CrossMark

REGULAR PAPER

SMTIBEA: a hybrid multi-objective optimization algorithm for configuring large constrained software product lines

Jianmei Guo[1] · **Jia Hui Liang[2]** · **Kai Shi[1]** · **Dingyu Yang[3]** · **Jingsong Zhang[4]** · **Krzysztof Czarnecki[2]** · **Vijay Ganesh[2]** · **Huiqun Yu[1]**

Abstract A key challenge to software product line engineering is to explore a huge space of various products and to find optimal or near-optimal solutions that satisfy all predefined constraints and balance multiple often competing objectives. To address this challenge, we propose a hybrid multi-objective optimization algorithm called SMTIBEA that combines the indicator-based evolutionary algorithm (IBEA) with the satisfiability modulo theories (SMT) solving. We evaluated the proposed algorithm on five large, constrained, real-world SPLs. Compared to the state-of-the-art, our approach significantly extends the expressiveness of constraints and simultaneously achieves a comparable performance. Furthermore, we investigate the performance influence of the SMT solving on two evolutionary operators of the IBEA.

Keywords Software product lines · Search-based software engineering · Multi-objective evolutionary algorithms · Constraint solving · Feature models

✉ Jianmei Guo
gjm@ecust.edu.cn

✉ Jingsong Zhang
jasun@dmbio.info

[1] School of Information Science and Engineering, East China University of Science and Technology, Shanghai, China

[2] Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada

[3] School of Electronic Information, Shanghai Dianji University, Shanghai, China

[4] Institute of Biochemistry and Cell Biology, Chinese Academy of Sciences, Shanghai, China

1 Introduction

Software product line (SPL) engineering is gaining momentum in academia and industry to reduce development cost, to enhance software quality, and to shorten time to market [1,43]. An SPL is a collection of software products that share a managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [7]. A *feature* is an essential abstraction of a functionality or product characteristic relevant to stakeholders [8]. All features of an SPL and the constraints defined among features are captured in a *feature model* that describes and manages the commonalities and variabilities of an SPL. Every product of an SPL is specified as a unique selection of features, called a *configuration*.

A central task in SPL engineering is to derive an optimal product that meets specific requirements of stakeholders, which we call the *configuration optimization problem*. It is non-trivial to address the SPL configuration optimization problem. Firstly, the high variability of different products expressed by the feature combinatorics in the feature model, gives rise to a huge configuration space, which often makes the configuration space exploration beyond human capabilities. Secondly, once a configuration is made, it must be validated to satisfy the constraints defined in the feature model, which usually must rely on an automated analysis technique. Thirdly, the configuration optimization problem often involves multiple competing objectives, such as low cost and high performance, so there is no single optimal solution that excels in all objectives and stakeholders usually would like to know a set of optimal or near-optimal solutions that trade-off the multiple objectives [22,49].

🙎 Springer

The SPL configuration optimization problem was first studied in 2008 by White et al. [54] who reduced the problem to a multi-dimensional multi-choice knapsack problem and proposed a method called Filtered Cartesian Flattening to solve it. To tackle the same problem, Guo et al. [19] proposed the first genetic algorithm in 2011 that incorporates a repair operator that fixes configurations conforming to the predefined constraints after each round of crossover and mutation. These approaches were originally designed for single-objective scenarios. Moreover, they were evaluated on randomly generated feature models that follow a set of empirically approximated assumptions, which may diverge from the characteristics of real-world SPLs [5].

In 2013, Sayyad et al. [49] first proposed to adapt multi-objective evolutionary algorithms (MOEAs) for the solving of the SPL configuration optimization problem. They investigated seven widely used MOEAs and identified the indicator-based evolutionary algorithm (IBEA) [58] to be superior to the other MOEAs for the problem. Later, they conducted the first study on large real-world feature models and improved the scalability of IBEA to a feature model as large as 6888 features [48]. Sayyad et al.'s previous works define the number of constraints that are violated in the generated configurations as one optimization objective to minimize, and they produce valid configurations without guarantee.

To deal with the constraints explicitly for the SPL configuration optimization problem, in 2015 Henard et al. [26] proposed the first hybrid method, called SATIBEA, that augments IBEA with SAT (Boolean satisfiability problem) solving. The SAT solving part of their approach encodes the feature model in conjunctive normal form (CNF) formulas and asks an off-the-shelf SAT solver to return a valid configuration. They incorporated the SAT solving into the mutation operator of IBEA to replace randomly mutated configurations with valid ones. Their empirical results on large real-world SPLs showed that their SATIBEA approach significantly outperforms Sayyad et al.'s IBEA approach, which established the current *state-of-the-art*.

The expressiveness of CNF formulas used in the state-of-the-art is limited to Boolean constraints. However, real-world SPLs embodies not only Boolean constraints but also non-Boolean constraints, such as those over infinite domain variables (integers, real numbers, and character strings) together with arithmetic, relational, and string operators [5,37,40–42]. SAT solving of such constraints is undecidable in general. To this end, we aim at an approach that supports richer constraints, including Boolean and non-Boolean constraints, for real-world SPLs. Moreover, we aim at a deep understanding of the influence of the constraint solving on the evolutionary operators of an MOEA. In this paper, we answer the following three research questions:

RQ1: Is it possible to design a hybrid approach that augments IBEA with support for expressing and solving richer constraints?

RQ2: How does the constraint solving influence the evolutionary operators of the IBEA?

RQ3: What is the performance of the proposed approach compared to the state-of-the-art?

Our contributions of this paper are summarized as follows:

– We propose SMTIBEA, the first hybrid method that augments IBEA with the satisfiability modulo theories (SMT) solving for the SPL configuration optimization problem. SMT combines standard SAT with richer theories, such as equality reasoning, linear arithmetic, bit vectors, and arrays [9].

– We demonstrate the effectiveness and scalability of our approach by experiments on five large real-world SPLs, ranging from 1244 to 6888 features and from 2468 to 343,944 constraints. We evaluate the performance of algorithms in terms of five metrics, considering quality, convergence, and diversity.

– Our evaluation is based on experimental results from 30 independent executions of algorithms. We conducted inferential statistical testing for significance and assessment of effect size.

– We design three SMTIBEA variants to investigate the performance influence of SMT solving on the mutation operator and the initial population generation of IBEA. Our empirical results reinforce the importance of augmenting the mutation operator with constraint solving. Moreover, augmenting the initial population generation with constraint solving is preferable as well.

– Our empirical results show that our approach obtains performance comparable to the state-of-the-art, given that the expressiveness of constraints has been significantly improved. In particular, in terms of a widely used quality metric *Hypervolume*, our approach outperforms the state-of-the-art for 4 out of 5 SPLs when all generated configurations are used for evaluation. When filtering out all invalid configurations from the results, our approach still works better for 2 SPLs.

– We implemented the SMTIBEA approach and made the source code of our implementation publicly available at http://github.com/jmguo/SMTIBEA.

The rest of this paper is organized as follows. Section 2 introduces background concepts on SPL engineering. Section 3 defines the SPL configuration optimization problem. Related methods of solving the problem are discussed in Sect. 4. Section 5 proposes our approach SMTIBEA. We describe our experimental setup in Sect. 6 and present the results in Sect. 7. Section 8 discusses our experimental

results, explains threats to validity, and describes our perspectives and the future work. Section 9 concludes the paper.

2 Background

Originated from feature-oriented domain analysis (FODA) proposed in 1990 [31], *SPL engineering* has emerged as one of the most promising software development paradigms for reducing development costs, enhancing software quality, and shortening time to market [1,7]. SPL engineering achieves software mass customization by exploiting the commonalities of a set of software systems that belong to an SPL and by managing the differences among those systems. Typically, SPL engineering is composed of two development processes: *domain engineering* and *application engineering* [43].

Domain engineering defines the commonalities and variabilities of an SPL in terms of features and organizes all features and their constraints in a domain feature model. A *feature* is an essential abstraction of a functionality or product characteristic relevant to stakeholders [8]. A *feature model* is a tree-like structure that defines the relationships among features in a hierarchical manner. For example, Fig. 1 shows a feature model of a sample mobile-phone SPL, adapted from Benavides et al. [4]. Each feature except the root feature MobilePhone has a parent. Relations between a feature and its group of child features are classified as And (no arc)-, Or (filled arc)-, and Alternative-groups (arc). The members of an And-group can be either mandatory (filled circle) or optional (empty circle). Cross-tree constraints comprise *requires* and *excludes* relations between features (e.g., Video requires Camera). According to Batory et al. [3], a feature model can be translated into a Boolean formula in CNF. The transla-
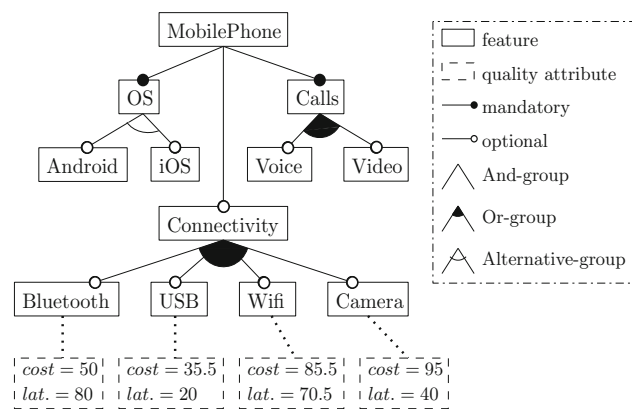
tion rules of feature-selection constraints are summarized in Table 1.

Application engineering configures the feature model established in the domain engineering and derives a particular software product to meet user- or market-specific requirements. Each product is specified as a unique selection of features, called a *configuration*. A configuration of features is *valid* when all constraints defined in the feature model are satisfied. For example, in Fig. 1, configuration (OS, Android, Calls, Video, Connectivity, Camera) is valid, but configuration (OS, Android, Calls, Video, Connectivity, Wifi) is not, because the latter violates the constraint: Video requires Camera.

Among a large number of valid configurations derived from the feature model, users desire the optimal one that meets certain quality requirements, such as low cost and low latency. To describe the product quality, in the SPL community, a feature model is often augmented with the quality attributes of each feature [19,48,49,54]. A quality attribute of a feature indicates the feature's influence on the quality attribute of a final product that can be implemented and measured. As shown in Fig. 1, selecting feature Bluetooth increases the cost by 50 and the latency by 80 in the final product. The quality attributes of a derived product can be calculated by aggregating the quality attributes of all selected features and potential feature interactions involved. Following the same settings as the state-of-the-art [26], we assume that we already have a feature model along with the quality attributes. For research on measuring and inferring the quality attributes, we refer the interested readers elsewhere [18,46,51,57].

3 Problem definition

Given a feature model with n features, the configuration optimization problem of an SPL 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

Fig. 1 A feature model defining a configuration optimization problem with two objectives for a sample mobile phone SPL. Each feature is augmented with two quality attributes

Cross-tree constraints : *Video requires Camera*

Objectives : *minimizing cost, minimizing latency*

Table 1 Feature-selection constraints in Boolean formulas (*P* represents a parent feature and $C_1, \ldots, C_i, \ldots, C_n$ are its child features; $M \subseteq \{1, \ldots, n\}$ denotes the mandatory features by their indices in an And-group; F_i and F_j denote arbitrary features)

Type	Boolean formulas
Mandatory	$C_i \leftrightarrow P$
Optional	$C_i \rightarrow P$
And-group	$(P \rightarrow \bigwedge_{i \in M} C_i) \wedge (\bigvee_{1 \leq i \leq n} C_i \rightarrow P)$
Or-group	$P \leftrightarrow \bigvee_{1 \leq i \leq n} C_i$
Alternative-group	$(P \leftrightarrow \bigvee_{1 \leq i \leq n} C_i) \wedge \bigwedge_{i < j}(\neg C_i \vee \neg C_j)$
Requires	$F_i \rightarrow F_j$
Excludes	$\neg(F_i \wedge F_j)$

domains of variables in X, C is a set of constraints on variables in X, and $F = \{f_1, \ldots, f_m\}$ is composed of m objective functions to minimize simultaneously. Without loss of generality, we consider only minimization problems here.

A *solution s* to the configuration 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 $s \in \mathcal{S}$. Note that multiple solutions may have the same values regarding all objectives.

For multi-objective problems considered in this paper, the optimization objectives are often *competing*, such as achieving low cost and high performance. Therefore, there is usually no single optimal solution that excels in all objectives. Decision makers would like to know diverse, ideally all, if their number is tractable, Pareto-optimal solutions that trade-off the multiple objectives, such that they can choose a posteriori the solution that best meets their needs.

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

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

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 exact Pareto front $\tilde{\mathcal{S}}$.

4 Related work

In this section, we discuss the related methods of solving the SPL configuration optimization problem. We categorize the related methods into two classes: *approximate* and *exact* methods. Also, we identify the state-of-the-art method for each class.

4.1 Approximate methods

The SPL configuration optimization problem has a natural connection to the community of *search-based software engineering (SBSE)*, a term coined in 2001 [23], referring to the use of computational search as a means of optimizing different software engineering problems [21,24,36,47]. A recent survey [22] reported the upsurge in interest and activity in the area of *SBSE for SPL*.

For the SPL configuration optimization problem, many previous approaches were proposed for single-objective optimization, such as [19,54]. Wu et al. [55] introduced a bi-objective optimization method that balances reliability and cost for a single case study of mail server system development. Lu et al. [33] 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. Also, they evaluated their approach on a single case study. Hierons et al. [28] proposed a method called SIP that optimizes first on the number of constraints and then on the other objectives. The case studies they used are either relatively small (at most 290 features) or randomly generated (e.g., the largest one with 10,000 features).

Sayyad et al. conducted the first and detailed study on MOEAs for the SPL configuration optimization problem in a series of papers [48–50]. Moreover, the feature models they used for evaluation come from large, constrained, real-world SPLs. To deal with the constraints defined in the feature model and produce valid configurations, they defined the number of constraints that were violated in the generated solution as an optimization objective to minimize. Other optimization objectives are relevant to user preferences, such as minimizing cost and minimizing the number of defects. They conducted experiments and compared the performance of seven widely used MOEAs, such as IBEA [58], Non-dominated Sorting Genetic Algorithm version 2 (NSGA-II) [11], and Strength Pareto Evolutionary Algorithm version 2 (SPEA2) [59]. Their experimental results demonstrated the superiority of IBEA to other algorithms for the SPL configuration optimization problem.

Henard et al. [26] proposed the first hybrid algorithm, called SATIBEA, by combining IBEA with SAT solving. Their SAT solving part encodes the feature model in a CNF formula and asks a SAT solver to return a valid solution [27]. They incorporated SAT solving into the mutation operator of IBEA. Concretely, SATIBEA has a probability to replace a randomly mutated configuration with a new valid one returned by the SAT solver, such that the predefined constraints are resolved explicitly and some configurations in the population are guaranteed to be valid. Moreover, to increase the diversity of solutions found, SATIBEA randomly permutes the parameters of the underlying solver at each execution of SAT solving. Empirical results on the same case studies as used by Sayyad et al.'s IBEA approach showed that SATIBEA significantly outperforms the IBEA approach on large real-world SPLs.

In this paper, we refer to SATIBEA as the state-of-the-art approximate optimization method for the SPL configuration optimization problem and use SATIBEA as a baseline algorithm for performance comparison.

4.2 Exact methods

Olaechea et al. [38,39] first adapted the Guided Improved Algorithm (GIA) [44] for the exact solving of the multi-objective configuration optimization problem of an SPL. The key idea of GIA relies on an *improved search* process, which uses a constraint solver to return a valid solution and then augments the current constraints with the superior constraint to search for solutions that dominate ones found already. Given a currently found solution s, the *superior constraint* augmented for the next search is defined as follows:

$$\sup(s) = \{s' \in \mathcal{S} : s' \prec s\}. \tag{3}$$

Adding the superior constraint will force the solver to return a solution dominating s; if there is no solution returned, then solution s is Pareto-optimal. For example, as shown in Fig. 2, the superior constraint of solution s_4 defines the subspace that contains all solutions that dominate s_4.

The superior constraint defined above specifies a group of linear inequalities, which go beyond the general expressiveness of SAT solving. Olaechea et al. [38] expressed the superior constraint and implemented the improved search using an off-the-shelf SMT solver. SMT significantly extends standard SAT by adding richer theories, such as equality reasoning, linear arithmetic, bit vectors, and arrays [9]. Also, a recent study [45] shows that SMT might be the most efficient reasoning formalism in checking model properties, compared to CSP (Constraint Satisfaction Problem) [52], Alloy [30], and ASP (Answer Set Programming) [35]. Empirical results showed that GIA is able to compute all exact Pareto-optimal solutions in reasonable time for SPL case studies with less than 45 features, but it is not scalable to a case study ESHOP with 290 features [38]. Later, Guo et al. proposed several parallel algorithms to improve the scalability of GIA, but their algorithms are not yet able to complete the exact solving of the ESHOP case in 6 days [20]. Note that the ESHOP case
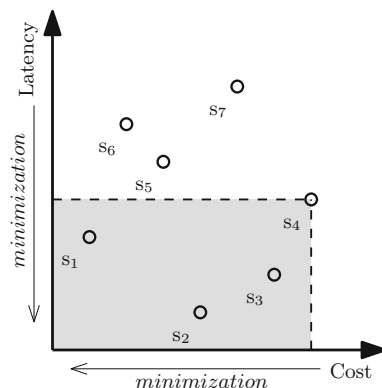

Fig. 2 Superior constraint of solution s_4 defines the gray area that contains all solutions that dominate s_4

with 290 features has approximately 5.02e+49 valid configurations [20], so probably no algorithm is able to produce the full, exact Pareto front in reasonable time. This demonstrates the need for an approximate approach for large SPLs.

To the best of our knowledge, GIA is the state-of-the-art exact optimization method for multi-objective SPL configuration optimization problem. GIA empirically works well for the problems with up to 44 features [38]. However, many real-world SPLs involve thousands of features, which poses a great challenge for the SPL configuration optimization. In this paper, we target large, constrained SPLs and keep the same case studies as [26,48]. We will not carry out performance comparison to GIA because of its insufficient scalability on large SPLs. But we will investigate the performance influence of the improved search and the general SMT solving when incorporating either of them into the mutation operator of the IBEA.

5 Our approach

In this section, we present our approach SMTIBEA that combines IBEA with SMT solving.

5.1 Framework and operators

The basic algorithmic framework of SMTIBEA follows the IBEA template of JMETAL, an open-source Java framework for multi-objective optimization [13,14]. The details of the classic IBEA can be found elsewhere [58].

Our approach adopts the algorithmic template supporting binary solution type, which is also used by the state-of-the-art (SATIBEA) [26]. That is, each feature is binary, and each configuration of a feature model 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 used in SMTIBEA include: single-point crossover, bit-flip mutation, and binary tournament selection. The standard initial population generation follows a *random* strategy; that is, the initial population is composed of a set of randomly generated binary strings, each of which represents an arbitrary configuration that may not be valid. SMTIBEA stops its execution when the predefined timeout is reached.

5.2 Specifications and SMT solving

SMTIBEA translates a feature model together with the quality attributes into a set of quantifier-free formulas in first-order logic [9]. By inputting the formulas to an off-the-shelf SMT solver, we are able to resolve the predefined constraints automatically and acquire a valid configuration conveniently. Given the case studies we evaluated, three types of variables

are supported in our specifications: Boolean, integer, and real. Each feature is encoded as a Boolean variable. Each quality attribute is represented as either an integer or a real variable, for example, the cost of each feature is real, and the number of defects of each feature is integer. Moreover, two standard theories, quantifier-free linear integer arithmetic and quantifier-free linear real arithmetic, are used for reasoning.

A quality attribute of a configuration is represented by an additional variable, and its value is the sum of the quality attributes of the features selected in the configuration. For example, the total cost of a configuration c is the sum of the cost of each feature f selected in c, i.e, $total\_cost(c) = \sum_{f \in c} cost(f)$.

As described in Sect. 2, the constraints defined between features are usually encoded into a Boolean formula and can be resolved automatically by SAT solving. However, in practice, there may exist non-Boolean constraints defined on quality attributes. Compared to traditional SAT solving, a distinguished advantage of SMT solving is able to conveniently represent and automatically resolve these non-Boolean constraints. For example, one can represent a constraint defined between the costs of two features (e.g., $cost(f_1) = 2 * cost(f_2) + 100$) or a constraint defining the threshold of a configuration's total cost (e.g., $total\_cost(c) <= 1000$).

By the additional variable for the total quality of a configuration, we can further specify the superior constraint defined in equation (3). For example, the superior constraint regarding the cost is defined as follows: $total\_cost(c') < total\_cost(c)$, where c' is a subsequent configuration returned by the underlying solver. Note that, for multi-objective problems, the superior constraint is specified as a combination of multiple linear inequalities. Take the case shown in Figure 1 as example, the superior constraint defined for the two objectives is $(total\_cost(c') < total\_cost(c)) \wedge (total\_latency(c') < total\_latency(c))$. The solver is repeatedly called, adding the superior constraint after each call to improve upon the total cost of previously found solutions, until a Pareto-optimal solution is found. This way, we achieve the *improved search* process introduced in Sect. 4.2.

We implement SMTIBEA with the efficient SMT solver Z3 [10]. Following the same technique of the state-of-the-art to increase the diversity of solutions returned by the underlying solver, we randomly permute a parameter of the Z3 solver: *random_seed*. The parameter controls the order of variables considered at each call to the solver. That is, SMTIBEA randomly permutes the variable order at each execution of SMT solving in order to increase the diversity of solutions generated.

Table 2 Characteristics of three SMTIBEA variants compared to the state-of-the-art (SATIBEA)

Algorithm	Mutation	Initial population generation
SATIBEA	SAT solving	Random generation
SMTIBEAv1	SMT solving	Random generation
SMTIBEAv2	SMT solving	SMT solving
SMTIBEAv3	Improved search	Random generation

5.3 SMTIBEA variants

To answer RQ2, we design three SMTIBEA variants by incorporating the SMT solving into the mutation operator and the initial population generation of IBEA. Table 2 summarizes the key characteristics of the SMTIBEA variants, compared to the state-of-the-art.

As a baseline algorithm, SATIBEA incorporates the SAT solving into the mutation operator of IBEA. In addition, SATIBEA adopts the standard, random way of initial population generation. Note that such an initial population may contain invalid configurations.

The first variant SMTIBEAv1 replaces the SAT solving in SATIBEA with the SMT solving. Once a mutation operator is enabled, SMTIBEAv1 has a probability of randomly picking a configuration in the current population and replacing it with a new valid one returned by the underlying SMT solver. Also, SMTIBEAv1 keeps using the standard way of initial population generation.

The second variant SMTIBEAv2 replaces the standard way of initial population generation in SMTIBEAv1 with a new way: The underlying SMT solver is repeatedly called to create valid configurations for the initial population until the population reaches a designated size. That is, all configurations in the initial population are guaranteed to be valid. Furthermore, as mentioned in previous section, SMTIBEAv2 randomly permutes the variable order during the SMT solving to increase the diversity of the configurations generated. Note that SMTIBEAv2 invokes the SMT solving twice in the mutation and in the initial population generation, respectively.

The third variant SMTIBEAv3 augments SMTIBEAv1 with the improved search using the superior constraint defined in Eq. (3). To develop the capability of constraint solving in the mutation operator, SMTIBEAv3 has a probability to invoke the improved search that forces the underlying solver to return a valid and better solution than the currently

found ones. Note that the SMT solving naturally supports the superior constraint that usually goes beyond the general expressiveness of SAT solving.

6 Experimental setup

In this section, we present the settings of our conducted experiments. Concretely, we introduce the subjects, the optimization objectives, and the performance metrics. Moreover, we describe the settings of our implementation, the measurement, and the statistical tests.

6.1 Subjects

Scalability is a key concern for addressing the SPL configuration optimization problem in practice. Since small problems can even be solved exactly, as described in Sect. 4.2, we aim at large, constrained, real-world SPL case studies in this paper. In particular, we target the real feature model that has more than 1000 features and is rich in constraints.

We performed our case studies on the same publicly available dataset deployed by the state-of-the-art. The dataset covers five large, constrained, real-world feature models taken from the Linux Variability Analysis Tools (LVAT) repository [34]. We reported the software version, the number of features, and the number of constraints for each subject in Table 3.

In the dataset we used, the original authors augmented each feature of each feature model with three quality attributes: `cost`, `defects`, and `used_before` [26]. They set the values for these attributes arbitrarily with a uniform distribution: `cost` takes real values between 5.0 and 15.0, `used_before` takes Boolean values, and `defects` takes integer values between 0 and 10. Also, they defined a dependency between two attributes: if (not `used_before`) then `defects` = 0.

6.2 Optimization objectives

In our experiments, we consider the following five objectives:

1. *Correctness*. We seek to minimize the number of constraints of the feature model that are violated by a configuration.

Table 3 Overview of subject SPLs

System	Version	#Features	#Constraints
eCos	3.0	1244	3146
FreeBSD	8.0.0	1396	62,183
Fiasco	2011081207	1638	5228
uClinux	20100825	1850	2468
Linux	2.6.28.6	6888	343,944

2. *Richness of features*. We seek to minimize the number of deselected features in a configuration.
3. *Features that were not used before*. We seek to minimize the number of features that were not used before in a configuration.
4. *Known defects*. We seek to minimize the number of known defects in a configuration.
5. *Cost*. We seek to minimize the cost of a configuration.

Note that the first two objectives are related to the structural information of a configuration, and the later three objectives come from the quality attributes that augment each feature. In practice, other objectives can be also considered. We selected the five objectives to ensure identical settings as those reported by the state-of-the-art.

6.3 Performance metrics

We evaluated 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. As suggested by Zitzler et al. [60], we measured the quality of an approximate (Pareto) front using the following three metrics:

– *Hypervolume (HV)* [6]. 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) [60]. 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)* [29]. 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)* [32]. 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 metric:

– *Generalized Spread (GS)* [12]. 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 optimization objectives. A higher GS indicates a better Pareto front.

Note that the quality aspect has a higher priority for performance evaluation, since users usually desire a high-quality population of solutions in the first place. Among a population of solutions produced, users desire more Pareto-optimal solutions, so the convergence matters next to the quality. In addition, the diversity ensures that users have a variety of solutions to choose when the quality and convergence are satisfying. Therefore, the comparisons of the convergence and the diversity are only important when there is quality in the solutions found [26].

The computation of all the above 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, which has been evidenced in [20,38] for the SPL configuration optimization problem. Therefore, we acquired the reference front *R* by calculating the best solution found so far by all the studied algorithms. Concretely, we made a union of all the approximate Pareto fronts found by each algorithm, and then we discarded all the dominated solutions in the union and used all the non-dominated solutions composing the reference front.

To minimize the effects of objective value scaling, we reported the *normalized* metrics (except for ER) for analysis. We first normalized the objective values of all solutions of an approximate front *A* and of the reference front *R* with respect to the maximum and minimum values for each objective. For example, suppose the `defects` values range from 0 to 50, and the `cost` values range from 100 to 100000. Since the magnitudes of the `cost` values are significantly larger, which is actually the case in our evaluated dataset, they may dictate each metric because the `defects` values are negligible. To address this issue, a `defects` value (resp. `cost` value) x is normalized to $\frac{x-0}{50-0}$ (resp. $\frac{x-100}{100000-100}$), which is a real value between 0.0 and 1.0. Afterward, each normalized metric is calculated by using the normalized objective values of solutions. Note that the ER metric is not affected by the issue of objective value scaling, so we reported it directly.

6.4 SMTIBEA settings

Table 4 lists the values of key parameters of SMTIBEA. To enable a fair comparison to the state-of-the-art, we used the same settings as the ones reported by Henard et al. [26]. Note that, once a mutation operator is enabled at a rate of 0.001, each of the three SMTIBEA variants has a probability of 0.98 to perform the standard bit-flip mutation. With a probability

Table 4 Overview of the parameter settings of SMTIBEA

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 SMT solving for mutation	0.02
Probability of using improved search for mutation	0.02
Internal timeout of a call to the SMT Solver	6 s

of 0.02, SMTIBEAv1 or SMTIBEAv2 returns a valid solution by using the SMT solving, while SMTIBEAv3 returns a valid and better solution than the currently found one by using the improved search.

SMTIBEA employs the Z3 solver for SMT solving. As reported by Henard et al. [26], each call to the SAT solver during the execution of SATIBEA takes less than 6 s. Therefore, to make a fair comparison, we set 6 s as the internal timeout of a call to the Z3 solver. That is, each call to the underlying solver during the execution of SMTIBEA has a timeout of 6 s to return a valid configuration; if the timeout is reached, a random solution that may not be valid will be returned. Furthermore, we set the *random_seed* parameter of Z3 solver to a random unsigned integer ranging from 0 to 5e+8, by which we randomly permute the variable order at each execution of SMT solving.

6.5 Measurement settings

All measurements were performed on the same Windows 7 Machine with Intel Core i5 CPU 3.5 GHz and 16 GB RAM. The algorithms are compute-bound and not memory intensive. Since Henard et at. reported that SATIBEA stabilizes on its ultimate solutions in 15 min [26], we performed each studied algorithm 15 min 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.

6.6 Statistical tests

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 nonparametric 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.

As suggested by Arcuri and Briand [2], we reported the nonparametric effect size measure $\hat{A}_{12}$, introduced by Vargha and Delaney [53]. It indicates the performance superiority, and it measures the extent to which an algorithm outperforms the other. Moreover, the superiority between algorithms is considered as small, medium and large when $\hat{A}_{12}$ value is over 0.56, 0.64 and 0.71, respectively.

7 Experimental results

In this section, we present our experimental results. Section 7.1 compares the performance of three SMTIBEA variants and chooses the best. Section 7.2 compares the selected SMTIBEA variant to the current state-of-the-art (SATIBEA).

7.1 Comparison of three SMTIBEA variants

Table 5 records the experimental results of three SMTIBEA variants when applied to five SPLs. The columns *SMTIBEAv1 (v1)*, *SMTIBEAv2 (v2)* and *SMTIBEAv3 (v3)* list the measured results of each approach. In particular, we report the median and mean values of the measured results for 30 executions. We conducted statistical analysis for the comparisons of any two approaches and calculated the p values and the effect sizes $\hat{A}_{12}$. The columns *v1 VS v3*, *v2 VS v3* and *v2 VS v1* list three pairs of statistical analysis results. From them, the other three pairs, including *v3 VS v1*, *v3 VS v2* and *v1 VS v2*, can be inferred, e.g., $\hat{A}_{12}(v1\ VS\ v3) + \hat{A}_{12}(v3\ VS\ v1) = 1$. So we reported only a half of $\hat{A}_{12}$ results. The rows of the table record the measured details for each subject SPL, including Hypervolume (row HV), Epsilon (row ϵ), Inverted Generational Distance (row IGD), Error Ratio (row ER) and Generalized Spread (row GS) for 30 runs per algorithm. We highlight the median and mean values in bold that are the best for each subject system and each performance metric.

Our results reveal that no one variant is absolutely better than the other two. In terms of HV, for example, SMTIBEAv1 achieves the highest HV only for FREEBSD; SMTIBEAv2 works the best for LINUX and ECOS; and SMTIBEAv3 produces the best results for FIASCO and UCLINUX.

We calculated all the p-values and the effect sizes $\hat{A}_{12}$ for the comparisons of any two approaches and presented three pairs of them in Table 5. In terms of the p-values, the statistical differences between SMTIBEAv1 and SMTIBEAv3 (*v1 VS v3*) and between SMTIBEAv2 and SMTIBEAv3 (*v2 VS v3*) are significant (below 0.05 significance level), while

those between SMTIBEAv2 and SMTIBEAv1 (*v2 VS v1*) are not quite significant (above 0.05 significance level) except for LINUX.

For the largest subject LINUX, SMTIBEAv2 achieves the best HV, and the statistical differences between SMTIBEAv2 and SMTIBEAv3 (*v2 VS v3*) and between SMTIBEAv2 and SMTIBEAv1 (*v2 VS v1*) are significant with a large effect size (i.e., $\hat{A}_{12}$ is over 0.71 and even equals to 1). Although SMTIBEAv1 produces a better ϵ and a better IGD than SMTIBEAv2, the effect sizes of SMTIBEAv1 versus SMTIBEAv2 are less than 0.56. Furthermore, the comparisons of the convergence and the diversity are only important when there is quality in the solutions found, e.g., HV [26].

Therefore, we identify SMTIBEAv2 as the best among three variants and use it for the following comparison to the state-of-the-art.

7.2 Comparison between SMTIBEAv2 and the state-of-the-art

Table 6 compares the performances of SMTIBEAv2 and the current state-of-the-art (SATIBEA) when applied to five SPLs. To illustrate the statistical differences straightforwardly, we reported the p-values and the effect sizes $\hat{A}_{12}$ of both SMTIBEAv2 versus SATIBEA (column *v2 VS SAT*) and SATIBEA versus SMTIBEAv2 (column *SAT VS v2*). We highlight the median and mean values in bold that are the best for each subject system and for each performance metric at a significance level of 5% and with a large effect size ($\hat{A}_{12} \geq 0.71$).

In terms of HV, our results show that SMTIBEAv2 outperforms SATIBEA for 4 out of 5 SPLs (except for UCLINUX), and their statistical differences are significant with a large effect size. In terms of the ϵ and IGD, we observed that SATIBEA seemed to work better than SMTIBEAv2 for 4 out of 5 SPLs (except for FIASCO), but the effect sizes $\hat{A}_{12}$ of SATIBEA versus SMTIBEAv2 are less than 0.56. Only for UCLINUX, SATIBEA outperforms SMTIBEAv2 in terms of HV significantly with a large effect size.

Although correctness is one of the optimization objectives defined in Sect. 6.2, invalid configurations usually make little sense in practice. Hence, we filter out all invalid configurations from our experimental results, and then we recalculate the performance metrics using the other four objectives. In such a case, all evaluated solutions are valid.

Table 7 presents the results of comparing SMTIBEAv2 to SATIBEA when filtering all invalid configurations out. We highlight all results at the significance level of 0.05 and with a large effect size. In terms of HV, SMTIBEAv2 significantly outperforms SATIBEA for ECOS and FREEBSD, while SATIBEA works better than SMTIBEAv2 for LINUX

Table 5 Experimental results of five performance metrics (HV, ε, IGD, ER, and GS) of three SMTIBEA variants for five subject SPLs; the number in bold indicates the best case for each subject and each metric

SPL	Metric	SMTIBEAv1 (v1)		SMTIBEAv2 (v2)		SMTIBEAv3 (v3)		v1 VS v3		v2 VS v3		v2 VS v1	
		Median	Mean	Median	Mean	Median	Mean	p value	$\hat{A}_{12}$	p value	$\hat{A}_{12}$	p value	$\hat{A}_{12}$
ECos	HV	2.8669e−1	2.8668e−1	**2.8730e−1**	**2.8690e−1**	2.4770e−1	2.4747e−1	3.0199e−11	1.000	3.0199e−11	1.000	5.2978e−1	0.548
	ε	8.0255e−2	8.4446e−2	**7.8922e−2**	**7.9683e−2**	3.9225e−1	3.9034e−1	2.9935e−11	0.000	2.9972e−11	0.000	4.0335e−1	0.437
	IGD	5.2873e−4	5.2809e−4	**5.2273e−4**	**5.2162e−4**	2.0914e−3	2.1458e−3	3.0199e−11	0.000	3.0199e−11	0.000	9.0490e−2	0.372
	ER	**1.9365e−1**	2.1677e−1	2.0033e−1	**2.0676e−1**	8.0409e−1	8.0727e−1	3.0104e−11	0.000	3.0066e−11	0.000	5.7916e−1	0.458
	GS	8.8475e−1	8.9431e−1	8.8565e−1	8.9282e−1	**9.7781e−1**	**9.8967e−1**	3.5201e−07	0.117	3.0811e−08	0.083	9.8231e−1	0.498
Fiasco	HV	2.0483e−1	2.0500e−1	2.0494e−1	2.0543e−1	**2.7864e−1**	**2.7799e−1**	3.0199e−11	0.000	3.0199e−11	0.000	5.7929e−1	0.542
	ε	6.8757e−1	6.8672e−1	6.8701e−1	6.8487e−1	**9.6749e−2**	**1.0067e−1**	2.8198e−11	1.000	2.9192e−11	1.000	3.7198e−1	0.433
	IGD	3.1444e−3	3.1397e−3	3.1427e−3	3.1306e−3	**4.9547e−4**	**5.0705e−4**	3.0199e−11	1.000	3.0199e−11	1.000	7.8446e−1	0.479
	ER	2.8000e−1	2.7198e−1	2.000e−1	2.2247e−1	**7.8462e−2**	**1.0036e−1**	1.5425e−09	0.954	2.2268e−06	0.856	5.9378e−2	0.358
	GS	9.0371e−1	9.0545e−1	9.0538e−1	9.0570e−1	**9.2439e−1**	**9.2647e−1**	1.1711e−02	0.310	1.3272e−02	0.313	8.5338e−1	0.514
FreeBSD	HV	**3.3935e−1**	**3.3759e−1**	3.3787e−1	3.3705e−1	1.4293e−1	1.4317e−1	3.0199e−11	1.000	3.0199e−11	1.000	6.7350e−1	0.468
	ε	**1.5168e−1**	**1.5561e−1**	1.5598e−1	1.5707e−1	3.9259e−1	3.9142e−1	3.0047e−11	0.000	3.0104e−11	0.000	8.5336e−1	0.514
	IGD	8.7272e−4	8.7762e−4	**8.5681e−4**	**8.6407e−4**	3.0534e−3	3.0466e−3	3.0199e−11	0.000	3.0199e−11	0.000	2.0095e−1	0.403
	ER	**6.4190e−2**	7.8589e−2	7.2504e−2	**7.7121e−2**	1.0000e−0	9.9839e−1	9.4000e−12	0.000	9.3870e−12	0.000	7.7311e−1	0.522
	GS	9.3447e−1	9.3743e−1	9.3146e−1	9.4187e−1	**9.5868e−1**	**9.7496e−1**	1.4412e−02	0.316	2.0681e−02	0.325	8.1875e−1	0.482
uClinux	HV	2.0977e−1	2.1036e−1	2.1131e−1	2.1159e−1	**2.6365e−1**	**2.6365e−1**	3.0199e−11	0.000	3.0199e−11	0.000	4.2896e−1	0.560
	ε	6.9740e−1	6.9754e−1	6.9957e−1	6.9761e−1	**8.1440e−2**	**7.9983e−2**	2.9766e−11	1.000	2.9803e−11	1.000	8.2994e−1	0.517
	IGD	5.2522e−3	5.2410e−3	5.2602e−3	5.2447e−3	**5.9554e−4**	**6.0314e−4**	3.0199e−11	1.000	3.0199e−11	1.000	9.5873e−1	0.504
	ER	**4.0167e−1**	**4.0889e−1**	4.4500e−1	4.6256e−1	4.8636e−1	4.9513e−1	1.6279e−02	0.319	4.5527e−01	0.443	4.1606e−1	0.562
	GS	7.9419e−1	7.9253e−1	7.9083e−1	7.8579e−1	**1.0184e−0**	**1.0177e−0**	3.0199e−11	0.000	3.0199e−11	0.000	3.1830e−1	0.424
Linux	HV	2.6080e−1	2.6104e−1	**2.6366e−1**	**2.6356e−1**	2.4026e−1	2.3970e−1	3.0199e−11	1.000	3.0199e−11	1.000	1.2477e−04	0.789
	ε	**1.6249e−1**	**1.5633e−1**	2.8539e−1	2.7474e−1	2.2071e−1	2.1338e−1	4.6836e−08	0.089	1.0901e−05	0.831	1.6938e−09	0.953
	IGD	**4.6300e−4**	**4.7421e−4**	8.1699e−4	8.1545e−4	5.5959e−4	5.8133e−4	2.9215e−08	0.053	3.1589e−10	0.973	3.0199e−11	1.000
	ER	**5.7047e−2**	6.6577e−2	6.0201e−2	**6.5135e−2**	4.5470e−1	4.5712e−1	3.0142e−11	0.000	3.0142e−11	0.000	9.0584e−01	0.491
	GS	8.5554e−1	8.6698e−1	9.1133e−1	9.1841e−1	**9.2155e−1**	**9.2921e−1**	1.3367e−05	0.172	4.1191e−01	0.438	1.8608e−06	0.859

Table 6 Experimental results (including invalid configurations) of five performance metrics (HV, ϵ, IGD, ER, and GS) of SMTIBEAv2 and SATIBEA for five subject SPLs; the number in bold indicates a significantly better case with a large effect size for each subject and each metric

SPL	Metric	SMTIBEAv2 (v2)		SATIBEA (SAT)		v2 VS SAT		SAT VS v2	
		Median	Mean	Median	Mean	p value	$\widehat{A}_{12}$	p value	$\widehat{A}_{12}$
ECOS	HV	**2.8664e–1**	**2.8638e–1**	2.8203e–1	2.8178e–1	1.0937e–10	0.986	1.0937e–10	0.014
	ϵ	1.3829e–1	1.3759e–1	6.4039e–2	6.6997e–2	3.2519e–11	0.999	3.2519e–11	0.001
	IGD	6.9226e–4	6.8459e–4	5.8313e–4	5.8368e–4	4.6159e–10	0.969	4.6159e–10	0.031
	ER	1.9167e–1	1.8585e–1	2.1000e–1	2.3000e–1	5.2712e–02	0.354	5.2712e–02	0.646
	GS	**9.5198e–1**	**9.4619e–1**	8.7854e–1	8.9229e–1	3.1830e–03	0.722	3.1830e–03	0.278
FIASCO	HV	**2.4856e–1**	**2.4881e–1**	2.4241e–1	2.4397e–1	1.7836e–4	0.782	1.7836e–4	0.218
	ϵ	1.0139e–1	1.0313e–1	1.0959e–1	1.0890e–1	1.6531e–2	0.319	1.6531e–2	0.681
	IGD	8.9115e–4	8.9490e–4	9.4019e–4	9.3008e–4	2.4157e–2	0.330	2.4157e–2	0.670
	ER	1.6333e–1	1.7976e–1	2.5417e–1	2.5009e–1	1.2986e–3	0.258	1.2986e–3	0.742
	GS	6.1394e–1	6.1753e–1	6.2685e–1	6.2895e–1	5.7929e–1	0.458	5.7929e–1	0.542
FREEBSD	HV	**3.2792e–1**	**3.2687e–1**	2.6101e–1	2.6096e–1	3.0199e–11	1.000	3.0199e–11	0.000
	ϵ	2.3596e–1	2.3250e–1	1.7366e–1	1.7394e–1	4.9712e–09	0.94	4.9712e–09	0.060
	IGD	1.7802e–3	1.7677e–3	1.7009e–3	1.7144e–3	7.0127e–02	0.637	7.0127e–02	0.363
	ER	4.1170e–2	4.3587e–2	6.7246e–1	6.6796e–1	3.0104e–11	0.000	3.0104e–11	1.000
	GS	9.4224e–1	9.5058e–1	9.0261e–1	9.2948e–1	3.5137e–02	0.659	3.5137e–02	0.341
UCLINUX	HV	2.0922e–1	2.0932e–1	**2.9389e–1**	**2.9321e–1**	3.0199e–11	0.000	3.0199e–11	1.000
	ϵ	3.5314e–1	3.5611e–1	6.7987e–2	6.7847e–2	2.9027e–11	1.000	2.9027e–11	0.000
	IGD	3.0566e–3	3.0972e–3	8.1260e–4	8.2772e–4	3.0199e–11	1.000	3.0199e–11	0.000
	ER	2.8000e–1	3.0311e–1	3.4500e–1	3.4244e–1	1.1703e–01	0.382	1.1703e–01	0.618
	GS	**6.3612e–1**	**6.6254e–1**	5.8158e–1	5.9492e–1	1.8916e–04	0.781	1.8916e–04	0.219
LINUX	HV	**2.2168e–1**	**2.2217e–1**	2.1870e–1	2.1841e–1	3.8481e–03	0.718	3.8481e–03	0.282
	ϵ	5.5890e–1	5.5324e–1	1.4448e–1	1.4340e–1	3.0199e–11	1.000	3.0199e–11	0.000
	IGD	2.7086e–3	2.6476e–3	1.0065e–3	9.8871e–4	3.0199e–11	1.000	3.0199e–11	0.000
	ER	2.5207e–2	2.9674e–2	2.8984e–1	2.8867e–1	3.0123e–11	0.000	3.0123e–11	1.000
	GS	9.5370e–1	9.5481e–1	9.4104e–1	9.5809e–1	4.8252e–01	0.553	4.8252e–01	0.447

and UCLINUX. Although SATIBEA tends to produce a better ϵ and a better IGD than SMTIBEAv2, the effect sizes are not large. Note that, in Table 7, a ER value of 0 indicates that all solutions produced by the corresponding algorithm are a part of the reference front. If all input data are 0, it is not necessary to perform statistical tests, and we mark '–' in the table.

8 Discussion

In this section, we discuss the experimental results and answer the research questions. We present our perspectives regarding several interesting aspects, including selection of performance metrics, convergence speed of algorithms, expressiveness of our approach, timeout of a call to a constraint solver, and strategies of handling constraints. Finally, we explain the threats to validity.

8.1 Research questions

Regarding RQ1, we proposed SMTIBEA, the first approach that augments IBEA with SMT solving to address richer constraints in real-world SPLs. SMTIBEA maps the SPL configuration optimization problem to quantifier-free first-order formulas, which significantly extend the expressiveness of the CNF Boolean formulas used in the state-of-the-art (SATIBEA). SMTIBEA enhances the capability of constraint reasoning by combining SAT solving with linear arithmetic over integers and real variables. In particular, SMTIBEA enables the representation and automated resolution of constraints defined not only on features but also on quality attributes. Moreover, SMTIBEA supports the improved search [38] using the superior constraint defined in equation (3), which essentially involves a combination of linear inequalities.

Regarding RQ2, we incorporated the SMT solving into the mutation operator and the initial population genera-

Table 7 Experimental results (excluding invalid configurations) of five performance metrics (HV, ϵ, IGD, ER, and GS) of SMTIBEAv2 and SATIBEA for five subject SPLs; the number in bold indicates a significantly better case with a large effect size for each subject and each metric

SPL	Metric	SMTIBEAv2 (v2)		SATIBEA (SAT)		v2 VS SAT		SAT VS v2	
		Median	Mean	Median	Mean	p value	$\widehat{A}_{12}$	p value	$\widehat{A}_{12}$
ECOS	HV	**2.8360e–1**	**2.8383e–1**	2.7914e–1	2.7918e–1	2.6695e–09	0.948	2.6695e–09	0.052
	ϵ	1.4959e–1	1.4872e–1	5.2745e–2	5.4492e–2	3.0085e–11	1.000	3.0085e–11	0.000
	IGD	2.5344e–3	2.6055e–3	1.4870e–3	1.5587e–3	6.0658e–11	0.992	6.0658e–11	0.008
	ER	3.6777e–3	8.0555e–3	3.5527e–3	7.1132e–3	8.0600e–01	0.518	8.0600e–01	0.482
	GS	8.4228e–1	8.4768e–1	8.1362e–1	8.2148e–1	6.7869e–02	0.638	6.7869e–02	0.362
FIASCO	HV	2.4589e–1	2.4390e–1	2.4766e–1	2.4745e–1	3.2553e–01	0.426	3.2553e–01	0.574
	ϵ	1.0607e–1	1.0582e–1	7.8301e–2	8.1879e–2	1.1058e–04	0.791	1.1058e–04	0.209
	IGD	9.4013e–3	9.1720e–3	6.8848e–3	6.4806e–3	1.3594e–07	0.897	1.3594e–07	0.103
	ER	0.0000e–0	1.4493e–3	0.0000e–0	1.2289e–2	2.0038e–03	0.337	2.0038e–03	0.663
	GS	7.1778e–1	7.2302e–1	6.9608e–1	6.8854e–1	2.0095e–01	0.597	2.0095e–01	0.403
FREEBSD	HV	**1.2933e–1**	**1.2645e–1**	6.3163e–2	7.2578e–2	2.2986e–2	0.767	2.2986e–2	0.233
	ϵ	6.3110e–1	6.2687e–1	0.0000e–0	6.5392e–2	1.8852e–06	1.000	1.8852e–06	0.000
	IGD	1.4785e–1	1.8820e–1	0.0000e–0	1.1499e–2	2.2958e–06	0.996	2.2958e–06	0.004
	ER	0.0000e–0	0.0000e–0	0.0000e–0	1.6265e–3	4.8778e–01	0.467	4.8778e–01	0.533
	GS	9.9790e–1	9.9346e–1	1.1063e–0	1.0894e–0	5.7888e–01	0.433	5.7888e–01	0.567
UCLINUX	HV	2.0922e–1	2.0932e–1	**2.9389e–1**	**2.9321e–1**	3.0199e–11	0.000	3.0199e–11	1.000
	ϵ	3.5314e–1	3.5611e–1	5.6624e–2	5.8685e–2	2.9210e–11	1.000	2.9210e–11	0.000
	IGD	1.3923e–2	1.4078e–2	2.8927e–3	2.9652e–3	3.0199e–11	1.000	3.0199e–11	0.000
	ER	0.0000e–0	5.3333e–3	2.3333e–2	2.2339e–2	9.3307e–06	0.184	9.3307e–06	0.816
	GS	**5.9739e–1**	**6.1310e–1**	5.3003e–1	5.4193e–1	2.4327e–05	0.818	2.4327e–05	0.182
LINUX	HV	1.2684e–1	1.1858e–1	**1.6351e–1**	**1.6490e–1**	9.5349e–09	0.064	9.5349e–09	0.936
	ϵ	8.1048e–1	8.1790e–1	1.2577e–1	1.0932e–1	4.4618e–11	1.000	4.4618e–11	0.000
	IGD	1.1803e–1	1.1613e–1	7.0226e–3	6.4436e–3	4.4618e–11	1.000	4.4618e–11	0.000
	ER	0.0000e–0	0.0000e–0	0.0000e–0	0.0000e–0	–	–	–	–
	GS	9.8525e–1	9.7513e–1	9.8389e–1	9.8174e–1	6.5466e–01	0.466	6.5466e–01	0.534

tion of IBEA, and we designed three approach variants SMTIBEAv1, SMTIBEAv2, and SMTIBEAv3 for performance comparison. Firstly, our experimental results reinforced the importance of incorporating the constraint solving into the mutation operator, which has been demonstrated by the state-of-the-art as well.

Secondly, the key difference between SMTIBEAv1 and SMTIBEAv2 is that SMTIBEAv2 starts with an initial population of all valid configurations. According to our results in Table 5, the statistical differences between SMTIBEAv1 and SMTIBEAv2 are not quite significant for 4 SPLs, but SMTIBEAv2 significantly outperforms SMTIBEAv1 with a large effect size for LINUX. Therefore, we conclude that incorporating constraint solving into the initial population generation of IBEA is preferable.

Thirdly, the variant SMTIBEAv3 augments the mutation operator of SMTIBEAv1 with the improved search that combines the SMT solving and the superior constraint to return a better solution than the currently found ones. Our empirical

results in Table 5 show that SMTIBEAv3 produces the worst HV among three variants for 3 out of 5 SPLs significantly with a large effect size. Hence, according to our experimental results, we do not recommend augmenting the mutation operator with the improved search.

Regarding RQ3, Our SMTIBEA approach achieves a comparable performance to the state-of-the-art (SATIBEA), given that SMTIBEA has significantly improved the expressiveness of constraints. As reported in Table 6, our approach significantly outperforms SATIBEA in terms of HV for 4 out of 5 SPLs (except for FIASCO) with a large effect size. After filtering out all invalid configurations from the results, as listed in Table 7, our approach still works significantly better than the state-of-the-art for 2 SPLs (ECOS and FREEBSD).

8.2 Performance metrics

The performance evaluation of MOEAs depends mainly on a number of heuristically chosen metrics, such as Hyper-

volume [6]. However, every metric provides some specific, but incomplete, quantifications of optimality and can only be used effectively under certain conditions [56]. In the original SATIBEA paper [26], the authors introduced a Pareto Front Size (PFS) metric to assess the diversity of the produced solutions. The PFS metric is defined as the number of solutions in an approximate front A produced by an algorithm. A higher PFS is preferred since more options are given to users.

Table 8 presents the PFS results comparing SMTIBEAv2 to SATIBEA when applied to five SPLs. We highlight the median and mean values in bold that are better for each subject at a significance level of 5% and with a large effect size ($\hat{A}_{12} \geq 0.71$). As shown in the table, SMTIBEAv2 gains a higher PFS than SATIBEA for 3 out of 5 SPLs with a large effect size. For subject FIASCO, the calculated p-value is not significant. For subject FREEBSD, the effect size reaches medium (over 0.64 but less than 0.71) but not large.

Even though we buy the idea of the PFS metric, we prefer a metric that has been widely accepted by the community. Technically, the PFS metric indicates the absolute cardinality of an approximate front A, while the ER metric calculates the percentage of solutions in an approximate front A that are not solutions in the reference front R. Thus, the PFS metric might be skewed to an algorithm that produces a larger approximate front A without checking the correlation to the global reference front R, whereas the ER metric is able to reflect the correlation and helps users to choose an algorithm that can find more "really good" solutions (globally Pareto-optimal in the reference front). Therefore, in this paper, we adopt the ER metric instead of the PFS metric to assess the diversity. Note that this diversity metric is only important when high quality is preserved.

8.3 Convergence speed

It is interesting to understand the convergence speed of our approach compared to the state-of-the-art. For each of the five subjects, we conducted experiments that performed SMTIBEAv2 and SATIBEA, respectively, within a certain time budget, from 5 to 30 min. We independently executed each algorithm 30 times for each subject and each time budget, and we took the mean values of the measurements for

analysis. Our experimental results indicated that the convergence speeds of SMTIBEAv2 and SATIBEA are comparable. As shown in Figure 3, for subject LINUX, which is the largest available SPL hitherto reported upon in the literature, both algorithms quickly converge to a relatively stable HV value even in 5 min. Moreover, from 5 to 30 min, the HV changes within a small range of 0.04. In addition, the HV values of SATIBEA present slight fluctuation, while those of SMTIBEAv2 tend to behave stably.

8.4 Expressiveness

The configuration optimization problems in real-world SPLs usually involve various complex constraints, including not only Boolean but also non-Boolean constraints, such as those over infinite domain variables (integers, real numbers, and character strings) together with arithmetic, relational, and string operators [5,37,40,41]. Previous approaches [3,4,26, 48] mostly support only Boolean constraints and thus suffer
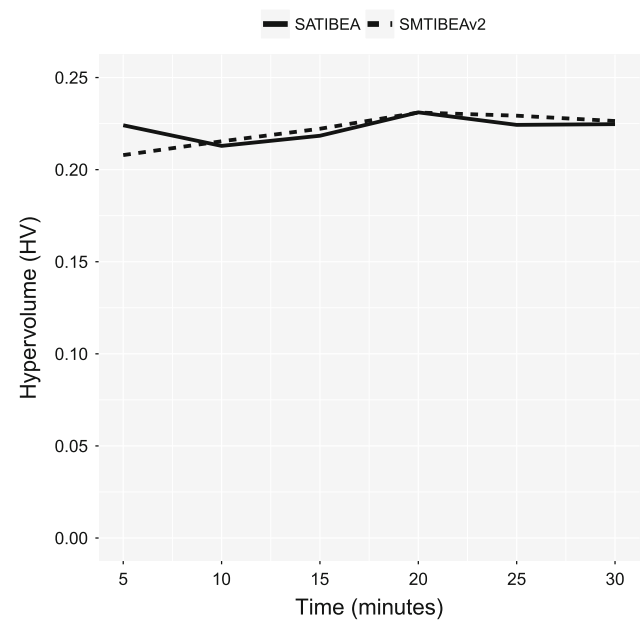

Fig. 3 Hypervolume over time for SMTIBEAv2 and SATIBEA on Linux

Table 8 Experimental results of the Pareto Front Size (PFS) metric of SMTIBEAv2 and SATIBEA for five subject SPLs; the number in bold indicates a significantly better case with a large effect size

SPL	SMTIBEAv2 (v2)		SATIBEA (SAT)		v2 VS SAT		SAT VS v2	
	Median	Mean	Median	Mean	p value	$\hat{A}_{12}$	p value	$\hat{A}_{12}$
ECOS	**182**	**181**	168	168	2.9283e–09	0.946	2.9283e–09	0.054
FIASCO	119	121	115	140	1.7807e–01	0.602	1.7807e–01	0.398
FREEBSD	223	224	220	218	3.5580e–02	0.658	3.5580e–02	0.342
UCLINUX	**116**	**117**	106	107	2.2941e–07	0.889	2.2941e–07	0.111
LINUX	**296**	**295**	290	290	4.7581e–08	0.909	4.7581e–08	0.091

from limited applicability. Our SMTIBEA approach significantly improves the expressiveness from Boolean constraints to quantifier-free first-order constraints, particularly without sacrificing much performance.

Theoretically, SMT solving may take more time than SAT solving, because it encodes and resolves more variables and more complex constraints. Given a subject SPL, let the number of features be N and the number of structural constraints defined between features M. For example, as defined in Table 3, the subject LINUX contains $N = 6\,888$ features and $M = 343\,944$ constraints. In general, the SAT encoding used by SATIBEA contains M structural constraints for each system and thus the SAT solver finds a valid solution that satisfies M constraints defined between features. In contrast, the SMT encoding used by SMTIBEA contains not only structural constraints but also quality constraints. For example, if a feature f of subject LINUX has a cost of 100, then there is a quality (cost) constraint encoded as $cost(f) = 100$, as explained in Sect. 5.2. In this way, for a certain quality attribute, a system has to encode N quality constraints for all N features. In our case studies, as mentioned in Sect. 6.1, there are three kinds of quality attributes: cost, defects, and used_before. Hence, the SMT encoding used by SMTIBEA contains $(M + 3N)$ constraints in total for each system.

We aim at a fair comparison to SATIBEA using the same experimental settings in this paper, which limits the expressiveness of SMT solving to some extent. To this end, we designed SMTIBEAv3 with the improved search that incorporates the superior constraints defined by Eq. (3). The superior constraints cannot be encoded and resolved directly by SAT solving, but we are aware that there exist techniques that translate some pseudo-Boolean constraints into SAT [15].

Our empirical studies on large, constrained SPLs demonstrated the effectiveness and scalability of SMTIBEA. Surprisingly, SMTIBEA even achieves a comparable performance to the state-of-the-art. Furthermore, we published the source code of our implementation of SMTIBEA, and we encourage researchers to use it to express and solve more SPL configuration optimization problems with various complex constraints.

8.5 Timeout setting

As mentioned above, SMT solving may take more time than SAT solving. To make a fair comparison to the state-of-the-art, we set the same timeout of a call to the underlying solver to 6 s in our experiments. What if we increase the timeout of a call as well as the total running time of an algorithm? We conducted an experiment that compares SMTIBEAv2 to SATIBEA using the same experimental setup mentioned in Sect. 6 but only tuned the timeout of a call to the underlying

Z3 solver in SMTIBEAv2 to be 12 s and the total running time of SMTIBEAv2 and SATIBEA to be 30 min. Table 9 shows the experimental results. SMTIBEAv2 outperforms SATIBEA for 3 out of 5 SPLs in terms of HV. For subject LINUX, SMTIBEAv2 gains a higher HV than SATIBEA, but the statistical results (p-value) are not significant. Comparing to the previous results in Table 6 where the timeout of a Z3 call is 6 s, our experimental results demonstrated that increasing the timeout of the underlying solver of SMTIBEAv2 may not necessarily improve the algorithm performance.

It is non-trivial to set an appropriate timeout of a call to the underlying constraint solver for a hybrid evolutionary algorithm (e.g., SMTIBEA or SATIBEA), because it involves the trade-off between constraint solving and evolutionary search within a certain time limit. If the timeout is set to be higher, the algorithm tends to spend more time in constraint solving to acquire a valid solution, but this significantly reduces the running time for evolutionary search. Moreover, the case becomes worse if the solver cannot find a valid solution and returns a random, probably invalid, solution in the end, and thus a higher timeout maybe leads to wasting more time. If the timeout is set to be lower, the solver may have a higher probability not to find a valid solution, which still wastes the running time.

The above trade-off becomes more complex for SMTIBEAv3 where the improved search is incorporated. The improved search is the key to the state-of-the-art exact multi-objective optimization approach for the SPL configuration optimization problem [18,38]. It relies on the incremental solving by augmenting additional constraints, which is widely used in the community of artificial intelligence and constraint programming [17,25]. Also, the incremental solving is supported by many off-the-shelf solvers, such as SAT4J and Z3. However, an improved search that finds a better solution by augmenting the superior constraint defined in equation (3), usually takes much more time than a straightforward call to the underlying solver to return an arbitrary solution. Furthermore, the running time of finding a better solution changes dynamically in terms of the currently augmented constraint and the specific problem. Hence, it is non-trivial to predefine an appropriate timeout for returning a better solution during the improved search. In the future, it deserves to study further how to tune the timeout and reach a sweet spot between the improved search and the evolutionary search, which might be interesting and meaningful for both communities.

8.6 Strategies of handling constraints

There are many hybrid algorithm variants we can design by incorporating SAT/SMT solving, random generation, and the improved search into different components of IBEA and by tuning the parameter values of IBEA. An interesting variant is to keep configurations valid "all along the way" that is both

Table 9 Experimental results of five performance metrics (HV, ϵ, IGD, ER, and GS) of SMTIBEAv2 and SATIBEA for five subject SPLs in 30 min; the timeout of a call to the underlying Z3 solver of SMTIBEAv2 is 12 s; the number in bold indicates a significantly better case with a large effect size for each subject and each metric

SPL	Metric	SMTIBEAv2 (v2)		SATIBEA (SAT)		v2 VS SAT		SAT VS v2	
		Median	Mean	Median	Mean	p value	$\widehat{A}_{12}$	p value	$\widehat{A}_{12}$
ECOS	HV	**2.8963e–1**	**2.8938e–1**	2.8666e–1	2.8632e–1	2.0152e–08	0.922	2.0152e–08	0.078
	ϵ	1.1366e–1	1.1885e–1	6.1366e–2	6.1956e–2	3.0085e–11	1.000	3.0085e–11	0.000
	IGD	5.3500e–4	5.3690e–4	4.3174e–4	4.3436e–4	4.9752e–11	0.994	4.9752e–11	0.006
	ER	1.6861e–1	1.8950e–1	2.3500e–1	2.5193e–1	1.8510e–03	0.266	1.8510e–03	0.734
	GS	8.3794e–1	8.3602e–1	8.2472e–1	8.2989e–1	5.7930e–01	0.542	5.7930e–01	0.458
FIASCO	HV	**2.5353e–1**	**2.5449e–1**	2.4970e–1	2.5102e–1	4.0330e–03	0.717	4.0330e–03	0.283
	ϵ	1.0669e–1	1.0667e–1	1.0828e–1	1.0717e–1	8.1863e–01	0.482	8.1863e–01	0.518
	IGD	9.5590e–4	9.6205e–4	9.9559e–4	9.9688e–4	8.7710e–02	0.371	8.7710e–02	0.629
	ER	1.7333e–1	1.8760e–1	2.4833e–1	2.4091e–1	1.3259e–02	0.313	1.3259e–02	0.687
	GS	6.2102e–1	6.2199e–1	6.6546e–1	6.5555e–1	6.9125e–04	0.244	6.9125e–04	0.756
FREEBSD	HV	**3.5881e–1**	**3.5856e–1**	2.8820e–1	2.8765e–1	3.0199e–11	1.000	3.0199e–11	0.000
	ϵ	1.5756e–1	1.6265e–1	1.6867e–1	1.7055e–1	7.8490e–02	0.367	7.8490e–02	0.633
	IGD	1.2342e–3	1.2346e–3	1.6901e–3	1.6951e–3	8.9934e–11	0.012	8.9934e–11	0.988
	ER	3.2193e–2	4.0784e–2	7.5007e–1	7.3555e–1	3.0066e–11	0.000	3.0066e–11	1.000
	GS	9.5904e–1	9.6559e–1	9.5024e–1	9.5308e–1	1.9073e–01	0.599	1.9073e–01	0.401
UCLINUX	HV	2.3109e–1	2.3189e–1	**3.0334e–1**	**3.0357e–1**	3.0199e–11	0.000	3.0199e–11	1.000
	ϵ	3.3993e–1	3.4103e–1	7.5908e–2	7.4576e–2	2.9229e–11	1.000	2.9229e–11	0.000
	IGD	2.4728e–3	2.4804e–3	8.3877e–4	8.5572e–4	3.0199e–11	1.000	3.0199e–11	0.000
	ER	2.4167e–1	2.6978e–1	3.3667e–1	3.4356e–1	1.3512e–02	0.315	1.3512e–02	0.685
	GS	**6.5184e–1**	**6.4823e–1**	5.9173e–1	5.9157e–1	8.3520e–08	0.903	8.3520e–08	0.097
LINUX	HV	2.2608e–1	2.2572e–1	2.2294e–1	2.2412e–1	1.1882e–01	0.618	1.1882e–01	0.382
	ϵ	5.3585e–1	5.3545e–1	1.2852e–1	1.2627e–1	3.0199e–11	1.000	3.0199e–11	0.000
	IGD	2.7879e–3	2.7641e–3	1.1992e–3	1.1655e–3	3.0199e–11	1.000	3.0199e–11	0.000
	ER	3.3445e–2	3.3288e–2	2.6845e–1	2.7740e–1	3.0142e–11	0.000	3.0142e–11	1.000
	GS	**9.4682e–1**	**9.5138e–1**	9.1734e–1	9.1713e–1	4.4592e–04	0.764	4.4592e–04	0.236

in the initial population and at each iteration of the algorithm. To meet this goal, we designed a variant called SMTIBEAv4 that changes three parameter values of SMTIBEAv2: The mutation rate is one, the probability of using standard bit-flip mutation is zero, and the probability of using SMT solving for mutation is one. Note that SMTIBEAv4 generates an initial population of all valid configurations, and it always performs mutation to produce a valid configuration at each iteration.

Table 10 shows the experimental results of SMTIBEAv2 and SMTIBEAv4 when applied to five SPLs. We highlight the median and mean values in bold that are better for each subject at a significance level of 5% and with a large effect size. As shown in the table, SMTIBEAv2 outperforms SMTIBEAv4 for all subject SPLs in terms of HV with a large effect size. Moreover, we counted the number of invalid configurations produced by SMTIBEAv2 for each subject SPL and listed the medians and means in Table 11. As expected, not all configurations produced by SMTIBEAv2 are valid, since correctness is used as an opti-

mization objective in the experimental settings. In contrast, all configurations produced by SMTIBEAv4 are valid, which is a distinct advantage if users prefer valid configurations all along the way. In this paper, we aim at a fair comparison to SATIBEA using the same experimental settings and further study on the validity and other concerns will be conducted in future.

Eiben and Smith [16] introduced three techniques to handle the constraints when carrying out evolutionary computing: (1) designing a penalty function to de-prioritize solutions that violate constraints, (2) designing a repair operator to ensure that each solution is fixed to satisfy constraints, and (3) modifying the combination and mutation operators so that only valid solutions are generated. Sayyad et al.'s IBEA approach [49] adopts the first technique, that is, using the number of constraints violated as an optimization objective to minimize. Henard et al.'s SATIBEA approach [26] and our three SMTIBEA variants (SMTIBEAv1, SMTIBEAv2 and SMTIBEAv3) adopt a combination of the first and the

Table 10 Experimental results of five performance metrics (HV, ϵ, IGD, ER, and GS) of SMTIBEAv2 and SMTIBEAv4 for five SPLs; the number in bold indicates a significantly better case with a large effect size for each subject and each metric

SPL	Metric	SMTIBEAv2 (v2)		SMTIBEAv4 (v4)		v2 VS v4		v4 VS v2	
		Median	Mean	Median	Mean	p value	$\widehat{A}_{12}$	p value	$\widehat{A}_{12}$
ECOS	HV	**2.8632e−1**	**2.8602e−1**	2.2691e−1	2.2158e−1	3.0199e−11	1.000	3.0199e−11	0.000
	ϵ	8.5066e−2	8.9225e−2	2.3157e−1	2.3404e−1	3.0047e−11	0.000	3.0047e−11	1.000
	IGD	9.0442e−4	9.0753e−4	1.7347e−3	1.8343e−3	3.0199e−11	0.000	3.0199e−11	1.000
	ER	1.2000e−1	1.3050e−1	9.1238e−1	9.1319e−1	3.0047e−11	0.000	3.0047e−11	1.000
	GS	**9.6413e−1**	**9.6193e−1**	6.5595e−1	6.5273e−1	3.0199e−11	1.000	3.0199e−11	0.000
FIASCO	HV	**2.4617e−1**	**2.4651e−1**	1.1048e−1	1.1000e−1	3.0199e−11	1.000	3.0199e−11	0.000
	ϵ	9.9678e−2	9.9815e−2	4.4150e−1	4.3990e−1	2.9580e−11	0.000	2.9580e−11	1.000
	IGD	1.0544e−3	1.0470e−3	7.7143e−3	7.7120e−3	3.0199e−11	0.000	3.0199e−11	1.000
	ER	1.2000e−1	1.4227e−1	8.9906e−1	9.0043e−1	3.0180e−11	0.000	3.0180e−11	1.000
	GS	6.0952e−1	6.1151e−1	**6.8368e−1**	**6.8527e−1**	4.9980e−09	0.060	4.9980e−09	0.940
FREEBSD	HV	**3.3788e−1**	**3.3687e−1**	1.8238e−1	1.8159e−1	3.0199e−11	1.000	3.0199e−11	0.000
	ϵ	1.4783e−1	1.4800e−1	4.0628e−1	4.0055e−1	3.0010e−11	0.000	3.0010e−11	1.000
	IGD	1.1314e−3	1.1442e−3	2.9916e−3	3.0234e−3	3.0199e−11	0.000	3.0199e−11	1.000
	ER	4.1169e−2	4.3589e−2	7.5555e−1	7.5015e−1	3.0123e−11	0.000	3.0123e−11	1.000
	GS	**9.3080e−1**	**9.4051e−1**	7.6730e−1	7.6822e−1	3.0199e−11	1.000	3.0199e−11	0.000
UCLINUX	HV	**2.9750e−1**	**2.9591e−1**	1.1080e−1	1.1282e−1	3.0199e−11	1.000	3.0199e−11	0.000
	ϵ	1.5959e−1	1.6022e−1	4.5333e−1	4.4872e−1	2.9991e−11	0.000	2.9991e−11	1.000
	IGD	1.1919e−3	1.2445e−3	5.7614e−3	5.7053e−3	3.0199e−11	0.000	3.0199e−11	1.000
	ER	2.2833e−1	2.6933e−1	1.0000e−0	9.9963e−1	2.3602e−12	0.000	2.3602e−12	1.618
	GS	5.0094e−1	5.2249e−1	**5.9722e−1**	**5.9640e−1**	1.5582e−08	0.074	1.5582e−08	0.926
LINUX	HV	**2.5851e−1**	**2.5885e−1**	2.4093e−1	2.4080e−1	3.0199e−11	1.000	3.0199e−11	0.000
	ϵ	2.3782e−1	2.2780e−1	3.8842e−1	3.6713e−1	4.2706e−07	0.119	4.2706e−07	0.881
	IGD	7.8999e−4	8.0719e−4	2.2598e−3	2.2540e−3	3.0199e−11	0.000	3.0199e−11	1.000
	ER	2.8523e−2	3.3374e−2	5.1905e−1	5.1811e−1	3.0123e−11	0.000	3.0123e−11	1.000
	GS	**8.5187e−1**	**8.4611e−1**	7.6980e−1	7.7058e−1	2.4386e−09	0.949	2.4386e−09	0.051

third techniques, especially incorporating constraint solving into the mutation operator. SMTIBEAv4 adopts the second technique, because the mutation essentially fixes a configuration to be always valid at each iteration of the algorithm. In the future, we plan to investigate more about the strengths and weaknesses of each technique and study algorithm variants by combining the above three techniques.

Table 11 Median and mean of the number of invalid configurations produced by SMTIBEAv2 for five SPLs

System	Median	Mean
ECOS	21	22.5
FREEBSD	294	293.1
FIASCO	228	229.7
UCLINUX	0	0
LINUX	294	293.0

8.7 Threats to validity

To enhance internal validity, we replicated the state-of-the-art (SATIBEA) using the exactly same code published by the original authors [26]. To make a fair comparison, our implementation of the SMTIBEA method uses the same framework and parameter settings as the state-of-the-art. As listed in Table 2, the key difference lies in the SMT solving and the improved search using the superior constraints. The improved search follows the standard method of GIA [38]. To avoid the misleading effects caused by random

fluctuation in measurements, each studied method 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 of SMTIBEA for any study on reproducibility.

Even though the feature models we used in our experiments are real, a threat to our analysis is the use of synthetic data as the three quality attributes of features, as described in Sect. 6.1. The data were generated randomly based on

distributions seen in historical datasets [49]. The difficulty of acquiring real data comes from the fact that the real data are often proprietary and not published. Future work should attempt to collect real data for evaluation in practice.

Many metrics were proposed to evaluate the performance of MOEAs. We are aware that every existing metric provides some specific, but incomplete, quantifications of performance and can only be used effectively under certain conditions [56]. To make our performance evaluation as comprehensive as possible, we used five performance metrics taking quality, convergence, and diversity into account.

To increase external validity, we evaluated five large SPL case studies. All these SPLs have been deployed and used in real-world scenarios, and they are highly constrained. The quality attributes of features and the optimization objectives are synthetic, since the actual data are not available for the studied subjects. To make a fair comparison, we used the same synthetic data provided by the state-of-the-art. However, we are aware that the results of our experiments may not be automatically transferable to all other configuration optimization problems of SPLs.

9 Conclusion

We proposed SMTIBEA, the first hybrid multi-objective optimization algorithm that combines IBEA with SMT solving for the SPL configuration optimization problem. SMTIBEA significantly extends the constraint expressiveness of the state-of-the-art from CNF Boolean formulas to quantifier-free first-order formulas and thus supports richer constraints in real-world SPLs.

We conducted experiments on five large, real-world, highly constrained SPLs, and we evaluated the algorithms in terms of five performance metrics. Empirical results demonstrated that our approach is comparable in performance to the state-of-the-art, given that the expressiveness of constraints has been significantly improved.

We designed three SMTIBEA variants to investigate the performance influence of SMT solving on the mutation operator and the initial population generation of the IBEA. Our empirical results reinforced the importance of augmenting the mutation operator with constraint solving. Augmenting the initial population generation with SMT solving tends to produce solutions with a better HV, so it is recommended.

Future work includes investigating the expressiveness of our approach on more complex SPL configuration optimization problems. Moreover, different ways of combining the improved search with the evolutionary search to improve the performance will be studied.

References

1. Apel, S., Batory, D.S., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines—Concepts and Implementation. Springer, Berlin (2013)
2. Arcuri, A., Briand, L.C.: A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of 33rd International Conference on Software Engineering (ICSE), pp. 1–10 (2011)
3. Batory, D.: Feature models, grammars, and propositional formulas. In: Proceedings of 9th International Software Product Line Conference (SPLC), pp. 7–20 (2005)
4. Benavides, D., Segura, S., Cortés, A.: Automated analysis of feature models 20 years later: a literature review. Inf. Syst. **35**(6), 615–636 (2010)
5. Berger, T., She, S., Lotufo, R., Wasowski, A., Czarnecki, K.: A study of variability models and languages in the systems software domain. IEEE Trans. Softw. Eng. **39**(12), 1611–1640 (2013)
6. Brockhoff, D., Friedrich, T., Neumann, F.: Analyzing hypervolume indicator based algorithms. In: Proceedings of 10th International Conference on Parallel Problem Solving from Nature (PPSN), pp. 651–660 (2008)
7. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Boston (2001)
8. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Boston (2000)
9. de Moura, L., Bjørner, N.: Satisfiability modulo theories: introduction and applications. Commun. ACM **54**(9), 69–77 (2011)
10. de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Proceedings of 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pp. 337–340 (2008)
11. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
12. Deb, K., Mohan, M., Mishra, S.: Towards a quick computation of well-spread Pareto-optimal solutions. In: Proceedings of Second International Conference on Evolutionary Multi-Criterion Optimization (EMO), pp. 222–236 (2003)
13. Durillo, J.J., Nebro, A.J.: jMetal: a java framework for multi-objective optimization. Adv. Eng. Softw. **42**, 760–771 (2011)
14. Durillo, J.J., Nebro, A.J., Alba, E.: The jMetal framework for multi-objective optimization: design and architecture. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC), pp. 4138–4325. Barcelona, Spain (2010)
15. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. J. Satisf. Boolean Model. Comput. **2**(1–4), 1–26 (2006)
16. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Natural Computing Series. Springer, Berlin (2003)
17. Gavanelli, M.: An algorithm for multi-criteria optimization in CSPs. In: Proceedings of 15th European Conference on Artificial Intelligence (ECAI), pp. 136–140 (2002)
18. Guo, J., Czarnecki, K., Apel, S., Siegmund, N., Wąsowski, A.: Variability-aware performance prediction: a statistical learning approach. In: Proceedings of 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 301–311 (2013)

19. Guo, J., White, J., Wang, G., Li, J., Wang, Y.: A genetic algorithm for optimized feature selection with resource constraints in software product lines. J. Syst. Softw. **84**(12), 2208–2221 (2011)

20. Guo, J., Zulkoski, E., Olaechea, R., Rayside, D., Czarnecki, K., Apel, S., Atlee, J.M.: Scaling exact multi-objective combinatorial optimization by parallelization. In: Proceedings of 29th ACM/IEEE International Conference on Automated Software Engineering (ASE), pp. 409–420 (2014)

21. Harman, M.: The current state and future of search based software engineering. In: Proceedings of International Workshop on the Future of Software Engineering (FOSE), pp. 342–357 (2007)

22. Harman, M., Jia, Y., Krinke, J., Langdon, W.B., Petke, J., Zhang, Y.: 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), pp. 5–18 (2014)

23. Harman, M., Jones, B.F.: Search-based software engineering. Inf. Softw. Technol. **43**(14), 833–839 (2001)

24. Harman, M., Mansouri, S., Zhang, Y.: Search based software engineering: a comprehensive analysis and review of trends techniques and applications. Tech. rep., King's College London TR-09-03 (2009)

25. Hartert, R., Schaus., P.: A support-based algorithm for the bi-objective Pareto constraint. In: Proceedings of 28th AAAI Conference on Artificial Intelligence (AAAI), pp. 2674–2679 (2014)

26. Henard, C., Papadakis, M., Harman, M., Traon, Y.L.: 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), pp. 517–528 (2015)

27. Henard, C., Papadakis, M., Perrouin, G., Klein, J., Traon, Y.L.: Multi-objective test generation for software product lines. In: Proceedings of 17th International Software Product Line Conference (SPLC), pp. 62–71 (2013)

28. Hierons, R.M., Li, M., Liu, X., Segura, S., Zheng, W.: SIP: optimal product selection from feature models using many-objective evolutionary optimization. ACM Trans. Softw. Eng. Methodol. **25**(2), 17 (2016)

29. Ishibuchi, H., Masuda, H., Tanigaki, Y., Nojima, Y.: Modified distance calculation in generational distance and inverted generational distance. In: Proceedings of 8th International Conference on Evolutionary Multi-Criterion Optimization (EMO), pp. 110–125 (2015)

30. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT, Cambridge (2006)

31. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-oriented domain analysis (FODA) feasibility study. Tech. rep., CMU SEI, SEI-90-TR-21 (1990)

32. Knowles, J., Corne, D.: On metrics for comparing nondominated sets. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC), pp. 711–716 (2002)

33. Lu, H., Yue, T., Ali, S., Zhang, L.: Nonconformity resolving recommendations for product line configuration. In: Proceedings of Ninth International Conference on Software Testing, Verification and Validation (ICST), pp. 57–68 (2016)

34. LVAT: Linux variability analysis tools. http://code.google.com/p/linux-variability-analysis-tools

35. Marek, V., Truszczynski, M.: Stable models and an alternative logic programming paradigm. In: The Logic Programming Paradigm: A 25-year Perspective. Springer (1999)

36. Mkaouer, W., Kessentini, M., Shaout, A., Koligheu, P., Bechikh, S., Deb, K., Ouni, A.: Many-objective software remodularization using NSGA-III. ACM Trans. Softw. Eng. Methodol. **24**(3), 1–45 (2015)

37. Nadi, S., Berger, T., Kästner, C., Czarnecki, K.: Where do configuration constraints stem from? An extraction approach and an empirical study. IEEE Trans. Softw. Eng. **41**(8), 820–841 (2015)

38. Olaechea, R., Rayside, D., Guo, J., Czarnecki, K.: Comparison of exact and approximate multi-objective optimization for software product lines. In: Proceedings of 18th International Software Product Line Conference (SPLC), pp. 92–101 (2014)

39. Olaechea, R., Stewart, S., Czarnecki, K., Rayside, D.: Modeling and multi-objective optimization of quality attributes in variability-rich software. In: Proceedings of Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages (NFPinDSML), pp. 2:1–2:6 (2012)

40. Passos, L.T., Berger, T., Novakovic, M., Czarnecki, K., Xiong, Y., Wasowski, A.: A study of non-boolean constraints in variability models of an embedded operating system. In: Proceedings of Third Workshop on Feature-Oriented Software Development (FOSD), pp. 9–16 (2011)

41. Passos, L.T., Guo, J., Teixeira, L., Czarnecki, K., Wasowski, A., Borba, P.: Coevolution of variability models and related artifacts: a case study from the Linux kernel. In: Proceedings of 17th International Software Product Line Conference (SPLC), pp. 91–100 (2013)

42. Passos, L.T., Teixeira, L., Dintzner, N., Apel, S., Wasowski, A., Czarnecki, K., Borba, P., Guo, J.: Coevolution of variability models and related software artifacts—a fresh look at evolution patterns in the Linux kernel. Empir. Softw. Eng. **21**(4), 1744–1793 (2016)

43. Pohl, K., Bockle, G., van der Linden, F.: Software Product line Engineering: Foundations, Principles, and Techniques. Springer, Berlin (2005)

44. Rayside, D., Estler, H.C., Jackson, D.: A guided improvement algorithm for exact, general purpose, many-objective combinatorial optimization. Tech. rep., MIT-CSAIL-TR-2009-033 (2009)

45. Saadatpanah, P., Famelis, M., Gorzny, J., Robinson, N., Chechik, M., Salay, R.: Comparing the effectiveness of reasoning formalisms for partial models. In: Proceedings of the Workshop on Model-Driven Engineering, Verification and Validation (MoDeVVa), pp. 41–46 (2012)

46. Sarkar, A., Guo, J., Siegmund, N., Apel, S., Czarnecki, K.: Cost-efficient sampling for performance prediction of configurable systems. In: Proceedings of 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 342–352 (2015)

47. Sarro, F., Petrozziello, A., Harman, M.: Multi-objective software effort estimation. In: Proceedings of 38th International Conference on Software Engineering (ICSE), pp. 619–630 (2016)

48. Sayyad, A., Ingram, J., Menzies, T., Ammar, H.: Scalable product line configuration: A straw to break the camel's back. In: Proceedings of 28th International Conference on Automated Software Engineering (ASE), pp. 465–474 (2013)

49. Sayyad, A., Menzies, T., Ammar, H.: 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), pp. 492–501. IEEE (2013)

50. Sayyad, A.S., Ingram, J., Menzies, T., Ammar, H.: 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), pp. 22–27 (2013)

51. Siegmund, N., Kolesnikov, S., Kästner, C., Apel, S., Batory, D., Rosenmüller, M., Saake, G.: Predicting performance via automated feature-interaction detection. In: Proceedings of 34th International Conference on Software Engineering (ICSE), pp. 167–177 (2012)

52. Tsang, E.: Foundations of constraint satisfaction. Academic, (1993)

53. Vargha, A., Delaney, H.D.: A critique and improvement of the CL common language effect size statistics of McGraw and Wong. J. Educ. Behav. Stat. **25**(2), 101–132 (2000)

54. White, J., Doughtery, B., Schmidt, D.C.: 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), pp. 209–216 (2008)

55. Wu, Z., Tang, J., Kwong, C.K., Chan, C.Y.: An optimization model for reuse scenario selection considering reliability and cost in software product line development. Int. J. Inf. Technol. Decis. Mak. **10**(5), 811–841 (2011)

56. Yen, G., He, Z.: Performance metric ensemble for multiobjective evolutionary algorithms. IEEE Trans. Evol. Comput. **18**(1), 131–144 (2014)

57. Zhang, Y., Guo, J., Blais, E., Czarnecki, K.: Performance prediction of configurable software systems by Fourier learning. In: Proceedings of 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 365–373 (2015)

58. Zitzler, E., Künzli, S.: Indicator-based selection in multiobjective search. In: Proceedings of 8th International Conference on Parallel Problem Solving from Nature (PPSN), pp. 832–842 (2004)

59. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In: Proceedings of the Conference on Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems (EUROGEN), pp. 95–100 (2001)

60. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., da Fonseca, V.G.: Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans. Evol. Comput. **7**(2), 117–132 (2003)

Kai Shi is a Ph.D. candidate in the Department of Computer Science and Engineering, East China University of Science and Technology. His main research interests include multi-objective evolutionary algorithms and parallelization, constraint solving, software product lines, search-based software engineering, and cloud computing.

Jianmei Guo is an Associate Professor of Computer Science and Engineering at East China University of Science and Technology (ECUST). Before coming to ECUST, he was a Postdoctoral Fellow at the University of Waterloo (2012–2015), Canada. He received his Ph.D. in computer science from Shanghai Jiao Tong University in 2011. His research interests involve software engineering and artificial intelligence, focusing on a systematical methodology of hardware–software co-design to achieve automated, verifiable and intelligent product development, configuration and integration. He received an ACM SIGSOFT Distinguished Paper Award at ASE 2015, and two Best Paper Awards at Canadian AI 2017 and SPLC 2016.

Dingyu Yang received the B.E. and M.E. degrees from the Kunming University of Science and Technology, and the Ph.D. degree from the Shanghai Jiao Tong University, all in computer science. He is currently an assistant professor at the Shanghai Dian Ji University, Shanghai, China. His research interests include resource prediction and anomaly detection in cloud computing and big data.

Jia Hui Liang is a Ph.D. student in the Electrical and Computer Engineering department at the University of Waterloo, under the supervision of Vijay Ganesh and Krzysztof Czarnecki. His research involves constraint solving and their application to efficiently instantiate and multi-objectively optimize high-level Clafer models. Jia is the main contributor to Clafer's choco solver as well as a contributor to the underlying Choco4 CSP library. His work on improving SAT solving received two first prizes at the main and application tracks of the 2016 SAT Competition.

Jingsong Zhang received the Ph.D. degree in the Department of Computer Science and Engineering in 2015 from Shanghai Jiao Tong University, China. He is currently a Postdoctoral Fellow in Institute of Biochemistry and Cell Biology, Shanghai Institutes for Biological Sciences, Chinese Academy of Sciences, China. His research interests include sequence analysis, bioinformatics, sequential pattern mining, data mining. He has been a reviewer for some academic journals and conferences.

Krzysztof Czarnecki is a Professor of Electrical and Computer Engineering at the University of Waterloo. Before coming to Waterloo, he was a researcher at DaimlerChrysler Research (1995–2002), Germany, focusing on improving software development practices and technologies in enterprise, automotive, and aerospace domains. He co-authored the book on "Generative Programming" (Addison-Wesley, 2000), which deals with automating software component assembly based on domain-specific languages. While at Waterloo, he held the NSERC/Bank of Nova Scotia Industrial Research Chair in Requirements Engineering of Service-oriented Software Systems (2008–2013) and has worked on a range of topics in model-driven systems and software engineering, including product lines engineering, design synthesis, variability modeling, model transformation, and domain-specific languages. He received the Premier's Research Excellence Award in 2004 and the British Computing Society in Upper Canada Award for Outstanding Contributions to IT Industry in 2008.

Huiqun Yu received his B.S. degree from Nanjing University in 1989, M.S. degree from East China University of Science and Technology (ECUST) in 1992, and Ph.D. degree from Shanghai Jiao Tong University in 1995, all in computer science. He is currently a Professor of computer science with the Department of Computer Science and Engineering at ECUST. From 2001 to 2004, he was a Visiting Researcher in the School of Computer Science at Florida International University. His research interests include software engineering, high confidence computing systems, cloud computing and formal methods. He is a senior member of the IEEE, a member of the ACM, and a senior member of the China Computer Federation.

Vijay Ganesh is an assistant professor at the University of Waterloo since 2012. Prior to that he was a research scientist at MIT and completed his PhD in computer science from Stanford University in 2007. Vijay's primary area of research is the theory and practice of automated reasoning aimed at software engineering, formal methods, security, and mathematics. In this context he has led the development of many SAT/SMT solvers, most notably, STP, The Z3 string solver, MapleSAT, and MathCheck. He has also proved several decidability and complexity results relating to the SATisfiability problem for various mathematical theories. For his research, he has won over 21 awards, honors, and medals including an ACM Test of Time Award at CCS 2016, two Google Faculty Research Awards in 2011 and 2013, and a Ten-Year Most Influential Paper Award at DATE 2008.