To Preserve or Not to Preserve Invalid Solutions in Search-Based Software Engineering: A Case Study in Software Product Lines

Jianmei Guo
Alibaba Group, China

Kai Shi
East China University of Science and Technology, China

ABSTRACT

Multi-objective evolutionary algorithms (MOEAs) have been successfully applied for software product lines (SPLs) to search for optimal or near-optimal solutions that balance multiple objectives. However, MOEAs usually produce invalid solutions that violate the constraints predefined. As invalid solutions are unbuildable in practice, we debate the preservation of invalid solutions during the search. We conduct experiments on seven real-world SPLs, including five largest SPLs hitherto reported and two SPLs with realistic values and constraints of quality attributes. We identify three potential limitations of preserving invalid solutions. Furthermore, based on the state-of-the-art, we design five algorithm variants that adopt different evolutionary operators. By performance evaluation, we provide empirical guidance on how to preserve valid solutions. Our empirical study demonstrates that whether or not to preserve invalid solutions deserves more attention in the community, and in some cases, we have to preserve valid solutions all along the way.

CCS CONCEPTS

• **Software and its engineering → Search-based software engineering**;

KEYWORDS

Search-based software engineering, software product lines, multi-objective evolutionary algorithms, constraint solving, validity

ACM Reference Format:
Jianmei Guo and Kai Shi. 2018. To Preserve or Not to Preserve Invalid Solutions in Search-Based Software Engineering: A Case Study in Software Product Lines. In *ICSE'18: 40th International Conference on Software Engineering, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3180155.3180163

1 INTRODUCTION

Search-based software engineering (SBSE) [26–29] has been successfully applied to many fields within the general area of software engineering, such as requirements [40, 43, 57, 76], design [1, 53, 59], testing [3, 21, 36, 47, 48, 56], and maintenance [7, 30, 38, 39, 52, 66]. SBSE reformulates software-engineering problems as search-based optimization problems and finds optimal or near-optimal solutions by metaheuristics. Typical metaheuristics, such as multi-objective evolutionary algorithms (MOEAs), usually produce *invalid* solutions that violate the constraints predefined, since they do not embody constraint checking and their embedded operators (e.g., mutation) may break the validity of solutions. However, many software-engineering problems are highly constrained [10, 42, 49, 50]. Invalid solutions are simply *unbuildable* [32] and thus *useless* in practice. Hence, a central research question is:

RQ1: Should we preserve invalid solutions during the search?

To answer the question, we conduct a case study in software product lines (SPLs). Recently, the area of SBSE for SPLs is gaining an upsurge in interest and activity [27]. An SPL is a collection of related software products, all of which share some core functionality, yet each of which differs in some specific features [4, 27]. A key task in SPL engineering is to derive an optimal solution (i.e., software product) by feature selection to meet specific requirements of stakeholders [58], which we call the *SPL optimization problem*. It is non-trivial to address this problem, because: 1) the high variability of different products expressed by feature combinatorics, gives rise to a huge search space; 2) there are usually a set of predefined constraints that determine whether or not a solution can be feasibly implemented; and 3) the problem often involves multiple competing objectives and generally no single solution excels in all objectives.

Many metaheuristics have been used to address the SPL optimization problem [24, 34, 45, 67, 71, 72]. Sayyad et al. first applied MOEAs to solve the problem and reported the indicator-based evolutionary algorithm (IBEA) to be the best algorithm [64]. Afterwards, studies of various improvements on IBEA [63, 74] have been proposed. In particular, Henard et al. [32] proposed a method called SATIBEA that combines IBEA with SAT (Boolean satisfiability problem) solving. They incorporated the SAT solving into the mutation operator of IBEA to replace some randomly-mutated solutions with valid ones. Their empirical results on large real-world SPLs showed that SATIBEA significantly outperforms the original IBEA, which established the current state-of-the-art.

All of these previous MOEAs for the SPL optimization problem permit invalid solutions. Specifically, invalid solutions, either randomly-generated or randomly-mutated ones, can be preserved both in the initial population and at each iteration of MOEAs. This tends to be a common sense in the community, and nobody questions it. To the best of our knowledge, there is no systematic study that debates the preservation of invalid solutions.

In this paper, by empirical study on seven real-world SPLs, we identify three potential limitations of preserving invalid solutions when using previous MOEAs for the SPL optimization problem. Firstly, previous methods produce valid solutions without guarantee, and in the worst case, they fail to find any valid solutions, which makes the costly search effort all in vain. Secondly, the performance evaluation of different MOEAs depends mainly on a number of metrics (e.g., hypervolume [12]) calculated on the population of

ICSE'18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5638-1/18/05…$15.00
https://doi.org/10.1145/3180155.3180163

solutions produced, but the population including invalid solutions may lead to misleading evaluation results and thus may recommend an inappropriate algorithm. Thirdly, previous studies mostly generate a fixed, synthetic value for each quality attribute of each feature, and they assume that the fitness of a solution can be calculated by simply aggregating the attribute values of all selected features. However, in practice, the attribute values of a feature may not be fixed, and more importantly, there may exist complex constraints defined among these quality attributes. In such a case, the fitness calculation of all previous MOEAs does not work, and valid solutions must be preserved "*all along the way*" that is both in the initial population and at each iteration of MOEAs.

Our aim is not only to make the community aware of the potential limitations of preserving invalid solutions, but also to provide guidance on how to preserve valid solutions. We use the state-of-the-art SATIBEA [32] as a baseline and design five algorithm variants that incorporate SAT solving into the initial population generation and the mutation operator of IBEA in different ways. Moreover, one algorithm designed to preserve valid solutions all along the way further incorporates a subroutine that resolves non-Boolean constraints over integer or real-number variables together with arithmetic or relational operators, which cannot be straightforwardly addressed by the state-of-the-art [32] relying on SAT solving. We compare the performance of different algorithms by experiments and then provide empirical guidance. In particular, we answer the following research questions:

RQ2: Should we keep valid solutions in the initial population?
RQ3: Should we fix invalid solutions after mutation?
RQ4: Should we preserve valid solutions all along the way?

In summary, we make the following contributions:

- We conduct the first study on the preservation of invalid solutions in SBSE for SPLs. By empirical study, we identify three potential limitations of preserving invalid solutions, which were usually ignored in the past.
- We perform experiments on seven real-world SPLs, ranging from 48 to 6,888 features and from 23 to 343,944 constraints. Our subjects include five largest SPLs (e.g., Linux) hitherto reported in the literature and two SPLs (AmazonEC2 and Drupal) with realistic attribute values. In particular, AmazonEC2 contains 17,049 real constraints defined among the quality attributes of features.
- By performance evaluation using five designed algorithms, a key guidance we recommend is that preserving valid solutions all along the way is necessary for some subjects (e.g., AmazonEC2) containing attribute constrains, which may represent many real-world cases, but it might not always be necessary for others without attribute constraints. Other empirical guidance includes: excluding invalid solutions at least in the final population for pragmatic performance evaluation, using a random strategy for initial population generation, and adopting a lower mutation rate and a lower probability to fix a solution to be valid after mutation.
- Our empirical study gives software engineers this take-home message: *Whether or not to preserve invalid solutions deserves more attention in SBSE, and in some cases, we have to preserve valid solutions all along the way during the search.*
- We make the source code of our algorithms and our experimental results publicly available at: https://github.com/jmguo/balanceValidity/.

2 PRELIMINARIES AND MOTIVATION

This section introduces background concepts on the SPL optimization problem and discusses the state-of-the-art approaches to the problem. Moreover, we describe the subjects evaluated in this paper. Finally, we analyze the examples of subjects and present preliminary experimental results to motivate our study.

2.1 Problem Statement

Originated from feature-oriented domain analysis (FODA) proposed in 1990 [37], SPL engineering has emerged as one of the most promising software reuse paradigms for reducing development cost, enhancing software quality, and shortening time to market [13, 58, 69]. An SPL is a collection of related software products, all of which share some core functionality, yet each of which differs in some specific features [27]. The commonalities and variabilities among software products in an SPL are expressed in terms of features.

A *feature* is an essential abstraction of a functionality or product characteristic relevant to stakeholders [9, 15]. All features of an SPL and the constraints defined among features are captured in a *feature model*. For example, Figure 1 shows a part of the feature model of Amazon Elastic Compute Cloud (EC2), excerpted from García-Galán et al. [20]. Each feature except the root has a parent. The selection of features can be mandatory (filled circle) or optional (empty circle). Relations between a feature and its group of child features can be Or (not shown)- and Alternative-groups (arc). In addition, there may exist cross-tree constraints that comprise *requires* and *excludes* relations between features. According to Batory [6], a feature model can be translated into a Boolean formula in conjunctive normal form (CNF), which enables automated analysis of feature models using off-the-shelf constraint solvers [8, 68].

Each product is specified as a unique selection of features, called a *configuration*, or a *solution* in our settings. Traditionally, the constraints in a feature model indicate those defined among features, such as cross-tree constraints. However, there may exist other complex constraints in practice [10, 42, 49, 50]. In this paper, we evidence the constraints defined among the quality attributes of features. Therefore, a solution is *valid* when all constraints defined among features and among quality attributes (if applicable) are satisfied.

Given a feature model with n features, the SPL optimization problem is a quadruple $\mathcal{P} = \langle X, D, C, F \rangle$, where $X = \{x_1, \ldots, x_n\}$ is a set of decision variables on all n features, $D = \{D_1, \ldots, D_n\}$ is a set of domains of variables in X, C is a set of constraints predefined, and $F = \{f_1, \ldots, f_m\}$ is composed of m objective functions to minimize simultaneously [22]. Without loss of generality, we consider only minimization problems here.

A solution s to the SPL optimization problem $\mathcal{P}$ is a valid configuration of features, that is, a total assignment of all variables in X to values in the corresponding domains, such that all constraints in C are satisfied. All solutions constitute the search space $\mathcal{S}$. Each objective function $f_i(\mathcal{S})$ assigns an objective value to each solution $s \in \mathcal{S}$. Given two solutions s and s' to the problem $\mathcal{P}$, we say that s *dominates* s', denoted by $s \prec s'$, if and only if s is not worse than

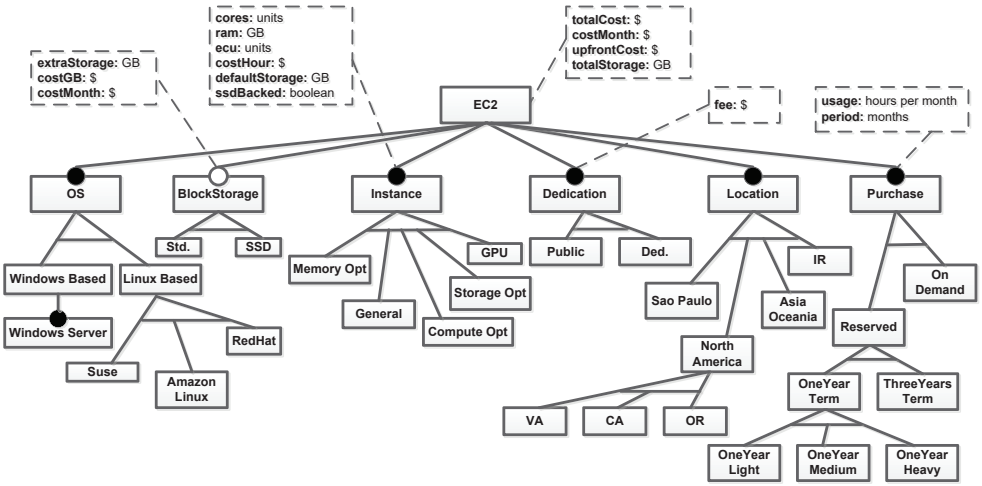

Figure 1: A sample feature model of Amazon Elastic Compute Cloud (EC2), excerpted from García-Galán et al. [20]

s' for any objective and s is better than s' regarding at least one objective, i.e.:

$$\forall i \in \{1, \ldots, m\} : f_i(s) \leq f_i(s') \quad and$$
$$\exists j \in \{1, \ldots, m\} : f_j(s) < f_j(s') \tag{1}$$

A solution to $\mathcal{P}$ is *Pareto-optimal*, which we denote by $\tilde{s}$, if and only if no other solution s in $\mathcal{S}$ dominates $\tilde{s}$, i.e.:

$$\nexists s \in \mathcal{S} : s \prec \tilde{s}. \tag{2}$$

All the Pareto-optimal solutions constitute the Pareto front $\tilde{\mathcal{S}}$.

2.2 State of the Art

To address the SPL optimization problem, many previous approaches were proposed for single-objective optimization, such as [24, 71]. Wu et al. [72] introduced a bi-objective optimization method that balances reliability and cost for a single case study of mail server system development. Tan et al. [67] proposed a feedback-directed mechanism to improve existing evolutionary algorithms. Lu et al. [45] proposed an approach called Zen-FIX to recommend solutions that conform to all predefined constraints and maximize the overall efficiency of an interactive product configuration process. Xue et al. [74] combined IBEA with differential evolution to enhance the correctness and diversity of solutions produced. Hierons et al. [34] proposed a method called SIP that optimizes first on the number of constraints and then on the other objectives. They first adopted two SPLs (AmazonEC2 and Drupal) with realistic attribute values.

Sayyad et al. conducted the first and detailed study on MOEAs for the SPL optimization problem in a series of papers [62–64]. They compared the performance of seven widely-used MOEAs and identified the IBEA to be superior to the other MOEAs for the problem [64]. To deal with the constraints defined in the feature model, they defined the number of constraints that are violated in the generated solutions as an optimization objective to minimize. Other optimization objectives are relevant to user preferences, such as minimizing cost and minimizing the number of defects.

To further enhance the scalability of IBEA to large SPLs (e.g., Linux with 6,888 features), Sayyad et al. [63] proposed an approach that improves the initial population of IBEA with a valid solution, i.e., a *seed*. They reported that IBEA is scalable to large SPLs with a *rich seed* that is a valid solution with many features selected, and using a rich seed is even more effective than using many seeds.

Henard et al. [32] proposed the first hybrid algorithm, called SATIBEA, by combining IBEA with SAT solving. Their SAT solving part encodes a typical feature model (without attribute constraints) in a CNF formula and asks a SAT solver to return a valid solution [33]. They incorporated SAT solving into the mutation operator of IBEA. Specifically, SATIBEA has a probability to replace a randomly-mutated (possibly invalid) solution with a new valid one returned by the SAT solver, such that the predefined constraints are resolved explicitly and some solutions in the population are guaranteed to be valid. Their empirical results on large SPLs demonstrated that SATIBEA significantly outperforms Sayyad et al.'s approaches [63, 64], which established the current state-of-the-art.

2.3 Subjects

Scalability is a key concern for addressing the SPL optimization problem in practice. Since small problems with fewer than 45 features have been reported to be solved exactly and efficiently [22, 25, 51], we aim at large, constrained, real-world SPLs in this paper. Table 1 lists the SPL subjects in our evaluation. For each SPL, we report the version, the number of features, the number of cross-tree constraints (i.e., constraints defined among features), the number of quality attributes of features, the type of attribute values (i.e., synthetic or realistic), and the number of attribute constraints (i.e., constraints defined among the quality attributes of features).

The five subjects (eCos, FreeBSD, Fiasco, uClinux and Linux) are the five largest available real-world SPLs hitherto reported in the literature. Their feature models were taken from the Linux Variability Analysis Tools (LVAT) repository.[1] Later, state-of-the-art methods [32, 63] augmented each feature of each feature model

[1] http://code.google.com/p/linux-variability-analysis-tools

Table 1: Overview of subjects. CTC – Cross-tree constraints; Attr – Attribute

SPL	Version	#Features	#CTC	#Attr	Type of Attr Values	#Attr Constraints
eCos	3.0	1,244	3,146	3	synthetic	N/A
FreeBSD	8.0.0	1,396	62,183	3	synthetic	N/A
Fiasco	2011081207	1,638	5,228	3	synthetic	N/A
uClinux	20100825	1,850	2,468	3	synthetic	N/A
Linux	2.6.28.6	6,888	343,944	3	synthetic	N/A
Drupal	7.23	47	23	22	realistic	N/A
AmazonEC2	12 June 2014	81	3,859	17	realistic	17,049

Table 2: Ratios of invalid solutions in the final populations produced by state-of-the-art approaches

SPL	IBEA [63]	SATIBEA [32]
eCos	0%	4.94%
FreeBSD	2%	89.02%
Fiasco	0%	63.86%
uClinux	69%	0.02%
Linux	100%	81.83%

with three quality attributes: `cost`, `defects`, and `used_before`. They set a fixed, synthetic value for each attribute arbitrarily with a uniform distribution. We use the same publicly-available dataset deployed by the state-of-the-art [32].

The other two subjects (Drupal and AmazonEC2) are small-scale, but they are, to the best of our knowledge, the only two available real-world SPLs with realistic attribute values [34]. Drupal [60, 65] represents the feature model of an open-source Web content management framework Drupal. This model provides realistic values of 22 quality attributes, extracted from Drupal's GIT repository and issue tracking system, including size, changes, defects, test cases, and developers. AmazonEC2 [20] records the configuration space of Amazon EC2 and computing service. It contains realistic values of 17 quality attributes mainly related to pricing, usage, and hardware constraints, including cost per hour, cost per month, RAM and storage. Moreover, different from all other SPL subjects, the attributes in AmazonEC2 do not have a fixed value. Instead, the value of each attribute is derived from the selected features using thousands of attribute constraints of the form like "`(M3.large AND OneYear AND Sydney AND RedHat AND Public) IMPLIES (upfrontCost==249 AND costHour==0.235)`". Hierons et al. also studied this subject, but they "assigned a random value to each attribute within its domain" [34]. In contrast, we stick with the real data regarding attribute values and attribute constraints reported by the original literature [20], in which all attribute values and constraints were captured from a real AWS (Amazon Web Services) snapshot on June 12, 2014.

2.4 Motivation Examples

2.4.1 Ratio of Invalid Solutions. To motivate our study, we first perform state-of-the-art approaches [32, 63] on the five large SPLs and calculate the ratio of invalid solutions in the final population produced by each algorithm for each SPL. In Table 2, the data of column "IBEA" were reported by Sayyad et al. [63], while the column "SATIBEA" records our experimental results (please refer to Section 4.1 for our experimental setup). As depicted in Table 2, IBEA produces less invalid solutions for three smaller SPLs (eCos, FreeBSD and Fiasco), while SATIBEA works better for two larger SPLs (uClinux and Linux). However, both approaches produce valid solutions without guarantee. For the largest subject Linux, both approaches have a high probability (above 81%) to produce invalid solutions; in the worst case, IBEA fails to produce any valid solutions, which makes the search effort all in vain. This indicates the need for the methods that handle constraints more effectively.

2.4.2 Realistic Attribute Constraints. In practice, the predefined constraints may exist not only among features but also among the quality attributes of features. In the past, most studies on the SPL optimization problem augment feature models with quality attributes in a simple way: *1) each feature has the same set of quality attributes; 2) each attribute of a feature has a fixed value; and 3) the fitness of a solution is calculated by simply aggregating the quality attributes of all selected features*. For example, the synthetic attribute values of the five large SPLs in Table 1 were generated in this way. In this case, even if a solution produced is invalid, the MOEA still works, because the fitness of the solution can be calculated anyway.

Different from all other SPL subjects, as shown in Figure 1, AmazonEC2 illustrates a more realistic and more challenging case: *1) each feature has a different set of attributes; 2) each attribute of a feature does not have a fixed value; and 3) the fitness of a solution is derived from the selected features using a number of attribute constraints*. For example, the calculation of the total cost of an EC2 instance (*Inst*) conforms to a set of attribute constraints, a part of which is illustrated as follows (*BlockS* indicates block storage):

$$\begin{aligned} EC2.totalCost &= EC2.costMonth * Purchase.period + \\ &\quad EC2.upfrontCost \\ EC2.costMonth &= (Inst.costHour + Inst.dedicatedFee) * \\ &\quad Purchase.signedUsage + BlockS.costMonth \\ BlockS.costMonth &= BlockS.extraSpace * BlockS.costGB \end{aligned}$$

Note that these attribute constraints are non-Boolean over integer or real-number variables together with arithmetic or relational operators [10, 23, 50, 54, 55], which cannot be straightforwardly addressed by the state-of-the-art [32]. In such a case, an invalid solution that violates either cross-tree constraints or attribute constraints, breaks previous MOEAs, because the fitness of the solution cannot be acquired by simple aggregation. This indicates the need for a new adaption of MOEAs to meet the complex constraints and to calculate the fitness of solutions.

3 ALGORITHMS

This section describes the framework and operators of the algorithms in our evaluation. Moreover, we explain five algorithm variants we design for answering the research questions **RQ2** to **RQ4**.

3.1 Framework and Operators

The basic algorithmic framework of our evaluated algorithms follows the IBEA template of jMetal, an open-source Java framework for multi-objective optimization [18, 19]. The details of the classic IBEA can be found elsewhere [77].

We adopt the algorithmic template supporting binary solution type. That is, each feature is binary, and each solution is encoded as a binary string, where the number of bits is equal to the number of features. If a feature is selected, then the corresponding bit value is set to 1, and 0 otherwise.

Standard operators include: single-point crossover, bit-flip mutation, and binary tournament selection. The standard initial population generation follows a *random* strategy: the initial population is composed of a set of randomly-generated binary strings, each of which represents an arbitrary solution that may not be valid.

Table 3 lists the values of key parameters of the state-of-the-art SATIBEA. We use the same settings as the ones reported by Henard et al. [32]. Note that, once a mutation operator is enabled at a rate of 0.001, SATIBEA has a probability of 0.98 to perform the standard bit-flip mutation, and with a probability of 0.02, SATIBEA returns a valid solution by using the SAT solving.

3.2 Algorithm Variants

We use the state-of-the-art SATIBEA as a baseline and design five algorithm variants by incorporating SAT solving into the initial population generation and the mutation operator of IBEA in different ways. Table 4 summarizes the key characteristics of our algorithm variants, compared to the state-of-the-art.

To answer the research question **RQ2**, we design two algorithm variants SATIBEAv1 and SATIBEAv2, and we compare them to the original SATIBEA. As listed in Table 4, we fix the mutation operator (columns λ and δ) and adopt three different strategies to generate the initial population (column TIP) for the algorithms. SATIBEA adopts the standard, *random* way of initial population generation, that is, such an initial population may contain invalid solutions. SATIBEAv1 adopts a *valid* strategy: the underlying SAT solver is repeatedly called to generate valid solutions until the initial population reaches a designated size, that is, all solutions in the initial population are valid. To improve the diversity of the solutions generated, we use the same strategy of randomly permuting the parameters of the underlying Sat4j SAT solver [11] as reported by Henard et al. [32]. SATIBEAv2 augments SATIBEA with the method suggested by Sayyad et al. [63]: the initial population starts with all randomly-generated solutions and then any one of them is replaced with a *rich seed*, that is, a valid solution with many features selected.

To answer the research question **RQ3**, we design two algorithm variants SATIBEAv3 and SATIBEAv4, and we compare them to the original SATIBEA. As listed in Table 4, the three algorithms all use random initial populations, but they adopt three different mutation strategies. SATIBEA has a rate of 0.001 to perform mutation, and it has a probability of 0.02 to fix a solution to be valid after mutation.

Table 3: Overview of the parameter settings of SATIBEA

Parameter	Value
Population size	300
Archive size	300
Single-point crossover rate	0.05
Mutation rate	0.001
Probability of using standard bit-flip mutation	0.98
Probability of using SAT solving for mutation	0.02

Table 4: Characteristics of SATIBEA and its five variants. λ – Mutation rate; δ – Probability of using SAT solving for mutation; TIP – Type of initial population generation

Algorithm	λ	δ	TIP
SATIBEA	0.001	0.02	random
SATIBEAv1	0.001	0.02	valid
SATIBEAv2	0.001	0.02	random + rich seed
SATIBEAv3	1	1	random
SATIBEAv4	0.001	1	random
SATIBEAv5	1	1	valid

SATIBEAv3 and SATIBEAv4 have different rates ($\lambda = 1$ and 0.001) to perform mutation, but they are both sure ($\delta = 1$) to fix a solution to be valid after mutation.

As explained in Section 2.4, the subject AmazonEC2 contains a number of attribute constraints, and we have to preserve valid solutions all along the way (that is, both in the initial population and at each iteration of MOEAs) to ensure that the fitness calculation and even the entire MOEA work. To answer the research question **RQ4**, we design an algorithm variant SATIBEAv5 that starts with all valid solutions in the initial population, and it is sure to perform mutation ($\lambda = 1$) and then fix a solution to be valid ($\delta = 1$). Moreover, SATIBEAv5 incorporates a subroutine that resolves non-Boolean attribute constraints and always preserves valid solutions.

4 EXPERIMENTS

In this section, we first describe the experimental setup and then present the results of a series of experiments.

4.1 Setup

We describe our subjects in Section 2.3 and the settings of our algorithms in Section 3. This section introduces performance metrics, measurement settings, and statistical tests. We will explain the optimization objectives later when presenting experimental results.

4.1.1 Performance Metrics. We evaluate the performance of the studied algorithms in terms of three aspects: 1) the quality of the approximate (Pareto) front produced, 2) the convergence to the exact Pareto front, and 3) the diversity of the solutions produced. Following the suggestions from Henard et al. [32], the comparisons of the convergence and the diversity are only important when there is quality in the solutions found.

As suggested by Zitzler et al. [78], we measure the quality of an approximate front using the following three metrics:

- *Hypervolume (HV)* [12]. This metric calculates the volume of the subspace covered by an approximate front A. It evaluates how well the approximate front A fulfills the optimization objectives. A higher HV indicates a better Pareto front.
- *Epsilon (ϵ)* [78]. This metric calculates the shortest distance that is required to transform every solution in an approximate front A to dominate the reference front R. It evaluates how close an approximate front A is to the reference front R. A lower ϵ indicates a better Pareto front.
- *Inverted Generational Distance (IGD)* [35]. This metric calculates the average distance from the solutions belonging to the reference front R to the closest solution in an approximate front A. It complements the ϵ metric to evaluate how close an approximate front A is to the reference front R. A lower IGD indicates a better Pareto front.

We measured the convergence to the exact Pareto front using the following metric:

- *Error Ratio (ER)* [41]. This metric calculates the percentage of solutions in an approximate front A that are not solutions in the reference front R. It evaluates the proportion of non-true solution in an approximate front A. A lower ER indicates a better Pareto front.

We measured the diversity using the following two metrics:

- *Generalized Spread (GS)* [16]. This metric calculates the extent of spread in the solutions of an approximate front A. It evaluates the distribution of all solutions among all the objectives. A higher GS indicates a better Pareto front.
- *Pareto Front Size* (PFS) [32]. This meric is the number of solutions in a Pareto front A. A higher Pareto front size is preferred, since more options are given to the user.

The computation of many metrics requires the reference front R. Ideally, the reference front is supposed to be the exact Pareto front. Since most multi-objective optimization problems are NP-hard, it is usually infeasible to acquire the exact Pareto front. Therefore, we acquired the reference front R by calculating the best solutions found so far by all the studied algorithms [75]. Specifically, we made a union of all the approximate fronts found by each algorithm, and then we discarded all the dominated solutions and used all the non-dominated solutions composing the reference front.

4.1.2 Measurement Settings. All measurements of each algorithm were performed on Windows 7 (64bit) Machine with Intel Core i5-4690K CPU 3.5GHz and 16GB RAM. Each algorithm is compute-bound and not memory intensive. Since Henard et at. reported that SATIBEA stabilizes on its ultimate solutions in 15 minutes [32], we performed each studied algorithm 15 minutes to compute the approximate Pareto front for each subject.

To reduce measurement fluctuations caused by randomness (e.g., the randomness of performing crossover and mutation), we independently executed each algorithm 30 times for each subject to support inferential statistical testing for significance and assessment of effect size. We took both the median and mean values of the measurements for analysis.

4.1.3 Statistical Tests. To make fair comparisons, we adopted the same techniques suggested by the state-of-the-art [5, 32] for statistical testing. Based on the experimental results of 30 executions per algorithm, we conducted a statistical test at the significance level of 5%. We performed the Mann-Whitney U test, which is a non-parametric test and makes fewer assumptions regarding the underlying populations. We calculated the estimated probability, i.e., p-value, that an algorithm produces different results than the other. Furthermore, we reported the non-parametric effect size measure $\hat{A}_{12}$ [70]. It indicates the performance superiority, and it measures the extent to which an algorithm outperforms the other. The superiority between algorithms are considered as small, medium and large when $\hat{A}_{12}$ value is over 0.56, 0.64 and 0.71, respectively.

4.2 Different Initial Populations

To answer the research question **RQ2**, we compare the performance of three algorithms (SATIBEA, SATIBEAv1 and SATIBEAv2) that adopt different strategies to generate the initial population.

To achieve a fair implementation of SATIBEAv2, we use the same subjects and the corresponding rich seeds, published by Sayyad et al. [63]. Hence, in this experiment, we evaluate only the five large subjects. Moreover, we consider the same optimization objectives as designed by the state-of-the-art [32, 63]. For each subject, we aim at minimizing the following five objectives:

(1) *Correctness*: the number of constraints that were not satisfied.
(2) *Richness of features*: the number of deselected features in a solution.
(3) *Features that were not used before*: the number of features that were not used before in a solution.
(4) *Known defects*: the number of known defects in a solution.
(5) *Cost*: the cost of a solution.

State-of-the-art methods use *Correctness* as one of the optimization objectives and permit invalid solutions. However, invalid solutions are unbuildable in practice. The results of performance metrics might be misleading when incorporating invalid solutions in the final population for evaluation. Hence, we conduct a controlled experiment that includes and excludes invalid solutions from the final population for performance evaluation. Note that a valid solution holds the minimum value of 0 regarding *Correctness*. We present our experimental results of including and excluding invalid solutions as follows.

4.2.1 Including Invalid Solutions. Table 5 records the experimental results of three algorithms for five subjects when including invalid solutions for performance evaluation. The columns "SATIBEA (v0)", "SATIBEAv1 (v1)", and "SATIBEAv2 (v2)" list the measured results of each algorithm. We report the median and mean values of the measured results for 30 executions. We conduct statistical analysis for the comparisons of any two approaches and calculate the p-values and the effect sizes $\hat{A}_{12}$. The columns "v0 VS v1", "v0 VS v2", and "v1 VS v2" list three pairs of statistical analysis results. From them, the other three counterparts (v1 VS v0, v2 VS v0, and v2 VS v1) can be inferred, e.g., $\hat{A}_{12}$(v0 VS v1) + $\hat{A}_{12}$(v1 VS v0) = 1. So we report only a half of $\hat{A}_{12}$ results. The rows of the table record the measured results of six performance metrics for each subject after 30 runs per algorithm. In the table, the highlighted number indicates the best case among three algorithms with statistical significance (p-value < 0.05 and $\hat{A}_{12}$ > 0.56), and the number

Table 5: Experimental results (including invalid solutions) of three algorithms for five subjects; the highlighted number indicates the best case among three algorithms with statistical significance; the number in bold indicates a better case between two algorithms with statistical significance

SPL	Metric	SATIBEA (v0)		SATIBEAv1 (v1)		SATIBEAv2 (v2)		v0 VS v1		v0 VS v2		v1 VS v2	
		Median	Mean	Median	Mean	Median	Mean	*p*-value	$\hat{A}_{12}$	*p*-value	$\hat{A}_{12}$	*p*-value	$\hat{A}_{12}$
eCos	HV	2.8198e-1	2.8172e-1	2.8159e-1	2.8131e-1	2.8101e-1	2.8111e-1	3.2553e-1	0.574	1.6238e-1	0.606	6.5204e-1	0.534
	ϵ	6.7416e-2	6.9620e-2	7.1178e-2	7.3257e-2	7.1465e-2	7.0994e-2	9.0401e-2	0.372	3.9514e-1	0.436	5.1045e-1	0.550
	IGD	5.1142e-4	5.0868e-4	4.9895e-4	4.9895e-4	5.0793e-4	5.0578e-4	6.3533e-2	0.640	6.2040e-1	0.538	2.2257e-1	0.408
	ER	2.3167e-1	2.4758e-1	2.6210e-1	2.5367e-1	2.2667e-1	2.4121e-1	4.5521e-1	0.443	8.0724e-1	0.519	2.9376e-1	0.579
	GS	8.7907e-1	8.9251e-1	9.0626e-1	9.0403e-1	8.8404e-1	8.8430e-1	3.2553e-1	0.426	9.1171e-1	0.509	1.1199e-1	0.620
	PFS	168	167.76	168	168.47	167	168.4	8.8224e-1	0.488	7.6712e-1	0.477	9.7040e-1	0.497
FreeBSD	HV	2.6281e-1	2.6289e-1	2.6528e-1	2.6497e-1	2.6366e-1	2.6377e-1	**6.2027e-4**	**0.242**	1.7145e-1	0.397	**1.5014e-2**	**0.683**
	ϵ	6.6622e-2	6.8798e-2	7.0486e-2	7.0977e-2	6.9307e-2	6.8553e-2	2.8362e-1	0.419	7.6744e-1	0.477	4.3322e-1	0.559
	IGD	4.5240e-4	4.5256e-4	4.5713e-4	4.5446e-4	4.3575e-4	4.4011e-4	6.7350e-1	0.468	5.1877e-2	0.647	6.7869e-2	0.638
	ER	1.6839e-1	1.6881e-1	1.0863e-1	1.2327e-1	1.8404e-1	1.9427e-1	3.7576e-3	0.718	1.7849e-1	0.398	1.5839e-4	0.216
	GS	8.7730e-1	9.0226e-1	8.9863e-1	9.1896e-1	9.3451e-1	9.2889e-1	2.1702e-1	0.407	**4.3584e-2**	**0.348**	4.8252e-1	0.447
	PFS	220	218.03	219	215.6	220.5	218.9	4.4130e-1	0.558	7.8992e-1	0.479	2.8309e-1	0.419
Fiasco	HV	2.4450e-1	2.4610e-1	2.4390e-1	2.4418e-1	2.4360e-1	2.4400e-1	3.4029e-1	0.572	2.4581e-1	0.588	8.7663e-1	0.512
	ϵ	1.0437e-1	1.0509e-1	1.0961e-1	1.1424e-1	1.0346e-1	1.0783e-1	2.4428e-2	0.331	8.3020e-1	0.483	7.4262e-2	0.634
	IGD	7.8360e-4	7.6171e-4	8.1268e-4	8.1795e-4	7.8501e-4	7.8700e-4	3.6439e-2	0.342	5.3951e-1	0.453	7.4827e-2	0.634
	ER	2.5833e-1	2.5670e-1	2.7333e-1	2.7500e-1	2.6000e-1	2.7424e-1	4.1179e-1	0.438	5.7913e-1	0.458	8.6497e-1	0.513
	GS	6.0657e-1	6.1202e-1	6.2190e-1	6.2696e-1	6.2803e-1	6.3613e-1	1.6687e-1	0.396	5.0120e-2	0.352	5.9969e-1	0.460
	PFS	115	139.67	112	111.93	113	113.57	7.3308e-2	0.635	3.1027e-1	0.577	3.4682e-1	0.429
uClinux	HV	2.9389e-1	2.9321e-1	2.9503e-1	2.9428e-1	2.9046e-1	2.9109e-1	4.2039e-1	0.439	**2.8129e-2**	**0.666**	**1.6798e-3**	**0.737**
	ϵ	6.4749e-2	6.4442e-2	6.3492e-2	6.4177e-2	6.2706e-2	6.2299e-2	8.2415e-1	0.517	2.5448e-1	0.586	2.7663e-1	0.582
	IGD	5.6107e-4	5.8913e-4	5.5922e-4	5.7262e-4	5.7586e-4	5.7781e-4	9.7052e-1	0.497	9.8231e-1	0.498	9.8231e-1	0.502
	ER	3.2667e-1	3.2656e-1	3.5500e-1	3.5778e-1	4.0000e-1	4.0322e-1	4.9166e-1	0.448	6.6721e-2	0.362	2.6739e-1	0.416
	GS	5.8160e-1	5.9499e-1	5.9202e-1	5.8721e-1	5.9410e-1	5.9710e-1	8.4180e-1	0.516	6.3088e-1	0.463	5.0114e-1	0.449
	PFS	106	107.33	107	108.2	109	108.37	5.8868e-1	0.459	4.3648e-1	0.441	8.3563e-1	0.483
Linux	HV	2.2424e-1	2.2401e-1	2.2664e-1	2.2750e-1	2.2406e-1	2.2415e-1	**4.0330e-3**	**0.283**	7.8446e-1	0.521	**1.5638e-2**	**0.682**
	ϵ	1.1176e-1	1.1089e-1	1.1232e-1	1.0806e-1	1.1355e-1	1.1337e-1	9.3519e-1	0.507	6.6273e-1	0.467	6.3087e-1	0.463
	IGD	3.5543e-4	3.5938e-4	3.5583e-4	3.7242e-4	3.4873e-4	3.5502e-4	8.4180e-1	0.484	7.9585e-1	0.520	4.9178e-1	0.552
	ER	2.4150e-1	2.4688e-1	1.7036e-1	1.6372e-1	2.3743e-1	2.4480e-1	1.0274e-6	0.868	8.2449e-1	0.517	3.4400e-6	0.151
	GS	9.4074e-1	9.5611e-1	9.5512e-1	9.8042e-1	9.3218e-1	9.6783e-1	4.2039e-1	0.439	8.9999e-1	0.510	2.8378e-1	0.581
	PFS	290	290.03	289	288.2	289.0	288.73	1.3105e-1	0.613	2.4888e-1	0.587	6.3867e-1	0.464

in bold indicates a better case between any two algorithms with statistical significance.

From Table 5, we observe that SATIBEAv1 outperforms the other two algorithms on the quality metric HV for two subjects (FreeBSD and Linux), with a large effect size ($\hat{A}_{12} > 0.71$). For uClinux, either SATIBEA or SATIBEAv1 works better than SATIBEAv2 in terms of HV. In addition, for eCos and Fiasco, no results with statistical significance can be observed.

4.2.2 Excluding Invalid Solutions. Table 6 presents the experimental results of three algorithms for five subjects when excluding invalid solutions for performance evaluation. As shown in the table, SATIBEA outperforms the other two algorithms only on the diversity metric PFS for Fiasco. In terms of the quality metric HV, SATIBEA works better than SATIBEAv2 for two subjects (Fiasco and uClinux), while SATIBEAv1 also works better than SATIBEAv2 for uClinux. For three subjects (eCos, FreeBSD and Linux), we did not observe statistically significant results in terms of any quality metric, which has a higher priority for performance evaluation.

4.3 Different Mutation Operators

To answer the research question **RQ3**, we compare the performance of three algorithms (SATIBEA, SATIBEAv3 and SATIBEAv4) that adopt different mutation strategies. In this experiment, we evaluate six subjects, including the five large SPLs and Drupal. Drupal has 22 quality attributes of features, and the values of these quality attributes are realistic. We use the same optimization objectives as designed by Hierons et al. [34] for Drupal as follows:

(1) *Correctness*: we seek to minimize the number of constraints that were not satisfied.
(2) *Richness of features*: we seek to minimize the number of deselected features in a solution.
(3) *Number of lines of code*: this should be minimized.
(4) *Cyclomatic complexity*: this should be minimized.
(5) *Test assertions*: this should be maximized.
(6) *Number of installations that contain the feature*: this should be maximized.
(7) *Number of developers*: this should be minimized.
(8) *Number of changes*: this should be minimized.

Again, we believe that invalid solutions are useless in practice and might bring misleading results of performance metrics. Hence, in this experiment, we consider only the evaluation results after excluding invalid solutions from the final population.

Table 7 presents our experimental results of three algorithms for six subjects when excluding invalid solutions for performance evaluation. In terms of the quality metric HV, SATIBEA outperforms the other two algorithms for four subjects (eCos, uClinux, Linux and Drupal), even with the largest effect size ($\hat{A}_{12}$=1). Moreover, SATIBEA works the best on GS for four subjects and on PFS for three subjects. Only for Fiasco, SATIBEAv3 works the best among three algorithms in terms of HV.

Table 6: Experimental results (excluding invalid solutions) of three algorithms for five subjects; the highlighted number indicates the best case among three algorithms with statistical significance; the number in bold indicates a better case between two algorithms with statistical significance

SPL	Metric	SATIBEA (v0)		SATIBEAv1 (v1)		SATIBEAv2 (v2)		v0 VS v1		v0 VS v2		v1 VS v2	
		Median	Mean	Median	Mean	Median	Mean	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$
eCos	HV	2.8004e-1	2.8012e-1	2.7883e-1	2.7920e-1	2.7974e-1	2.7955e-1	7.0127e-2	0.637	1.3732e-1	0.612	5.9969e-1	0.460
	ϵ	6.7416e-2	6.9535e-2	7.1178e-2	7.3228e-2	7.0873e-2	7.0878e-2	8.4913e-2	0.370	3.9926e-1	0.436	4.9630e-1	0.552
	IGD	3.7109e-4	3.7310e-4	3.6892e-4	3.7368e-4	3.7972e-4	3.7667e-4	9.4696e-1	0.494	3.7904e-1	0.433	4.5530e-1	0.443
	ER	2.3063e-1	2.5022e-1	2.5922e-1	2.5527e-1	2.3355e-1	2.4201e-1	4.7335e-1	0.446	7.2826e-1	0.527	2.8046e-1	0.582
	GS	8.1640e-1	8.2488e-1	8.1309e-1	8.2515e-1	8.0083e-1	8.0512e-1	7.5059e-1	0.476	4.4642e-1	0.558	2.5805e-1	0.586
	PFS	160	160.2	157.5	158.73	158	160.23	3.1413e-1	0.576	7.0042e-1	0.529	6.2514e-1	0.463
FreeBSD	HV	7.0157e-2	8.1413e-2	9.3336e-2	8.5262e-2	5.6607e-2	8.0614e-2	8.1875e-1	0.482	9.9410e-1	0.501	7.0617e-1	0.529
	ϵ	3.5237e-1	3.5548e-1	3.6444e-1	3.4277e-1	3.5300e-1	3.5404e-1	8.3025e-1	0.483	7.8446e-1	0.521	5.4930e-1	0.546
	IGD	7.7385e-3	8.6702e-3	9.2703e-3	8.6230e-3	7.0031e-3	8.9212e-3	7.1719e-1	0.472	5.0114e-1	0.551	3.4029e-1	0.572
	ER	3.8589e-1	3.7616e-1	1.9798e-1	3.0608e-1	4.5238e-1	4.2922e-1	2.9601e-1	0.579	4.1170e-1	0.438	8.3133e-2	0.369
	GS	1.0346e-0	1.1289e-0	1.0414e-0	1.1106e-0	1.2429e-0	1.2260e-0	7.9585e-1	0.480	7.9782e-2	0.368	9.6263e-2	0.374
	PFS	15	16.9	9.5	13.77	11.5	13.67	5.0577e-2	0.647	9.0043e-1	0.628	8.8217e-1	0.488
Fiasco	HV	2.5548e-1	2.5639e-1	2.5364e-1	2.5388e-1	2.5260e-1	2.5233e-1	2.6433e-1	0.584	**2.0681e-2**	**0.674**	1.0233e-1	0.623
	ϵ	1.0811e-1	1.0890e-1	1.1957e-1	1.2373e-1	1.1218e-1	1.1897e-1	5.3049e-3	0.290	1.5352e-1	0.392	1.5783e-1	0.607
	IGD	1.9590e-3	1.9082e-3	2.0830e-3	2.0658e-3	1.9883e-3	1.9903e-3	1.5638e-2	0.318	3.6322e-1	0.431	5.3685e-2	0.646
	ER	3.0707e-1	2.8904e-1	3.1218e-1	3.1584e-1	3.4543e-1	3.4525e-1	4.0353e-1	0.437	2.1503e-2	0.327	1.8085e-1	0.399
	GS	7.0761e-1	7.0058e-1	7.2691e-1	7.1262e-1	6.7259e-1	6.8723e-1	7.0617e-1	0.471	5.9969e-1	0.540	1.6238e-1	0.606
	PFS	34	39.43	32	31.7	31	31.77	**1.8908e-2**	**0.676**	**1.6666e-2**	**0.679**	7.3706e-1	0.526
uClinux	HV	2.9389e-1	2.9321e-1	2.9503e-1	2.9428e-1	2.9046e-1	2.9109e-1	4.2039e-1	0.439	**2.8129e-2**	**0.666**	**1.6798e-3**	**0.737**
	ϵ	6.4749e-2	6.4442e-2	6.3492e-2	6.4177e-2	6.2706e-2	6.2299e-2	8.2415e-1	0.517	2.5448e-1	0.586	2.7663e-1	0.582
	IGD	5.3137e-4	5.6145e-4	5.2942e-4	5.4339e-4	5.4694e-4	5.4873e-4	9.5873e-1	0.504	9.2344e-1	0.508	9.8231e-1	0.502
	ER	3.2667e-1	3.2654e-1	3.5500e-1	3.5778e-1	4.0000e-1	4.0322e-1	4.9166e-1	0.448	6.6721e-2	0.362	2.6739e-1	0.416
	GS	5.3045e-1	5.4256e-1	5.4181e-1	5.3675e-1	5.4442e-1	5.4690e-1	9.3519e-1	0.507	5.6922e-1	0.457	5.4933e-1	0.454
	PFS	106	107.3	107	108.2	109	108.37	5.7343e-1	0.457	4.2785e-1	0.440	8.3563e-1	0.484
Linux	HV	1.6473e-1	1.6597e-1	1.6424e-1	1.6473e-1	1.5979e-1	1.6268e-1	4.9178e-1	0.552	7.2446e-2	0.636	3.7904e-1	0.567
	ϵ	1.2114e-1	1.1766e-1	1.1987e-1	1.1781e-1	1.3141e-1	1.2393e-1	8.7663e-1	0.512	1.6685e-1	0.396	2.2823e-1	0.409
	IGD	4.7509e-4	4.7752e-4	4.7885e-4	5.1449e-4	4.7619e-4	4.8519e-4	2.7071e-1	0.417	6.8432e-1	0.469	4.3764e-1	0.559
	ER	3.0485e-1	3.0681e-1	2.5352e-1	2.5996e-1	2.8132e-1	2.8193e-1	6.5156e-3	0.705	1.1874e-1	0.618	1.6911e-1	0.396
	GS	9.8430e-1	9.8237e-1	1.0189e-0	1.0338e-0	9.5972e-1	9.7451e-1	9.3341e-2	0.373	3.6322e-1	0.569	**2.3243e-2**	**0.671**
	PFS	47.5	49.73	49	48.03	49	48.73	4.0217e-1	0.563	9.0563e-1	0.509	5.7354e-1	0.457

4.4 Preserving Validity All Along the Way

To answer the research question **RQ4**, we design an new algorithm SATIBEAv5 that preserves valid solutions all along the way, that is, both in the initial population and at each iteration of the algorithm. Moreover, we compare the performance of SATIBEAv5 and SATIBEA for all seven subjects, including the five large SPLs, Drupal, and AmazonEC2. AmazonEC2 contains 17 quality attributes, along with realistic attribute values, and 17,049 realistic attribute constraints. We use the same optimization objectives as designed by Hierons et al. [34] for AmazonEC2 as follows:

(1) *Correctness*: we seek to minimize the number of constraints that were not satisfied.
(2) *Richness of features*: we seek to minimize the number of deselected features in a solution.
(3) *EC2.costMonth*: this should be minimized.
(4) *Instance.cores*: this should be maximized.
(5) *Instance.ecu*: this should be maximized.
(6) *Instance.ram*: this should be maximized.
(7) *Instance.costHour*: this should be minimized.
(8) *Instance.ssdBacked*: this should be maximized.

Note that, for AmazonEC2, Hierons et al. "assigned a random value to each attribute within its domain" [34], so that invalid solutions can survive during the search of the algorithm. In contrast, we stick with the real attribute values and constraints reported by the original literature [20]. In this case, as explained in Section 2.4, invalid solutions break the fitness calculation of previous MOEAs that is based on simple aggregation. Therefore, our algorithm SATIBEAv5 preserves only valid solutions.

Table 8 presents our experimental results of two algorithms for seven subjects when excluding invalid solutions for performance evaluation. In terms of the quality metric HV, SATIBEA outperforms SATIBEAv5 for four subjects (eCos, uClinux, Linux and Drupal), while SATIBEAv5 works better for two subjects (FreeBSD and Fiasco), both with a large effect size ($\hat{A}_{12} > 0.71$). As expected, only SATIBEAv5 works for AmazonEC2.

5 DISCUSSION

5.1 Potential Limitations

By empirical study, we demonstrate three potential limitations of preserving invalid solutions when using previous MOEAs for the SPL optimization problem. Firstly, previous MOEAs produce valid solutions without guarantee. As shown in Table 2, state-of-the-art approaches [32, 63] have a high probability to produce invalid solutions, especially for large SPLs. In the worst case, IBEA [63] fails to find any valid solutions. Note that invalid solutions are unbuildable and useless in practice, so producing many invalid solutions makes the costly search effort in vain.

Secondly, the performance evaluation of different MOEAs depends mainly on various performance metrics calculated on the

Table 7: Experimental results (excluding invalid solutions) of three algorithms for six subjects; the highlighted number indicates the best case among three algorithms with statistical significance; the number in bold indicates a better case between two algorithms with statistical significance

SPL	Metric	SATIBEA (v0)		SATIBEAv3 (v3)		SATIBEAv4 (v4)		v0 VS v3		v0 VS v4		v3 VS v4	
		Median	Mean	Median	Mean	Median	Mean	*p*-value	$\hat{A}_{12}$	*p*-value	$\hat{A}_{12}$	*p*-value	$\hat{A}_{12}$
eCos	HV	2.8010e-1	2.8017e-1	2.3625e-1	2.3595e-1	2.3406e-1	2.3428e-1	**3.0199e-11**	**1.000**	**3.0199e-11**	**1.000**	1.3732e-1	0.612
	ϵ	6.1871e-2	6.5651e-2	1.1835e-1	1.1913e-1	1.2398e-1	1.2790e-1	2.9935e-11	0.000	2.9935e-11	0.000	5.9557e-3	0.293
	IGD	8.3866e-4	8.5297e-4	1.0277e-3	1.0333e-3	1.0496e-3	1.0510e-3	3.0199e-11	0.000	3.0199e-11	0.000	1.6285e-2	0.319
	ER	1.0671e-1	1.2274e-1	6.6207e-1	6.5564e-1	6.5696e-1	6.5744e-1	3.0180e-11	0.000	3.0180e-11	0.000	8.0727e-1	0.481
	GS	8.1603e-1	8.2438e-1	6.5141e-1	6.5166e-1	6.5806e-1	6.6119e-1	**4.9752e-11**	**0.994**	**4.0772e-11**	**0.997**	2.8378e-1	0.419
	PFS	160	160.20	166	165.03	165.50	165.03	**9.1421e-03**	**0.304**	**1.1600e-02**	**0.310**	8.0109e-1	0.519
FreeBSD	HV	7.0509e-2	8.1759e-2	2.2086e-1	2.1665e-1	2.1795e-1	2.1768e-1	**3.0199e-11**	**0.000**	**3.0199e-11**	**0.000**	9.4696e-1	0.494
	ϵ	3.2874e-1	3.5379e-1	1.0296e-1	1.1103e-1	1.1772e-1	1.1476e-1	3.0161e-11	1.000	3.0161e-11	1.000	6.5203e-1	0.466
	IGD	3.6405e-3	4.9820e-3	3.5362e-4	3.6756e-4	3.6149e-4	3.5771e-4	3.0199e-11	1.000	3.0199e-11	1.000	6.3088e-1	0.537
	ER	3.1884e-1	2.9185e-1	5.6803e-1	5.9251e-1	5.5573e-1	5.6882e-1	4.0254e-07	0.119	1.9646e-06	0.142	3.2190e-1	0.575
	GS	1.0364e-0	1.1295e-0	5.7200e-1	5.6888e-1	5.6848e-1	5.6889e-1	**3.0199e-11**	**1.000**	**3.6897e-11**	**0.998**	6.4142e-1	0.536
	PFS	15	16.90	162	175.40	158	169.13	**2.9450e-11**	**0.000**	**2.9506e-11**	**0.000**	2.2780e-1	0.591
Fiasco	HV	2.5125e-1	2.5229e-1	2.7127e-1	2.6977e-1	2.6479e-1	2.6638e-1	**5.0723e-10**	**0.032**	2.2273e-09	0.050	**2.1506e-2**	**0.673**
	ϵ	1.1828e-1	1.1998e-1	6.4516e-2	6.2780e-2	6.9892e-2	6.8166e-2	2.8972e-11	1.000	2.9543e-11	1.000	6.9827e-3	0.297
	IGD	1.6558e-3	1.6168e-3	8.6523e-4	8.6408e-4	8.3411e-4	8.3955e-4	3.0199e-11	1.000	3.0199e-11	1.000	1.3832e-2	0.686
	ER	2.1437e-1	2.1240e-1	5.3782e-1	5.6829e-1	7.3277e-1	6.7284e-1	1.7759e-10	0.020	7.3891e-11	0.010	4.3581e-2	0.348
	GS	7.0888e-1	7.0147e-1	4.0601e-1	4.2529e-1	4.3305e-1	4.2703e-1	**8.9934e-11**	**0.988**	**9.9186e-11**	**0.987**	8.7663e-1	0.488
	PFS	34	39.43	93.50	100.63	124.50	119.73	**2.9376e-11**	**0.000**	**2.9376e-11**	**0.000**	1.2610e-1	0.257
uClinux	HV	2.9389e-1	2.9321e-1	1.1807e-1	1.3439e-1	1.1390e-1	1.3403e-1	**3.0199e-11**	**1.000**	**3.0199e-11**	**1.000**	6.3088e-1	0.537
	ϵ	6.1512e-2	6.2685e-2	2.6685e-1	2.5889e-1	2.7080e-1	2.6404e-1	2.9617e-11	0.000	2.9468e-11	0.000	1.0515e-1	0.378
	IGD	8.2835e-4	8.6534e-4	4.2938e-3	4.0822e-3	4.3783e-3	4.1129e-3	3.0199e-11	0.000	3.0199e-11	0.000	6.1000e-1	0.461
	ER	1.9500e-1	1.9617e-1	4.6564e-1	4.6673e-1	4.6154e-1	4.5959e-1	3.0066e-11	0.000	3.6783e-11	0.002	4.9173e-1	0.552
	GS	5.3027e-1	5.4248e-1	9.1723e-1	9.5887e-1	8.9549e-1	9.2659e-1	**3.0199e-11**	**0.000**	**3.0199e-11**	**0.000**	2.3985e-1	0.589
	PFS	106	107.30	60	59.07	58.50	56.50	**2.9137e-11**	**1.000**	**2.9339e-11**	**1.000**	2.7957e-1	0.582
Linux	HV	1.6442e-1	1.6566e-1	2.6396e-7	2.8052e-7	2.6166e-7	2.4631e-7	**3.0199e-11**	**1.000**	**3.0199e-11**	**1.000**	2.3985e-1	0.589
	ϵ	1.2143e-1	1.1752e-1	9.9648e-1	9.9659e-1	9.9648e-1	9.9670e-1	2.8021e-11	0.000	2.8092e-11	0.000	3.8570e-1	0.435
	IGD	6.8655e-4	6.8310e-4	3.3407e-2	3.3408e-2	3.3407e-2	3.3414e-2	3.0199e-11	0.000	3.0199e-11	0.000	2.3399e-1	0.410
	ER	1.6667e-1	1.6155e-1	0.0000e-0	4.6667e-1	0.0000e-0	4.3333e-1	6.5763e-01	0.533	3.7107e-01	0.567	8.0355e-1	0.517
	GS	9.8419e-1	9.8226e-1	1.0000e-0	1.0000e-0	1.0000e-0	1.0000e-0	1.5722e-01	0.400	1.5722e-01	0.400	N/A	0.500
	PFS	47.5	49.73	1.0	1.0	1.0	1.0	**1.1834e-12**	**1.000**	**1.1834e-12**	**1.000**	N/A	0.500
Drupal	HV	1.7385e-1	1.7382e-1	1.0050e-1	1.0054e-1	1.0000e-1	1.0103e-1	**3.0199e-11**	**1.000**	**3.0199e-11**	**1.000**	8.0727e-1	0.481
	ϵ	9.1875e-2	9.2260e-2	2.1739e-1	2.1779e-1	2.1662e-1	2.1416e-1	2.8646e-11	0.000	2.8718e-11	0.000	3.4394e-1	0.572
	IGD	4.1593e-3	4.1669e-3	2.6269e-3	2.6199e-3	2.5710e-3	2.5965e-3	3.0199e-11	1.000	3.0199e-11	1.000	8.7710e-2	0.629
	ER	0.0000e-0	4.1111e-3	3.4667e-1	3.2869e-1	3.3833e-1	3.4167e-1	3.1411e-12	0.000	3.1340e-12	0.000	5.7410e-1	0.457
	GS	4.5876e-1	4.5631e-1	3.6211e-1	3.7023e-1	3.5540e-1	3.5448e-1	**2.9215e-09**	**0.947**	**4.5043e-11**	**0.996**	3.7904e-1	0.567
	PFS	169	169.63	81	82.40	85	86.30	**2.9339e-11**	**1.000**	**2.9468e-11**	**1.000**	7.1986e-1	0.526

final population of solutions produced, but the population including invalid solutions may lead to misleading evaluation results and thus may recommend an inappropriate algorithm. As demonstrated in Section 4.2, we conduct a controlled experiment that includes and excludes invalid solutions for performance evaluation. From Table 5 (including invalid solutions), we prefer SATIBEAv1 because it outperforms the other two algorithms on HV for two subjects with a large effect size. In contrast, from Table 6 (excluding invalid solutions), the superiority of SATIBEAv1 disappears, and it is hard to statistically distinguish SATIBEA and SATIBEAv1 in terms of any quality metric. The different results of the controlled experiment demonstrate the impact of invalid solutions on the calculation of performance metrics. Again, since invalid solutions are unbuildable in practice, we recommend excluding invalid solutions, *at least*, in the final population for pragmatic performance evaluation.

Thirdly, as illustrated in Section 2.4, AmazonEC2 brings a more realistic and more challenging case: each feature attribute does not have a fixed value, and the fitness of a solution is derived from the selected features using a number of attribute constraints. This breaks all previous MOEAs that permit invalid solutions, because the fitness of an invalid solution cannot be calculated by simple aggregation. As demonstrated in Table 8, only SATIBEAv5 that preserves valid solutions all along the way, works for AmazonEC2.

5.2 Research Questions and Perspectives

We first answer the research questions **RQ2** to **RQ4**, and then we get back to the central question **RQ1**. Regarding **RQ2**, as explained in Section 5.1, invalid solutions influence the calculation of performance metrics, and we recommend a pragmatic performance evaluation that considers only valid solutions, at least, in the final population. Hence, as shown in Table 6 (excluding invalid solutions), we cannot statistically distinguish three algorithms adopting different initial populations in terms of any quality metric. However, in terms of the effort of generating the initial population, SATIBEA is the least and the easiest to build, because it simply uses randomly-generated solutions. Therefore, we recommend the random strategy to generate the initial population.

Regarding **RQ3**, as shown in Table 7, our experimental results on six subjects demonstrate the superiority of SATIBEA among three algorithms adopting different mutation operators. We recommend the mutation operator that has a lower mutation rate of 0.001 and a lower probability of 0.02 to fix a solution to be valid after mutation.

Table 8: Experimental results (excluding invalid solutions) of two algorithms for seven subjects; the number in bold indicates a better case with statistical significance

SPL	Metric	SATIBEA (v0)		SATIBEAv5 (v5)		v0 VS v5	
		Median	Mean	Median	Mean	p-value	$\hat{A}_{12}$
eCos	HV	**2.8009e-1**	**2.8017e-1**	2.2629e-1	2.2605e-1	3.0199e-11	1.000
	ϵ	6.1512e-2	6.4943e-2	1.3881e-1	1.4259e-1	3.0123e-11	0.000
	IGD	6.7217e-4	6.7671e-4	1.3830e-3	1.3880e-3	3.0199e-11	0.000
	ER	1.0672e-1	1.2275e-1	5.1630e-1	5.1857e-1	3.0180e-11	0.000
	GS	8.1603e-1	8.2438e-1	**9.9866e-1**	**9.8890e-1**	9.8329e-08	0.099
	PFS	160	160.20	**166**	**166.40**	4.3833e-03	0.286
FreeBSD	HV	7.0768e-2	8.1951e-2	**1.4023e-1**	**1.4047e-1**	9.5207e-04	0.251
	ϵ	3.3407e-1	3.5161e-1	2.0648e-1	2.0250e-1	1.2416e-07	0.898
	IGD	6.1370e-3	6.6500e-3	1.1740e-3	1.1840e-3	8.1527e-11	0.989
	ER	2.6007e-1	2.6741e-1	7.3604e-1	7.4426e-1	7.5855e-10	0.038
	GS	**1.0360e-0**	**1.1293e-0**	6.8856e-1	6.8271e-1	9.7555e-10	0.960
	PFS	15	16.90	**233**	**232.13**	2.9137e-11	0.000
Fiasco	HV	2.5172e-1	2.5278e-1	**2.5830e-1**	**2.5845e-1**	1.8575e-03	0.266
	ϵ	1.0753e-1	1.0782e-1	6.8608e-2	6.8744e-2	2.2194e-10	0.977
	IGD	1.8580e-3	1.8010e-3	8.2382e-4	8.2485e-4	3.0199e-11	1.000
	ER	2.0010e-1	1.9836e-1	8.2141e-1	8.1841e-1	3.0180e-11	0.000
	GS	**7.0869e-1**	**7.0143e-1**	3.6266e-1	3.6282e-1	3.0199e-11	1.000
	PFS	34	39.43	**238**	**237.73**	2.9190e-11	0.000
uClinux	HV	**2.9389e-1**	**2.9321e-1**	1.0551e-1	1.0615e-1	3.0199e-11	1.000
	ϵ	6.1512e-2	6.2686e-2	2.7954e-1	2.7990e-1	2.9339e-11	0.000
	IGD	8.0522e-4	8.4610e-4	4.9290e-3	4.9220e-3	3.0199e-11	0.000
	ER	1.9500e-1	1.9617e-1	2.8470e-1	2.8498e-1	2.1299e-05	0.180
	GS	5.3027e-1	5.4248e-1	5.6622e-1	5.6575e-1	6.1452e-02	0.359
	PFS	**106**	**107.30**	83.5	83.43	2.9321e-11	1.000
Linux	HV	**1.6455e-1**	**1.6578e-1**	1.3151e-1	1.3216e-1	2.0338e-09	0.951
	ϵ	1.2114e-1	1.1761e-1	1.7652e-1	1.7316e-1	2.6681e-09	0.052
	IGD	3.1842e-4	3.3354e-4	7.7584e-4	7.7920e-4	6.0658e-11	0.008
	ER	2.7726e-1	2.7200e-1	5.8067e-1	5.7535e-1	3.0161e-11	0.000
	GS	**9.8436e-1**	**9.8248e-1**	8.3832e-1	8.3303e-1	6.0459e-07	0.876
	PFS	47.5	49.73	**259.0**	**259.47**	2.9284e-11	0.000
Drupal	HV	**1.6477e-1**	**1.6479e-1**	8.0825e-2	8.0440e-2	3.0199e-11	1.000
	ϵ	8.3278e-2	8.7599e-2	2.7798e-1	2.7867e-1	2.6437e-11	0.000
	IGD	5.2930e-3	5.2910e-3	2.5080e-3	2.5290e-3	3.0199e-11	1.000
	ER	0.0000e-0	4.1120e-3	5.9217e-1	5.8845e-1	3.1602e-12	0.000
	GS	**4.2290e-1**	**4.2159e-1**	3.3713e-1	3.3957e-1	5.4941e-11	0.993
	PFS	169	169.63	**269**	**269.17**	2.9119e-11	0.000
AmazonEC2	HV	N/A	N/A	1.8369e-1	2.2194e-1	N/A	N/A
	ϵ	N/A	N/A	5.1613e-1	4.4232e-1	N/A	N/A
	IGD	N/A	N/A	1.6370e-1	1.6280e-1	N/A	N/A
	ER	N/A	N/A	9.2872e-1	9.2438e-1	N/A	N/A
	GS	N/A	N/A	6.4402e-1	6.5201e-1	N/A	N/A
	PFS	N/A	N/A	103	101.71	N/A	N/A

This mutation operator tends to enhance the exploration capability of MOEAs and improve the diversity of solutions produced.

Regarding **RQ4**, as demonstrated in Table 8, valid solutions must be preserved all along the way for AmazonEC2. However, for the other subjects without attribute constraints, SATIBEA works better for 4 out of 6 sujects. This indicates that preserving valid solutions all along the way *might not always be necessary*. As highlighted in Section 2.4, the key difference between AmazonEC2 and other subjects lies in the characteristics regarding feature attributes and constraints, which can be used for decision making.

Regarding **RQ1**, it is a big question for the entire SBSE community. This paper conducts only a case study in the area of SBSE for SPLs. Our aim is not to answer the big question completely in this paper, but to make the community pay more attention to the question. In particular, by our empirical study, we demonstrate that whether or not to preserve invalid solutions deserves more attention and more study in SBSE. In some cases, as analyzed for AmazonEC2, we have to preserve valid solutions all along the way during the search of SBSE.

5.3 Threats to Validity

To enhance internal validity, we replicated the state-of-the-art using the exactly same code published by the original authors [32]. To avoid the misleading effects caused by random fluctuation in measurements, each studied algorithm was performed 30 times for each subject. We took the median and mean values for analysis and performed inferential statistical tests for significance and assessment of effect size. Furthermore, we published the source code of our implementation for the study on reproducibility.

To increase external validity, we evaluated seven SPLs. All the subjects have been deployed in real-world scenarios, and they are highly constrained. Five subjects are the five largest SPLs hitherto reported in the literature, while two subjects have realistic attribute values and constraints. We are aware that these subjects might be yet too limited to draw a complete conclusion and our empirical results may not be automatically transferable to all other SPLs.

6 CONCLUSIONS

Metaheuristics used in SBSE, such as MOEAs, usually produce invalid solutions that violate the constraints predefined. Since these invalid solutions are unbuildable and useless in practice, we debate the preservation of invalid solutions during the search of SBSE.

In this paper, we conduct a case study in SPLs, focusing on the MOEAs for the SPL optimization problem. We perform experiments on seven real-world SPLs, including five largest SPLs hitherto reported in the literature and two SPLs with realistic attribute values and constraints. By empirical study, we identify three potential limitations of preserving invalid solutions. Moreover, by using the state-of-the-art as a baseline, we design five algorithm variants that adopt different initial populations and different mutation operators. By performance evaluation, we provide empirical guidance on how to preserve valid solutions during the search of MOEAs.

In short, our empirical study demonstrates that whether or not to preserve invalid solutions deserves more attention in SBSE, and in some cases, we have to preserve valid solutions all along the way during the search of SBSE.

Future work will collect more real-world SPLs and investigate the impact of realistic constraints [17, 50, 73] on the feasibility and scalability of MOEAs and other metaheuristics for the SPL optimization problem. Also, we plan to extend our study on the preservation of invalid solutions to other application fields of SBSE, such as testing [2, 14, 31] and maintenance [44, 46, 61].

ACKNOWLEDGMENTS

The authors would like to thank Dr. Jesús García-Galán for providing the original data of AmazonEC2. Also, the authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. The work is partially supported by National Natural Science Foundation of China (No. 61772200), Shanghai Pujiang Talent Program (No. 17PJ1401900), Shanghai Municipal Natural Science Foundation (No. 17ZR1406900), and Alibaba Group.

REFERENCES

[1] Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziolek, and Indika Meedeniya. 2013. Software architecture optimization methods: A Systematic literature review. *IEEE Trans. Software Eng.* 39, 5 (2013), 658–683.

[2] Aldeida Aleti, Irene Moser, and Lars Grunske. 2017. Analysing the fitness landscape of search-based software testing problems. *Autom. Softw. Eng.* 24, 3 (2017), 603–621.

[3] Shaukat Ali, Lionel C. Briand, Hadi Hemmati, and Rajwinder Kaur Panesar-Walawege. 2010. A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Trans. Software Eng.* 36, 6 (2010), 742–762.

[4] Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-oriented software product lines - Concepts and implementation.* Springer.

[5] Andrea Arcuri and Lionel C. Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of 33rd International Conference on Software Engineering (ICSE)*. 1–10.

[6] Don Batory. 2005. Feature models, grammars, and propositional formulas. In *Proceedings of 9th International Software Product Line Conference (SPLC)*. 7–20.

[7] Gabriele Bavota, Massimiliano Di Penta, and Rocco Oliveto. 2014. Search based software maintenance: Methods and tools. In *Evolving Software Systems*. 103–137.

[8] David Benavides, Sergio Segura, and Antonio Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (2010), 615–636.

[9] Thorsten Berger, Daniela Lettner, Julia Rubin, Paul Grünbacher, Adeline Silva, Martin Becker, Marsha Chechik, and Krzysztof Czarnecki. 2015. What is a feature?: a qualitative study of features in industrial software product lines. In *Proceedings of the 19th International Conference on Software Product Line (SPLC)*. 16–25.

[10] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wasowski, and Krzysztof Czarnecki. 2013. A study of variability models and languages in the systems software domain. *IEEE Trans. Software Eng.* 39, 12 (2013), 1611–1640.

[11] Daniel Le Berre and Anne Parrain. 2010. The Sat4j library, release 2.2. *JSAT* 7, 2-3 (2010), 59–6.

[12] Dimo Brockhoff, Tobias Friedrich, and Frank Neumann. 2008. Analyzing hypervolume indicator based algorithms. In *Proceedings of 10th International Conference on Parallel Problem Solving from Nature (PPSN)*. 651–660.

[13] Paul Clements and Linda Northrop. 2001. *Software product lines: Practices and patterns.* Addison-Wesley.

[14] Myra B. Cohen. 2017. The evolutionary landscape of SBST: A 10 year perspective. In *Proceedings of 10th IEEE/ACM International Workshop on Search-Based Software Testing (SBST)*. 47–48.

[15] Krzysztof Czarnecki and Ulrich Eisenecker. 2000. *Generative programming: Methods, tools, and applications.* Addison-Wesley.

[16] Kalyanmoy Deb, Manikanth Mohan, and Shikhar Mishra. 2003. Towards a quick computation of well-spread Pareto-optimal solutions. In *Proceedings of Second International Conference on Evolutionary Multi-Criterion Optimization (EMO)*. 222–236.

[17] Xavier Devroey, Gilles Perrouin, Mike Papadakis, Axel Legay, Pierre-Yves Schobbens, and Patrick Heymans. 2016. Featured model-based mutation analysis. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. 655–666.

[18] Juan J. Durillo and Antonio J. Nebro. 2011. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software* 42 (2011), 760–771.

[19] Juan J. Durillo, Antonio J. Nebro, and Enrique Alba. 2010. The jMetal framework for multi-objective optimization: Design and architecture. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*. 1–8.

[20] Jesús García-Galán, Pablo Trinidad, Omer F. Rana, and Antonio Ruiz Cortés. 2016. Automated configuration support for infrastructure migration to the cloud. *Future Generation Comp. Syst.* 55 (2016), 200–212.

[21] Brady J. Garvin, Myra B. Cohen, and Matthew B. Dwyer. 2011. Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering* 16, 1 (2011), 61–102.

[22] Jianmei Guo, Eric Blais, Krzysztof Czarnecki, and Peter van Beek. 2017. A worst-case analysis of constraint-based algorithms for exact multi-objective combinatorial optimization. In *Proceedings of the 30th Canadian Conference on Artificial Intelligence (Canadian AI)*. 117–128.

[23] Jianmei Guo, Jia Hui Liang, Kai Shi, Dingyu Yang, Jingsong Zhang, Krzysztof Czarnecki, Vijay Ganesh, and Huiqun Yu. 2017. SMTIBEA: a hybrid multi-objective optimization algorithm for configuring large constrained software product lines. *Software & Systems Modeling* (2017), 1–20. https://doi.org/10.1007/s10270-017-0610-0

[24] Jianmei Guo, Jules White, Guangxin Wang, Jian Li, and Yinglin Wang. 2011. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software* 84, 12 (2011), 2208–2221.

[25] Jianmei Guo, Edward Zulkoski, Rafael Olaechea, Derek Rayside, Krzysztof Czarnecki, Sven Apel, and Joanne M. Atlee. 2014. Scaling exact multi-objective combinatorial optimization by parallelization. In *Proceedings of 29th ACM/IEEE International Conference on Automated Software Engineering (ASE)*. 409–420.

[26] Mark Harman. 2007. The current state and future of search based software engineering. In *Proceedings of International Workshop on the Future of Software Engineering (FOSE)*. 342–357.

[27] Mark Harman, Yue Jia, Jens Krinke, William B. Langdon, Justyna Petke, and Yuanyuan Zhang. 2014. Search based software engineering for software product line engineering: A survey and directions for future work. In *Proceedings of 18th International Software Product Line Conference (SPLC)*. 5–18.

[28] Mark Harman and Bryan F. Jones. 2001. Search-based software engineering. *Information & Software Technology* 43, 14 (2001), 833–839.

[29] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.* 45, 1 (2012), 11:1–11:61.

[30] Hadi Hemmati, Meiyappan Nagappan, and Ahmed E. Hassan. 2015. Investigating the effect of "defect co-fix" on quality assurance resource allocation: A search-based approach. *Journal of Systems and Software* 103 (2015), 412–422.

[31] 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)*. 523–534.

[32] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. 2015. Combining multi-objective search and constraint solving for configuring large software product lines. In *Proceedings of 37th IEEE/ACM International Conference on Software Engineering (ICSE)*. 517–528.

[33] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. 2013. Multi-objective test generation for software product lines. In *Proceedings of 17th International Software Product Line Conference (SPLC)*. 62–71.

[34] Robert M. Hierons, Miqing Li, Xiaohui Liu, Sergio Segura, and Wei Zheng. 2016. SIP: Optimal product selection from feature models using many-objective evolutionary optimization. *ACM Trans. Softw. Eng. Methodol.* 25, 2 (2016).

[35] Hisao Ishibuchi, Hiroyuki Masuda, Yuki Tanigaki, and Yusuke Nojima. 2015. Modified distance calculation in generational distance and inverted generational distance. In *Proceedings of 8th International Conference on Evolutionary Multi-Criterion Optimization (EMO)*. 110–125.

[36] Yue Jia, Myra B. Cohen, Mark Harman, and Justyna Petke. 2015. Learning combinatorial interaction test generation strategies using hyperheuristic search. In *Proceedings of 37th IEEE/ACM International Conference on Software Engineering (ICSE)*. 540–550.

[37] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study.* Technical Report. CMU SEI, SEI-90-TR-21.

[38] Muhammad Rezaul Karim, Günther Ruhe, Md. Mainur Rahman, Vahid Garousi, and Thomas Zimmermann. 2016. An empirical investigation of single-objective and multiobjective evolutionary algorithms for developer's assignment to bugs. *Journal of Software: Evolution and Process* 28, 12 (2016), 1025–1060.

[39] Marouane Kessentini, Usman Mansoor, Manuel Wimmer, Ali Ouni, and Kalyanmoy Deb. 2017. Search-based detection of model level changes. *Empirical Software Engineering* 22, 2 (2017), 670–715.

[40] Fitsum Meshesha Kifetew, Angelo Susi, Denisse Muñante, Anna Perini, Alberto Siena, and Paolo Busetta. 2017. Towards multi-decision-maker requirements prioritisation via multi-objective optimisation. In *Proceedings of the Forum and Doctoral Consortium Papers Presented at the 29th International Conference on Advanced Information Systems Engineering (CAiSE)*. 137–144.

[41] Joshua Knowles and David Corne. 2002. On metrics for comparing nondominated sets. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*. 711–716.

[42] Alexander Knüppel, Thomas Thüm, Stephan Mennicke, Jens Meinicke, and Ina Schaefer. 2017. Is there a mismatch between real-world feature models and product-line research?. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 291–302.

[43] Lingbo Li, Mark Harman, Fan Wu, and Yuanyuan Zhang. 2017. The value of exact analysis in requirements selection. *IEEE Trans. Software Eng.* 43, 6 (2017), 580–596.

[44] Han Liu, Chengnian Sun, Zhendong Su, Yu Jiang, Ming Gu, and Jiaguang Sun. 2017. Stochastic optimization of program obfuscation. In *Proceedings of the 39th International Conference on Software Engineering (ICSE)*. 221–231.

[45] Hong Lu, Tao Yue, Shaukat Ali, and Li Zhang. 2016. Nonconformity resolving recommendations for product line configuration. In *Proceedings of Ninth International Conference on Software Testing, Verification and Validation (ICST)*. 57–68.

[46] Ruchika Malhotra and Megha Khanna. 2017. An exploratory study for software change prediction in object-oriented systems using hybridized techniques. *Autom. Softw. Eng.* 24, 3 (2017), 673–717.

[47] Bogdan Marculescu, Robert Feldt, and Richard Torkar. 2016. Using exploration focused techniques to augment search-based software testing: An experimental evaluation. In *Proceedings of 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 69–79.

[48] Phil McMinn, Mark Harman, Gordon Fraser, and Gregory M. Kapfhammer. 2016. Automated search for good coverage criteria: moving from code coverage to fault coverage through search-based software engineering. In *Proceedings of the 9th*

International Workshop on Search-Based Software Testing (SBST). 43–44.
[49] Jens Meinicke, Chu-Pan Wong, Christian Kästner, Thomas Thüm, and Gunter Saake. 2016. On essential configuration complexity: measuring interactions in highly-configurable systems. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 483–494.
[50] Sarah Nadi, Thorsten Berger, Christian Kästner, and Krzysztof Czarnecki. 2015. Where do configuration constraints stem from? An extraction approach and an empirical study. *IEEE Trans. Software Eng.* 41, 8 (2015), 820–841.
[51] Rafael Olaechea, Derek Rayside, Jianmei Guo, and Krzysztof Czarnecki. 2014. Comparison of exact and approximate multi-objective optimization for software product lines. In *Proceedings of 18th International Software Product Line Conference (SPLC)*. 92–101.
[52] Vinicius Paulo L. Oliveira, Eduardo F. D. Souza, Claire Le Goues, and Celso G. Camilo-Junior. 2016. Improved crossover operators for genetic programming for program repair. In *Proceedings of 8th International Symposium on Search Based Software Engineering (SSBSE)*. 112–127.
[53] Ali Ouni, Marouane Kessentini, Mel Ó Cinnéide, Houari A. Sahraoui, Kalyanmoy Deb, and Katsuro Inoue. 2017. MORE: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells. *Journal of Software: Evolution and Process* 29, 5 (2017).
[54] Leonardo Teixeira Passos, Jianmei Guo, Leopoldo Teixeira, Krzysztof Czarnecki, Andrzej Wasowski, and Paulo Borba. 2013. Coevolution of variability models and related artifacts: A case study from the Linux kernel. In *Proceedings of 17th International Software Product Line Conference (SPLC)*. 91–100.
[55] Leonardo Teixeira Passos, Leopoldo Teixeira, Nicolas Dintzner, Sven Apel, Andrzej Wasowski, Krzysztof Czarnecki, Paulo Borba, and Jianmei Guo. 2016. Coevolution of variability models and related software artifacts - A fresh look at evolution patterns in the Linux kernel. *Empirical Software Engineering* 21, 4 (2016), 1744–1793.
[56] Justyna Petke, Myra B. Cohen, Mark Harman, and Shin Yoo. 2015. Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection. *IEEE Trans. Software Eng.* 41, 9 (2015), 901–924.
[57] Antônio Mauricio Pitangueira, Paolo Tonella, Angelo Susi, Rita Suzana Pitangueira Maciel, and Márcio de Oliveira Barros. 2017. Minimizing the stakeholder dissatisfaction risk in requirement selection for next release planning. *Information & Software Technology* 87 (2017), 104–118.
[58] Klaus Pohl, Günter Bockle, and Frank van der Linden. 2005. *Software product line engineering: Foundations, principles, and techniques.* Springer.
[59] Outi Räihä. 2010. A survey on search-based software design. *Computer Science Review* 4, 4 (2010), 203–249.
[60] Ana B. Sánchez, Sergio Segura, José Antonio Parejo, and Antonio Ruiz Cortés. 2017. Variability testing in the wild: the Drupal case study. *Software & Systems Modeling* 16, 1 (2017), 173–194.
[61] Federica Sarro, Alessio Petrozziello, and Mark Harman. 2016. Multi-objective software effort estimation. In *Proceedings of 38th International Conference on Software Engineering (ICSE)*. 619–630.
[62] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. 2013. Optimum feature selection in software product lines: Let your model and values guide your search. In *Proceedings of 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*. 22–27.
[63] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. 2013. Scalable product line configuration: A straw to break the camel's back. In *Proceedings of 28th International Conference on Automated Software Engineering (ASE)*. 465–474.
[64] 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 35th International Conference on Software Engineering (ICSE)*. IEEE, 492–501.
[65] Sergio Segura, Amador Durán, Ana B. Sánchez, Daniel Le Berre, Emmanuel Lonca, and Antonio Ruiz Cortés. 2015. Automated metamorphic testing of variability analysis tools. *Softw. Test., Verif. Reliab.* 25, 2 (2015), 138–163.
[66] Mozhan Soltani, Annibale Panichella, and Arie van Deursen. 2017. A guided genetic algorithm for automated crash reproduction. In *Proceedings of the 39th International Conference on Software Engineering (ICSE)*. 209–220.
[67] Tian Huat Tan, Yinxing Xue, Manman Chen, Jun Sun, Yang Liu, and Jin Song Dong. 2015. Optimizing selection of competing features via feedback-directed evolutionary algorithms. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA)*. 246–256.
[68] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. 2014. A classification and survey of analysis strategies for software product lines. *ACM Comput. Surv.* 47, 1 (2014), 6:1–6:45.
[69] Tassio Vale, Ivica Crnkovic, Eduardo Santana de Almeida, Paulo Anselmo da Mota Silveira Neto, Yguaratã Cerqueira Cavalcanti, and Silvio Romero de Lemos Meira. 2016. Twenty-eight years of component-based software engineering. *Journal of Systems and Software* 111 (2016), 128–148.
[70] A. Vargha and H. D. Delaney. 2000. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal on Educational and Behavioral Statistics* 25, 2 (2000), 101–132.
[71] Jules White, Brian Doughtery, and Douglas C. Schmidt. 2008. Filtered cartesian flattening: An approximation technique for optimally selecting features while adhering to resource constraints. In *Proceedings of First International Workshop on Analyses of Software Product Lines (ASPL)*. 209–216.
[72] Zhiqiao Wu, Jiafu Tang, C. K. Kwong, and Ching-Yuen Chan. 2011. An optimization model for reuse scenario selection considering reliability and cost in software product line development. *International Journal of Information Technology and Decision Making* 10, 5 (2011), 811–841.
[73] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. 2015. Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 307–319.
[74] Yinxing Xue, Jinghui Zhong, Tian Huat Tan, Yang Liu, Wentong Cai, Manman Chen, and Jun Sun. 2016. IBED: Combining IBEA and DE for optimal feature selection in software product line engineering. *Appl. Soft Comput.* 49 (2016), 1215–1231.
[75] Shin Yoo and Mark Harman. 2007. Pareto efficient multi-objective test case selection. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. 140–150.
[76] Yuanyuan Zhang, Anthony Finkelstein, and Mark Harman. 2008. Search based requirements optimisation: Existing work and challenges. In *Proceedings of 14th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*. 88–94.
[77] Eckart Zitzler and Simon Künzli. 2004. Indicator-based selection in multiobjective search. In *Proceedings of 8th International Conference on Parallel Problem Solving from Nature (PPSN)*. 832–842.
[78] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. 2003. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Trans. Evolutionary Computation* 7, 2 (2003), 117–132.