecessary solving attempts.

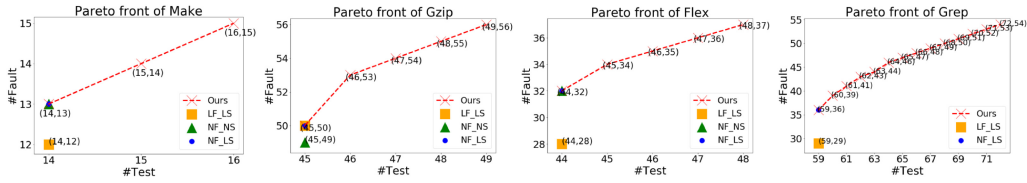
6.2.4 Summary. Throughout the pairwise comparison of the four configurations, we find that these four configurations yield identical Pareto fronts, but there exist performance gaps among them. Generally, Config.2 (Big-M+CWMOIP) is the most efficient method, while Config.3 (Or-relation + ϵ-constraint) is the least efficient method. Comparing the results of the two bi-criteria problems (classic and variant), we observe that ϵ-constraint-based methods take more time on the variant problem than on the classic problem. On the contrary, CWMOIP-based methods take more time on the classic problem than on the variant problem, owing to the boost in less solving attempts owing to CWMOIP.

6.3 Answer to RQ2

6.3.1 Configuration. In this section, we mainly compare the nondominated solutions found by our MOIP methods with one nondominated solution returned by the state-of-the-art single-objective IP (SOIP) methods (LF_LS, NF_NS, and NF_LS in Reference [55]). Strictly, it is not fair to compare the MOIP methods and the SOIP methods using indicators such as HV and spread. Hence, we mainly check whether the nondominated solutions found by the SOIP methods are always included in those found by the MOIP methods. For simplicity, we just use Config.3 (Or-relation + ϵ-constraint, also the least efficient method) to represent our methods. Besides, we will roughly compare the average time of finding a nondominated solution for each method.

6.3.2 Comparisons on the Classic Bi-criteria Problem. As shown in Table 5, except on Sed, there exist multiple nondominated solutions in the Pareto front of each program. Actually, on Sed, our methods, NF_NS, NF_LS all find the only unique global nondominated solution $(12, 25)$, while LF_LS finds a dominated solution $(12, 21)$ that is not optimal. After scrutiny, we find the reason why Sed has only one nondominated solution: (1) using 12 test cases can cover all the statements, (2) using 12 test cases can only find all the 25 faults.*Only under such circumstance (using a set of test cases can optimize both criteria at the same time), there exists a global optimal nondominated solution.* However, such circumstance cannot be achieved on most programs. Therefore, in Figure 5, for a clear comparison, we visualize the solution(s) found by each method for the programs except Sed.

Figure 5 clearly shows the true Pareto front constructed by our method(s) on each program—the leftmost node represents the nondominated solution optimizing the first criterion (using the minimal set of test cases to cover all statements); the rightmost node represents the



(a) Make (b) Gzip (c) Flex (d) Grep

Fig. 5. Pareto fronts and solutions found by different IP methods for the **classic bi-criteria** problem. Note that program Sed is not included above, as it has only one global nondominated solution (12,25) that is found by our methods and NF_LS.

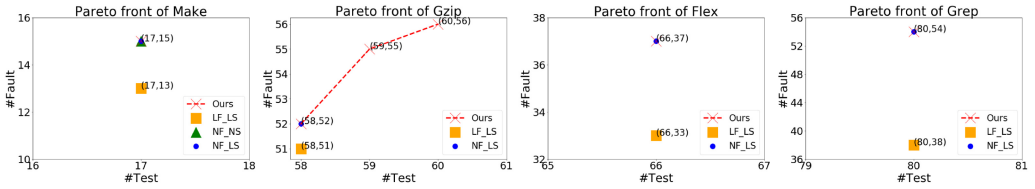
nondominated solution optimizing the second criterion (using a set of test cases to reveal a maximal set of faults); and in-between there exist alternative nondominated solutions that represent some trade-off decisions between two extremes. We can observe the results for other methods: (1) NF_LS is proved to be sound, as it can always find the leftmost node optimizing the criteria on test case size; (2) LF_LS (a.k.a., MINT) is not sound due to its inaccurate formulation, and it always produces the non-optimal solution contained by the Pareto front (below the frontier curve of the true Pareto front); (3) NF_NS is neither sound nor efficient due to its nonlinear solving part, and it finds a non-optimal solution on Gzip and also fails for timeout on Grep. Hence, the above results of existing IP methods exactly comply with the theoretic analysis in Table 2 in Section 3.2.

Regarding *soundness* of our methods, the linear formulation and linear solving parts are both sound in theory. We further write code to recheck the validity of the found nondominated solutions in practice, and all these together verify the soundness of our ILP methods. For the *completeness* of our methods, as illustrated in Figure 5, for each *consecutive* node (along the axle of #Test (number of test cases) or #Fault) between the leftmost and rightmost nodes, we calculate the maximized number of faults for the given number of the test cases. In other words, outside the Pareto front, there will not exist valid yet nondominated solutions.

For example, on the program Flex in Figure 5, there are five nondominated solutions, namely, (44, 32), (45, 34), (46, 35), (47, 36), and (48, 37). The soundness and completeness of the solutions could be checked via the following three steps:

(1) The leftmost solution represents the solution that uses the least number of test cases (44) to cover all the statements and reveal the maximum faults (32) for that number of test cases (44). We can verify this solution by checking whether (43, 32) exists—specifically, using the same constraints and the first objective equal to −43 (to minimize a positive integer is to maximize a negative integer) and the second objective equal to 32. Via the ILP solver, it returns no solution as a result.
(2) The rightmost solution represents the solution that uses the least number of test cases (48) to cover all the statements and reveal all the faults (37). We can verify this solution by checking whether (47, 37) exists. Similarly, the ILP solver returns no solution as a result.
(3) For the nonconsecutive point on the Pareto front, we need to verify whether there exists a better solution. For example, between (44, 32), (45, 34), it is needed to check whether there exists a better solution (44, 33) that uses 44 test cases to cover all statements and reveal 33 faults. Similarly, the ILP solver returns no solution as a result. Hence, (44, 32) is a nondominated solution.

6.3.3 Comparisons on the Variant Bi-criteria Problem. In Figure 6, we visualize the solution(s) found by each method for programs except Sed. Compared with the Pareto fronts in Figure 5, those in Figure 6 are quite small—with only one global nondominated solution found on Make,



(a) Make (b) Gzip (c) Flex (d) Grep

Fig. 6. Pareto fronts and solutions found by different IP methods for the **variant bi-criteria** problem. We include the same four programs to be consistent with Figure 5—program Sed is not included above, on which only one global nondominated solution (32, 25) is found by our methods and NF_LS.

Flex, and Grep, and only three nondominated solutions on Gzip. The reason is that the constraints are quite strict, and it needs many test cases to achieve the required execution frequency. For example, on Flex, for the classic problem, it only needs 48 test cases to reveal all 37 faults and cover all statements, while for the variant problem it needs up to 66 test cases to reveal the 37 faults and meanwhile execute some statements with certain frequencies.

Benefits of knowing the True Pareto Front: Only on Gzip, there exist three nondominated solutions—the leftmost node (58, 52) represents the solution minimizing the test-suite size (at least 58 test cases to satisfy the coverage constraints); the rightmost node (60, 56) represents the solution maximizing the number of revealed faults (totally 56 faults); and the middle node (59, 55) represents a nondominated solution for a trade-off decision. In practice, the solutions (59, 55) and (60, 56) could be more helpful than (58, 52), as adding 1 (or 2) more test cases will help to reveal 3 (or 4) more faults. *Hence, the existing method NF_LS, which focuses on minimizing test-suite size via returning (58, 52), may fail to provide the testing engineers such a broad range of optimization decisions, while our IP methods offer more nondominated trade-off solutions.* For such purpose, building the full Pareto front with our IP methods will be helpful in practice.

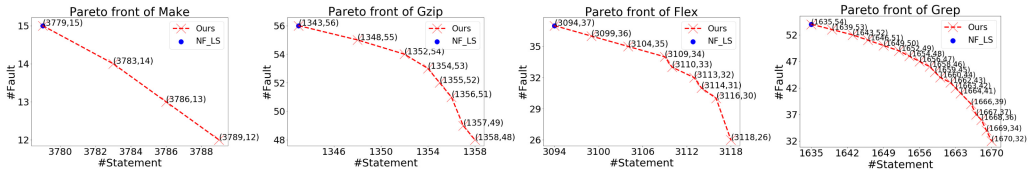
Results of Existing IP methods: Figure 6 shows the following results for existing methods. First, the NF_LS method is still sound and always finds the leftmost node in the Pareto front. It is also efficient, taking one second on each program. Second, the NF_NS method only succeeds on the smallest program (i.e., Make) and fails for timeout on other programs. The reason is that the variant bi-criteria problem has more complicated constraints, and it usually takes more time in solving than that in the classic bi-criteria problem. Third, the LF_LS method always finds the suboptimal solutions for the four programs in Figure 6, due to its inaccurate formulation.

6.3.4 Comparisons on the Tri-criteria Problem. As observed and explained in Section 6.2, Big-M + CWMOIP (Config.2) exhibits to be the most efficient method in solving the classic and variant bi-criteria problem, and Or-relation + ϵ-constraint (Config.3) is proven to be the least efficient. So, these two methods represent the upper limit and lower limit of our IP methods. We adopt these two methods to repeat the experiments that are conducted on the tri-criteria problem in Reference [55], using the same subject programs and parameters (i.e., the required size of the reduced test-suites). The experimental results are shown in Table 9.

In Table 9, **test size** denotes the required ratio of the reduced test-suite size to the original size; **t(s)** denotes the execution time in seconds; $|\mathcal{P}|$ denotes the size of the Pareto front; **#A** denotes the times of solving attempts; $|C|$ denotes the size of the constraints of inequalities or equities (K referring to one thousand unit). Experimental results show the following facts: (1) Config.2 and Config.3 find the Pareto fronts of the same size for any given scenario. After scrutiny, it is confirmed they yield the same solutions. (2) In general, Config.2 is 10–50 times faster than Config.3 on this problem, which shows a larger performance gap than that on the bi-criteria problem. The

Table 9. Effectiveness of the Fastest (Config.2) and the Slowest (Config.3) Methods of Ours on the **Tri-criteria** Problem

Test Size	Setup	Make				Sed				Gzip				Flex				Grep			
		t(s)	$\|\mathcal{P}\|$	#A	$\|C\|$	t(s)	$\|\mathcal{P}\|$	#A	$\|C\|$	t(s)	$\|\mathcal{P}\|$	#A	$\|C\|$	t(s)	$\|\mathcal{P}\|$	#A	$\|C\|$	t(s)	$\|\mathcal{P}\|$	#A	$\|C\|$
5%	Config. 2	26	4	9	7.6K	7	1	6	1.9K	9	8	13	2.9K	121	9	15	6.3K	111	19	24	3.4K
	Config. 3	388	4	16	395K	64	1	26	183K	154	8	57	145K	2,834	9	38	934K	2,248	19	55	599K
10%	Config. 2	16	1	6	7.6K	7	1	7	1.9K	8	5	10	2.9K	80	1	7	6.3K	35	1	7	3.4K
	Config. 3	369	1	16	395K	61	1	26	183K	154	5	57	145K	2,498	1	38	934K	1,984	1	55	599K
15%	Config. 2	13	1	7	7.6K	6	1	7	1.9K	8	1	7	2.9K	72	1	7	6.3K	33	1	7	3.4K
	Config. 3	355	1	16	395K	61	1	26	183K	141	1	57	145K	2,564	1	38	934K	1,999	1	55	599K
20%	Config. 2	11	1	7	7.6K	5	1	7	1.9K	7	1	7	2.9K	61	1	7	6.3K	37	1	7	3.4K
	Config. 3	296	1	16	395K	53	1	26	183K	121	1	57	145K	2,236	1	38	934K	1,792	1	55	599K



(a) Make (b) Gzip (c) Flex (d) Grep

Fig. 7. Pareto fronts and solutions found by our methods and NF_LS for the **tri-criteria** problem when the sizes of the reduced suites are 5%. Note that LF_LS and NF_NS are not compared due to their bad results reported in Reference [55]. Besides, program Sed is not included above, as it only has one global nondominated solution ($|\mathcal{P}| = 1$).

reason is that the tri-criteria problem is more complicated. (3) As the constraint on test-suite size becomes relaxed (e.g., from 5% to 10%), more statements can be covered and almost all faults will be revealed. When test-suite size is not used as a constraint, the two objectives (revealing more faults and covering more statements) are not competing—the Pareto front will have only one solution.

For the scenarios $|\mathcal{P}| = 1$ (i.e., only one globally optimal solution) in Table 9, we compare the solution found by our methods with that found by NF_LS. Results prove that NF_LS and our methods can always find the globally optimal solution, indicating both of them are sound. For the scenarios $|\mathcal{P}| \neq 1$ (i.e., there exists alternative optimal solutions), in Figure 7, we visualize the solution(s) found by our methods and NF_LS for programs except Sed, when the reduced test-suite size is 5%. As shown in Figure 7, we observe that NF_LS can always find the leftmost node in the Pareto fronts of these programs. This fact implies that NF_LS is actually maximizing the single objective of the revealed faults, not maximizing the covered statements. However, the tri-criteria problem is meant to maximize both or balance them when they compete (see Definition 7 in Section 2.3).

In contrast, our methods aim to find all the alternative nondominated solutions for each program. On the first three programs (Make, Gzip, and Flex), our methods return the consecutive nodes (nondominated solutions) between the leftmost and rightmost nodes. On the last program (i.e., Grep), on the Pareto front of 19 nodes, there exist some nodes such as (1664, 41) and (1666, 39) that are not consecutive on x-axis or y-axis. After inspecting the details, we find that (1665, 40) is not solvable—there would not exist a solution that can cover 1,665 statements and reveal 40 faults. Therefore, our methods are *sound* and capable in building the *complete* Pareto fronts for the tri-criteria problem.

Table 10. Efficiency of Different Methods on the Classic Bi-criteria and Tri-criteria Problems

Setup	Make			Sed			Gzip			Flex			Grep		
	$t_2(s)$	$t_3(s)$	Ratio	$t_2(s)$	$t_3(s)$	Ratio	$t_2(s)$	$t_3(s)$	Ratio	$t_2(s)$	$t_3(s)$	Ratio	$t_2(s)$	$t_3(s)$	Ratio
Config. 2	0.87	16	18	0.34	7	21	0.39	8	21	2.36	80	34	2.85	35	12
Config. 3	1.4	369	264	1.08	61	56	1.76	154	88	6.93	2,498	360	7.27	1,984	273
LF_LS	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
NF_NS	20	n/a	n/a	21,854	n/a	n/a	2,813	n/a	n/a	1,708	n/a	n/a	n/a	n/a	n/a
NF_LS	1	2,673	2,673	1	2,976	2,976	1	1,565	1,565	1	43,608	43,608	1	14,099	14,099

6.3.5 Performance Comparison. We collect the execution time of different methods on the classic bi-criteria problem and the tri-criteria problem and compare them in Table 10. Note that we also include only Config.2 (Big-M+CWMOIP) and Config.3 (Or-relation + ϵ-constraint), which represent the fastest and slowest configuration, respectively. Besides, $\boldsymbol{t_2(s)}$ denotes the execution time for the classic bi-criteria problem, $\boldsymbol{t_3(s)}$ denotes the execution time for the tri-criteria problem, and **Ratio** denotes for the ratio of $\boldsymbol{t_3(s)}$ to $\boldsymbol{t_2(s)}$. As the execution time of the variant bi-criteria problem is quite close to that of the classic bi-criteria problem, we do not include it in Table 10.

As shown in Table 10, LF_LS seems to be the most efficient method among all, using one second for solving in all cases—the reason is that it performs one attempt of ILP solving and adopts only a number of $|T|$ decisive variables, no matter for classic bi-criteria or tri-criteria problem. Considering LF_LS is running on NEOS servers, the core solving part should take less than one second if the time for networking package transition is deducted. However, LF_LS is not sound, as the solutions are usually suboptimal and dominated by other methods (see Figures 5 and 6 for examples). On the contrary, NL_NS is the most time-consuming method, which cannot be finished in eight hours and causes timeout issues in most scenarios. Hence, only NF_LS and our methods represent the reasonably effective methods, for which it is worth investigating further.

Interestingly, on the classic bi-criteria problem, the performance of NF_LS seems to be slightly worse than Config.2 and slightly better than Config.3. Nevertheless, their purposes are different—our methods aim to find the full Pareto front with dozens of ILP solving attempts, while NF_LS aims to find one nondominated solution minimizing the test-suite size. As NF_LS executes ILP solving just once, it can be finished within one second when the problem size is not too big. *As the number of decisive variables in NF_LS is $|T| \times |F|$ for the bi-criteria problem, at most around 40 thousands of auxiliary variables are needed on Grep, and the problem is still solvable in one second by NEOS server or modern LP solvers. In contrast, our methods require just a linear number of decisive variables (i.e., $|T| + |F|$) for the bi-criteria problem, but may have several to dozens of ILP solving attempts.* Therefore, for the bi-criteria problem, our methods (ranging from the slowest Config.3 to the fastest Config.2) are comparable with NF_LS in terms of scalability. Note that, regarding solving efficiency for per optimal solution, our MOIP methods perform better, as seeking each nondominated optimal solution via our MOIP-based methods is like applying a SOIP-based method (e.g., NF_LS) once.

On the tri-criteria problem, the performance of NF_LS seems to be unsatisfactory, indicating the fact that the inherent complexity of the nonlinear formulation hinders the scalability of NF_LS. Since NF_LS aims to conduct the ILP solving just once, the overheads mainly come from the huge size of decisive variables introduced by NF_LS. After looking into the implementation of NF_LS, we find that it may introduce hundreds of thousands of auxiliary variables—*the theoretic upper limit could be as large as $|T| \times |F| + |T| \times |S|$, as a number of $|T| \times |F|$ auxiliary variables are introduced to model the dependency between test cases and faults, and other $|T| \times |S|$ auxiliary variables are further introduced to model the dependency between test cases and statements.* Compared with such

overheads brought by the nonlinear formulation in NF_LS, the benefits of our linear formulation methods (Big-M and Or-relation) gradually exhibit. In Table 10, the ratio for NF_LS ranges from 1,565 to 43,608, while it only ranges from 56 to 360 for Config.3 and from 12 to 34 for Config.2. *The small ratio values are attributed to the complexity of linear formulation for the tri-criteria problem, namely,* $|T| + |F| + |S|$. Hence, from the bi-criteria to the tri-criteria problem, the complexity (i.e., the size of decisive variables) of the NF_LS method increases from $|T| \times |F|$ to $|T| \times |F| + |T| \times |S|$, while the complexity of our methods increases from just $|T| + |F|$ to $|T| + |F| + |S|$.

6.3.6 Summary. Among all IP methods, LF_LS is efficient but not sound, i.e., always yielding a suboptimal solution. NF_NS is neither sound nor efficient, as it adopts the nonlinear solving part that is time-consuming and cannot guarantee the optimality of found solutions. NF_LS, as a sound method, can always find the leftmost solution inside the Pareto front (optimizing one objective among multiple), but it is not efficient on the tri-criteria problem. Overall, our methods, represented by Config.2 (Big-M+CWMOIP), are generally superior to NF_LS in terms of the completeness of the Pareto front and the performance. Especially, on the tri-criteria problem, the Config.2 method can build the complete Pareto front by using only several-hundredths of the time that is required by NF_LS for obtaining one nondominated solution.

6.4 Answer to RQ3

Besides the comparison between existing IP methods and our methods, we also want to compare our ILP methods with the state-of-the-art MOEAs. MOEAs, as the meta-heuristic algorithms, typically cannot compete with the existing IP methods on the small problem instances for finding an exact solution (running the evolution process itself may take several seconds, within which the solving of IP methods can be done on the small instances). However, MOEAs are scalable and suitable for the big problem instances with a huge solution space. In such cases, we apply two MOEAs (i.e., NSGA-II [23] and MOEA/D [105]) on the tri-criteria problem and compare the results with ours.

6.4.1 The Relaxed Formulation of Tri-criteria Problem Using MOEAs. As can be seen in Equation (30), two criteria (i.e., the number of covered statements and the number of revealed faults) are adopted as objectives, while the criteria of the size of reduced test-suite is used as the constraint. Different from the single objective evolution algorithms that support constraint solving during the evolution process (e.g., NPGA [18]), MOEAs cannot explicitly support constraint solving. If all found solutions that cannot satisfy the constraint on the size of reduced test-suite are abandoned during evolution, MOEAs need to regenerate a large number of candidate solutions, and the evolution process will thus take a very long time. Under such circumstance, we have to relax the problem—converting the constraint on the size of the reduced test-suite (in Equation (33), $\sum_{i=1}^{n} t_i = \eta n$) to the third objective minimizing the gap between the actual size ($\sum_{i=1}^{n} t_i$) and the desired size of the reduced test-suite (ηn). Furthermore, as all the objectives are usually to be optimized in MOEAs, we can now formally formulate this tri-criteria problem using MOEAs as:

$$\begin{aligned} \text{Max} \quad & O_1(T) = \sum_{k=1}^{p} s_k, \\ \text{Max} \quad & O_2(T) = \sum_{j=1}^{q} f_j, \\ \text{Max} \quad & O_3(T) = -|\eta n - \sum_{i=1}^{n} t_i|, \end{aligned} \tag{33}$$

where n denotes the number of original test cases, p denotes the number of the statements to cover, q denotes the number of faults to reveal, η denotes the targeted percentage of the test-suite, s_k is a binary variable indicating whether kth statement is covered by the reduced test-suite from T, f_j is a binary variable indicating whether jth fault is revealed by the reduced test-suite from T, and t_i is a binary decisive variable indicating whether ith test case from T is selected into the reduced test-suite.

The constraint on the test-suite size in Definition 7 is a strict constraint. Different from the single objective evolution algorithms that support constraint solving during the evolution process (e.g., NPGA [18]), MOEAs cannot explicitly support constraint solving. If all found solutions that cannot satisfy the constraint on the size of reduced test-suite are abandoned during evolution, MOEAs need to regenerate a large number of candidate solutions and the evolution process will thus take very long time (sometimes, it will hang). Hence, the relaxed tri-criteria formation is proposed for a practical and efficient application of MOEAs. Generally, this formulation relaxation idea is inspired by the MOEA application on the optimal feature selection problem in SPLC [82], which treats the constraints on correctness (the number of satisfied feature dependencies) as an objective to enable gradual evolution from invalid solutions to valid solutions, as MOEAs cannot directly handle constraint solving by itself.

6.4.2 MOEA Implementation and Experiment Setup. As methods presented by Reference [55] and our ILP methods are all implemented in Python, we also implement the MOEA solutions based on the Python language and libraries for ease of comparison. Specifically, we adopt Pygmo 2.1 [22], the Python version of Pagmo, which is a widely used scientific library for massive parallel optimization (e.g., multi-objective optimization). It supports 13 common SOEAs, which include Particle Swarm Optimization (PSO), Artificial Bee Colony (ABC), Corana's Simulated Annealing (SA), and so on. It also supports two MOEAs, namely, NSGA-II and MOEA/D, which are exactly the ones to be experimented in our study. For problem encoding, the original test-suite T is mapped to a binary bit vector, where the number of bits is equal to the number of test cases in T. If the i-bit value is *true*, then the ith test case (t_i) is selected, otherwise t_i deselected. The fitness functions are also straightforward: Knowing which test cases are selected can help to calculate the covered statements and revealed faults, as such knowledge can be aggregated from the information in the *cov.info* and *fault.info* files of each target program. Notably, in this study, we just use the by-default encoding and the standard MOEAs (NSGA-II and MOEA/D) to serve as the baseline tool for the tri-criteria MCTSM problem.

For parameters setup, we set the number of total evaluations as 200K for each MOEA. As we set the population size as 100 (the same as that in Reference [109]) and the maximum number of generations as 2K. Zheng et al. [109] have investigated several MOEAs (namely, Greedy, NSGA-II, and MOEA/D) on the MCTSM problem. They tuned up the parameters and finally chose population size as 100 and the total evaluations ($evaluations = population\ size * generation\ number$) as 100K. Hence, the best parameters from the previous practice are population size as 100 and generation number as 1K. Compared to the total 100K evaluations in Reference [109], we double the number of evaluations to assure that the MOEAs will converge on different target programs. Besides, we use the search operators of the standard NSGA-II and MOEA/D, the same as those used in Reference [109].

For the quality indicator of results, we adopt the built-in quality indicator of Pygmo, namely, HV, to compare the two MOEAs. The reasons to choose HV as the only indicator are twofold: first, it is observed to be the most frequently used performance indicator in evolutionary multi-objective optimization (EMO) conferences [77]; second, without the knowledge of the true (or approximated) Pareto front, calculating HV is comparatively easier than calculating other indicators, e.g.,

Table 11. Comparison of NSGA-II and MOEA/D on the **Relaxed Tri-criteria** Problem

Test Size	Tri-obj	Make		Sed		Gzip		Flex		Grep	
		HV	t(s)	HV	t(s)	HV	t(s)	HV	t(s)	HV	t(s)
5%	NSGA-II	**1**	**141**	**1**	**66**	**0.999**	**76**	**1**	**303**	**0.994**	**275**
	MOEA/D	0.787	257	0.77	127	0.486	197	0.52	1,211	0.51	871
10%	NSGA-II	**1**	**184**	**1**	**100**	**1**	**89**	**1**	**424**	**1**	**298**
	MOEA/D	0.97	276	0.929	138	0.522	212	0.582	1,244	0.531	842
15%	NSGA-II	**1**	**259**	**1**	**137**	**1**	**108**	**1**	**577**	**1**	**402**
	MOEA/D	0.965	314	0.973	151	0.623	210	0.565	1,295	0.591	850
20%	NSGA-II	1	338	**1**	165	**1**	**134**	**1**	**778**	**1**	**511**
	MOEA/D	0.989	333	0.983	164	0.831	208	0.691	1,303	0.665	860

The **bold** font highlights the average HV value that is significantly better and the average execution time that is significantly shorter, according to $p < 0.001$ in Mann–Whitney U test [57] on basis of 30 runs.

generational distance (GD), inverted generational distance (IGD), epsilon, and so on. Given several reference points, HV can be efficiently calculated for a set of returned solutions. In our case, experiments in Section 6.3.4 have found some best and worst solutions on the first two objectives (i.e., maximizing the covered statements and maximizing the revealed faults), and we can use these solutions for this relaxed problem as reference points. Note that the indicator *spread* is not suitable for this problem, as we find many cases where only one global optimal solution exists (e.g., for the program Sed).

6.4.3 Analysis of Results of MOEAs. Table 11 shows the average HV and t(s) of applying the two MOEAs on the relaxed tri-criteria problem for 30 runs, where **HV** denotes the normalized Hypervolume and **t(s)** denotes the execution time in seconds of an algorithm. We observe three facts: (1) *In most cases,* NSGA-*II produces better results than MOEA/D.* Their differences are still not huge on the small programs (e.g., Make and Sed), while the performance gap could be quite huge on the two big programs (e.g., Flex and Grep)—MOEA/D may use up to triple execution time of NSGA-II but only attain half of the HV of NSGA-II. (2) *As the allowed test-suite size (η) increases from 5% to 20%, the performance gap between the two MOEAs is gradually mitigated.* For example, the two MOEAs produce significantly different results on Make for $\eta = \{5\%, 10\%, 15\%\}$, but they produce statistically similar results on Make for $\eta = 20\%$. In other words, they both find the optimal solutions in this case—the rationale is that the number of valid optimal solutions is becoming larger as the test-suite size to be allowed becomes big. Hence, the difficulty in searching valid solutions decreases, the gaps between different MOEAs become less notable. (3) For the cases where there exists a global optimal solution (see the cases with $|\mathcal{P}| = 1$ in Table 9), we find NSGA-II can reach the sole optimal solution in most situations. In contrast, MOEA/D seems to achieve the great diversity of the solutions and keep in evolution many solutions that are not good at O_3 in Equation (33)—*the weak convergence on O_3 finally makes MOEA/D fail to find the global optimal solution.*

The above experiments show the basic capability of the *standard* state-of-the-art MOEAs in solving the tri-criteria problem. For the usage of the two MOEAs, the importance is to assure that each MOEA is given enough generations to allow the convergence, which avoids the underestimation of a MOEA.

6.4.4 Parameters Tuning in Two MOEAs. In addition to using population size as 100 (the same as that in Reference [109]) and the maximum number of generations as 2K, we also tune up these two parameters on the tri-criteria problem. We find that parameters tuning fails to help find more

Table 12. The Number of Hitting Runs (h) That Find the Nondominated Solutions for the Tri-criteria Problem When the Sizes of the Reduced Suites Are **10%**, **15%**, and **20%**

Test Size η	Tri-criteria	Make h	Sed h	Gzip h	Flex h	Grep h
10%	NSGA-II	27	30	28	30	13
	MOEA/D	8	3	0	0	0
15%	NSGA-II	30	30	30	30	30
	MOEA/D	16	15	0	0	0
20%	NSGA-II	30	30	30	30	30
	MOEA/D	28	20	0	0	0

Note that the following 14 cases all have one global nondominated solution—except on Gzip with $\eta = 10\%$ where exist five nondominated solutions ($|\mathcal{P}| = 5$, see Table 9). For the case Gzip with $\eta = 10\%$, if a run of the MOEA can cover any nondominated solution from the true Pareto front, we consider it a hitting run.

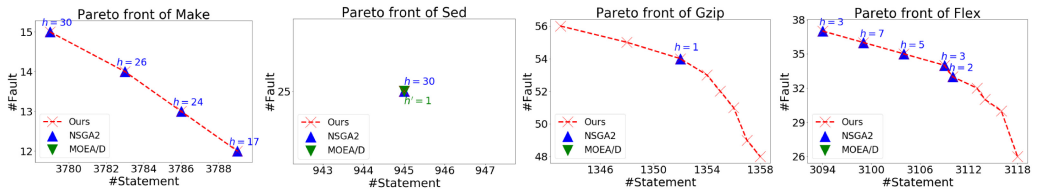
Table 13. Parameters Tuning-up for the Two MOEAs: The Number of Hitting Runs (h) Out of 30 Runs, Which Find the Nondominated Solutions for the Tri-criteria Problem When the Sizes of the Reduced Suites is **20%**

Test Size η	Tri-criteria	Population	Generations	Make h	Sed h	Gzip h	Flex h	Grep h
20%	NSGA-II	50	2,000	30	30	30	30	30
	MOEA/D	50	2,000	9	7	0	0	0
	NSGA-II	100	1,000	30	30	30	30	30
	MOEA/D	100	1,000	8	7	0	0	0
	NSGA-II	**100**	**2,000**	**30**	**30**	**30**	**30**	**30**
	MOEA/D	**100**	**2,000**	**28**	**20**	**0**	**0**	**0**
	NSGA-II	100	3,000	30	30	30	30	30
	MOEA/D	100	3,000	17	16	1	2	0

Note that the following five cases all have one global nondominated solution with $\eta = 20\%$.

nondominated solutions in most cases, even for the easiest problem instance when $\eta = 20\%$. As shown in Table 12, as the size of the reduced suite (η) increases, the constraint becomes relaxed and the two MOEAs can more easily find the nondominated solution(s). Hence, we want to first tune up parameters for the problem instance with $\eta = 20\%$. If parameter tuning cannot help much in the problem instance with $\eta = 20\%$, it would be highly unlikely that it helps in other problem instances with $\eta = 5\%$, 10%, or 15%.

In general, we observe three facts from Table 13: (1) The parameters tuning does not affect NSGA-II, as it already converges to find the global nondominated solution in all cases using 100K evaluations. (2) Neither population size nor generation number can determine alone, and only their product (i.e., the total evaluations) affects—the first four rows in Table 13 prove that setting population size as 50 and generation number as 2K yields almost same results as setting population size as 100 and generation number as 1K. (3) For MOEA/D, using more evaluations may not help to find the nondominated solutions—the last row (using 300K evaluations for MOEA/D) in Table 13 actually performs worse than the last row but two (using 200K evaluation for MOEA/D) on program Make and Sed. Hence, we can summarize that the parameters used in Section 6.4.3 (i.e., population size as 100 and generation number as 2K) already yield the best results.



(a) Make (b) Sed (c) Gzip (d) Flex

Fig. 8. Pareto fronts and solutions found by our methods and the two MOEAs for the **tri-criteria** problem when the sizes of the reduced suites are **5%**, where h (or h') denotes the number of hitting runs out of 30 runs for a nondominated solution by NSGA-II (or MOEA/D). Note that program Grep is not included above, as neither NSGA-II nor MOEA/D finds any nondominated solution out of 30 runs.

6.4.5 Comparison between MOEAs and Our ILP Methods. The results presented in Section 6.4.3 actually are for the *relaxed* tri-criteria problem, which treats the criteria on the size of reduced test-suite as an objective rather than a strict constraint. To have a fair comparison with our IP-based methods that find solutions that strictly satisfy the constraints, we will keep just the solutions with $O_3 = 0$ (i.e., the valid solutions found by the two MOEAs for the tri-criteria problem). The intuition for keeping invalid solutions in evolution and removing them from final results is that an invalid ($O_3 \neq 0$) solution in early generations has chance to be mutated to a valid ($O_3 = 0$) solution at the end of evolution. To sum up, although making the size of reduced test-suite as O_3 in the relaxed formulation, we still logically treat it as a constraint in the final results.

In Figure 8 and Table 12, we show how NSGA-II and MOEA/D cover the true Pareto front built by our IP-methods for the tri-criteria problem, when the size of the reduced test-suite grows from 5% to 20%. As there often exist various nondominated solutions when $\eta = 5\%$ on these programs, we use the scatter-plot to visualize the coverage and the number of hitting runs of the Pareto front. However, when $\eta = 10\%$, 15%, and 20%, since there exists one globally nondominated solution in most cases, we just list the number of hitting runs in Table 12.

As shown in Figure 8, we can observe the following three facts: (1) In general, the MOEAs cannot fully cover the whole Pareto front built by our IP-methods on most programs. Only on Make and Sed, NSGA-II seems to be quite effective—15 out of 30 runs cover the whole Pareto front on Make (i.e., four nondominated solutions in Figure 8(a)) and 30 out of 30 runs cover that on Sed (i.e., one solo nondominated solution in Figure 8(b)). However, when the search space becomes large on other programs, NSGA-II or MOEA/D cannot fully cover the true Pareto front in any run. NSGA-II, as the better one, could at most partially cover several points in the true Pareto front (see that in Figure 8(c) and Figure 8(d)). Unfortunately, none of them could cover any point in true Pareto front on Grep. (2) MOEA/D seems to be quite ineffective in finding the nondominated solutions. For 30 runs on each of the five programs, MOEA/D only finds the solo nondominated solution on Sed for once ($h' = 1$ in Figure 8(b)), while it fails to find any nondominated solution in other cases. As analyzed in Section 6.4.3, MOEA/D is prone to retaining many solutions with $O_3 \neq 0$—or, to put it another way, we find that MOEA/D has a slower convergence on O_3 than on O_1 and O_2. Hence, the results in Table 11 and Figure 8 indicate that solutions found by MOEA/D are often dominated by those by NSGA-II and our IP-methods. (3) In general, NSGA-II has more hitting runs for points on the left side in the true Pareto than those on the right side—for example, on Make in Figure 8(a), it hits the leftmost point (3779, 15) for 30 times while the rightmost point (3789, 12) for 17 times. The rationale is that the solution space of O_1 (e.g., 15 faults in Make) is much larger than that of O_2 (e.g., 3,803 statements in Make).

We can gain the following findings from Table 12: First, NSGA-II performs quite well when the solution space is less constrained, after η is increased to 10%–20%. This is observed from the fact that NSGA-II only hits a nondominated solution less than 10 times on Gzip and Flex in Figure 8, while it hits about 30 times in Table 12. However, in most cases when η = 10%–20%, there exists one solo nondominated solution. NSGA-II seems to be good at finding such a globally optimal solution. Note that for η = 10% on Gzip, if NSGA-II hits any of the five nondominated solutions inside the true Pareto front, we consider it a hit case. Second, MOEA/D gradually yields better results on small programs as η increases, but still fails for big programs. For example, on Make, the number of hitting runs h is 8 when η = 10%, and it is much improved (h = 28) when η = 20%. Since 28 out of 30 runs of MOEA/D find the globally optimal solution and 30 out of 30 runs of NSGA-II find that, this fact explains that there are no significant differences for HV on Make with η = 20% in Table 11. However, MOEA/D cannot find any nondominated solutions on Gzip, Flex, and Grep.

6.4.6 Summary. In general, NSGA-II is capable in finding the solo nondominated solution in cases where η = 10%–20%, while it could only partially cover the true Pareto front (built by our IP-methods) when η = 5%, especially on Gzip and Grep basically unable to hit any nondominated solution in Pareto front. MOEA/D is even worse, being just able to find the solo nondominated solution when η = 10%–20% on Make and Sed. Regarding efficiency, our IP-methods are very fast, solving the problem by at least 7 seconds and at most 35 seconds for Config.2 (our fast method). In contrast, MOEAs is much less efficient, one execution taking at least 60 seconds and at most 700 seconds. To summarize, our IP-methods could build up the *complete* Pareto front, while being at least 10 times faster than two adopted MOEAs in our experiments.

7 DISCUSSION

In this section, we first briefly review the threats to validity in the experimental study. Furthermore, we summarize the merits and applicability of MOEAs and MOIP methods. Last, we conclude the generality of MOIP via the current research on the MCTSM problem in regression testing and on the optimal feature selection problem in SPLE.

7.1 Threat to Validity

Threats to external validity are twofold: the selection of experimental subjects and the choice of objectives for optimization, which may affect the extent to which the results could be generalized from our experiments. Regarding the experimental subjects, we select exactly the same systems that have been used and published by Lin et al. [55] to facilitate direct comparison. There are other subject systems that are used in Reference [109], including schedule-v2, tcas-v1, and tot_info-v1. However, the information on coverage and faults of these systems is not published. In the future, to evaluate on more subject systems, we will make endeavors to collect the coverage information via GCOV and the faults via checking the bug-tracking systems (if available) or generating faults by Trivia Compiler Equivalence (TCE) [72].

The second threat to external validity comes from the objectives for optimization. Currently, we mainly limit our study to the **three criteria** of the TSM problem—that is, the statement coverage, the reduced test-suite size, and the number of revealed faults. No matter the classic/variant bi-criteria problem or the tri-criteria problem, they use the above three types of information as constraints or objectives. In reality, to minimize the test-suite, it is also feasible to consider other types of code coverage, e.g., branch coverage [75], Modified Condition/Decision Coverage (MC/DC) [1], and so on. One concern is that importing various types of coverage will inevitably raise the question on the effectiveness of different coverage types, which is out of the scope of our study. Hence,

our study focuses on the statement coverage and on the modeling and solving of the MCTSM problem itself.

The threat to internal validity stems from the choice and setup of the MOEAs evaluated in our experiments. In Section 6.4, we compare our MOIP methods with the two MOEAs. The MOIP methods are all deterministic and require no parameter configurations. On the contrary, MOEAs are nondeterministic and require deliberate configuration—the randomness of the MOEAs by nature as well as the setup parameters such as population size, generation number, ratios of different evolutionary operations, and so on. First, to have a fair comparison with Nemo, we implement our IP methods in Python. Meanwhile, we also implement the MOEA solutions (for NSGA-II and MOEA/D) with the Python library Pygmo, which is a popular MOEA library in the GitHub community and has about 250 stars and 60 forks. Second, to mitigate the effect of randomness of MOEAs, we repeat the experiment 30 times and compare on the basis of the average normalized HV and execution time. Third, to cope with the combinatorial explosion of parameter options, we refer to Reference [109] for the population size and the total number of evaluations and use the default setup of Pygmo for other parameters. As shown in Table 12, our confgurations allow the convergence of MOEAs—NSGA-II can find the nondominated solutions in most cases, while MOEA/D cannot succeed on those large programs due to its own issues. Even given a larger population size and more generations, we find that MOEA/D can be only slightly improved. In the future, it will be interesting to tune up and enhance the existing MOEAs (especially MOEA/D) to yield better results.

7.2 Answers to the Unsolved RPs

After the evaluations, we shed light on the answers to the unsolved RPs in Section 1.

7.2.1 Answer to RP1. Clearly, we can answer that the MCTSM problem is not necessary to be formulated as an INP problem in Reference [55]. Instead, it can be ideally modeled as an ILP problem. As shown in Section 4, there are at least two linear modeling methods, namely, the Big-M method and the OR-relation method. In Section 4.4, we also prove the equivalence of the two modeling methods in theory. Furthermore, throughout the evaluations on RQ1, we prove that the two methods (Config.1 and Config.3 for ϵ-constraint, Config.2 and Config.4 for CWMOIP) always yield the same results on the five evaluated programs. In addition, as shown in Table 3, it is also not necessary to make use of a number of $|T| \times |F|$ decisive variables for the classic bi-criteria problem. In contrast, the two linear modeling methods applied by us just require a number of $|T| + |F|$ decisive variables. Hence, to summarize, for the MCTSM problem with linear objectives, we can model it as an ILP problem and solve it with a linear number of decisive variables.

7.2.2 Answer to RP2. By adopting the two solving methods (i.e., ϵ-constraint and CWMOIP) in Section 5, we avoid the weighted-sum approach (i.e., scalarizing multiple criteria to one objective) and implement a general approach that aims at finding all nondominated solutions in the Pareto front of the MCTSM problem. According to our previous study on applying MOIP methods to the optimal feature selection problem in Reference [98] and the evaluation results for RQ1 in Section 6.2, these two solving methods yield the same results for a given model, just having some performance gaps (CWMOIP being significantly faster than ϵ-constraint). Furthermore, based on the results for RQ2 in Section 6.3, we find that the existing weighted-sum approaches (i.e., LF_LS, NF_NS, and NF_LS) can only find at most one nondominated solution in the Pareto front, while our solving methods can find all valid nondominated solutions in the Pareto front, which helps achieve the completeness of the set of sound solutions.

7.2.3 Answer to RP3. For the tri-criteria TSM problem, the NF_LS method proposed in Reference [55] needs a number of $T| \times |F| + |T| \times |S|$ decisive variables and hence takes a considerably long solving time. We also apply the standard MOEAs, namely, NSGA-II and MOEA/D to the *relaxed* tri-criteria TSM problem (as MOEAs cannot directly handle the constraints on the test-suite size). Results of RQ3 in Section 6.4 show that no matter the NF_LS method or the standard MOEAs, in many cases, cannot find all the nondominated solutions. The reasons behind that are twofold: (1) The NF_LS method applies the weighted-sum approach and hence only targets for one optimal solution—actually, such a solution is just a nondominated solution in the Pareto front, if treating the MCTSM in a way of MOO. (2) The standard MOEAs are essentially heuristic approaches, which are actually no guarantee of the non-dominance and the completeness of the found solutions. As shown in Table 12, if the MCTSM instance is small, NSGA-II seems to find the nondominated solutions in most runs, but the number of hitting-runs out of 30 (i.e., the probability) decreases when the size of MCTSM instance increases. In contrast, on these programs that are also used in References [37, 55], our MOIP methods can find the complete set of nondominated solutions in all cases.

7.3 Assumption and Benefits of Practical Application of MOIP Methods on MCTSM

To enable the practical application of our MOIP methods on MCTSM in real-world industry scenarios, it is important to understand the assumption, benefits, and limitations of our MOIP methods.

7.3.1 Assumption of Practical Application. In general, TSM is usually encountered in a scenario of regression testing—the test-suite has been executed before and now it requires to be reduced for saving the cost to rerun it. Hence, *in a TSM problem, it is often assumed that the information on fault detection and statement coverage of each test case is available.* For example, on servers of data centers of Microsoft, many internal tools or packages require to be rebuilt and retested online and daily (for purpose of fast integration of new features from CodeFlow).[2] To mitigate the performance burden on servers, the test-suite to rerun daily and online needs to be reduced. However, the information collection and test-suite reduction process could be done offline. In other words, a TSM problem we encounter is usually a white-box TSM problem rather than a black-box one that is addressed in Reference [4]. Even without knowing the information of the whole test-suite, testing engineers can resort to some tools to get such information. For example, the tool gcov [29] can be adopted to collect the coverage information; the fault information can be retrieved via the bug tracking (e.g., Bugzilla) or version control systems (e.g., Git or CodeFlow), and sometimes even a simple execution of the whole test-suite can yield the fault information relevant to test cases. After all, the central concern of the TSM problem is to save the rerunning cost of the test-suite at deployment or production time, while the offline preprocessing of the whole test-suite for collecting necessary information is not the major concern.

7.3.2 Benefits of Practical Application. The benefits brought by the practical application of our MOIP methods may come from three aspects. First, compared with the complicated non-linear formulation of NF_LS, our formulations (i.e., the Big-M and OR-relation in Table 3) are easy to understand and apply. Hence, testing engineers should have a comparatively low learning curve in understanding the formulations and applying them for the real-world MCTSM instances. Second, we implement and publish the MOIP solving methods (i.e., ϵ-constraint and CWMOIP) as an open-source Python library, which could be easily reused and extended by testing engineers for their own problem. Last but not the least, the true Pareto front built by our MOIP methods could

[2]This is based on the observation from the first author when he was working for seven months in O365 production group, Microsoft.

be useful in practice, especially helping testing engineers to conduct the on-demand test-suite reduction [35]—some extent of test-suite reduction (minimization) reduces the cost of testing at the price of compromising the capability of fault revealing, and testing engineers need to answer to what size the test-suite should be reduced. A good TSM solution should take into consideration many factors: the total allowed execution time (or the total number of test cases to execute), the maximum acceptable loss in fault revealing (how many faults are not necessary to be re-exposed), and also the minimum statement coverage to assure. With the knowledge of true Pareto fronts, test engineers can have a comprehensive understanding of the solution space and answer the question like "using 10% of the original test-suite for two hours to reveal 80% faults and cover 85% lines is more effective than using 20% of the test-suite for four hours to reveal 85% faults and cover 90% lines.''

7.3.3 Limitation in Practical Application. The limitations of our MOIP methods in practical application stem from two aspects. In this article, although we adopt the formations (Big-M and OR-relation) to have a linear formulation for the relation between a fault and multiple test cases, our MOIP methods cannot handle the non-linear objectives in nature. According to the recent survey [45], the objectives adopted in existing TSM studies belong to the three traditional test criteria: namely, test-suite cost (e.g., the size or the execution time), fault detection capability, and coverage (e.g., line or branch coverage). As formulated in this article, these goals are all linear. However, a recent study proposes the criterion of similarity between test cases for the TCP problem [62], which cannot be directly modeled as a linear objective—it is hard to have a linear formulation to minimize the total similarity among selected test cases. Hence, the MOIP methods need to be extended to support such objectives (e.g., using linear similarity measure). Second, the MCTSM instances we solve and evaluate are all small-to-medium size, which require less than 100K decisive variables. In a TSM problem recently reported by Microsoft [76], the O365 online service system owns millions of test cases, which pose a challenge for IP methods due to the huge number to decisive variables. For an MCTSM instance at the scale of O365 service system, our exact methods (ϵ-constraint and CWMOIP) that aim at the complete Pareto front will probably result to timeouts—under such circumstance, true Pareto front is not much concerned, while a feasible and efficient method (i.e., the data-driven prediction method proposed in Reference [76]) is needed in practice.

7.4 The Generality and Other Applications of MOIP

Following the success of applying MOIP methods onto the optimal feature selection problem in SPLE [98], in this study, we also have proven that MOIP methods could be sound, complete, and efficient for the MCTSM problem. Nevertheless, *the important assumption* of applying MOIPs should be admitted—*all the constraints and objectives need to be linear*. Both the feature selection and the MCTSM problem satisfy this assumption and could be essentially reduced to the minimum set covering problem [28]. The major difference between these two problems lies in that the solution space of the MCTSM problem is generally smaller than that of the systems in the optimal feature selection problem, and hence the method SolRep [98] that samples the nondominated solutions from the huge solution space is not necessarily needed in this study.

The assumption may not hold in practical application scenario, since many real-world MOO problems in SBSE have non-linear objectives, e.g., the objective of semantics (vocabulary-based similarity for two modules) in the optimal software remodularization problem [64]. Still, some other optimization problems in SBSE usually adopt the conventional goals (e.g., total cost, total time, total revenue) in engineering management, which are the arithmetic summation of the coefficient of each decisive variable. For instance, in software project management area, the requirement

scheduling problem (RSP) [51, 89] is also similar to the MCTSM problem—the dependency among tasks constitute the constraints, and the objectives include maximizing of the revenue gained for the scheduled tasks, minimization of the time, and the cost taken for those tasks. In addition, the multi-objective next release problem (MONRP) [106, 107] is essentially a type of the minimum set cover problem, which aims to satisfy the requirements from customers, meanwhile maximizing the total score of importance of requirements and minimizing the total cost required for the satisfaction of such requirements. Considering the nature of RSP and MONRP, MOIP methods should be ideally applicable to these two problems for discovering the sound and complete Pareto front when they own multiple linear objectives.

8 RELATED WORK

Our study is relevant to two lines of research in SBSE.

8.1 IP Methods and Heuristics for Test Case Minimization, Selection, and Prioritization

According to the report by Khan et al. [45], as an active research area, there are about 2,510 studies on the topic of test case minimization, selection, and prioritization between 1996 and 2016. Among this huge number of studies, only 260 are published in peer-reviewed journals or conferences and could be categorized into three subproblems [101]: TSM (or termed *TSR* in Reference [45]), which is addressed in this study, TCS, and TCP.

Before detailing the studies on TSM, we observe that the existing studies on TSM are usually in the following two lines: (1) focusing on the formulations suitable for IP techniques [9, 21, 35, 39, 55, 70] mostly in the case of *white-box TSM* or (2) focusing on the formulations suitable for MOEAs or heuristic methods (e.g., with non-linear optimization objectives for TSM) [7, 31, 38, 48, 59, 69, 73, 79, 86, 100, 109, 110] mostly in the case of *black-box TSM*. In the article, we focus on the formulations suitable for IP techniques. Hence, our study is mainly to improve and compare with the state-of-the-art IP techniques. Besides, to further make the evaluations more comprehensive, we also compare our MOIP techniques with MOEAs for the TSM formulation (i.e., the tri-criteria problem) used in the first line of studies. Notably, white-box TSM refers to those TSM problems where the source code, the information of tests coverage, and faults could be achieved by some means (e.g., GCOV [29] for the coverage information and bug tracking or version control systems for the fault information). On the contrary, black-box TSM means that the source code, coverage, and fault information relevant to test cases are unknown or incomplete—for example, if most of the information in Table 1 is not available, we cannot apply the MOIP techniques to get the optimal or non-dominated solutions; while MOEAs or heuristic methods can still adopt some testing criteria (I/O based in Reference [37] or test similarity based in References [20, 62]) to get some near-optimal approximated solutions.

8.1.1 IP for TSM. Besides heuristic approaches, IP methods serve as the exact techniques that can guarantee to find the optimal solutions (if properly modeled), playing an indispensable role in solving the TSM problem. Though there exist only several studies on IP methods [9, 21, 35, 39, 55, 70], the IP methods exhibit the effectiveness in both single-criterion TSM or multi-criteria TSM. Early in 1999, Csöndes et al. [21] propose to model the test-suite reduction with ILP for the first time. Specifically, they present three models, namely, linear model, "All or nothing" model, and "One is enough" model, which own the same linear objective and only differ in the constraints. Witnessing the limitation of a single-criterion for TSM, in 2004, Black et al. [9] propose the bi-criteria (i.e., the test-suite size and the error detection rate) ILP model, which actually has one single objective function taking a weighted sum of the two criteria. In 2009, Hsu and Orso present

MINTS [39], the general framework and tool for TSM, which supports four criteria such as statement coverage, execution time, setup effort, and fault detection. However, in its ILP model, the objective function takes a weighted minimization policy for different criteria—the same as Reference [9], a proper weight scheme is required. In 2012, Dan et al. [35] propose and solve the problem of on-demand test-suite reduction, which actually defines some *variant* of TSM. Similar to the triple-criteria problem defined in Definition 8, on-demand test-suite reduction concedes most l% loss in fault-detection capability but retains at least c% of the test cases. In 2017, Panda and Mohapatra [70] propose to employ model cohesion values as new criteria of fault detection for the TSM and formulate the ILP model on the basis of one single objective of the sum of cohesion values of all the nodes covered by the reduced test-suite. Recently, pinpointed by Lin et al. [55], the ILP model in previous studies [9, 39] (e.g., MINT) is not sound, as the dependency between test cases and faults is not considered. Hence, they model this problem with INP, convert it to an ILP model by introducing extra decisive variables, and solve it with a weighted-sum single objective. Besides, Baller et al. [5] apply ILP to solve the test-suite minimization problem for the sets of software variants under test, and they also use a weighted summation policy for multiple conflicting optimization objectives.

Different from all above studies, we apply MOIP methods for the TSM problem, not using a weighted summation policy for various optimization criteria. Instead of using INP, we also show that the TSM problem could be ideally modeled with ILP.

8.1.2 Heuristics for TSM. As TSM is NP-complete, it is generally not easy to find the optimal solutions using the exact methods (i.e., ILP methods). Hence, most of the existing studies are heuristic. According to Reference [45], 95 out of 113 studies on TSM present new approaches, which can be classified into four types of heuristics on the basis of employed algorithms: greedy-based (66%, 63/95, as the majority), clustering-based (3%, 3/95), search-based (20%, 19/95), and hybrid approaches (8%, 8/95). The greedy-based algorithms, called HGS, were initially proposed by Reference [36] to randomly add the test cases for the test-suite with the minimal cardinality. After that, several other greedy-based algorithms [17, 67, 97] are proposed to make optimal decisions between the factors of test execution orderings, coverage, and fault detection capability. Recently, References [3, 11, 103] represent the state-of-the-art greedy-based algorithms for TSM. The clustering-based algorithms [13, 56, 81] usually define a similarity function, which takes as input two test cases and returns a value representing the similarity of the given test cases according to cost and/or effectiveness measures. For these clustering-based algorithms, the redundant test cases will be found by sampling methods and removed for the reduced size of test-suite. Another major branch of algorithms are search-based, including those metaheuristics such as Evolutionary Algorithm [27, 30, 108]. Last, various techniques are combined together as the hybrid approach, which has a wide scope for application. For example, Lin et al. [54] proposes to combine the HGS algorithm and the heuristic search; in some studies [32, 44, 74], the greedy search or the EA is integrated with the clustering algorithm to reduce the executing cost of the regression suite without diminishing its capability in fault detection; besides, dynamic program analysis techniques (i.e., program slicing and delta debugging) are combined to minimize the randomly generated test cases in Reference [88] and formal concept analysis is applied with heuristic search to better balance between requirements satisfaction and the size of test-suite [49].

8.1.3 MOEAs for MCTSM. As usually multiple testing criteria (e.g., various types of coverage [80], fault detection [84, 85], and energy consumption [8, 52]) are involved in the TSM problem, multiple goals or optimization functions are adopted by the previous studies in References [7, 31, 38, 48, 59, 69, 73, 79, 86, 100, 109, 110]. Before 2009 and 2010, it is common to apply heuristic approaches (e.g., HGS) to optimize each of the above criteria separately or strike for a balance

between them. Since Smith et al. apply 2-optimal greedy (2OPT) in 2009 [86] and Yoo et al. adopt the hybrid algorithm for Pareto efficiency on TSM in 2010 [100], it becomes a trend to make use of Pareto dominance as well as MOEAs for equally treating multiple objectives in TSM. Among the above studies on multi-objective optimization of the TSM, References [48, 79, 109, 110] represent the state-of-the-art methods, among which References [109] and [79] are most relevant—in their problem definition, the multiple objectives are all linear; further, the same MOEAs (i.e., NSGA-II and MOEA/D) are used. In contrast, non-linear objectives are used in Reference [48] and many objectives (more than five, so called "many" rather than "multiple") are defined in Reference [110]. Hence, in this study, we compare our IP methods with NSGA-II and MOEA/D, not other MOEAs.

8.1.4 MOEAs for TCS and TCP. In general, the TSM problem is also relevant to the TCS problem [26] and the TCP problem [46]. There is a large number of studies on these two problems, and we cannot elaborate on the concrete works due to the page limit. Similar to the TSM, heuristic approaches including those MOEAs are commonly used to solve these two problems. For instance, Marchetto et al. employ various metrics and rely on NSGA-II to prioritize test cases on 21 Java applications [58], while Panichella et al. propose to enhance the multi-objective genetic algorithms (MOGAs) with injected diversity on 11 programs for TCS [71]. In the future, our IP methods have the potential to be applied to the TCS and TCP problems owing to their commonality with the TSM problem.

To summarize, on the TSM problem, IP methods (including our MOIP techniques) are good at handling TSM instances with a small-to-medium size (less than 10K test cases and 100K lines of code) and less than five types of criteria. Especially, for white-box test-suite optimization, MOIP applied in this study can help to find the sound and complete Pareto front on the five subject programs in Table 4. However, for TSM instances with millions of tests (e.g., the case in Reference [76]), MOIP techniques need to be tested and improved in terms of scalability. Besides, for black-box test-suite optimization, heuristic methods (including MOEAs) seem to be more suitable due to their scalability. In the future, more research endeavors should be made to enhance MOIP techniques for improving the scalability and fitting them for black-box test-suite optimization.

8.2 IP for Other SBSE Problems

8.2.1 SOIP for Other SBSE Problems. On TCS and TCP, there are also some studies that employ IP methods [63, 104]. In Reference [104], Zhang et al. consider two criteria (i.e., time and statement coverage) for the TCP problem and apply ILP using the criterion of time as constraints and the criterion of coverage as the single objective. However, the ILP model is unsound and may yield suboptimal solutions, as the objective function fails to consider the dependency between test cases and statements. In Reference [63], Mirarab et al. also consider three criteria (i.e., the size of selected test cases, fault detection capability, and code coverage), and they use test-suite size as the constraints and the last two criteria for the objective. These two criteria are combined by formulating an objective function equal to the weighted-sum of the individual criteria [63]—it relies on the weighted minimization policy for different criteria on the TCS problem.

Except regression testing, IP has also been used in other SE domains, e.g., SPLE and software project management. In SPLE, to initialize products, the developers need to select a set of features to fulfill the customer requirements, called the valid product feature selection problem [90]. In the requirement engineering domain, ILP is applied to solve the problem of requirement interdependencies [15]. After that, Brinkkemper and his colleagues apply ILP for the problem of flexible release planning [51, 89], in which the objective is to maximize the revenue of the scheduled tasks (or to minimize their cost correspondingly) and the constraints include budget, manpower, team capability, and other factors on each task. Last, in Reference [87], Stylianou et al. survey the

studies on human resource allocation and scheduling for software project management and categorize the mathematical modeling approaches (e.g., ILP) and computational intelligence approaches (e.g., EA).

To summarize, all the studies mentioned in Sections 8.1.1 and 8.2.1 do not have a MOIP modeling. Instead, these studies usually apply a weight scheme for the multicriteria and convert them into one single objective function. Besides, neither Big-M formulation nor CWMOIP is applied in these studies. In general, we think that the recent progress on operational research is not well applied onto the SBSE problems. Hence, in general, IP methods are not as popular as heuristics due to the scalability issue and the strict limitation on the application. Meanwhile, owing to the easy usage and recent advances of computational intelligence, MOEAs seem to become the default method for the MOO problems in SBSE.

8.2.2 MOIP for Other SBSE Problems. To the best of our knowledge, MOIP is only applied on the next release problem (NRP), except our previous research on the optimal feature selection with MOIP. In 2015, Veerapen et al. [92] solve the single- and bi-objective NRP with ILP model. For the single-objective NRP, the objective is the arithmetic sum of the profits associated to selected requirements, and the constraints are the costs of requirements. For the bi-objective NRP, the costs of requirements become the second objective. To solve this, Veerapen et al. employ NSGA-II and the ϵ-constraint method, and the latter one works quite efficiently. Finally, Veerapen et al. conclude that heuristic algorithms (e.g., greedy or NSGA-II) can be discarded in favor of the exact method for both the single-objective and the small bi-objective NRP instances [92]. However, only ϵ-constraint method is applied; *no other IP techniques (e.g., the Big-M formulation and CWMOIP solving method)* are adopted to further optimize the model formulation and speed up the solving in Reference [92]. Recently, in our previous study on optimal feature selection in SPLE [98], we model the four-objective optimization with the ILP model and solve it with ϵ-constraint, CWMOIP, and also SolRep. Note that SolRep is designed to efficiently find evenly distributed nondominated solutions from the true Pareto front when the solution space is too big. In this study, as the MCTSM instances are all small, CWMOIP can ideally handle them, and hence SolRep is not necessarily needed. Also different from Reference [98], we present the Big-M formulation and prove their equivalence with the Or-relation formulation in Section 4.4.

9 CONCLUSION

In this study, by virtue of the Big-M or the OR-relation formulation, we prove that this problem could be *soundly* modeled with ILP, not necessarily with INP. Besides, instead of using a weighted-sum minimization policy in all previous studies, we treat various criteria equally and present an ILP model with multiple objectives. In principle, the Pareto-optimal solution sets should contain the single solution achieved by the weighted-sum policy. We further solve this model with two solving methods (i.e., ϵ-constraint and CWMOIP) and find the *complete* solutions on the target programs in evaluation.

Additionally, in evaluation, the standard MOEAs are just adopted to serve as the baseline to compare with our exact ILP methods and used to check whether the set of found nondominated solution is sound and complete. In the future, we plan to apply ILP methods for the black-box TSM problem and conduct a more complete and general comparison between IP methods and MOEAs on those large programs such as Closure Compiler, Commons Lang, Commons Math, JfreeChart, Joda-Time. Other than applying the exact methods, we will apply the exact IP methods RepSol (proposed by us for large instances of the optimal feature selection problem in SPLC [98]) and compare with some state-of-the-art MOEAs. As heuristic methods are more active and supportive of more formulations and test criteria [45], we plan to apply the IP methods with these formulations and test criteria.

Following the success of the application of MOIP methods on the MCTSM problem in regression testing and the optimal feature selection problem in SPLE, in the future, we plan to apply MOIP methods to solve more particular SBSE problems with linear constraints and objectives (e.g., NRP and RSP). Along the journey of applying MOIP methods, we will further consider their combination with heuristics approaches (e.g., MOEAs) to support the general SBSE problems with both linear and non-linear objectives.

REFERENCES

[1] Radio Technical Commission for Aeronautics (RTCA). 1992. DO-178B: Software Considerations in Airborne Systems and Equipment Certification. https://en.wikipedia.org/wiki/DO-178B.

[2] Sven Apel and Christian Kästner. 2009. An overview of feature-oriented software development. *J. Obj. Technol.* 8, 5 (2009), 49–84. DOI: https://doi.org/10.5381/jot.2009.8.5.c5

[3] Stephan Arlt, Andreas Podelski, and Martin Wehrle. 2014. Reducing GUI test suites via program slicing. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'14)*. 270–281. DOI: https://doi.org/10.1145/2610384.2610391

[4] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Ainhoa Arruabarrena, Leire Etxeberria, and Goiuria Sagardui. 2019. Pareto efficient multi-objective black-box test case selection for simulation-based testing. *Inf. Softw. Technol.* 114 (2019), 137–154. DOI: https://doi.org/10.1016/j.infsof.2019.06.009

[5] Hauke Baller, Sascha Lity, Malte Lochau, and Ina Schaefer. 2014. Multi-objective test-suite optimization for incremental product family testing. In *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation ICST'14)*. 303–312. DOI: https://doi.org/10.1109/ICST.2014.43

[6] Don S. Batory. 2005. Feature models, grammars, and propositional formulas. In *Proceedings of the 9th International Conference on Software Product Lines (SPLC'05)*. 7–20. DOI: https://doi.org/10.1007/11554844_3

[7] Benoit Baudry, Franck Fleurey, and Yves Le Traon. 2006. Improving test suites for efficient fault localization. In *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*. 82–91. DOI: https://doi.org/10.1145/1134285.1134299

[8] Reyhaneh Jabbarvand Behrouz, Alireza Sadeghi, Hamid Bagheri, and Sam Malek. 2016. Energy-aware test-suite minimization for Android apps. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'16)*. ACM, 425–436.

[9] Jennifer Black, Emanuel Melachrinoudis, and David R. Kaeli. 2004. Bi-criteria models for all-uses test suite reduction. In *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*. 106–115. DOI: https://doi.org/10.1109/ICSE.2004.1317433

[10] Marcel Böhme and Abhik Roychoudhury. 2014. CoREBench: Studying complexity of regression errors. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'14)*. 105–115. DOI: https://doi.org/10.1145/2610384.2628058

[11] Prasad Bokil, Padmanabhan Krishnan, and R. Venkatesh. 2015. Achieving effective test suites for reactive systems using specification mining and test suite reduction techniques. *ACM SIGSOFT Softw. Eng. Notes* 40, 1 (2015), 1–8. DOI: https://doi.org/10.1145/2693208.2693226

[12] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press, New York, NY.

[13] Lionel C. Briand, Yvan Labiche, Zaheer Bawar, and Nadia Traldi Spido. 2009. Using machine learning to refine category-partition test specifications and test suites. *Inf. Softw. Technol.* 51, 11 (2009), 1551–1564. DOI: https://doi.org/10.1016/j.infsof.2009.06.006

[14] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. 2008. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI'08)*. 209–224. Retrieved from http://www.usenix.org/events/osdi08/tech/full_papers/cadar/cadar.pdf.

[15] Pär Carlshamre. 2002. Release planning in market-driven software product development: Provoking an understanding. *Requir. Eng.* 7, 3 (2002), 139–151.

[16] Ryan Carlson, Hyunsook Do, and Anne Denton. 2011. A clustering approach to improving test case prioritization: An industrial case study. In *Proceedings of the IEEE 27th International Conference on Software Maintenance (ICSM'11)*. 382–391. DOI: https://doi.org/10.1109/ICSM.2011.6080805

[17] Tsong Yueh Chen and Man Fai Lau. 1996. Dividing strategies for the optimization of a test suite. *Inf. Proc. Lett.* 60, 3 (1996), 135–141. DOI: https://doi.org/10.1016/S0020-0190(96)00135-4

[18] Carlos A. Coello and Efrén Mezura-Montes. 2002. Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Adv. Eng. Inform.* 16, 3 (2002), 193–203.

[19] Domenico Cotroneo, Roberto Pietrantuono, and Stefano Russo. 2013. A learning-based method for combining testing techniques. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*. 142–151. DOI: https://doi.org/10.1109/ICSE.2013.6606560

[20] Emilio Cruciani, Breno Miranda, Roberto Verdecchia, and Antonia Bertolino. 2019. Scalable approaches for test suite reduction. In *Proceedings of the 41st International Conference on Software Engineering (ICSE'19)*. 419–429. DOI: https://doi.org/10.1109/ICSE.2019.00055

[21] Tibor Csöndes, Sarolta Dibuz, and Balázs Kotnyek. 1999. Test suite reduction in conformance testing. *Acta Cybern.* 14 (1 1999), 229–238.

[22] Dario Izzo and Francesco Biscani. 2019. Pagmo (C++) or Pygmo (Python): A scientific library for massively parallel optimization. Retrieved from https://esa.github.io/pagmo2/.

[23] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* 6, 2 (2002), 182–197. DOI: https://doi.org/10.1109/4235.996017

[24] Orit Edelstein, Eitan Farchi, Evgeny Goldin, Yarden Nir, Gil Ratsaby, and Shmuel Ur. 2003. Framework for testing multi-threaded Java programs. *Concurr. Comput. Pract. Exper.* 15, 3–5 (2003), 485–499. DOI: https://doi.org/10.1002/cpe.654

[25] Sebastian Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. 2002. Test case prioritization: A family of empirical studies. *IEEE Trans. Softw. Eng.* 28, 2 (Feb. 2002), 159–182. DOI: https://doi.org/10.1109/32.988497

[26] Emelie Engström, Per Runeson, and Mats Skoglund. 2010. A systematic review on regression test selection techniques. *Inf. Softw. Technol.* 52, 1 (2010), 14–30. DOI: https://doi.org/10.1016/j.infsof.2009.07.001

[27] Usman Farooq and Chiou Peng Lam. 2009. Evolving the quality of a model based test suite. In *Proceedings of the 2nd International Conference on Software Testing Verification and Validation (ICST'09)*. 141–149. DOI: https://doi.org/10.1109/ICSTW.2009.27

[28] Michael R. Garey and David S. Johnson. 1990. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY.

[29] GCC 2018. gcov—A Test Coverage Program. Retrieved from https://gcc.gnu.org/onlinedocs/gcc/Gcov.html.

[30] Jingyao Geng, Zheng Li, Ruilian Zhao, and Junxia Guo. 2016. Search based test suite minimization for fault detection and localization: A co-driven method. In *Proceedings of the 8th International Symposium on Search-based Software Engineering (SSBSE'16)*. 34–48. DOI: https://doi.org/10.1007/978-3-319-47106-8_3

[31] Qing Gu, Bao Tang, and Daoxu Chen. 2010. Optimal regression testing based on selective coverage of test requirements. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA'10)*. 419–426. DOI: https://doi.org/10.1109/ISPA.2010.62

[32] Liucheng Guo, Jiangfang Yi, Liang Zhang, Xiaoyin Wang, and Dong Tong. 2011. CGA: Combining cluster analysis with genetic algorithm for regression suite reduction of microprocessors. In *Proceedings of the IEEE International System-on-Chip Conference (SoCC'11)*. 207–212. DOI: https://doi.org/10.1109/SOCC.2011.6085105

[33] Brent Hailpern and Padmanabhan Santhanam. 2001. *Software Debugging, Testing, and Verification.* IBM Research Report RC22220 (W0110-083). IBM, Thomas J. Watson Research Center, NY.

[34] Yacov Y. Haimes, Leon S. Lasdon, and David A. Wismer. 1971. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Trans. Syst. Man Cybern.* SMC-1, 3 (July 1971), 296–297. DOI: https://doi.org/10.1109/TSMC.1971.4308298

[35] Dan Hao, Lu Zhang, Xingxia Wu, Hong Mei, and Gregg Rothermel. 2012. On-demand test suite reduction. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*. 738–748. DOI: https://doi.org/10.1109/ICSE.2012.6227144

[36] Mary Jean Harrold, Rajiv Gupta, and Mary Lou Soffa. 1993. A methodology for controlling the size of a test suite. *ACM Trans. Softw. Eng. Methodol.* 2, 3 (1993), 270–285. DOI: https://doi.org/10.1145/152388.152391

[37] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. 2016. Comparing white-box and black-box test prioritization. In *Proceedings of the 38th International Conference on Software Engineering (ICSE'16)*. 523–534. DOI: https://doi.org/10.1145/2884781.2884791

[38] Mark Hennessy and James F. Power. 2005. An analysis of rule coverage as a criterion in generating minimal test suites for grammar-based software. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE'05)*. 104–113. DOI: https://doi.org/10.1145/1101908.1101926

[39] Hwa-You Hsu and Alessandro Orso. 2009. MINTS: A general framework and tool for supporting test-suite minimization. In *Proceedings of the 31st International Conference on Software Engineering (ICSE'09)*. 419–429. DOI: https://doi.org/10.1109/ICSE.2009.5070541

[40] I. Griva, S. G. Nash, and A. Sofer. 2009. *Linear and Nonlinear Optimization* (2nd. ed.). Society for Industrial Mathematics.

[41] IBM Inc. 2019. CPLEX Optimizer. Retrieved from https://www.ibm.com/analytics/cplex-optimizer

[42] Laura Inozemtseva and Reid Holmes. 2014. Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*. 435–445. DOI: https://doi.org/10.1145/2568225.2568271

[43] Wei Jin. 2013. Reproducing and debugging field failures in house. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*. 1441–1443. DOI: https://doi.org/10.1109/ICSE.2013.6606738

[44] Alireza Khalilian and Saeed Parsa. 2009. Bi-criteria test suite reduction by cluster analysis of execution profiles. In *Proceedings of the Central and East European Conference on Software Engineering Techniques (CEE-SET'09)*. 243–256. DOI: https://doi.org/10.1007/978-3-642-28038-2_19

[45] Saif Rehman Khan, Sai Peck Lee, Nadeem Javaid, and Wadood Abdul. 2018. A systematic review on test suite reduction: Approaches, experiment's quality evaluation, and guidelines. *IEEE Access* 6 (2018), 11816–11841. DOI: https://doi.org/10.1109/ACCESS.2018.2809600

[46] Muhammad Khatibsyarbini, Mohd Adham Isa, Dayang N. A. Jawawi, and Rooster Tumeng. 2018. Test case prioritization approaches in regression testing: A systematic literature review. *Inf. Softw. Technol.* 93 (2018), 74–93.

[47] Adriaan Labuschagne, Laura Inozemtseva, and Reid Holmes. 2017. Measuring the cost of regression testing in practice: A study of Java projects using continuous integration. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'17)*. 821–830. DOI: https://doi.org/10.1145/3106237.3106288

[48] Remo Lachmann, Michael Felderer, Manuel Nieke, Sandro Schulze, Christoph Seidl, and Ina Schaefer. 2017. Multi-objective black-box test case selection for system testing. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'17)*. 1311–1318. DOI: https://doi.org/10.1145/3071178.3071189

[49] Andreas Leitner, Manuel Oriol, Andreas Zeller, Ilinca Ciupa, and Bertrand Meyer. 2007. Efficient unit test case minimization. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07)*. 417–420. DOI: https://doi.org/10.1145/1321631.1321698

[50] Hareton K. N. Leung and Lee White. 1991. A cost model to compare regression test strategies. In *Proceedings of the International Conference on Software Maintenance*. 201–208. DOI: https://doi.org/10.1109/ICSM.1991.160330

[51] Chen Li, Marjan van den Akker, Sjaak Brinkkemper, and Guido Diepen. 2010. An integrated approach for requirement selection and scheduling in software release planning. *Requir. Eng.* 15, 4 (2010), 375–396. DOI: https://doi.org/10.1007/s00766-010-0104-x

[52] Ding Li, Yuchen Jin, Cagri Sahin, James Clause, and William G. J. Halfond. 2014. Integrated energy-directed test suite optimization. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'14)*. ACM, 339–350.

[53] Yan-Fu Li and Enrico Zio. 2017. RAMS optimization principles. *Handbook of Safety Principles*. Wiley, 514–539.

[54] Jun-Wei Lin and Chin-Yu Huang. 2009. Analysis of test suite reduction with enhanced tie-breaking techniques. *Inf. Softw. Technol.* 51, 4 (2009), 679–690. DOI: https://doi.org/10.1016/j.infsof.2008.11.004

[55] Jun-Wei Lin, Reyhaneh Jabbarvand, Joshua Garcia, and Sam Malek. 2018. Nemo: Multi-criteria test-suite minimization with integer nonlinear programming. In *Proceedings of the 40th International Conference on Software Engineering (ICSE'18)*. 1039–1049. DOI: https://doi.org/10.1145/3180155.3180174

[56] Yue Liu, Kang Wang, Wang Wei, Bofeng Zhang, and Hailin Zhong. 2011. User-session-based test cases optimization method based on agglutinate hierarchy clustering. In *Proceedings of the IEEE International Conference on Internet of Things and International Conference on Cyber, Physical, and Social Computing (iThings & CPSCom'11)*. 413–418. DOI: https://doi.org/10.1109/iThings/CPSCom.2011.135

[57] Henry B. Mann and Donald R. Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Stat.* 18, 1 (1947), 50–60.

[58] Alessandro Marchetto, Md. Mahfuzul Islam, M. Waseem Asghar, Angelo Susi, and Giuseppe Scanniello. 2016. A multi-objective technique to prioritize test cases. *IEEE Trans. Softw. Eng.* 42, 10 (2016), 918–940. DOI: https://doi.org/10.1109/TSE.2015.2510633

[59] Alessandro Marchetto, Md. Mahfuzul Islam, Angelo Susi, and Giuseppe Scanniello. 2013. A multi-objective technique for test suite reduction. In *Proceedings of the 8th International Conference on Software Engineering Advances (ICSEA'13)*. 18–24.

[60] Martina Marré and Antonia Bertolino. 2003. Using spanning sets for coverage testing. *IEEE Trans. Softw. Eng.* 29, 11 (2003), 974–984. DOI: https://doi.org/10.1109/TSE.2003.1245299

[61] Marcílio Mendonça, Thiago Tonelli Bartolomei, and Donald Cowan. 2008. Decision-making coordination in collaborative product configuration. In *Proceedings of the ACM Symposium on Applied Computing (SAC'08)*. ACM, New York, NY, 108–113. DOI: https://doi.org/10.1145/1363686.1363715

[62] Breno Miranda, Emilio Cruciani, Roberto Verdecchia, and Antonia Bertolino. 2018. FAST approaches to scalable similarity-based test case prioritization. In *Proceedings of the 40th International Conference on Software Engineering (ICSE'18)*. 222–232. DOI: https://doi.org/10.1145/3180155.3180210

[63] Siavash Mirarab, Soroush Akhlaghi, and Ladan Tahvildari. 2012. Size-constrained regression test case selection using multicriteria optimization. *IEEE Trans. Softw. Eng.* 38, 4 (2012), 936–956. DOI: https://doi.org/10.1109/TSE.2011.56

[64] Wiem Mkaouer, Marouane Kessentini, Adnan Shaout, Patrice Koligheu, Slim Bechikh, Kalyanmoy Deb, and Ali Ouni. 2015. Many-objective software remodularization using NSGA-III. *ACM Trans. Softw. Eng. Methodol.* 24, 3 (2015), 17:1–17:45. DOI: https://doi.org/10.1145/2729974

[65] Lev Nachmanson, Margus Veanes, Wolfram Schulte, Nikolai Tillmann, and Wolfgang Grieskamp. 2004. Optimal strategies for testing nondeterministic systems. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'04)*. 55–64. DOI: https://doi.org/10.1145/1007512.1007520

[66] George L. Nemhauser and Laurence A. Wolsey. 1988. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY.

[67] Jefferson Offutt, Jie Pan, and Jeffrey M. Voas. 1995. Procedures for reducing the size of coverage-based test sets. In *Proceedings of the 12th International Conference on Testing Computer Software*. 111–123.

[68] Melih Özlen and Meral Azizoglu. 2009. Multi-objective integer programming: A general approach for generating all non-dominated solutions. *Eur. J. Oper. Res.* 199, 1 (2009), 25–35. DOI: https://doi.org/10.1016/j.ejor.2008.10.023

[69] Lili Pan, Junyi Li, Beiji Zou, and Hao Chen. 2005. Bi-objective model for test-suite reduction based on modified condition/decision coverage. In *Proceedings of the 11th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'05)*. 235–244. DOI: https://doi.org/10.1109/PRDC.2005.22

[70] Subhrakanta Panda and Durga Prasad Mohapatra. 2017. Regression test suite minimization using integer linear programming model. *Softw. Pract. Exper.* 47, 11 (2017), 1539–1560. DOI: https://doi.org/10.1002/spe.2485

[71] Annibale Panichella, Rocco Oliveto, Massimiliano Di Penta, and Andrea De Lucia. 2015. Improving multi-objective test case selection by injecting diversity in genetic algorithms. *IEEE Trans. Softw. Eng.* 41, 4 (2015), 358–383. DOI: https://doi.org/10.1109/TSE.2014.2364175

[72] Mike Papadakis, Yue Jia, Mark Harman, and Yves Le Traon. 2015. Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique. In *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE'15), Volume 1*. 936–946. DOI: https://doi.org/10.1109/ICSE.2015.103

[73] Saeed Parsa and Alireza Khalilian. 2010. On the optimization approach towards test suite minimization. *Int. J. Softw. Eng. Appl.* 4, 1 (2010), 15–28.

[74] Saeed Parsa, Alireza Khalilian, and Y. Fazlalizadeh. 2009. A new algorithm to test suite reduction based on cluster analysis. In *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology*. 189–193. DOI: https://doi.org/10.1109/ICCSIT.2009.5234742

[75] Tanay Kanti Paul and Man Fai Lau. 2014. A systematic literature review on modified condition and decision coverage. In *Proceedings of the Symposium on Applied Computing (SAC'14)*. 1301–1308. DOI: https://doi.org/10.1145/2554850.2555004

[76] Adithya Abraham Philip, Ranjita Bhagwan, Rahul Kumar, Chandra Shekhar Maddila, and Nachiappan Nagappan. 2019. FastLane: Test minimization for rapidly deployed large-scale online services. In *Proceedings of the 41st International Conference on Software Engineering (ICSE'19)*. 408–418. DOI: https://doi.org/10.1109/ICSE.2019.00054

[77] Nery Riquelme, Christian von Lücken, and Benjamín Barán. 2015. Performance metrics in multi-objective optimization. In *Proceedings of the Latin American Computing Conference/Conferencia Latinoamericana En Informatica (CLEI'15)*. IEEE, 1–11.

[78] Gregg Rothermel and Mary Jean Harrold. 1993. A safe, efficient algorithm for regression test selection. In *Proceedings of the Conference on Software Maintenance (ICSM'93)*. 358–367. DOI: https://doi.org/10.1109/ICSM.1993.366926

[79] Takfarinas Saber, Florian Delavernhe, Mike Papadakis, Michael O'Neill, and Anthony Ventresque. 2018. A hybrid algorithm for multi-objective test case selection. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'18)*. 1–8. DOI: https://doi.org/10.1109/CEC.2018.8477875

[80] Sreedevi Sampath, Renée C. Bryce, and Atif M. Memon. 2013. A uniform representation of hybrid criteria for regression testing. *IEEE Trans. Softw. Eng.* 39, 10 (2013), 1326–1344. DOI: https://doi.org/10.1109/TSE.2013.16

[81] Sreedevi Sampath, Sara Sprenkle, Emily Gibson, Lori L. Pollock, and Amie L. Souter. 2005. Analyzing clusters of web application user sessions. *ACM SIGSOFT Softw. Eng. Notes* 30, 4 (2005), 1–7. DOI: https://doi.org/10.1145/1082983.1083255

[82] Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. 2013. On the value of user preferences in search-based software engineering: A case study in software product lines. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*. 492–501. DOI: https://doi.org/10.1109/ICSE.2013.6606595

[83] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. 2006. Feature diagrams: A survey and a formal semantics. In *Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE'06)*. 136–145. DOI: https://doi.org/10.1109/RE.2006.23

[84] August Shi, Alex Gyori, Milos Gligoric, Andrey Zaytsev, and Darko Marinov. 2014. Balancing trade-offs in test-suite reduction. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14)*. 246–256. DOI: https://doi.org/10.1145/2635868.2635921

[85] August Shi, Tifany Yung, Alex Gyori, and Darko Marinov. 2015. Comparing and combining test-suite reduction and regression test selection. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/SIGSOFT FSE'15)*. ACM, 237–247.

[86] Adam M. Smith and Gregory M. Kapfhammer. 2009. An empirical study of incorporating cost into test suite reduction and prioritization. In *Proceedings of the ACM Symposium on Applied Computing (SAC'09)*. 461–467. DOI: https://doi.org/10.1145/1529282.1529382

[87] Constantinos Stylianou and Andreas S. Andreou. 2014. *Human Resource Allocation and Scheduling for Software Project Management*. Springer Berlin, 73–106. DOI: https://doi.org/10.1007/978-3-642-55035-5_4

[88] Sriraman Tallam and Neelam Gupta. 2005. A concept analysis inspired greedy algorithm for test suite minimization. In *Proceedings of the ACM Workshop on Program Analysis For Software Tools and Engineering (PASTE'05)*. 35–42. DOI: https://doi.org/10.1145/1108792.1108802

[89] Marjan van den Akker, Sjaak Brinkkemper, Guido Diepen, and Johan Versendaal. 2008. Software product release planning through optimization and what-if analysis. *Inf. Softw. Technol.* 50, 1–2 (2008), 101–111. DOI: https://doi.org/10.1016/j.infsof.2007.10.017

[90] Pim van den Broek. 2010. Optimization of product instantiation using integer programming. In *Proceedings of the Software Product Lines Conference Workshops (SPLC'10)*. Lancaster University, 107–112.

[91] Vandana and Ajmer Singh. 2017. Multi-objective test case minimization using evolutionary algorithms: A review. In *Proceedings of the International Conference of Electronics, Communication and Aerospace Technology (ICECA'17)*, Vol. 1. 329–334. DOI: https://doi.org/10.1109/ICECA.2017.8203698

[92] Nadarajen Veerapen, Gabriela Ochoa, Mark Harman, and Edmund K. Burke. 2015. An integer linear programming approach to the single and bi-objective next release problem. *Inf. Softw. Technol.* 65 (2015), 1–13. DOI: https://doi.org/10.1016/j.infsof.2015.03.008

[93] Sergiy A. Vilkomir and Jonathan P. Bowen. 2001. Formalization of software testing criteria using the Z notation. In *Proceedings of the 25th International Computer Software and Applications Conference (COMPSAC'01)*. 351–356. DOI: https://doi.org/10.1109/CMPSAC.2001.960638

[94] Shuai Wang, Shaukat Ali, Tao Yue, Yan Li, and Marius Liaaen. 2016. A practical guide to select quality indicators for assessing Pareto-based search algorithms in search-based software engineering. In *Proceedings of the 38th International Conference on Software Engineering (ICSE'16)*. 631–642. DOI: https://doi.org/10.1145/2884781.2884880

[95] Laurence A. Wolsey. 1998. *Integer Programming*. Wiley-Interscience, New York, NY.

[96] Eric Wong, Joseph R. Horgan, Saul London, and Hiralal Agrawal. 1997. A study of effective regression testing in practice. In *Proceedings of the 8th International Symposium on Software Reliability Engineering (ISSRE'97)*. 264–274. DOI: https://doi.org/10.1109/ISSRE.1997.630875

[97] Eric Wong, Joseph Robert Horgan, Aditya P. Mathur, and Alberto Pasquini. 1999. Test set size minimization and fault detection effectiveness: A case study in a space application. *J. Syst. Softw.* 48, 2 (1999), 79–89. DOI: https://doi.org/10.1016/S0164-1212(99)00048-5

[98] Yinxing Xue and Yan-Fu Li. 2018. Multi-objective integer programming approaches for solving optimal feature selection problem: A new perspective on multi-objective optimization problems in SBSE. In *Proceedings of the 40th International Conference on Software Engineering (ICSE'18)*. 1231–1242. DOI: https://doi.org/10.1145/3180155.3180257

[99] Yinxing Xue 2018. IP-Method for Multi-objective Optimization in SPL. Retrieved from https://sites.google.com/view/ip-method-repsol/.

[100] Shin Yoo and Mark Harman. 2010. Using hybrid algorithm for Pareto efficient multi-objective test suite minimisation. *J. Syst. Softw.* 83, 4 (2010), 689–701. DOI: https://doi.org/10.1016/j.jss.2009.11.706

[101] Shin Yoo and Mark Harman. 2012. Regression testing minimization, selection and prioritization: A survey. *Softw. Test. Verif. Reliab.* 22, 2 (Mar. 2012), 67–120. DOI: https://doi.org/10.1002/stv.430

[102] Shin Yoo, Mark Harman, and Shmuel Ur. 2011. Highly scalable multi-objective test suite minimisation using graphics cards. In *Proceedings of the 3rd International Symposium on Search-based Software Engineering (SSBSE'11)*. 219–236. DOI: https://doi.org/10.1007/978-3-642-23716-4_20

[103] Chaoqiang Zhang, Alex Groce, and Mohammad Amin Alipour. 2014. Using test case reduction and prioritization to improve symbolic execution. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'14)*. 160–170. DOI: https://doi.org/10.1145/2610384.2610392

[104] Lu Zhang, Shan-Shan Hou, Chao Guo, Tao Xie, and Hong Mei. 2009. Time-aware test-case prioritization using integer linear programming. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'09)*. ACM, 213–224.

[105] Qingfu Zhang and Hui Li. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Trans. Evolut. Comput.* 11, 6 (2007), 712–731.

[106] Yuanyuan Zhang, Mark Harman, and S. Afshin Mansouri. 2007. The multi-objective next release problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'07)*. 1129–1137. DOI: https://doi.org/10.1145/1276958.1277179

[107] Yuanyuan Zhang, Mark Harman, Gabriela Ochoa, Guenther Ruhe, and Sjaak Brinkkemper. 2018. An empirical study of meta- and hyper-heuristic search for multi-objective release planning. *ACM Trans. Softw. Eng. Methodol.* 27, 1 (2018), 3:1–3:32. DOI: https://doi.org/10.1145/3196831

[108] Yikun Zhang, Jiceng Liu, Yingan Cui, Xinhong Hei, and Minghui Zhang. 2011. An improved quantum genetic algorithm for test suite reduction. In *Proceedings of the IEEE International Conference on Computer Science and Automation Engineering*, Vol. 2. 149–153. DOI: https://doi.org/10.1109/CSAE.2011.5952443

[109] Wei Zheng, Robert M. Hierons, Miqing Li, Xiaohui Liu, and Veronica Vinciotti. 2016. Multi-objective optimisation for regression testing. *Inf. Sci.* 334–335 (2016), 1–16. DOI: https://doi.org/10.1016/j.ins.2015.11.027

[110] Wei Zheng, Xiaoxue Wu, Shichao Cao, and Jun Lin. 2018. MS-guided many-objective evolutionary optimisation for test suite minimisation. *IET Softw.* 12, 6 (2018), 547–554. DOI: https://doi.org/10.1049/iet-sen.2018.5133

[111] Eckart Zitzler and Lothar Thiele. 1999. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Trans. Evolut. Comput.* 3, 4 (1999), 257–271. DOI: https://doi.org/10.1109/4235.797969

Received May 2019; revised March 2020; accepted March 2020