Configuring Software Product Lines by Combining Many-Objective Optimization and SAT Solvers

YI XIANG, YUREN ZHOU, and ZIBIN ZHENG, Sun Yat-sen University, China
MIQING LI, University of Birmingham, UK

A feature model (FM) is a compact representation of the information of all possible products from software product lines. The optimal feature selection involves the simultaneous optimization of multiple (usually more than three) objectives in a large and highly constrained search space. By combining our previous work on many-objective evolutionary algorithm (i.e., VaEA) with two different satisfiability (SAT) solvers, this article proposes a new approach named *SATVaEA* for handling the optimal feature selection problem. In SATVaEA, an FM is simplified with the number of both features and constraints being reduced greatly. We enhance the search of VaEA by using two SAT solvers: one is a stochastic local search–based SAT solver that can quickly repair infeasible configurations, whereas the other is a conflict-driven clause-learning SAT solver that is introduced to generate diversified products. We evaluate SATVaEA on 21 FMs with up to 62,482 features, including two models with realistic values for feature attributes. The experimental results are promising, with SATVaEA returning 100% valid products on almost all FMs. For models with more than 10,000 features, the search in SATVaEA takes only a few minutes. Concerning both effectiveness and efficiency, SATVaEA significantly outperforms other state-of-the-art algorithms.

CCS Concepts: • **Software and its engineering → Search-based software engineering**; • **Mathematics of computing → Optimization with randomized search heuristics**;

Additional Key Words and Phrases: Optimal feature selection, many-objective optimization, satisfiability (SAT) solvers, vector angle–based evolutionary algorithm (VaEA)

ACM Reference format:
Yi Xiang, Yuren Zhou, Zibin Zheng, and Miqing Li. 2018. Configuring Software Product Lines by Combining Many-Objective Optimization and SAT Solvers. *ACM Trans. Softw. Eng. Methodol.* 26, 4, Article 14 (February 2018), 46 pages.
https://doi.org/10.1145/3176644

This work was supported by the National Basic Research Program of China (973 Project No. 2014CB347701); the National Natural Science Foundation of China under grants 61773410, 61673403, and 61472143; the Scientific Research Special Plan of Guangzhou Science and Technology Programme under grant 201607010045; the Program for Guangdong Introducing Innovative and Enterpreneurial Teams under grant 2016ZT06D211; and the Excellent Graduate Student Innovation Program from the Collaborative Innovation Center of High Performance Computing (Recipient: Y. Xiang).
Authors' addresses: Y. Xiang, Y. Zhou (co-corresponding author), and Z. Zheng (co-corresponding author), No. 132, East of Outer Ring Road, Guangzhou Higher Education Mega Center, Panyu District, Guangzhou, 510006, Guangdong Province, P.R. China; emails: gzhuxiang_yi@163.com; {zhouyuren, zhzibin}@mail.sysu.edu.cn; M. Li, School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK; email: limitsing@gmail.com.

https://doi.org/10.1145/3176644

1 INTRODUCTION

A software product line (SPL) [12] defines a family of software products that are systematically configured using a set of reusable modular software components. Generally, they share some common functionalities, but each of them differs in some specific features that are related to some particular aspects of the system functionality. The set of products within an SPL is compactly described by a feature model (FM) [4] that defines the constraints among features and so specifies which combinations of features are valid[1] [35]. The SPLs are used by software engineers to reduce software cost, increase software reusability, and facilitate software maintenance [32, 43].

Product configuration for an SPL is an optimal feature selection problem, which needs to find a set of features from an FM. However, without automated support, it is difficult for the feature selection process to achieve the optimum. It requires simultaneously satisfying multiple objectives, such as matching users' preferences, maximizing the number of selected features, minimizing product cost, and satisfying all constraints defined by the FM, which may contain thousands of features and constraints. The determination of "optimal" products in such large and constrained spaces is beyond human intuition, and therefore automated techniques for feature selection are needed to reduce configuration efforts.

Studies on the many-objective[2] optimal feature selection problem are probably due to the (potential) existence of these types of problems in practice. Sayyad et al. [75] first decided to consider five objectives in the study of finding optimal products from SPLs. Then, they replicated their empirical study [72] and reported the efforts of parameter tuning [73]. Subsequently, they enhanced the performance of evolutionary many-objective optimization (EMO) algorithms on large-scale FMs [74] to find more valid configurations.[3] Later, the studies on the many-objective feature selection problem were continued by Henard et al. [32], Tan et al. [82], Xue et al. [94], and several others [31, 42, 54, 63, 71]. For a detailed literature review, please refer to related works in Section 6. These works consider the artificial many-objective feature selection problem where the attribute values for calculating optimization objectives, as first suggested by Sayyad et al. [75], are generated arbitrarily according to uniform distributions. To the best of our knowledge, many-objective optimal feature selection for industrial SPLs has not yet been directly reported (probably due to confidentiality issues in the industry [15]). Recently, however, Hierons et al. [35] used realistic values for the attributes of the Amazon [26] and Drupal [69] FMs. For the Drupal model, the real attributes and values are obtained by using repository mining [69], whereas realistic attribute values for the Amazon model are randomly generated according to the constraints on the values for each real attribute name [26]. Based on these real attributes, the configuration of the preceding two SPLs leads to two more practical many-objective optimal feature selection problems where eight objectives are handled simultaneously [35].

For the many-objective optimal feature selection problem, pure EMO algorithms (e.g., NSGA-II, SPEA2, IBEA, and SPEA2+SDE [49]) are found to be insufficient in obtaining a set of valid products for large-scale SPLs, leading to the enhancements of them by adding a valid product as a seed in the initial population [74], or introducing smart satisfiability (SAT) solver–based mutation and replacement operators [32], or providing a novel encoding (that shrinks the representation of the

[1]A valid product refers to the configuration that satisfies all of the constraints in the FM. Usually, the software engineer is only interested in valid products. Since *valid* and *feasible* have the same meaning, the two words are used interchangeably in the article; the same applies for the words *invalid* and *infeasible*.
[2]The term *many-objective* is used here because it refers in particular to four or more objectives in the evolutionary computation community, whereas *multiobjective* means only two or three objectives. It is widely accepted that "many-objective" optimization is much more difficult than "multiobjective" optimization.
[3]A configuration of an FM corresponds to a product within an SPL, and therefore the terms *configuration* and *product* are used interchangeably.

problem) accompanied by the $1 + n$ approach (which optimizes first on the number of violated constraints and only then on the other objectives) [35]. However, these enhancements have their own disadvantages:

(1) It takes a long time to obtain a "feature-rich" product as a seed. Sayyad et al. [74] used a two-objective optimization with IBEA to get a feature-rich seed. However, this technique can be time consuming for large FMs. For the 2.6.28.6-icse11 model with 6,888 features, it takes nearly 3 hours to search for a seed.
(2) Since SATIBEA [32] treats all objectives (including the number of violated constraints as the first optimization objective) equally, invalid products may survive in the final population if they perform particularly well in other objectives. In addition, the smart SAT solver–based mutation and replacement operators are more time consuming than traditional ones and may slow down the whole search process [35]. As a result, SATIBEA may struggle to find a large ratio of valid products in the final population.
(3) Although the novel encoding in Hierons et al. [35] can shrink the representation, it needs more effort to first encode the FMs and then decode the representations. Compared to the direct encoding method where each feature is directly represented by a Boolean variable with *true* being selected and *false* deselected, the presented novel encoding is complicated because it should be made to cater to different FMs with usually quite different structures in terms of both structural and cross-tree constraints (CTCs) [35]. The proposed ShrInk Prioritise (SIP) method in Hierons et al. [35] has not yet been tested on large-scale real-world FMs. Although the largest FM is with 10,000 features, it is a randomly generated one without representing a real system.

From the perspective of software engineers and end users, prior works (e.g., SATIBEA [32] and SIP-based algorithms [35]) may be ineffective in maintaining the "diversity" among valid solutions in some cases. As will be shown in Section 5.8, SATIBEA can obtain only one valid solution for the real Amazon FM for which eight optimization objectives are used. Although SIP-based algorithms, such as SIP+SPEA2+SDE and SIP+NSGA-II [35], have the ability to find a large proportion of valid products, these products are quite similar to each other. Inspired by the well-established works of Henard et al. [32], Hierons et al. [35], and Sayyad et al. [75], this article proposes a new method called *SATVaEA* for the same optimal feature selection problem by combining our previously developed many-objective optimization framework named *VaEA* [92] and SAT solvers. In the proposed method, the optimal feature selection is converted into a many-objective optimization problem (MaOP) with constraints, and is solved based on the main framework of VaEA. The new proposal uses two SAT solvers for different purposes: one is a stochastic local search (SLS)-style SAT solver (i.e., the fast WalkSAT [10]) that aims to quickly repair an infeasible configuration, and the other is a DPLL/CDCL[4]-style SAT solver [16] (i.e., Sat4j [8]) that is used for diversity promotion (DP) [32]. The frequency of using the two solvers is controlled by a parameter named θ. We first evaluate the proposed SATVaEA method on 14 real-world FMs, ranging from 544 to 62,482 features with respect to seven performance metrics. Then, the proposed SATVaEA is compared to SATIBEA [32] and two SIP-based algorithms on seven FMs taken from Hierons et al. [35], including two FMs with realistic values for feature attributes. Experimental results have shown the effectiveness and efficiency of the proposed method in finding a large number of diversified valid products in a short time. In particular, SATVaEA finds 100% valid solutions for almost all of the

[4]DPLL and CDCL are short for Davis-Putnam-Logemann-Loveland procedure [16] and the conflict-driven clause-learning algorithm [57], respectively.

FMs considered in less than 10 minutes even for FMs with 60,000+ features. For small-scale FMs, it takes only a few seconds.

The main contributions of this article are summarized as follows:

(1) *An effective FM simplification method.* As in Sayyad et al. [74], Henard et al. [32], and Hierons et al. [35], mandatory features (features that must appear in any product) and dead features (features that cannot present in any product) are fixed in the representations to reduce the search space. On top of this, the article first proposes to simplify constraints of a given FM via the Boolean constraint propagation (BCP) procedure [98]. To the best of our knowledge, the idea of simplifying constraints via BCP has not been done in prior works in the area of many-objective optimal feature selection (e.g., Sayyad et al. [74], Henard et al. [32], Hierons et al. [35], and Tan et al. [82]). According to our results, the number of features (after removing fixed ones) and the number of constraints (after applying the BCP procedure) are decreased by 67.2% and 75.4% in the best cases, respectively.

(2) *The first use of both SLS-style and DPLL/CDCL-style SAT solvers for the many-objective optimal feature selection problem.* On one hand, SLS-style SAT solvers are computationally efficient when repairing an invalid configuration. On the other hand, DPLL/CDCL-style SAT solvers can find valid solutions (if any) for a given FM to make final solutions diversified. The probability of using both solvers is controlled by a parameter θ. A larger θ means more invocations to the SLS-style SAT solver and hence less invocations to the other one. Experimental results demonstrate that these techniques are useful.

(3) *A two-level ranking method used to distinguish solutions in the environmental selection.* The first-level ranking is based on the number of violated constraints (i.e., solutions with the same number of violated constraints are put into the same layer). The second-level ranking is actually the nondominated sorting procedure [20] that divides solutions in the same layer into different sublayers according to Pareto dominance. The two-level ranking method emphasizes individuals with less violated constraints and individuals performing well concerning Pareto dominance.

(4) *A comprehensive experimental study that evaluates SATVaEA on 21 FMs with respect to seven performance metrics.* To the best of our knowledge, the largest real-world FM under wide investigation is the Linux kernel model (2.6.28.6-icse11) with 6,888 features (e.g., [32, 74, 82, 94]). In some other works, such as Hierons et al. [35], Mendonca et al. [59], and Guo et al. [29], the largest FM has 10,000 features. However, all of these FMs are generated randomly without representing a real-world system. In this work, we adopt extremely large real-world FMs, with even 62,482 features, from the Linux Variability Analysis Tools (LVAT) repository.[5]

The remainder of this article is organized as follows. In Section 2, we give a brief introduction to FMs and many-objective optimization. Section 3 describes the SATVaEA method, and Section 4 outlines the experimental setup. Section 5 gives the results and analyses of the experiments. In Section 6, related works on the feature selection problem are reviewed. Finally, Section 7 concludes the work and gives possible directions for future studies.

[5]Note that Liang et al. [53] also used the LVAT repository, including the same largest FM as in our work; however, they used these FMs to demonstrate feasibilities of SAT solvers in analyzing large real-world FMs rather than configure SPLs by solving the many-objective optimal feature selection problem.

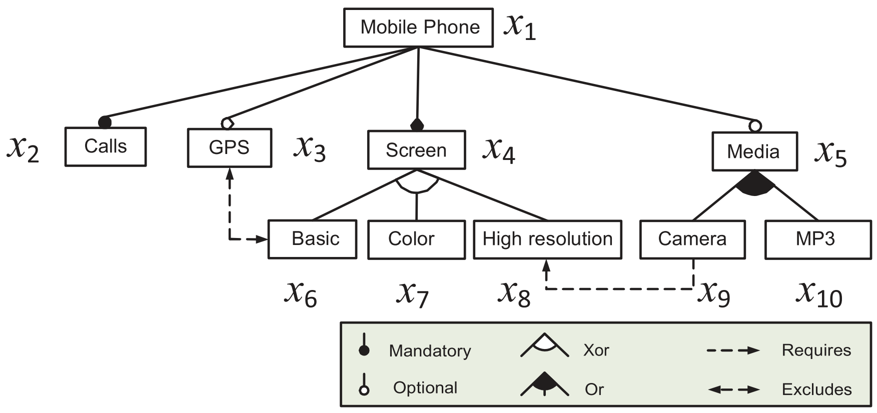


Fig. 1. A simple FM for a mobile phone SPL, which is adapted from Benavides et al. [5].

2 BACKGROUNDS

In this section, we provide necessary backgrounds regarding FMs (Section 2.1) and EMO approaches (Section 2.2).

2.1 Feature Models

An FM compactly represents the information of all possible products of an SPL in terms of features and relationships among them [40]. Visually, an FM is represented as a tree-like structure where each node represents a feature, composed by

(1) Parental relationships—relationships between a parent feature and its child features (or subfeatures)
(2) CTCs—inclusion or exclusion statements in the following form: if feature F is included, then features A or B must also be included (or excluded) [74].

Figure 1, adapted from Benavides et al. [5], illustrates a simplified FM from the mobile phone industry. It contains 10 features, and the root feature ('Mobile Phone') identifies the SPL. Some features are *mandatory* (i.e., included in any valid product), such as the 'Calls' feature, whereas some are *optional* (e.g., the 'GPS' feature that can be optionally included in products that contain its parent feature, i.e., the 'Mobile Phone' feature). The relation among a group of child features is called *xor* if exactly one feature can be selected when its parent feature is a part of the product. For example, the mobile phone must provide either a 'Basic,' or a 'Color,' or a 'High resolution' 'Screen,' but not all or any two of them in a same configuration. A set of child features are said to have an *or*-relation with their parent when at least one of them can be included in the products where their parent feature appears. For example, a 'Camera,' an 'MP3,' or both of them can be selected when the parent feature 'Media' appears in a same product. For cross-tree relationships, some features are constrained to co-occur. For example, the 'Camera' feature *requires* the 'High resolution' feature, whereas other features *exclude* each other (i.e., they cannot exist simultaneously in a same product). In Figure 1, the 'GPS' excludes the 'Basic' feature.

An FM can be translated to a propositional formula and then converted into an equivalent formula that is in conjunctive normal form (CNF). In this form, an FM is expressed as a conjunction of M clauses, $C_1, C_2, \ldots, C_M$, where a clause is a disjunction of several literals, each of which is a feature that is selected (x_j) or not ($\neg x_j$). For example, the FM in Figure 1 can be expressed in the following propositional formula:

$$\begin{aligned} \text{FM} = {} & \text{Mobile Phone } (x_1) \\ & \wedge \{\text{Mobile Phone } (x_1) \leftrightarrow \text{Calls } (x_2)\} \\ & \wedge \{\text{Mobile Phone } (x_1) \leftrightarrow \text{Screen } (x_4)\} \\ & \wedge \{\text{GPS } (x_3) \rightarrow \text{Mobile Phone } (x_1)\} \\ & \wedge \{\text{Media } (x_5) \rightarrow \text{Mobile Phone } (x_1)\} \\ & \wedge \{\text{Screen } (x_4) \leftrightarrow \mathit{xor}\, \{\text{Basic } (x_6), \text{Color } (x_7), \text{High resolution } (x_8)\}\} \\ & \wedge \{\text{Media } (x_5) \leftrightarrow \text{Camera } (x_9) \vee \text{MP3 } (x_{10})\} \\ & \wedge \{\text{Camera } (x_9) \rightarrow \text{High resolution } (x_8)\} \\ & \wedge \neg\{\text{GPS } (x_3) \wedge \text{Basic } (x_6)\} \end{aligned}$$

Then it can be written in the following CNF: $FM = x_1 \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_4) \wedge (x_1 \vee \neg x_4) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge (x_4 \vee \neg x_7) \wedge (x_4 \vee \neg x_8) \wedge (\neg x_4 \vee x_6 \vee x_7 \vee x_8) \wedge (\neg x_6 \vee \neg x_7) \wedge (\neg x_6 \vee \neg x_8) \wedge (\neg x_7 \vee \neg x_8) \wedge (\neg x_6 \vee \neg x_7 \vee \neg x_8) \wedge (x_5 \vee \neg x_9) \wedge (x_5 \vee \neg x_{10}) \wedge (\neg x_5 \vee x_9 \vee x_{10}) \wedge (x_8 \vee \neg x_9) \wedge (\neg x_3 \vee \neg x_6)$.

Most FMs used in this study are obtained from the LVAT repository.[6] These models have the DIMACS format, which expresses each model as a formula in CNF. Hence, SAT solvers can be directly applied to these models.

2.2 Evolutionary Many-Objective Optimization

Many real-world problems involve simultaneous optimization of several conflicting objectives. This type of problem is formally known as multi-objective optimization problems (MOPs) [18]. A key concept in MOPs is Pareto dominance. A vector $\mathbf{u} = (u_1, u_2, \ldots, u_m)$ is said to Pareto dominate another vector $\mathbf{v} = (v_1, v_2, \ldots, v_m)$ if and only if $u_i \leq v_i$ for all $i \in \{1, 2, \ldots, m\}$ and there exits at least one $j \in \{1, 2, \ldots, m\}$ such that $u_j < v_j$. Often, there is no single optimal solution available for MOPs, but a set of nondominated solutions known as a Pareto front (PF) [18, 101]. MOPs with four or more objectives are referred to as MaOPs [24, 46].

Since the goal of solving MOPs is to find a set of nondominated solutions, the Pareto dominance relation is naturally used as the selection criterion. Pareto dominance–based multiobjective evolutionary algorithms (MOEAs), such as PAES [44], NSGA-II [20], SPEA2 [103], MOPSO [14], and eMOABC [93], were widely used to handle MOPs with two or three objectives. However, they suffer from the insufficient selection problem as the number of objectives increases [37, 83]. Recently, non-Pareto dominance–based algorithms (which do not use Pareto dominance as the main selection criterion, such as MOEA/D [99] and HypE [1]), and improved Pareto dominance–based algorithms (e.g., NSGA-III [19] and GrEA [95]) have been shown to be promising in dealing with MaOPs.

In one of our previous studies, we proposed a vector angle–based evolutionary algorithm (VaEA) [92] for handling MaOPs. VaEA shares a common framework that was widely employed by other evolutionary algorithms. First, a population is initialized randomly in the decision space, then mating selection is applied to choose parent individuals. Next, offspring solutions are obtained by using crossover and mutation operators. Finally, through the environmental selection procedure, the population for the next generation is constructed by solutions chosen from the union set of both parent and offspring populations. The preceding steps are repeated until a termination condition is satisfied.

What makes VaEA different from other many-objective optimizers is the environmental selection where two principles are adopted to promote convergence and diversity. One is the *maximum-vector-angle-first* principle that is used to maintain the distribution of solutions, whereas the other is the *worse-elimination* principle that allows worse solutions in terms of the convergence

[6]LVAT is available at http://code.google.com/p/linux-variability-analysis-tools.

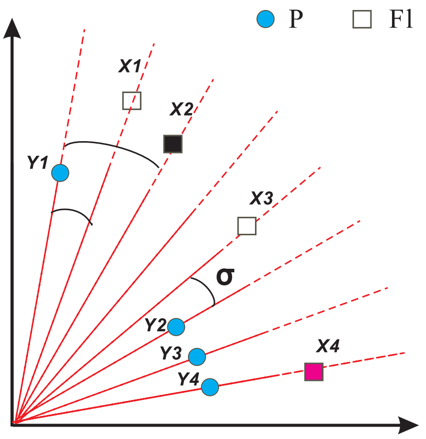


Fig. 2. Illustration of the maximum-vector-angle-first principle in VaEA. This figure is borrowed from Xiang et al. [92].

(measured by the sum of normalized objectives) to be conditionally replaced by other individuals. The environmental selection first divides solutions into different layers by using the nondominated sorting procedure [20]. Then, solutions in the first layer are added first, solutions in the second layer are added second, and so on. This procedure is continued until the population is full, or not full but cannot accommodate solutions in the next layer, which is often called the *critical layer*, denoted as F_l. Solutions in F_l are then added one by one according to the maximum-vector-angle-first principle. Figure 2 gives an illustration on how this principle works. Suppose that $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3$, and $\mathbf{y}_4$ are solutions already added into the current population P, and $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$, and $\mathbf{x}_4$ are individuals in F_l. According to Xiang et al. [92], each solution $\mathbf{x}$ in F_l has an angle to the current population, which is defined as the angle between $\mathbf{x}$ and its target solution—that is, the member in P to which $\mathbf{x}$ has the minimum angle. For example, the angle from $\mathbf{x}_3$ to the population P is σ, whereas the angle from $\mathbf{x}_2$ to P is 2σ. Since $\mathbf{x}_2$ has the maximum angle (2σ) to P, according to the maximum-vector-angle-first principle, $\mathbf{x}_2$ will be added. Clearly, the improvement of the diversity of P by adding $\mathbf{x}_2$ is more obvious than by adding other solutions (e.g., $\mathbf{x}_3$ and $\mathbf{x}_4$). After $\mathbf{x}_2$ is added, the region around it can be exploited and better solutions along the direction of $\mathbf{x}_2$ may be attainable. Now suppose that two more solutions need to be added; the choice will be $\mathbf{x}_1$ and $\mathbf{x}_3$. However, $\mathbf{x}_4$ will never be chosen because it has a zero angle to P. This is interpretable because a better solution $\mathbf{y}_4$ in the same direction as $\mathbf{x}_4$ has already been included in the population P. For the worse-elimination principle, it first identifies the individual in F_l that has the minimum angle to P. Then, the target solution of this individual may be replaced if it performs worse in terms of the convergence. The basic idea behind this principle is that it is often unnecessary to keep multiple solutions along a same (or similar) search direction.

In this article, we consider using VaEA as the main search framework because it was shown to be very effective when handling MaOPs, and it has the following good properties [92]: (1) it is free from a set of supplied reference points or weight vectors, (2) it has less algorithmic parameters to be tuned, and (3) the time complexity of VaEA is low.

3 THE PROPOSED SATVAEA

The framework of our proposed method is shown in Algorithm 1. The basic idea of SATVaEA is to combine many-objective optimization with SAT solvers. The SLS-style SAT solver aims at quickly repairing invalid solutions, whereas the DPLL/CDCL-style SAT solver is introduced to promote the diversity among solutions. A parameter $\theta \in [0, 1]$ is adopted to control the frequency of using the two styles of solvers (lines 7 through 11 in Algorithm 1). In line 7 of Algorithm 1, *rnd* is a random number between 0 and 1. Clearly, a larger θ means more invocations to the SLS-style SAT solver and hence less invocations to the other one. Generally, the DPLL/CDCL-style SAT solver is more time consuming than an SLS-style SAT solver. Therefore, if the allowed running time is highly limited, more computational resources should be given to the SLS-style SAT solver to produce more valid solutions quickly. In other words, SATVaEA prefers a large θ, which will be experimentally demonstrated in Section 5.5.

In SATVaEA, the main algorithm components include the simplification of the FM (line 1), population initialization and genetic operators (lines 2 and 4), two different SAT solvers (lines 8 and 10), and the environmental selection (line 13). In the following sections, the preceding main components will be described in detail.

ALGORITHM 1: Framework of the Proposed SATVaEA

```
Input: FM (a feature model to be configured), $N$ (population size), and $\theta$ (a parameter for controlling the
    frequency of using the two SAT solvers)
Output: Nondominated valid solutions in the final population.
 1: Simplify the FM
 2: Initialize population $P$ with $N$ random solutions
 3: while the termination condition is not fulfilled do
 4:     Generate an offspring population $Q$ by genetic operators
 5:     $S \leftarrow P \cup Q$ // $S$ is the union of $P$ and $Q$
 6:     if there exist invalid solutions in $S$ then
 7:         if $rnd < \theta$ then
 8:             Apply the SLS-style SAT solver to repair a random invalid solution
 9:         else
10:             Apply the DPLL/CDCL-style SAT solver to generate a valid solution, and use it to replace an
                invalid one
11:         end if
12:     end if
13:     Perform the environmental selection to $S$ to select $N$ diversified solutions for the new population $P$
14: end while
15: return Nondominated valid solutions in $P$
```

3.1 Simplification Method Used

Since an FM can be expressed in a CNF propositional formula, the simplification method BCP [98] is used to preprocess FMs in this work. As a standard step in DPLL/CDCL-style SAT solvers, the BCP is applied after each branching step (and also during preprocessing) and is used for identifying variables that must be assigned a specific Boolean value. However, BCP is not commonly used inside SLS-style SAT solvers. In addition, to the best of our knowledge, the idea of simplifying constraints via BCP has not been done in prior works in this area of many-objective optimal feature selection (e.g., [32, 35, 74, 82]). What prior works mainly focused on was the simplification of the representation of FMs. By preprocessing CNF formulas via BCP, the work intends to simplify both representation and constraints of an FM, which prepares for the subsequent applications of both styles of SAT solvers.

In the CNF formulas representing FMs, some disjunctions (or constraints) contain only one feature (or variable, denoted by x), which should be either a *positive* literal (x) or a *negative* literal ($\neg x$). If it is a positive one, meaning that the feature must always be selected, the variable x is then fixed and assigned to *true*, and all disjunctions that contain x can be eliminated because they are already satisfied. Since the satisfiability of constraints containing $\neg x$ is equal to that of the counterpart without $\neg x$, the literal $\neg x$ can be removed from those constraints. Otherwise, the variable x is assigned to *false*, and all disjunctions that contain $\neg x$ are eliminated. For disjunctions having the literal x, it is unnecessary to keep this literal because the false assignment has no impact on the satisfiability of these disjunctions. Note that we repeat the preceding procedures to ensure that there are no constraints with only single literals.

A fixed feature is either *mandatory* or *dead*. The truth assignment for the variable of this feature is fixed—that is, *true* for a mandatory feature and *false* for a dead feature. Thus, the number of features that allow flipping declines after excluding mandatory and dead features. According to the preceding procedure, the simplification method may also reduce the number of literals in each constraint, as well as the total number of constraints. Thus, it will be time saving when calculating the number of violated constraints for a given solution. As will be shown in Section 5.1, the preceding simplification method indeed reduces the complexity of a given FM.

Finally, we give an illustrative example to show how this simplification works. As shown in Section 2.1, the FM in Figure 1 can be represented in the following CNF formula: $FM = x_1 \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_4) \wedge (x_1 \vee \neg x_4) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge (x_4 \vee \neg x_7) \wedge (x_4 \vee \neg x_8) \wedge (\neg x_4 \vee x_6 \vee x_7 \vee x_8) \wedge (\neg x_6 \vee \neg x_7) \wedge (\neg x_6 \vee \neg x_8) \wedge (\neg x_7 \vee \neg x_8) \wedge (\neg x_6 \vee \neg x_7 \vee \neg x_8) \wedge (x_5 \vee \neg x_9) \wedge (x_5 \vee \neg x_{10}) \wedge (\neg x_5 \vee x_9 \vee x_{10}) \wedge (x_8 \vee \neg x_9) \wedge (\neg x_3 \vee \neg x_6)$. Since x_1 is the root, it must be selected (i.e., $x_1 \leftarrow true$). Because x_1 is satisfied, all constraints containing x_1 are satisfied as well. Thus, constraints $x_1 \vee \neg x_2$, $x_1 \vee \neg x_4$, $x_1 \vee \neg x_3$ and $x_1 \vee \neg x_5$ are removed, whereas constraints containing $\neg x_1$ can be simplified by removing $\neg x_1$. After the first round of simplifications, the FM is in the following form: $FM = x_2 \wedge x_4 \wedge (x_4 \vee \neg x_6) \wedge (x_4 \vee \neg x_7) \wedge (x_4 \vee \neg x_8) \wedge (\neg x_4 \vee x_6 \vee x_7 \vee x_8) \wedge (\neg x_6 \vee \neg x_7) \wedge (\neg x_6 \vee \neg x_8) \wedge (\neg x_7 \vee \neg x_8) \wedge (\neg x_6 \vee \neg x_7 \vee \neg x_8) \wedge (x_5 \vee \neg x_9) \wedge (x_5 \vee \neg x_{10}) \wedge (\neg x_5 \vee x_9 \vee x_{10}) \wedge (x_8 \vee \neg x_9) \wedge (\neg x_3 \vee \neg x_6)$. Since x_2 and x_4 are now positive literal, they must be selected. Similar procedures as in the first round of simplifications are repeated until there is no single literal. The final form of the original model will be as follows: $FM = (x_6 \vee x_7 \vee x_8) \wedge (\neg x_6 \vee \neg x_7) \wedge (\neg x_6 \vee \neg x_8) \wedge (\neg x_7 \vee \neg x_8) \wedge (\neg x_6 \vee \neg x_7 \vee \neg x_8) \wedge (x_5 \vee \neg x_9) \wedge (x_5 \vee \neg x_{10}) \wedge (\neg x_5 \vee x_9 \vee x_{10}) \wedge (x_8 \vee \neg x_9) \wedge (\neg x_3 \vee \neg x_6)$, where the number of constraints is only half of that in the original formula.

3.2 Population Initialization and Genetic Operations

Once features have been detected to be either mandatory or dead in the simplification procedure, we fix them in the initial population. For all other features, they are subject to random configurations. Specifically, each feature is either selected or deselected with an equal probability of 0.5. In this way, N random solutions are generated in the initial population (line 2 in Algorithm 1).

The population is then updated by using genetic operators. First, two parents are selected based on a binary tournament selection method, where the individual with a smaller number of violated constraints is preferred. In the case that the two individuals have the same number of violated constraints, one of them will be chosen randomly. Second, two offspring are produced by applying the single-point crossover operator [81, 97] to the parents, which exchanges the bits of the first parent, from the beginning to the crossover point, with those of the second one. Third, with the bit-flip mutation [11, 81], bits for each of the offspring solutions are flipped with a specific probability. It should be mentioned that we restrict the bit-flip mutation only on features that are not fixed, because flipping any fixed feature will result in invalid configurations.

The preceding procedures are repeated until N offspring are obtained. The parent and offspring populations are then combined into a union population S that contains $2N$ individuals. The union population will be updated by SAT solvers (in Sections 3.3 and 3.4), and then the environmental selection (see Section 3.5) will be performed to choose N diversified individuals. The new population is evolved continually until the termination condition is fulfilled. Note that both the maximum running time [32] and a predefined number of evaluations [35] can be used as a termination condition.

3.3 Fast Local Search With an SLS-Style SAT Solver

As shown in line 8 of Algorithm 1, an SLS-style SAT solver is introduced to repair a random invalid solution in the union population S. Actually, this repair is implemented by conducting a fast local search to this invalid solution. As one of the most famous local search algorithms, WalkSAT [76] was shown to be very efficient in solving large-scale SAT instances. Although WalkSAT has low time complexity at each step, it is still possible to further improve its efficiency. Recently, Cai [10] proposed a more efficient implementation of WalkSAT, which is called the *fast WalkSAT* in this work. The algorithm mainly utilizes the property *break* to pick a variable to flip from a falsified clause. For a variable x, its break value denoted by $break(x)$ is the number of satisfied clauses that would become falsified by flipping x. When picking variables, the variable with a small *break* value is preferred. Ideally, this value can be zero.

In each step, fast WalkSAT works as follows. First, a falsified clause C is selected at random. If there exist variables in clause C whose break value is 0, then one of such variables is flipped, with ties broken randomly. Otherwise, with a certain probability p (the noise parameter), a random variable in C is selected and flipped; in the remaining cases (i.e., with a probability $1 - p$), a random variable with the minimum *break* value from C is selected and flipped.

One important issue surrounding fast WalkSAT is the efficiency of computing the *break* value of variables. Before describing the method for computing *break* values, we give the following concepts and data structures:

- A *true literal*. Given an assignment π, if a literal is evaluated to be true under π, then it is a true literal. For example, x is a *true literal* if x is assigned to *true*; $\neg x$ is a *true literal* if x is assigned to *false*.
- A *false literal*. Given an assignment π, if a literal is evaluated to be false under π, then it is a false literal. For example, x is a *false literal* if x is assigned to *false*.
- *TrueLitCount*. This stores the number of true literals for all clauses. For example, $TrueLitCount(0) = 1$ means that the first clause (if the index starts from 0) has only 1 true literal.
- $PosLitClause(x)$ *for each variable* x. This stores the index numbers of clauses where the *positive* literal x appears. For example, $PosLitClause(x) = \{1, 2, 6\}$ means that the positive literal x appears (only) in clauses whose indexes are 1, 2, and 6.
- $NegLitClause(x)$ *for each variable* x. This stores the index numbers of clauses where the *negative* literal $\neg x$ appears.

Fast WalkSAT relies on the following observation: *For a variable* x*, a clause contributes (one) to* $break(x)$ *only when the clause has only one true literal, which is a true literal of* x [10]. To compute $break(x)$, we only need to scan one of the two index arrays for the variable x (i.e., $PosLitClause(x)$ or $NegLitClause(x)$). For a variable x, its break value $break(x)$ is initialized as 0. If the truth value of x is true under the current assignment, then we scan the array $PosLitClause(x)$. For each $c \in PosLitClause(x)$, if $TrueLitCount(c) = 1$, which means that the clause c contains exactly one true literal (and literal x is that true literal), then flipping the value of x would make clause c become falsified from satisfied, and thus $break(x)$ should be increased by one. If the truth value of x is false under the current assignment, then we scan the array $NegLitClause(x)$ and compute $break(x)$

similarly. According to Cai [10], the preceding computing procedure is very efficient, and fast WalkSAT was demonstrated to be significantly faster than other related state-of-the-art solvers.

In this article, fast WalkSAT is imported to quickly repair an invalid solution (if any). Specifically, an invalid solution (if any) is randomly selected from the current union population S. Starting from this solution (or assignment), fast WalkSAT is performed to search for a valid one. The search procedure is terminated if a valid product is found, or not found but the number of flips reaches a predefined value. The original solution will be replaced by the returned individual, which is either valid or invalid but has less constraint violations than the old solution. The primary aim of fast WalkSAT is speeding up the convergence of SATVaEA to a large amount of valid solutions. As will be demonstrated in Section 5.3, this technique indeed improves the convergence of the proposed SATVaEA.

3.4 DP With a DPLL/CDCL-Style SAT Solver

A set of diversified valid solutions is of importance to meet various preferences from both software engineers and end users. To improve the diversity among solutions, a DPLL/CDCL-style SAT solver [32] is used to find dissimilar products by randomly permuting control parameters of the SAT solver. Specifically, there are three different SAT parameters that need to be permuted [32]:

- *Constraint order*. This is the order in which the constraints are considered.
- *Literal order*. This is the order in which the literals of each constraint are considered.
- *Phase selection*. This is the order {*true*; *false*} in which assignments to variables are instantiated.

In the article, the SAT solver we choose is the well-known Sat4j [8], which is a Java library for solving Boolean satisfaction and optimization problems. In fact, Sat4j started in 2004 as an implementation in Java of the Minisat SAT solver [22]. Over the years, Sat4j has been used by numerous research groups, especially in software engineering. As will be shown in Table 16 in Section 6, Sat4j was widely used in the software product line engineering (SPLE) domain (e.g., [32, 33, 34, 39, 53, 59, 82, 94]). Given the popularity of Sat4j, we also choose it in our algorithm.

The Sat4j solver (with permuted parameters) is invoked to search for a valid solution, which is used to randomly substitute an invalid one (if any) in the union population S. The diversity of solutions could be improved by randomly permuting the preceding three parameters at each iteration of the SAT execution, which will be experimentally verified later in Section 5.4. In this section, we will also empirically assess the degree of DP created by this through a performance metric called *pure diversity* (PD), as defined in Wang et al. [84].

Finally, it should be noted here that if all solutions in S are valid, then we do not need to perform the two SAT solvers. Only in the case that invalid solutions appear will one of the two SAT solvers be awakened in each iteration. Furthermore, a parameter θ is used to control the frequency of using the two solvers. A detailed study on the effect of θ will be available in Section 5.5.

3.5 Environmental Selection

The aim of the environmental selection is to choose N diversified solutions for the next generation. Here, we propose a two-level ranking method to classify solutions. The first-level ranking is based on the number of violated constraints—that is, solutions with the same number of violated constraints are put into the same layer. The second one is actually the nondominated ranking procedure [20], which divides solutions in the same layer into different sublayers according to Pareto dominance.

More specifically, the first-level ranking works as follows. For individuals in the union population, those with the fewest number of violated constraints belong to the first layer, those with the second fewest number of violated constraints belong to the second layer, and so on. To achieve this, we can simply sort individuals according to the number of violated constraints in an ascending order, and individuals with the same number of violated constraints are naturally grouped into a same layer. The preceding operation requires only $O(NlogN)$ comparisons by using the quick sort procedure [36].

After the first-level ranking, individuals are chosen according to the follow principle: those in the first layer are chosen first, those in the second layer are chosen second, and so on. The purpose of this is to emphasize solutions with small number of violated constraints. The preceding procedure is continued until the population is full, or not full but cannot accommodate individuals in the next layer, which is often called the *critical layer*. The second case happens quite often because there could be many solutions with the same number of violated constraints. In this case, individuals in the critical layer are further classified according to the second-level ranking procedure.

Suppose that we need to choose 10 individuals from all 20 individuals in the union set. There are 4 individuals with zero violated constraints, 3 individuals with one violated constraint, 8 individuals with two violated constraints, and 5 individuals with three violated constraints. According to the first-level ranking, the first 4 individuals fall into the first layer and are chosen first, and the next 3 individuals fall into the second layer and are chosen second. There are already 7 individuals and the population cannot accommodate all 8 individuals in the third layer (critical layer). In this case, 3 individuals will be selected from the critical layer according to the second-level ranking.

For the second-level ranking, it divides individuals in the critical layer into different sublayers by the so-called nondominated sorting procedure [19, 20]. Similarly, individuals in the first sublayer are chosen first, individuals in the second sublayer are chosen second, and so on. The preceding procedure is continued until the new population is full, or not full but cannot hold all individuals in the next sublayer (called the *critical sublayer*). In the latter case, the maximum-vector-angle-first principle [92] is used to select individuals one by one from the critical sublayer to fill the remaining slots of the new population. A brief introduction to the maximum-vector-angle-first principle, along with an illustrative example, can be found in Section 2.2.

Note that before applying the maximum-vector-angle-first principle, individuals who have already been included into the population, together with those in the critical sublayer, are suggested to be normalized according to the method introduced in Deb and Jain [19]. After normalization, each individual in the critical sublayer is associated with its target solution (whose definition is available in Section 2.2). According to the maximum-vector-angle-first principle, the individual with the maximum vector angle to its target solution is chosen and added into the population. Once a new member has been added, some individuals in the critical sublayer may need to update their target solution [92]. The preceding principle is repeated until the population is full.

As a summary, we list some merits of the proposed environmental selection. In the first-level ranking, individuals are distinguished based on the number of violated constraints. Thus, more emphases are put on feasible individuals or individuals with a small number of violated constraints. If individuals have the same constraint violations, they are further differentiated by the nondominated ranking procedure. Therefore, individuals performing better in terms of Pareto dominance have more chances to survive. Finally, according to the maximum-vector-angle-first principle, individuals in the critical sublayer are dynamically selected such that the diversity among solutions is maintained as much as possible. Therefore, the used principle can ensure that solutions for the next generation have a proper distribution in the objective space.

Finally, note that SATVaEA could be used as a general tool in other areas where an optimization problem is expressed in a CNF formula. In fact, according to Algorithm 1, SATVaEA uses some common techniques that are based on the structure of the CNF formula, such as the BCP simplification method and two SAT solvers. For other algorithmic components, such as the generation of

Table 1. Feature Models Used in the Empirical Study

FM	#Features (N_f)	#Constraints (N_c)	$\alpha = \frac{N_c}{N_f}$
toybox	544	1,020	1.875
axTLS	684	2,155	3.151
freebsd-icse11	1,396	62,183	44.544
fiasco	1,638	5,228	3.192
uClinux	1,850	2,468	1.334
busybox-1.18.0	6,796	17,836	2.624
2.6.28.6-icse11	6,888	343,944	49.933
uClinux-config	11,254	31,637	2.811
coreboot	12,268	47,091	3.839
buildroot	14,910	45,603	3.059
embtoolkit	23,516	180,511	7.676
freetz	31,012	102,705	3.312
2.6.32-2var	60,072	268,223	4.465
2.6.33.3-2var	62,482	273,799	4.382

Note: Three of the models (i.e., the ones named 2.6.*) represent the Linux kernel configuration options for the x86 architecture. The 2.6.28.6-icse11 was widely used in related studies (e.g., [32, 74, 82, 94]).

offspring and the environmental selection, they are also common in the area of evolutionary computation. Therefore, if a MOP/MaOP can be represented in a CNF formula, the proposed algorithm can be directly applied to find solutions for this problem.

4 EXPERIMENTAL SETUP

This section gives details of the experimental setup, including FMs used, optimization objectives, implementation details, and performance metrics.

4.1 Feature Models Used in This Study

The FMs used in this empirical study are obtained from the LVAT repository, which collects the works of Berger et al. [6, 7] and She et al. [78]. These models are reverse engineered from such practical projects as the Linux kernel, uClinux, and FreeBSD operating systems, and other large-scale projects like the embedded systems toolkit. The models in this repository have distinct differences from those in SPLOT [58], which were published by academic researchers without representing any real system. Compared to academic models, LVAT models have much more features, more constraints, and higher branching factors [74]. Recently, LVAT models have been widely used as benchmarks in the SPLE domain [32, 53, 74].

The LVAT repository contains 15 models, and 14 of them are chosen in this article.[7] The characteristics of the FMs are summarized in Table 1. For each of them, the table presents the name of the model, the number of features (N_f), the number of constraints (N_c), and the ratio of the number of constraints to the number of features (i.e., $\alpha = \frac{N_c}{N_f}$). According to Table 1, the size of the FMs ranges from 544 to 62,482, whereas the number of constraints also changes widely, from the smallest 1,020 to the largest 343,944. Based on the number of features, all FMs can be classified into three groups: small-scale FMs with $N_f \leq 1{,}000$ (toybox and axTLS), medium-scale FMs with $1{,}000 < N_f \leq 6{,}000$

[7]According to Saayad et al. [74], the eCos model in LVAT accepts the "zero-feature" configuration, which definitely is a bug. In our experimental study, we find the same observation. Therefore, the eCos model is not considered here.

Table 2. Details for Generating Seeds

FM	Time (s)	Flips Required (#)
toybox	0.08	272
axTLS	0.11	527
freebsd-icse11	0.42	323
fiasco	0.22	4,585
uClinux	0.19	826
busybox-1.18.0	0.57	3,437
2.6.28.6-icse11	1.98	2,785
uClinux-config	0.96	5,775
coreboot	1.57	185,877
buildroot	2.17	1,043,673
embtoolkit	1,284.21	530,939,088
freetz	2.59	25,172
2.6.32-2var	89.17	81,845,069
2.6.33.3-2var	6.33	1,294,899

(freebsd-icse11, fiasco, and uClinux), and large-scale FMs with $N_f > 6{,}000$ (from busybox-1.18.0 to 2.6.33.3-2var in Table 1). For the ratio α, small values are observed for all FMs, except for freebsd-icse11 and 2.6.28.6-icse11. According to the observations in Mendonca et al. [59], the SAT-based approaches were found to be easy when analyzing and configuring FMs with small values of α.

Following the suggestions by Sayyad et al. [74, 75] and Henard et al. [32], each feature is augmented with three attributes: *cost*, *used before* and *defects*. The values of these attributes are generated arbitrarily according to uniform distributions. Specifically, the values of *cost* are distributed uniformly between 5.0 and 15.0, whereas those of *defects* are random integers between 0 and 10. The *used before* takes random Boolean values. However, there is a dependency between *used before* and *defects*: if (not *used before*), then *defects* = 0. Note that the ranges of the attributes are selected following the practice of the prior works [32, 35, 74, 75].

Since the role of seeds will be investigated in Section 5.6, we need to generate a seed for each FM. Although the two-objective optimization with IBEA [74] can be used to generate a rich seed, this method is very time consuming: it took nearly 3 hours to produce a valid product as a seed for the 2.6.28.6-icse11 model [74]. Instead, in this work, we use fast WalkSAT[8] [10] to generate random seeds. Table 2 gives the time (in seconds) and the number of required flips for generating a random seed. This experiment is conducted in a computer with the following hardware configurations: Intel Core i5-5200U and a CPU @ 2.20GHz with 8GB of RAM. As shown in Table 2, fast WalkSAT is highly efficient in generating a seed for all the small- and medium-scale FMs (within 1 second), and for some large-scale FMs such as busybox-1.18.0 and uClinux-config. For the remaining large-scale FMs, it is not always easy to obtain a random seed. For example, it takes nearly $1{,}284.21/60 \approx 21$ minutes to generate a valid configuration for embtoolkit, which is a very difficult FM to handle.

4.2 Optimization Objectives

In our proposed SATVaEA, the following optimization objectives are considered:

- *Richness of features (how many features are selected).* We seek to minimize the number of deselected features in a configuration.

[8]The code of fast WalkSAT can be downloaded from http://lcs.ios.ac.cn/~caisw/index.html.

- *Features that were used before.* Since features previously not used are more likely to be faulty [35], we seek to minimize the number of features that were not used before.
- *Known defects.* We seek to minimize the number of known defects in a configuration.
- *Cost.* We seek to minimize the sum of the costs of selected features.

In practice, based on the experience of software engineers and the need of end users, other optimization objectives can be also considered. Note that in prior works, such as SATIBEA [32] and SIP-based algorithms [35], usually five optimization objectives were adopted. Apart from the preceding four objectives, there was another objective named *Correctness* being widely used as the first objective in these works. The *Correctness* refers to the number of constraints that are violated by a configuration. Since SATVaEA treats the optimal feature selection problem as a constrained MaOP, we follow a general constraints-first and objectives-second pattern to solve the problem. As described in Section 3.5, the number of violated constraints (i.e., the *Correctness*) is first used to divide solutions into different layers (hence, the priority is given to those in lower layers). Then, if necessary, solutions with the identical number of violated constraints are distinguished by the preceding four optimization objectives based on Pareto dominance. This idea is somewhat similar to the $1 + n$ approach: *The first objective (i.e., the number of violated constraints) is viewed as the main objective to be considered first and then the remaining objectives as secondary objectives to be optimised equally* [35]. When dealing with MOPs or MaOPs with constraints, however, the method used in SATVaEA is more common in general multiobjective optimizers, such as NSGA-II [20], CNSGA-III [38], and CVaEA [91].

Therefore, we do not use the *Correctness* as an objective in SATVaEA. Instead, it is considered in the constraint-handling process. However, whether using the *Correctness* or not as the first objective makes no significant differences in our proposed algorithm. First, following the work principles of SATVaEA, the number of violated constraints is always used to distinguish individuals whether or not the *Correctness* is used as an objective. Second, Pareto dominance is only applied to solutions with an equal number of violated constraints. Therefore, if five objectives are considered, these solutions are all having equal values in the first objective (i.e., *Correctness*). According to the definition of Pareto dominance (refer to Section 2.2), the Pareto dominance relation among these solutions is ultimately determined by the other four objectives (because the first objective is equal and it has no impact on the final Pareto dominance relation). Finally, as suggested in Hierons et al. [35], only four objectives (after excluding the first one, i.e., *Correctness*) of valid solutions in the population are used to calculate performance metrics (as the first objective is always zero for valid solutions). The preceding facts well explain why no obvious differences are observed whether or not four or five objectives are used in SATVaEA. However, in practice, it would be simpler using only four objectives. An experimental verification of this, together with an illustrative example, will be found later in Section 5.7.

Finally, to make fair performance evaluations, for each algorithm only valid solutions are used to compute performance metrics. Moreover, for peer algorithms (e.g., SATIBEA [32] and SIP-based algorithms [35]), they are ultimately evaluated and compared considering the same objectives after excluding the *Correctness*, which was suggested in Hierons et al. [35]. Therefore, it is fair to make comparisons between SATVaEA and related algorithms even though different numbers of objectives are used.

4.3 Implementation Details

All of the algorithms are independently run 30 times on each FM to reduce the impact of the randomness [35]. Following the suggestions in Sayyad et al. [74], the termination criterion used in

Table 3. Maximum Running Time for Each FM

Time (s)	FM
6	toybox, axTLS (2 FMs)
30	freebsd-icse11, fiasco, uClinux, busybox-1.18.0 (4 FMs)
200	2.6.28.6-icse11,uClinux-config (2 FMs)
400	buildroot, freetz (2 FMs)
600	coreboot, embtoolkit, 2.6.32-2var, 2.6.33.3-2var (4 FMs)

Note: The maximum running time for each FM is set according to our experience. We find that it is enough for our algorithm, in most cases, to obtain 100% valid configurations in a single run by setting the running time to the values presented in this table.

all of the algorithms is the predefined maximum running time (in seconds), which is set according to Table 3. The size of the population for all of the algorithms is set to 100.

In SATVaEA and its derived algorithms, the following parameters are used: the crossover probability $p_c = 1.0$ and the mutation probability $p_m = 1/N_f$ (where N_f is the number of features) [35]. In the fast WalkSAT solver, the noise parameter is set to 0.567 according to Cai [10], and the number of the predefined flips is set to 4,000 for all of the FMs. With this number of flips, according to Table 2, this solver can guarantee that a valid solution can be found for almost half of the FMs.

For the peer algorithm SATIBEA, the parameters are set according to its developers [32]. The archive size is the same as the population size (i.e., 100), and the crossover rate is set to 0.05. The rate to use the bit-flip mutation is set to 0.98, where the probability to flip a feature is set to 0.001 per feature. The rate to use the smart mutation and the smart replacement is 0.01 for both cases. Since these values are recommended by the corresponding literature, we do not require a tuning phase.

All algorithms in the experiments are implemented in Java based on the framework of jMetal [21]. We obtain the implementation of SATIBEA from the standard toolkit.[9] SATVaEA and its derived algorithms are implemented by our codes.[10] All experiments are performed on a notebook PC with an Intel Core i5-5200U Quad Core @ 2.20GHz with 8GB of RAM.

4.4 Performance Metrics

In this study, we use seven performance metrics to measure the quality of solutions. Since software engineers are particularly interested in valid products, the rate of valid products [35], denoted VR, is employed to evaluate the ability of an algorithm to find valid products. Among all valid products, some of them may be dominated by others. The number of nondominated solutions, denoted NNDS, is a performance metric used to assess the ability of an algorithm to return nondominated valid products. For both VR and NNDS, the larger the value is, the better the quality of the solution set will be.

Hypervolume (HV) [104], inverted generational distance (IGD) [13, 100], and PD [84] are popular performance metrics in the EMO area. HV measures the volume of the objective space dominated by solutions in A and bounded by a reference point $\mathbf{z}^r = (z_1^r, z_2^r, \ldots, z_m^r)$, namely

$$HV(A) = Vol\left(\bigcup_{\mathbf{a}\in A}[a_1, z_1^r] \times [a_2, z_2^r] \ldots [a_m, z_m^r]\right), \tag{1}$$

[9]The code of SATIBEA is available at http://research.henard.net/SPL/ICSE_2015/.
[10]The code of SATVaEA can be found at https://www.researchgate.net/profile/Xiang_Yi9/publications.

where m is the number of objectives; $\mathbf{a} = (a_1, a_2, \ldots, a_m)$ is an objective vector in A; and $Vol(\cdot)$ indicates the Lebesgue measure [25, 47]. For HV, a large value reflects good performance of the solution set in terms of both convergence and diversity. When calculating HV, we need to address the following two issues well. One is the scaling of the objective space, and the other is the choice of the reference point [50]. We normalize each objective value of approximate solutions to [0,1] according to the ranges of the problems in the objective space and set reference point $\mathbf{z}^r$ to the nadir point $(1.0, 1.0, \ldots, 1.0)$.

Let $A = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{|A|}\}$ be an approximate objective vector set, and let $Z = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_{|Z|}\}$ be the reference point set containing a number of nondominated points properly distributed along the true PF. The IGD metric is the average distance from the solutions in Z to the closest solution in the set A, which is calculated as follows:

$$IGD(A) = \frac{1}{|Z|} \sum_{j=1}^{|Z|} d_j, \tag{2}$$

where d_j is the Euclidean distance from $\mathbf{z}_j$ to its nearest objective vector in A. If $|Z|$ is large enough to cover the true PF very well, then both convergence and diversity of the set A can be measured by $IGD(A)$. For calculating IGD, a set of reference points is needed. In this study, all nondominated valid solutions found by all of the algorithms over 30 runs are used to construct the reference set Z.

The PD metric is a PD assessment in many-objective optimization, which is an accumulation of the dissimilarity in the approximate set, where an L_p-norm-based ($p < 1$) distance is adopted to measure the dissimilarity among solutions [84]. Mathematically,

$$PD(A) = \max_{\mathbf{a}_i \in A} \{PD(A \backslash \{\mathbf{a}_i\}) + d(\mathbf{a}_i, A \backslash \{\mathbf{a}_i\})\}, \tag{3}$$

where $d(\mathbf{a}_i, A \backslash \{\mathbf{a}_i\})$ is the dissimilarity from $\mathbf{a}_i$ to its nearest point $\mathbf{s} = (s_1, s_2, \ldots, s_m) \in A \backslash \{\mathbf{a}_i\}$. The dissimilarity is defined as follows:

$$dissimilarity(\mathbf{a}_i, \mathbf{s}) = \left(\sum_{j=1}^{m} (a_{ij} - s_j)^p \right)^{1/p}, \tag{4}$$

where $p < 1$ for which a value of 0.1 is suggested in Wang et al. [84]. The details of the calculation of PD can be found in Wang et al. [84]. In the study, this metric is implemented by the code provided in Wang et al. [84].[11]

The TT50% (i.e., the time to obtain 50% valid solutions) was first introduced in Sayyad et al. [74] to measure the speed of the convergence to a large amount of valid products. Since TT50% shows who arrived faster at the 50% milestone, it is a useful comparison indicator when the final VR value is the same. Based on our experimental results, we find that our algorithm and some of its derived algorithms could obtain 100% valid solutions on the majority of the FMs. Therefore, we add another metric, TT100% (i.e., the time to obtain 100% valid solutions), to compare among algorithms when they are incomparable in terms of VR. Both TT50% and TT100% are metrics for measuring the convergence speed, and throughout the article, they are measured in seconds.

Finally, it is necessary to mention that when computing HV, IGD, and PD, we use only valid nondominated solutions in the population, as invalid solutions are meaningless to software engineers and end users. In addition, the TT50% or TT100% is marked by N/A if the rate of returned valid solutions never arrives at 50% or 100%.

[11]The code of PD can be found at http://www.surrey.ac.uk/cs/people/handing_wang/.

Table 4. Percentage of Decrements of Features and Constraints After Simplification

FM	Features That Allow Flipping (#)	Decrement	Constraints After Simplification (#)	Decrement
toybox	181	66.7%	477	53.2%
axTLS	300	56.1%	1,657	23.2%
freebsd-icse11	1,392	0.3%	54,351	12.6%
fiasco	631	61.5%	3,314	36.6%
uClinux	606	67.2%	606	75.4%
busybox-1.18.0	2,845	58.2%	12,145	31.9%
2.6.28.6-icse11	6,742	2.2%	227,009	34.0%
uClinux-config	5,227	53.6%	23,951	24.3%
coreboot	7,566	38.3%	40,736	13.5%
buildroot	8,150	45.3%	37,294	18.2%
embtoolkit	16,641	29.2%	171,027	5.3%
freetz	16,481	46.9%	85,671	16.6%
2.6.32-2var	27,077	54.9%	189,883	29.2%
2.6.33.3-2var	28,115	55.0%	195,815	28.5%
Average		**42.5%**		**27.8%**

5 RESULTS

In this section, we start by showing that the simplification method used in SATVaEA indeed reduces the complexity of an FM; we then show the original VaEA is not powerful enough for configuring SPLs. Next, we explore the effect of the fast local search, DP, the seeding technique, and the parameter θ. Finally, we compare SATVaEA to the state-of-the-art SATIBEA and two SIP-based algorithms.

5.1 The Simplification Method Reduces the Complexity

In SATVaEA, the simplification method is based on the idea that the corresponding feature of a single literal (as a constraint) is either mandatory or dead. The truth assignment for this feature is fixed—that is, *true* for a mandatory feature and *false* for a dead feature. Thus, the number of features that allow flipping declines after excluding mandatory and dead features, and hence the number of the constraints. To show the positive effect of this technique, Table 4 gives the percentage of decrements of features and constraints after simplification. As can be seen from the table, the simplification strategy allows the number of features to decrease by 0.3% in the worst case, 67.2% in the best case, and 42.5% on average. For the number of constraints, it decreases by 5.3% and 75.4% in the worst and best cases, respectively. On average, the number of total constraints decreases by 27.8%.

According to the preceding statistics, the simplification approach indeed reduces the complexity of an FM, and the benefits of this are twofold. On one hand, the decrement in the number of features helps reduce the search space of the problem. The original search space is 2^{N_f}. If the number of features decreases by β, then search space will reduce by $\eta = 1 - (\frac{1}{2})^{\beta N_f}$. As βN_f is very large in general, $(\frac{1}{2})^{\beta N_f}$ tends to 0. Hence, η is almost 100%. In other words, the search space reduces almost 100% for all models in this study, except for freebsd-icse11, for which η is equal to 94%. On the other hand, since the number of constraints also declines after simplification, it takes less time to compute the total number of violated constraints for a solution (because the number of constraints to be checked is less than the original one). As a positive result, the algorithm has more time to explore other possible solutions.

5.2 The Performance of VaEA Needs Improvements

In this section, we show that the performance of the original VaEA needs improvements when configuring SPLs. The only difference between VaEA and SATVaEA is that VaEA uses no SAT solvers. Table 5 lists medians of VR, NNDS, HV, IGD, TT50%, and TT100% for both SATVaEA and VaEA on all FMs. To make the comparison easy, the best results are shown in bold in this table. The symbol • indicates that SATVaEA significantly improves VaEA at a 0.05 level of significance by the Wilcoxon's rank sum test [90], whereas ○ indicates the opposite (i.e., VaEA shows significant improvements over SATVaEA). If no significant differences are detected, the entry will be marked by the symbol ≈. The symbols have the same meanings in other tables.

As shown in Table 5, SATVaEA shows significant improvements over VaEA on almost all FMs in terms of all performance metrics. Specifically, SATVaEA performs better than VaEA in 61 out of $14 \times 6 = 84$ cases (there are 14 FMs and 6 performance metrics for each) and worse in only 8 cases. In the remaining 15 cases, they perform competitively with each other. Table 6 gives a summary of comparisons between SATVaEA and VaEA. In this table, one can find the number of FMs on which SATVaEA is better than (•), similar to (≈), and worse than (○) VaEA for each performance metric. As seen, SATVaEA performs better than or similarly to VaEA on all FMs in terms of both VR and NNDS. Moreover, SATVaEA significantly outperforms VaEA on all models regarding both HV and IGD. However, there are four models on which SATVaEA is inferior to VaEA concerning both TT50% and TT100%.

Now we analyze the behaviors of VaEA in depth. For some FMs, such as toybox, axTLS, 2.6.28.6-icse11, and 2.6.32-2var, the VaEA could find 100% valid solutions in the final population; however, it only returned a small number of valid products on other FMs, such as busybox-1.18.0, coreboot, embtoolkit, and freetz. Although VaEA could obtain 100% valid products on FMs such as toybox, axTLS, and fiasco, the number of nondominated solutions is less than the population size. For example, VaEA found 100% valid solutions for axTLS, but there are only 30 nondominated ones. For some FMs, such as freebsd-icse11, uClinux, 2.6.28.6-icse11, buildroot, 2.6.32-2var, and 2.6.33.3-2var, 100% valid nondominated solutions are returned by VaEA; however, its performance is obviously worse than that of SATVaEA in terms of HV and IGD. This phenomenon may be attributed to the fact that solutions obtained by VaEA are very similar to each other, leading to the poor diversity among solutions. Figure 3 shows, by parallel coordinates [51], the final solutions found by both SATVaEA and VaEA on freebsd-icse11, uClinux, and buildroot. The solution set is associated with a particular run where the HV value is closest to the median over 30 runs. As shown in Figure 3, solutions of SATVaEA cover much more widely than those of VaEA in each objective. According to Figure 3(d), (e) and (f), solutions found by VaEA concentrate on only a small part in the objectives space. In each objective, objective values are very close to each other. As a natural consequence, the diversity among solutions is really poor. Similar observations can be found on other FMs. Hence, compared to SATVaEA, VaEA shows significantly worse HV and IGD values. For the convergence speed, VaEA is better than SATVaEA on only four FMs (i.e., freebsd-icse11, 2.6.28.6-icse11, 2.6.32-2var, and 2.6.33.3-2var). However, as mentioned previously, SATVaEA presents obviously better HV and IGD values on these FMs.

As a summary, we have the following observations:

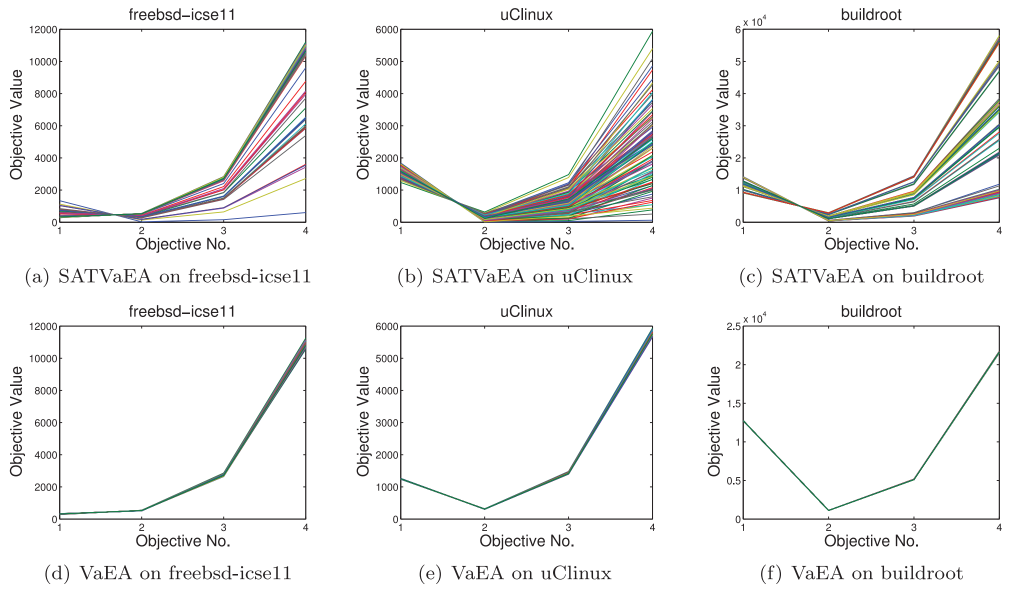
- The VaEA is unable to find 100% valid solutions on some FMs.
- Even though 100% valid solutions are obtained for some FMs, the number of nondominated ones is insufficient.

Table 5. Medians of the Performance Metrics for SATVaEA and VaEA, With the Best Results Shown in Bold

FM	Metric	SATVaEA	VaEA		FM	Metric	SATVaEA	VaEA	
toybox	VR	**100%**	**100%**	≈	axTLS	VR	**100%**	**100%**	≈
	NNDS	**100**	69	●		NNDS	**100**	30	●
	HV	**0.2859**	0.1540	●		HV	**0.2578**	0.1270	●
	IGD	**16.7143**	172.5668	●		IGD	**23.3812**	303.4900	●
	TT50%	**0.1615**	1.1150	●		TT50%	**0.9815**	3.2845	●
	TT100%	**0.2310**	1.5990	●		TT100%	**1.3175**	4.3265	●
freebsd-icse11	VR	**100%**	**100%**	≈	fiasco	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	49	●
	HV	**0.1962**	0.0004	●		HV	**0.2441**	0.0981	●
	IGD	**120.8886**	3,279.7933	●		IGD	**27.5199**	682.3682	●
	TT50%	6.6715	**5.9310**	○		TT50%	**4.2555**	4.9015	●
	TT100%	7.7195	**6.5130**	○		TT100%	**5.7280**	7.4070	●
uClinux	VR	**100%**	**100%**	≈	busybox-1.18.0	VR	**100%**	2%	●
	NNDS	**100**	**100**	≈		NNDS	**100**	2	●
	HV	**0.2735**	0.0001	●		HV	**0.2082**	0.0635	●
	IGD	**65.7827**	2,836.7113	●		IGD	**205.7477**	2,661.5083	●
	TT50%	**0.4045**	2.8865	●		TT50%	**12.2265**	N/A	●
	TT100%	**0.5285**	4.0255	●		TT100%	**23.0120**	N/A	●
2.6.28.6-icse11	VR	**100%**	**100%**	≈	uClinux-config	VR	**100%**	76%	●
	NNDS	**100**	**100**	≈		NNDS	**100**	30	●
	HV	**0.0789**	0.0000	●		HV	**0.1774**	0.0868	●
	IGD	**2,010.1401**	21,347.5147	●		IGD	**223.7868**	6,570.6469	●
	TT50%	30.7295	**20.1750**	○		TT50%	**16.4295**	145.6005	●
	TT100%	33.5210	**22.1110**	○		TT100%	**23.1060**	179.6610	●
coreboot	VR	**100%**	1%	●	buildroot	VR	**100%**	**100%**	≈
	NNDS	**100**	1	●		NNDS	**100**	**100**	≈
	HV	**0.1494**	0.0837	●		HV	**0.2136**	0.1080	●
	IGD	**228.2323**	11,380.6516	●		IGD	**255.7752**	13,292.1960	●
	TT50%	**129.8440**	N/A	●		TT50%	**29.1185**	36.7375	●
	TT100%	**244.9315**	N/A	●		TT100%	**39.4955**	45.1490	●
embtoolkit	VR	**100%**	2%	●	freetz	VR	**100%**	15%	●
	NNDS	**80**	2	●		NNDS	**100**	8	●
	HV	**0.2216**	0.0976	●		HV	**0.1479**	0.0949	●
	IGD	**621.2291**	9,096.2478	●		IGD	**354.6108**	30,377.6262	●
	TT50%	**312.0060**	N/A	●		TT50%	**92.1285**	365.6640	●
	TT100%	**485.3255**	N/A	●		TT100%	**148.5355**	N/A	●
2.6.32-2var	VR	**100%**	**100%**	≈	2.6.33.3-2var	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.1672**	0.1023	●		HV	**0.1355**	0.0882	●
	IGD	**948.7174**	40,303.6378	●		IGD	**1,018.2231**	35,770.2168	●
	TT50%	159.5415	**125.2175**	○		TT50%	172.6755	**127.2975**	○
	TT100%	224.2705	**164.9730**	○		TT100%	232.6125	**161.9255**	○

Table 6. Summary of the Comparison Between SATVaEA and VaEA

SATVaEA vs. VaEA	Better (●)	Similar (≈)	Worse (○)
VR	5	9	0
NNDS	8	6	0
HV	14	0	0
IGD	14	0	0
TT50%	10	0	4
TT100%	10	0	4



(a) SATVaEA on freebsd-icse11 (b) SATVaEA on uClinux (c) SATVaEA on buildroot

(d) VaEA on freebsd-icse11 (e) VaEA on uClinux (f) VaEA on buildroot

Fig. 3. Final solutions obtained by SATVaEA and VaEA on the selected FMs, shown by parallel coordinates.

- Even though 100% valid nondominated solutions are returned for some FMs, VaEA struggles to yield a set of high-quality solutions regarding both HV and IGD, which is attributed to the poor diversity among solutions.
- The convergence speed of VaEA should be improved.

Since VaEA encounters great difficulties in configuring SPLs, the space for improvement is possible when applying the framework of VaEA to the feature selection problem. The improvement of convergence and the promotion of diversity are two feasible avenues along this direction. In the following two sections, the effect of the fast local search and DP will be investigated experimentally.

5.3 The Effect of the Fast Local Search

Recall that the fast local search in SATVaEA is implemented by using an SLS-style SAT solver (i.e., fast WalkSAT). This section investigates how the fast local search affects the performance of the proposed approach. For this purpose, by turning off the SLS-style SAT solver, we obtain a new algorithm, denoted by SATVaEA-LS (the "-LS" means that the local search is removed). Note that the DPLL/CDCL-style SAT solver (for DP) is retained in SATVaEA-LS. Table 7 lists the medians

Table 7. Medians of the Performance Metrics for SATVaEA and SATVaEA-LS Where the Local Search Is Removed

FM	Metric	SATVaEA	SATVaEA-LS		FM	Metric	SATVaEA	SATVaEA-LS	
toybox	VR	**100%**	**100%**	≈	axTLS	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.2859**	0.2420	•		HV	**0.2578**	0.2240	•
	IGD	**16.7143**	19.9759	•		IGD	**23.3812**	35.3152	•
	TT50%	**0.1615**	1.1020	•		TT50%	**0.9815**	2.7380	•
	TT100%	**0.2310**	1.5650	•		TT100%	**1.3175**	3.7570	•
freebsd-icse11	VR	**100%**	**100%**	≈	fiasco	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.1962**	0.0790	•		HV	**0.2441**	0.2385	•
	IGD	**120.8886**	865.3401	•		IGD	27.5199	**26.6864**	≈
	TT50%	6.6715	**6.4690**	≈		TT50%	**4.2555**	5.7415	•
	TT100%	7.7195	**6.9615**	≈		TT100%	**5.7280**	9.0815	•
uClinux	VR	**100%**	**100%**	≈	busybox-1.18.0	VR	**100%**	17%	•
	NNDS	**100**	**100**	≈		NNDS	**100**	17	•
	HV	**0.2735**	0.2697	•		HV	**0.2082**	0.1296	•
	IGD	65.7827	**64.8386**	○		IGD	**205.7477**	748.4476	•
	TT50%	**0.4045**	0.8400	•		TT50%	**12.2265**	N/A	•
	TT100%	**0.5285**	1.0090	•		TT100%	**23.0120**	N/A	•
2.6.28.6-icse11	VR	**100%**	**100%**	≈	uClinux-config	VR	**100%**	92%	•
	NNDS	**100**	**100**	≈		NNDS	**100**	80	•
	HV	**0.0789**	0.0744	≈		HV	**0.1774**	0.1500	•
	IGD	**2,010.1401**	2,741.3387	•		IGD	**223.7868**	383.2769	•
	TT50%	30.7295	**26.4660**	○		TT50%	**16.4295**	116.6810	•
	TT100%	33.5210	**29.5465**	○		TT100%	**23.1060**	155.4745	•
coreboot	VR	**100%**	**100%**	≈	buildroot	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.1494**	0.1414	•		HV	**0.2136**	0.2081	•
	IGD	**228.2323**	353.4427	•		IGD	**255.7752**	316.1408	•
	TT50%	**129.8440**	213.1270	•		TT50%	**29.1185**	51.6480	•
	TT100%	**244.9315**	426.9770	•		TT100%	**39.4955**	67.2090	•
embtoolkit	VR	**100%**	40%	•	freetz	VR	**100%**	42%	•
	NNDS	**80**	40	•		NNDS	**100**	41	•
	HV	0.2216	**0.2237**	≈		HV	**0.1479**	0.1443	•
	IGD	621.2291	**560.4852**	≈		IGD	**354.6108**	406.1335	•
	TT50%	**312.0060**	560.8335	•		TT50%	**92.1285**	380.4265	•
	TT100%	**485.3255**	N/A	•		TT100%	**148.5355**	N/A	•
2.6.32-2var	VR	**100%**	**100%**	≈	2.6.33.3-2var	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.1672**	0.1623	•		HV	0.1355	**0.1384**	≈
	IGD	**948.7174**	1,081.9910	•		IGD	1,018.2231	**951.6649**	≈
	TT50%	**159.5415**	206.7590	•		TT50%	**172.6755**	214.9755	•
	TT100%	**224.2705**	259.7790	•		TT100%	**232.6125**	274.5100	•

Table 8. Summary of the Comparison Between SATVaEA and SATVaEA-LS Where the Local Search Is Removed

SATVaEA vs. SATVaEA-LS	Better (●)	Similar (≈)	Worse (○)
VR	4	10	0
NNDS	4	10	0
HV	11	3	0
IGD	10	3	1
TT50%	12	1	1
TT100%	12	1	1

of the performance metrics for both algorithms. It can be found from this table that SATVaEA performs better than SATVaEA-LS in 53 out of 84 cases and worse in only 3 cases. In the remaining 28 cases, the two algorithms are comparable to each other. Overall, SATVaEA is much better than SATVaEA-LS.

As shown in Table 7, SATVaEA-LS is unable to obtain 100% valid solutions on some FMs, such as busybox-1.18.0, uClinux-config, embtoolkit, and freetz. Even though on FMs such as toybox, axTLS, coreboot, buildroot, and 2.6.32-2var it can find 100% valid nondominated products, its performance in terms of the quality of solutions and the convergence speed is not satisfied. Table 8 gives a summary of the comparison between SATVaEA and SATVaEA-LS for each performance metric. As seen, the two algorithms have similar performance in terms of both VR and NNDS: SATVaEA performs better than SATVaEA-LS on 4 FMs and comparably on 10 FMs. For HV and IGD, SATVaEA significantly outperforms its competitor, obtaining better results on 11 and 10 out of 14 FMs, respectively. According to both TT50% and TT100%, SATVaEA is found to be faster than SATVaEA-LS on 12 FMs.

Since the original intention to introduce the fast local search is to speed up the convergence, we are particularly interested in the comparison of the time required to obtain 50% and 100% valid solutions by turning on and turning off the fast local search procedure (i.e., the SLS-style SAT solver). Table 9 gives the time increased (in a percentage) when obtaining 50% and 100% valid solutions by turning off the fast local search in SATVaEA (which is actually SATVaEA-LS). In the table, since TT50% and TT100% for SATVaEA-LS on busybox-1.18.0 are unknown (see Table 7), we are unable to compute the time increment. Instead, it is denoted by "N/A." It is the same for TT100% on embtoolkit and freetz. We can find from the table that the time required after turning off the local search reduces slightly on only two FMs (i.e., freebsd-icse11 and 2.6.28.6-icse11). In addition, the largest percentage of decrement is only 14%, which is observed on 2.6.28.6-icse11 for the TT50% metric. However, on the majority of FMs, SATVaEA-LS needs much more time to obtain 50% and 100% valid solutions. As seen in Table 9, the largest percentages of time increments are 610% and 577% for TT50% and TT100%, respectively. Moreover, TT50% and TT100% are increased by at least 24% and 16%, respectively. According to the preceding results, the fast local search does play an important role in speeding up the convergence of the algorithm.

Finally, we are going to explain why the convergence improvement also contributes to the enhancement of the quality of solutions. As shown in Table 7, SATVaEA presents much better HV and IGD results than SATVaEA-LS. As the local search method may quickly find a valid solution starting from an invalid one, the time point when obtaining 50% or 100% valid solutions in the whole population arrives in advance. Since the allowed total running time for both algorithms is the same, SATVaEA has more chances to explore more valid products from which solutions of high quality can be selected and retained by the algorithm. Here, the quality of solutions is

Table 9. Time Increased by Turning Off the Fast Local Search When Obtaining 50% and 100% Valid Solutions

FM	TT50%	TT100%
toybox	582%	577%
axTLS	179%	185%
freebsd-icse11	−3%	−10%
fiasco	35%	59%
uClinux	108%	91%
busybox-1.18.0	N/A	N/A
2.6.28.6-icse11	−14%	−12%
uClinux-config	610%	573%
coreboot	64%	74%
buildroot	77%	70%
embtoolkit	80%	N/A
freetz	313%	N/A
2.6.32-2var	30%	16%
2.6.33.3-2var	24%	18%

related to both the convergence and diversity. For example, the solutions dominating more solutions are deemed to have better convergence than those dominating less solutions, and solutions with larger angles to the current population are deemed to contribute more to the diversity than those with smaller angles to the population. Since more valid products can be visited and high-quality solutions will be retained by the nondominated sorting procedure [20] and the maximum-vector-angle-first principle [92], the final population found by SATVaEA has naturally better performance in terms of the metrics HV and IGD.

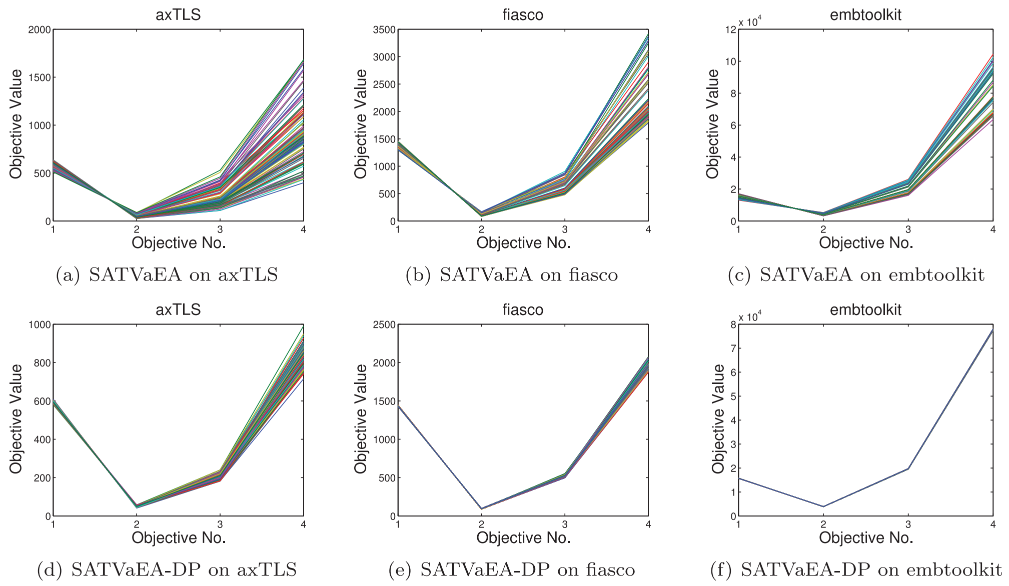
5.4 The Effect of DP

The diversity in SATVaEA is promoted by a DPLL/CDCL-style SAT solver. To examine the effect of DP on the performance of SATVaEA, a new algorithm, denoted by SATVaEA-DP, is obtained by turning off the corresponding SAT solver. Note that the SLS-style SAT solver is kept in SATVaEA-DP. Medians of performance metrics (VR, NNDS, HV, IGD, and PD) for SATVaEA and SATVaEA-DP are tabulated in Table 10. Since the original intention of importing the DPLL/CDCL-style SAT solver is to improve the diversity of the algorithm, we use the diversity metric PD to show that whether the algorithm is improved or not.

As shown by VR and NNDS results in Table 10, both of the algorithms can find 100% valid nondominated solutions on all 14 FMs, except for embtoolkit. For this model, SATVaEA-DP is slightly better than SATVaEA concerning VR and NNDS. However, SATVaEA significantly outperforms its competitor in terms of HV, IGD, and PD. In fact, our proposed approach shows obviously better performance than SATVaEA-DP on all FMs for HV, IGD, and PD. The only exception is that the two algorithms obtained similar PD values on freebsd-icse11. Both algorithms perform similarly in terms of VR and NNDS but quite differently for metrics HV, IGD, and PD. This phenomenon can be explained as follows. Since the fast local search procedure is kept in both algorithms, there are naturally no significant differences in terms of the rate of valid solutions. However, due to the lack of DP in SATVaEA-DP, the obtained solutions are distributed not widely in the whole objective space, which is evident from Figure 4. In this figure, solutions found by SATVaEA-DP concentrate on a small part of the PFs. As a natural consequence, the diversity among solutions is insufficient,

Table 10. Medians of the Performance Metrics for SATVaEA and SATVaEA-DP Where DP Is Removed

FM	Metric	SATVaEA	SATVaEA-DP		FM	Metric	SATVaEA	SATVaEA-DP	
toybox	VR	**100%**	**100%**	≈	axTLS	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.2859**	0.2542	●		HV	**0.2578**	0.1686	●
	IGD	**16.7143**	67.2513	●		IGD	**23.3812**	251.0669	●
	PD	**1.518E+8**	5.234E+7	●		PD	**1.958E+8**	1.604E+7	●
freebsd-icse11	VR	**100%**	**100%**	≈	fiasco	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.1962**	0.1752	●		HV	**0.2441**	0.1371	●
	IGD	**120.8886**	191.4611	●		IGD	**27.5199**	621.7578	●
	PD	**1.735E+9**	1.521E+9	≈		PD	**2.538E+8**	1.382E+7	●
uClinux	VR	**100%**	**100%**	≈	busybox-1.18.0	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.2735**	0.1345	●		HV	**0.2082**	0.0859	●
	IGD	**65.7827**	509.1445	●		IGD	**205.7477**	2.253.4476	●
	PD	**1.800E+9**	1.131E+9	●		PD	**1.616E+9**	8.267E+7	●
2.6.28.6-icse11	VR	**100%**	**100%**	≈	uClinux-config	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.0789**	0.0062	●		HV	**0.1774**	0.0952	●
	IGD	**2.010.1401**	15.311.7809	●		IGD	**223.7868**	6.383.1576	●
	PD	**6.554E+9**	1.798E+9	●		PD	**2.778E+9**	6.600E+7	●
coreboot	VR	**100%**	**100%**	≈	buildroot	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.1494**	0.1085	●		HV	**0.2136**	0.1152	●
	IGD	**228.2323**	10.431.5465	●		IGD	**255.7752**	12.980.8446	●
	PD	**2.975E+9**	3.014E+8	●		PD	**5.634E+9**	1.811E+8	●
embtoolkit	VR	100%	**100%**	○	freetz	VR	**100%**	**100%**	≈
	NNDS	80	**99**	○		NNDS	**100**	**100**	≈
	HV	**0.2216**	0.1022	●		HV	**0.1479**	0.0973	●
	IGD	**621.2291**	8.742.3104	●		IGD	**354.6108**	30.179.2565	●
	PD	**6.359E+9**	2.468E+7	●		PD	**5.630E+9**	5.550E+7	●
2.6.32-2var	VR	**100%**	**100%**	≈	2.6.33.3-2var	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.1672**	0.1064	●		HV	**0.1355**	0.0897	●
	IGD	**948.7174**	39.823.4126	●		IGD	**1.018.2231**	35.557.4200	●
	PD	**1.099E+10**	2.184E+8	●		PD	**9.200E+9**	1.009E+8	●



(a) SATVaEA on axTLS (b) SATVaEA on fiasco (c) SATVaEA on embtoolkit

(d) SATVaEA-DP on axTLS (e) SATVaEA-DP on fiasco (f) SATVaEA-DP on embtoolkit

Fig. 4. Final solutions obtained by SATVaEA and SATVaEA-DP on the models axTLS, fiasco, and embtoolkit, shown by parallel coordinates. The selected three models are representatives of the small-, medium-, and large-scale FMs considered.

Table 11. Results of VR and NNDS (in parentheses) for Different Values of θ

FM	$\theta = 0.7$	$\theta = 0.8$	$\theta = 0.9$	$\theta = 1.0$
toybox	100% (99.5)	100% (100)	100% (100)	100% (100)
fiasco	99.5% (99.5)	100% (100)	100% (100)	100% (100)
uClinux-config	100% (100)	100% (100)	100% (100)	100% (100)
coreboot	100% (100)	100% (100)	100% (100)	100% (100)
freetz	100% (100)	99.5% (99.5)	100% (100)	100% (100)

which directly leads to the poor performance in terms of HV and IGD that measure in part the diversity of the final solution set. According to the statistics shown in Table 10 and the distribution of the final solutions in Figure 4, DP with a DPLL/CDCL-style SAT solver does improve the diversity and hence the quality of returned solutions.

5.5 The Impact of the Parameter θ

In this section, we will investigate the effect of the parameter θ, which controls the frequency of using the two SAT solvers. We perform the experiments on five FMs: toybox, fiasco, uClinux-config, freetz, and coreboot. The preceding models are chosen mainly because they have different termination conditions. According to Table 3, each of the five models has a distinct running time. According to our empirical study, θ should be set to a relatively large value because small values may threaten to find invalid or dominated solutions. To show this, Table 11 gives the results of VR and NNDS on the five FMs for the θ values varying from 0.7 to 1.0 with a step size of 0.1. As can be found in the table, the algorithm cannot find 100% valid solutions on fiasco when $\theta = 0.7$, and on freetz when $\theta = 0.8$. Although 100% valid products can be found on toybox, some of them are dominated because NNDS is 99.5 less than 100. The preceding phenomenon can be interpreted

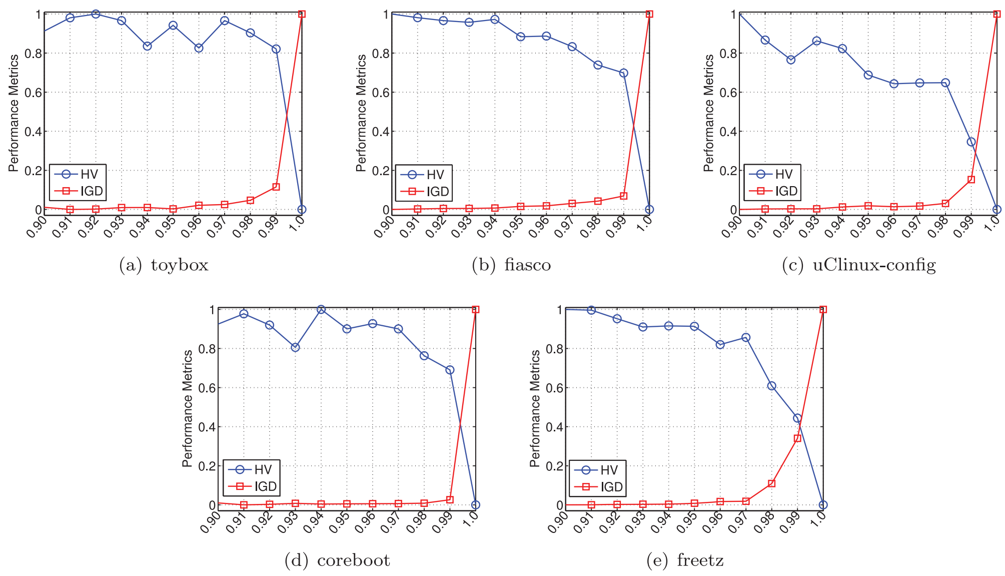


Fig. 5. The curves of HV and IGD with respect to θ values varying from 0.9 to 1.0 with a step size of 0.01. The values of HV and IGD for each FM are normalized to [0,1] according to the maximum and minimum values obtained under all θ values considered.

as follows. Since the DPLL/CDCL-style solver is much more time consuming than the SLS-style solver (the former needs to traverse the whole search space), the calls to the SLS-style solver are limited if more CPU resources are given to the DPLL/CDCL-style solver (by setting θ to a small value). Because the main function of the SLS-style solver is to quickly repair an invalid solution, there may be invalid solutions not operated by this solver if less time is given. Therefore, a small θ has the risk of producing invalid configurations.

To further examine the effect of θ, we change its value from 0.9 to 1.0 with a step size of 0.01. For each value, HV and IGD for toybox, fiasco, uClinux-config, coreboot, and freetz are recorded and plotted in Figure 5. The preceding two performance metrics are chosen because they can provide a combined measurement of both convergence and diversity, whereas VR and NNDS are omitted because the same results are obtained under different θ values. According to Figure 5, the curves of both HV and IGD change similarly on different FMs. Generally, as θ increases from 0.9 to 1.0, the curves of HV decline, whereas those of IGD rise. The worse performance is observed when $\theta = 1.0$, indicating again the usefulness of introducing DP. For general usages, the optimal value of θ is suggested to be 0.9, as high performance is obtained by SATVaEA under this parameter setting.

5.6 The Role of Seeds

It was reported in Sayyad et al. [74] and Henard et al. [32] that the "seedin" technique is a key to the scalability. One feature-rich valid seed in the initial population helps generate a considerable number of valid products. The experimental results also show that it is the quality, not quantity, of seeds that has the most impact on the ability to produce valid configurations. In our proposed method, does the seeding technique play a similar role in increasing the number of valid solutions?

To answer this question, SATVaEA is compared to a new algorithm that uses no seeds, denoted by SATVaEA-SD. The results of performance metrics for both algorithms are shown in Table 12.

Table 12. Medians of the Performance Metrics for SATVaEA and SATVaEA-SD Where No Seeds Are Used

FM	Metric	SATVaEA	SATVaEA-SD		FM	Metric	SATVaEA	SATVaEA-SD	
toybox	VR	**100%**	**100%**	≈	axTLS	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.2859**	0.2778	•		HV	**0.2578**	0.2527	•
	IGD	16.7143	**16.6869**	≈		IGD	**23.3812**	27.4373	•
freebsd-icse11	VR	**100%**	**100%**	≈	fiasco	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	0.1962	**0.2105**	≈		HV	**0.2441**	0.2340	•
	IGD	**120.8886**	380.7491	•		IGD	**27.5199**	28.1330	≈
uClinux	VR	**100%**	**100%**	≈	busybox-1.18.0	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.2735**	0.2727	≈		HV	**0.2082**	0.2002	≈
	IGD	65.7827	**65.4390**	≈		IGD	**205.7477**	242.1989	≈
2.6.28.6-icse11	VR	**100%**	**100%**	≈	uClinux-config	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	0.0789	**0.0968**	○		HV	0.1774	**0.2059**	○
	IGD	**2,010.1401**	2,257.8713	≈		IGD	**223.7868**	255.1597	≈
coreboot	VR	**100%**	**100%**	≈	buildroot	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.1494**	0.1389	•		HV	**0.2136**	0.1960	•
	IGD	**228.2323**	357.6060	•		IGD	**255.7752**	1,027.0338	•
embtoolkit	VR	**100%**	93%	≈	freetz	VR	**100%**	**100%**	≈
	NNDS	**80**	75	≈		NNDS	**100**	**100**	≈
	HV	**0.2216**	0.2192	≈		HV	**0.1479**	0.1022	•
	IGD	**621.2291**	713.5343	•		IGD	**354.6108**	2,603.7570	•
2.6.32-2var	VR	**100%**	**100%**	≈	2.6.33.3-2var	VR	**100%**	**100%**	≈
	NNDS	**100**	**100**	≈		NNDS	**100**	**100**	≈
	HV	**0.1672**	0.1187	•		HV	**0.1355**	0.1210	•
	IGD	**948.7174**	8,334.8404	•		IGD	**1,018.2231**	3,328.4471	•

As seen, the two algorithms have no significant differences on all FMs in terms of both VR and NNDS. For HV, SATVaEA is better than, similar to, and worse than SATVaEA-SD on 8, 4, and 2 out of 14 FMs, respectively. For IGD, SATVaEA outperforms SATVaEA-SD on 8 FMs and obtains comparable results on the remaining 6 FMs.

According to the preceding observations, the seeding technique seems to have no significant influence on the number of valid solutions. This phenomenon is inconsistent with the findings in Sayyad et al. [74]. However, it can be well explained as follows. First of all, the main factor for yielding valid configurations in both SATVaEA and SATVaEA-SD is the fast local search implemented by the SLS-style SAT solver rather than the slow diffusion of seeds, which is just the case in the method proposed in Sayyad et al. [74]. Second, during the search process, the DPLL/CDCL-style SAT solver generates valid solutions that can serve as seeds. Therefore, the diffusion of seeds also exists in SATVaEA-SD. Hence, it is unnecessary to plant an initial seed in the population.

Since the effect of seeds is subtle, we can use no seeds when applying the proposed method, which will be both labor- and time saving. According to Sayyad et al. [74], the rich seeds were precomputed by using the two-objective optimization with IBEA, which was very time consuming for large-scale FMs. For example, it took nearly 3 hours to produce a rich seed for the 2.6.28.6-icse11 model. Even though random seeds can be used, the time to generate such seeds may be also long for large-scale FMs. As shown in Section 4.1, it takes nearly 1,284.21/60 $\approx$ 21 minutes to generate a random seed for the embtoolkit model.

5.7 Comparing SATVaEA to the State-of-the-Art SATIBEA

In this section, SATVaEA is compared to the state-of-the-art SATIBEA [32], a recent method for configuring large-scale SPLs. In SATVaEA, we suggest to use only four optimization objectives (see Section 4.2). However, there are five objectives in SATIBEA. As discussed in Section 4.2, whether or not the *Correctness* is used as the first objective makes no significant differences in our proposed algorithm. To experimentally verify this, we introduce a new version of SATVaEA, denoted SATVaEA*, where exactly the same five objectives as in SATIBEA are used.

The medians of VR, NNDS, HV, IGD, TT50%, and TT100% are listed in Table 13. As shown, SATVaEA and SATVaEA* perform competitively in 55 out of 84 cases, and the number of cases where SATVaEA is significantly better and worse than SATVaEA* is 14 and 15, respectively. The differences are mainly observed on the time-related performance metrics TT50% and TT100%. This suggests that the two algorithms have similar overall performance, indicating the slight impact of using four or five objectives in our proposed algorithm. The following can be possible explanations for this. According to the environmental selection in Section 3.5, solutions are first divided into different layers according to the number of violated constraints no matter which number of objectives is used, and thus individuals in the same layers have the same number of violated constraints. This, at the same time, makes these individuals in SATVaEA* have identical values for the first objective (i.e., *Correctness*), which is defined as the number of violated constraints. Since the subsequent Pareto dominance check is only applied among solutions in the same layer, using four or five objectives will make no difference at all. For example, let $A = (0, 1, 5, 8, 10)$ and $B = (0, 4, 5, 10, 12)$ be two solutions in the first layer; according to the definition of Pareto dominance (stating that A Pareto dominates B if and only if all objectives of A are not worse than those of B and there exists at least one objective for which A is better than B), we have that A Pareto dominates B if all the five objectives are considered. However, if we only consider the last four objectives, A still Pareto dominates B. Due to the fact that, for SATVaEA*, the first objective is always the same for solutions in the same layer, the Pareto dominance relation will not change whether four or five objectives are considered. Therefore, SATVaEA can obtain similar performance compared to SATVaEA*.

Table 13. Medians of the Performance Metrics for SATVaEA, SATVaEA*, and SATIBEA

FM		SATVaEA	SATVaEA*	SATIBEA		FM	SATVaEA	SATVaEA*	SATIBEA	
toybox	VR	**100%**	**100%**	≈ 38%	●	axTLS	**100%**	**100%**	≈ 34%	●
	NNDS	**100**	**100**	≈ 37	●		**100**	**100**	≈ 32	●
	HV	**0.2859**	0.2858	≈ 0.2027	●		**0.2599**	0.2580	≈ 0.2009	●
	IGD	16.3326	**15.6506**	○ 37.7503	●		23.1346	**22.1175**	≈ 48.6505	●
	TT50%	0.1615	**0.1595**	≈ 4.7020	●		**0.9815**	1.0280	≈ N/A	●
	TT100%	0.2310	**0.2145**	≈ N/A	●		**1.3175**	1.4355	≈ N/A	●
freebsd-icse11	VR	**100%**	**100%**	≈ 2%	●	fiasco	**100%**	**100%**	≈ 70%	●
	NNDS	**100**	**100**	≈ 2	●		**100**	**100**	≈ 68	●
	HV	0.1953	**0.2110**	○ 0.0350	●		0.2484	**0.2532**	○ 0.2433	●
	IGD	131.8589	**89.8811**	○ 2,331.7843	●		27.8857	**23.9550**	○ 26.9817	≈
	TT50%	6.6715	**4.6945**	○ N/A	●		4.2555	**3.0265**	○ 17.1390	●
	TT100%	7.7195	**5.4410**	○ N/A	●		5.7280	**4.0980**	○ N/A	●
uClinux	VR	**100%**	**100%**	≈ 53%	●	busybox-1.18.0	**100%**	**100%**	≈ 17%	●
	NNDS	**100**	**100**	≈ 53	●		**100**	**100**	≈ 17	●
	HV	**0.2735**	0.2651	● 0.1413	●		0.2091	**0.2167**	≈ 0.1287	●
	IGD	**64.0759**	73.8407	● 396.3365	●		191.6017	**159.0700**	○ 608.0932	●
	TT50%	**0.4045**	0.6340	● 14.1070	●		**12.2265**	16.4275	● N/A	●
	TT100%	**0.5285**	0.9315	● N/A	●		**23.0120**	30.7845	● N/A	●
2.6.28.6-icse11	VR	**100%**	**100%**	≈ 8%	●	uClinux-config	**100%**	**100%**	≈ 48%	●
	NNDS	**100**	**100**	≈ 8	●		**100**	**100**	≈ 48	●
	HV	0.0790	0.0672	● **0.0985**	○		0.1744	**0.1909**	≈ 0.1485	●
	IGD	498.9620	**471.9560**	≈ 1,717.1896	●		209.8006	**191.0471**	≈ 376.3777	●
	TT50%	30.7295	**25.3945**	○ N/A	●		**16.4295**	18.7100	● 176.1710	●
	TT100%	33.5210	**27.8990**	○ N/A	●		**23.1060**	29.8930	● N/A	●
coreboot	VR	**100%**	**100%**	≈ 46%	●	buildroot	**100%**	**100%**	≈ 53%	●
	NNDS	**100**	**100**	≈ 46	●		**100**	**100**	≈ 53	●
	HV	**0.1494**	0.1475	≈ 0.1350	●		0.2148	**0.2169**	≈ 0.2052	●
	IGD	158.8231	**151.8182**	≈ 245.2294	●		**274.5503**	298.4087	● 368.8591	●
	TT50%	129.8440	**115.7305**	○ N/A	●		29.1185	**27.1125**	≈ 316.2660	●
	TT100%	244.9315	**206.3725**	○ N/A	●		39.4955	**36.1040**	≈ N/A	●
embtoolkit	VR	**100%**	**100%**	≈ 27%	●	freetz	**100%**	**100%**	≈ 37%	●
	NNDS	80	**87**	≈ 27	●		**100**	**100**	≈ 37	●
	HV	0.2211	**0.2234**	≈ 0.2151	●		0.1480	**0.1491**	≈ 0.1419	●
	IGD	565.5901	**473.8143**	○ 717.5449	●		**345.4515**	378.9928	● 410.2653	●
	TT50%	312.0060	**283.7145**	≈ N/A	●		92.1285	**87.4545**	≈ N/A	●
	TT100%	485.3255	**477.4910**	≈ N/A	●		148.5355	**147.7720**	≈ N/A	●
2.6.32-2var	VR	**100%**	**100%**	≈ 11%	●	2.6.33.3-2var	**100%**	**100%**	≈ 10%	●
	NNDS	**100**	**100**	≈ 11	●		**100**	**100**	≈ 10	●
	HV	**0.1668**	0.1654	≈ 0.1546	●		0.1336	**0.1349**	≈ 0.1268	●
	IGD	**788.7588**	945.8770	● 1,687.0840	●		949.1848	**916.6801**	≈ 1,948.5122	●
	TT50%	**159.5415**	177.2285	● N/A	●		**172.6755**	175.7185	≈ N/A	●
	TT100%	**224.2705**	248.8455	● N/A	●		**232.6125**	236.8475	≈ N/A	●

Table 14. Results of VR and TT50% for SATVaEA and SATIBEA With $N = 100, 200$, and 300

			toybox	fiasco	uClinux-config	freetz	coreboot
SATVaEA	$N = 100$	VR	100%	100%	100%	100%	100%
		TT50%	0.1652	2.9941	15.4965	93.0008	112.8800
	$N = 200$	VR	100%	100%	100%	100%	98%
		TT50%	0.4447	6.5697	32.6881	200.7604	286.8451
	$N = 300$	VR	100%	100%	100%	84%	81%
		TT50%	0.7502	9.9248	50.4580	306.7343	428.4902
SATIBEA	$N = 100$	VR	38%	69%	47%	37%	46%
		TT50%	4.7020	17.4750	182.3292	N/A	N/A
	$N = 200$	VR	13%	14%	13%	18%	19%
		TT50%	4.5530	N/A	N/A	N/A	N/A
	$N = 300$	VR	10%	11%	11%	16%	16%
		TT50%	N/A	N/A	N/A	N/A	N/A

It is obvious from the table that the proposed SATVaEA significantly outperforms SATIBEA on all FMs in terms of almost all performance metrics. The only two exceptions are that our method obtains similar results to SATIBEA on fiasco concerning the IGD metric and performs worse than the counterpart on 2.6.28.6-icse11 in terms of HV. The superiorities of our proposed algorithm are strongly demonstrated by the experimental results. According to Table 13, the main problem of SATIBEA is its low rate of valid configurations. In fact, the largest and smallest VR for SATIBEA are only 70% and 2%, respectively. The extremely low rate of valid solutions accounts for the poor performance of the algorithm related to quality metrics such as NNDS, HV, and IGD. It can be inferred from TT50% and TT100% that much more time is needed for SATIBEA to return a large amount of valid solutions.

Now we investigate the reasons SATIBEA performs poorly. First of all, SATIBEA treats all of the five optimization objectives equally, and no more emphases are put on the first objective (i.e., the number of violated constraints). As a result, the selection pressure toward valid solutions may be insufficient. Second, solutions in SATIBEA are evaluated and selected according to the fitness value that is calculated based on all of the objectives. A underlying risk of this is that valid solutions may be unselected if they perform well only in the first objective and poorly in all of the other objectives. Third, SATIBEA uses the indicator HV to guide the search process. As we know, the calculation of HV is computationally expensive, especially when a large number of objective is considered [86]. As a natural consequence, the number of solutions explored by the algorithm may be inadequate if the allowed running time is limited.

Finally, to investigate the effect of the population size N on the performance of the algorithms, we carry out the following experiments where the population size N in both SATVaEA and SATIBEA is changed from 100 to 300 with a step size of 100. With all other settings kept the same as in Section 4.3, the VR and TT50% results under different N values are recorded in Table 14, where five FMs as in Section 5.4 are chosen for performance evaluations. This table suggests that SATVaEA is still able to find almost 100% valid products even when N reaches 300. The only two exceptions are the two large-scale FMs (i.e., freetz and coreboot) for which more than 80% valid products are obtained. In contrast, SATIBEA fails to generate a large proportion of valid solutions even when $N = 100$. Moreover, as the population size increases, the ratio of valid products decreases in general. Since N is increased, it is natural that the time when 50% valid products are obtained comes later. It should be noted that the maximum running time for both $N = 200$ and $N = 300$ is the same as $N = 100$. One can imagine that if more time is given, more valid products will be found. However, from the perspective of practical applications, it could be enough for an algorithm to maintain 100 solutions from which users can choose their preferred ones. This population size was also chosen by Hierons et al. [35] in their SIP studies. Therefore, in the following section, SATVaEA will be compared to SIP-based approaches by setting N to 100 for all of the algorithms.

5.8 Comparing to SIP-Based Methods

In this section, the proposed SATVaEA is compared to two SIP-based methods (i.e., SIP+SPEA2+SDE and SIP+NSGA-II), together with SATIBEA on seven FMs taken from Hierons et al. [35]. It was observed in Hierons et al. [35] that there was no clear "best" EMO algorithm within the SIP framework. Therefore, we choose SIP+SPEA2+SDE and SIP+NSGA-II as two representatives. The seven FMs[12] used are WebPortal (43), E-shop (290), Drupal (48), Amazon (79), Random-10000 (10,000), RealAmazon (79), and RealDrupal (48), where WebPortal, E-shop, Drupal, and Amazon have been widely used in previous works on optimal SPL product selection [29, 63, 70], and Random-10000 is a randomly generated model with 10,000 features. The last two FMs—RealAmazon and RealDrupal—are models with realistic attribute values (either real values or ranges) [35]. In this experiment, SATVaEA uses exactly the same optimization objectives as in the peer algorithms. Specifically, for the first five FMs, we use the same five optimization objectives as Sayyad et al. [75], Henard et al. [32], and Hierons et al. [35]. For RealAmazon and RealDrupal, we consider eight objectives that are calculated based on realistic attribute values. For convenience, thepreceding seven models are referred to as SIP FMs. More details on SIP FMs can be found in Hierons et al. [35]. We use exactly the same models to conduct experiments, and data for these FMs are downloaded from http://www.cs.bham.ac.uk/~limx/Data/SIP(data).rar.

The implementations are kept the same as in Hierons et al. [35]. For all four algorithms (i.e., SATVaEA, SATIBEA, SIP+SPEA2+SDE, and SIP+NSGA-II),[13] they are executed 30 times on each FM and are terminated when the number of evaluations reaches 50,000. The size of the population for all algorithms is set to 100. In all algorithms, the uniform crossover and bit-flip mutation are used, with crossover and mutation probabilities being 1.0 and $1/N_f$ (where N_f denotes the number of features), respectively. Following the suggestions in Hierons et al. [35], only valid solutions are used for performance evaluations, and so are all objectives except for the first one (i.e., the *Correctness*).

Table 15 summarizes the results of performance metrics on the SIP FMs. In this comparison, as done in Hierons et al. [35], we include a new performance metric named *VN*, which is the number of executions where there is at least one valid solution in the final population. According to VN values, all four algorithms could succeed in finding at least one valid solution in each of the 30 runs on WebPortal, E-shop, Drupal, Amazon, and RealDrupal. However, for Random FM, both SIP+SPEA2+SDE and SIP+NSGA-II cannot find any valid solution in 5 out of 30 runs. Similarly, SATIBEA fails to obtain feasible solutions on the RealAmazon model in six executions.

For the VR metric, SATIBEA is worse than the other three algorithms on almost all FMs considered, and the only exception is the Random model on which SATIBEA outperforms the two SIP-based algorithms. It is observed that SATIBEA has difficulties in obtaining 100% valid solutions. For IGD and PD, our proposed SATVaEA performs significantly better than other algorithms, obtaining the best or the second best values on all FMs. Finally, considering the runtime, SIP+NSGA-II, followed by SATVaEA, may be the fastest algorithm. However, compared to the preceding two algorithms, SATIBEA and SIP+SPEA2+SDE take much more time when the search is terminated.

As a summary, we have the following conclusions:

- Compared to SATIBEA, the proposed SATVaEA is able to find (much) more valid products on all SIP FMs. Notably, SATIBEA can obtain only 1% valid solutions on the RealAmazon model. This is consistent with the findings in Hierons et al. [35] that IBEA-based algorithm (i.e., SIP+IBEA in Hierons et al. [35]) performed poorly on the real Amazon model. One

[12]The integer after each model is the number of features for that model.
[13]The codes of SIP+SPEA2+SDE and SIP+NSGA-II are downloaded from http://www.cs.bham.ac.uk/~limx/Codes/SIP.rar.

Table 15. Results of the Performance Metrics on SIP FMs, Where the Best and the Second Best Results Are Shown With a Dark and a Light Gray Background, Respectively

FM	Metric	SATVaEA	SATIBEA		SIP+SPEA2+SDE		SIP+NSGA-II	
WebPortal	VN (/30)	30	30		30		30	
	VR	100%	97%	•	100%	≈	100%	≈
	IGD	10.6366	16.1697	•	10.7513	≈	10.5212	≈
	PD	363,610.6200	571,731.2018	○	171,846.5337	•	151,310.5094	•
	Runtime	3.4810	16.0985	•	27.5430	•	0.6380	○
E-shop	VN (/30)	30	30		30		30	
	VR	100%	97%	•	100%	≈	100%	≈
	IGD	54.9058	44.8114	○	239.1933	•	171.2694	•
	PD	1,175,298.6980	1,011,220.8917	•	153,724.1969	•	524,246.0306	•
	Runtime	7.3775	19.4395	•	35.2810	•	0.9650	○
Drupal	VN (/30)	30	30		30		30	
	VR	100%	99%	•	100%	≈	100%	≈
	IGD	14.1677	17.5598	•	18.4402	•	14.9311	•
	PD	250,535.5281	554,239.5165	○	213,341.3709	•	159,993.2761	•
	Runtime	4.1250	16.1085	•	27.1520	•	0.6065	○
Amazon	VN (/30)	30	30		30		30	
	VR	100%	7%	•	100%	≈	100%	≈
	IGD	8.6380	28.0032	•	9.6209	•	10.0130	•
	PD	54,577.8655	184,690.1368	○	22,153.5554	•	38,384.7146	•
	Runtime	2.7470	17.4000	•	25.0440	•	0.5890	○
Random	VN (/30)	30	30		25		25	
	VR	100%	99%	•	83%	•	83%	•
	IGD	657.1179	701.8627	•	22,821.3747	•	24,091.6980	•
	PD	1.174,550.6969	1,082,824.5819	•	15,592.2866	•	15,703.5480	•
	Runtime	118.1385	126.4740	•	80.0140	○	47.2255	○
RealAmazon	VN (/30)	30	24		30		30	
	VR	100%	1%	•	100%	≈	100%	≈
	IGD	930.7528	69,436.0647	•	2,524.9864	•	2,043.4153	•
	PD	99,155,161.4696	0.0000	•	37,419,788.5432	•	53,111,385.1628	•
	Runtime	5.2790	41.4035	•	25.3040	•	0.7620	○
RealDrupal	VN (/30)	30	30		30		30	
	VR	100%	98%	•	100%	≈	100%	≈
	IGD	395,494.3475	540,523.4180	•	494,246.0175	•	348,347.1508	○
	PD	469,976,187.0195	211,561,691.2596	•	187,016,008.0179	•	309,700,406.6789	•
	Runtime	5.8650	41.4255	•	28.0990	•	0.7525	○

possible explanation for this is that for this model, we use eight objectives rather than five. As the number of objectives increases, the performance of IBEA-based algorithms may degenerate greatly [48, 83].

- Compared to two SIP-based algorithms, SATVaEA can find high-quality solutions, which is evident from the IGD and PD values. It seems that SIP-based algorithms perform well on small-size FMs (note that all SIP FMs, except for the Random one, have features less than 300) but poorly on the large Random model. A most likely explanation for this is that SIP-based algorithms are ineffective in searching for diversified solutions in large decision spaces. As shown in Figure 6, for the Random model, solutions found by both SIP+SPEA2+SDE and SIP+NSGA-II are very similar to each other, whereas those obtained by SATVaEA and SATIBEA are distributed widely in the objective space. Thanks to the DPLL/CDCL-style SAT solvers, both SATVaEA and SATIBEA are able to find a set of diversified products on this random model.
- Compared to SATIBEA and SIP+SPEA2+SDE, SATVaEA is much more time saving. As explained in Section 5.7, SATIBEA needs to calculate the HV indicator whose time complexity increases exponentially to the number of objectives [1, 86]. This, in turn, explains why much

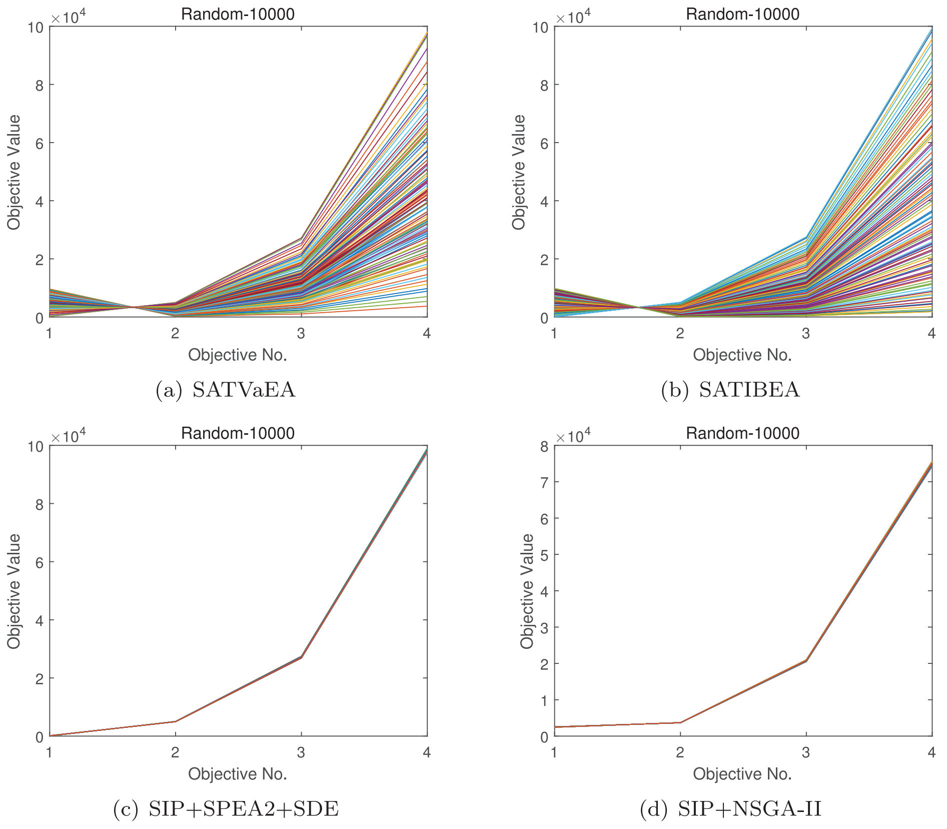


Fig. 6. Final solutions on the Random model, shown by parallel coordinates.

more time is required by SATIBEA on RealAmazon and RealDrupal, for which we consider eight objectives. For SIP+SPEA2+SDE, it maintains an external archive, and solutions in both the current population and the archive are considered when calculating the fitness value of a given individual, leading to the increase of the actual running time. This is consistent with the observations in Hierons et al. [35] that SIP+SPEA2+SDE runs the slowest among all SIP-based algorithms.

Finally, we are going to show how software engineers and end users can benefit from returning diversified valid configurations. For example, as shown in Figure 6, compared to the solutions of SIP+SPEA2+SDE and SIP+NSGA-II on the random FM with 10,000 features [35], those of SATVaEA are distributed more widely in the objective space, providing more choices for software engineers and end users. In practice, one may have various preferences to the software product to be configured. Given that the fourth objective is the cost and some user has a sufficient budget (larger than 10×10^4), she or he can easily find a preferred software configuration from solutions returned by both SATVaEA and SIP-based algorithms. However, in the case in which another user has a budget below 2×10^4, there are no feasible solutions that she or he can choose from those obtained by SIP-based algorithms. In contrast, SATVaEA still provides multiple feasible solutions within that budget for this user. The preceding may well illustrate the merits of returning various diversified solutions. As to how a software engineer or an end user determines the proper trade-off between various solutions, comprehensive evaluation mathematical models can be applied. A simple example would be putting a weight to each objective and then ranking solutions according to the weighted-sum method. Preferences will be given to solutions with lower ranks.

6 RELATED WORKS

The configuration of SPLs can be modeled as a single-objective or a many-objective optimal feature selection problem. We start by describing approaches that used single-objective optimization.

6.1 Single-Objective Optimization

Yeh and Wu [96] proposed a genetic algorithm (GA)-based SPL configuration approach where they defined a cost function to realize the goal of customers who would like to minimize the cost when configuring products. In the proposed approach, the SPL is modeled as a minimum-cost flow problem, where the product configuration network is represented as a flow network. Customized products can be easily obtained by finding the shortest path in the corresponding product configuration network. Their computational results showed that the solution quality of GAs retains 93.89% for a complex configuration problem. One of the disadvantages of this approach is that it does not handle the CTCs.

White et al. [87, 88] addressed the optimal feature selection problem by transforming it to a multidimensional multichoice knapsack problem (MMKP). They developed a polynomial time approach called *filtered Cartesian flattening* (FCF) to generate optimal feature configurations subject to global resource constraints. Their evaluations showed that FCF can find approximate solutions for models with up to 10,000 features in seconds. Guo et al. [29] proposed a GA named *GAFES* to handle the same problem. In GAFES, various weighted objectives are aggregated into a single one by using weights. As a part of the algorithm, they used a repair operator to transform an arbitrary (maybe invalid) configuration into a valid one. The evaluations showed that GAFES outperforms FCF on synthetically generated SPLs; however, the proposed GAFES was not tested on any real-world SPL.

Bagheri and Ensan [2] presented a reliability-aware configuration method that aims to satisfy reliability bounds (lower and upper) when searching for a configuration in an SPL. As one of the search engines, GA was used to handle the optimization objective and functional constraints. This approach was tested on randomly generated SPLs between 1,000 and 10,000 features, and between 5% and 20% of CTCs. It was found that the GA-based approach is useful for larger models (i.e., models with more than 3,000 features).

Müller [61] used a simulated annealing algorithm for selecting features from an SPL. This approach used a value-based portfolio optimization, where the search was guided using a single objective function to maximize the profit defined as a fixed trade-off between revenue and cost. Wang and Pang [85] presented an approach that used ant colony optimization to get an approximation solution of the feature selection optimization problem in polynomial time.

Compared to many-objective optimization, the disadvantage of these approaches is that single-objective search techniques use less information about the features or need to impose weights on each objective to aggregate objectives to form a scalar value.

6.2 Many-Objective Optimization

Recently, EMO algorithms have been widely used to improve the performance and scalability of selecting optimal products in SPLs. Sayyad et al. [75] studied the optimal feature selection problem based on many-objective optimization, and they compared seven EMO algorithms (including NSGA-II [20], IBEA [102], and SPEA2 [103]) on two academic FMs (i.e., Web Portal and E-Shop) with up to 290 features and synthetically generated attributes (i.e., *costs*, *defects*, and *used* *before*). In their evaluations, five optimization objectives were considered. The results showed that IBEA outperforms the other six EMO algorithms with regard to the quality, correctness, and satisfaction to user preferences. However, no valid products were found by IBEA on the model E-Shop after 50,000 evaluations. For IBEA, 50 million evaluations (or nearly 3 hours) were needed to obtain 52% of valid products for the preceding model. In the subsequent work [74], Sayyad et al. evaluated their original IBEA approach on large-scale FMs and found that this approach tended to not generate valid configurations. This led to two enhancements: one was to remove both mandatory and dead features, and the other was to plant a valid product as a seed in the initial population. These heuristics were evaluated on seven FMs from the LVAT repository. They showed that the results were promising, and 30 valid solutions could be found within 30 minutes when configuring the 2.6.28.6-icse11 model with 6,888 features.

Later Sayyad [71] proposed significant improvements to IBEA by employing two strong heuristic techniques: the PUSH method and the PULL method. The PUSH technique forces the evolutionary search to obey certain rules and dependencies defined by FMs, whereas the PULL technique gives a higher weight to the number of violated constraints as an optimization objective. The motivation for the PULL method is that software engineers are only interested in valid products, and therefore the number of violated constraints is deemed as the most important optimization objective.

Olaechea et al. [63] evaluated the guided improvement algorithm (GIA) [68] and IBEA on five FMs, including the two models used by Sayyad et al. [75]. According to their empirical results, the GIA can produce optimal solutions for models with up to 44 features and up to seven objectives in less than 2 hours, but it fails for large models like E-Shop with 290 features. For this model, it took more than 15 days to find at least one Pareto-optimal solution. For IBEA, it can produce approximate solutions with an average accuracy of at least 42% in less than 20 minutes even for the large SPLs with 290 features. However, the application of IBEA requires substantial effort to find the best parameter settings. Guo et al. [30] proposed five GIA-based novel parallel algorithms to effectively solve multiobjective combinatorial optimization (MOCO) problems. Their algorithms search for Pareto-optimal solutions using off-the-shelf solvers, and the search is parallelized via collaborative communication and divide and conquer. They evaluated the proposed algorithms on three software system product line design models, and empirical results demonstrated the feasibility and performance of these parallel algorithms. Recently, Guo et al. [28] suggested a hybrid multiobjective optimization algorithm called *SMTIBEA*, which combines IBEA with satisfiability modulo theories (SMT) solving. The proposed SMTIBEA was used to find optimal or near-optimal solutions that satisfy all predefined constraints and balance multiple often-competing objectives in a huge space of various products. Experimental results on five large, constrained, and real-world SPLs have demonstrated the high performance of the proposed algorithm.

Lian and Zhang [52] proposed the multiobjective optimization algorithm IVEA to optimize the selection of features with both functional requirements and nonfunctional requirements. The proposed IVEA was designed as a polynomial-time algorithm where a two-dimensional fitness function and a violation dominance principle were proposed and used in the environmental and mating selections, respectively. In the two-dimensional fitness function, the first dimension named *infeasibility* measures the rule violations of a configuration, whereas the second dimension calculates user preference for the optimization of multiple objectives. According to their experimental results on two models as in Sayyad et al. [75], IVEA outperformed the other four EMO algorithms in terms of both the quality of solutions and the running speed. However, for a testing scenario with 500,000 evaluations, the algorithm took almost 30 minutes to produce 183 valid configurations for the E-Shop model (with 290 features). The efficiency of the algorithm needs to be improved, especially for large-scale FMs.

Tan et al. [82] presented the feedback-directed mechanism to improve the performance of EMO algorithms for optimal feature selection in SPLs. The idea of this mechanism is that the violated constraints on a chromosome provide an important clue on which features need to be modified. They used this clue as a feedback to guide the mutation and crossover operators. The positions of the features involved in the violation are known as *error positions*. Genes in these positions are mutated with a larger probability than those in the *nonerror positions*, whereas the feedback-directed crossover operator considers values in the nonerror positions to preserve good genes in the offspring. In addition, a preprocessing technique was used to reduce the search space by filtering away the fixed features (i.e., mandatory and dead features) with the help of a SAT solver. The proposed technique was integrated into four EMO algorithms, including IBEA and NSGA-II. As shown by the evaluations, the feedback-directed IBEA performed much better than its competitors. Compared to the unguided IBEA, the feedback-directed IBEA successfully obtained 72.33% and 75% more valid solutions for case studies in SPLOT and LVAT repositories, respectively. By importing the seeding method proposed by Sayyad et al. [74], the feedback-directed IBEA greatly reduced the running time for finding a certain number of valid products. However, the proposed methods found no valid products for the 2.6.28.6-icse11 model. Thus, the scalability of the proposed approach should be well addressed.

Henard et al. [32] developed a search-based SPL feature selection algorithm (i.e., SATIBEA) by combining a many-objective search technique (i.e., IBEA) with a SAT solver, which was used to implement new mutation and replacement operators. The proposed SATIBEA searches for valid products by considering five objectives simultaneously. The algorithm was evaluated on five large real-world SPLs with the number of features ranging from 1,244 to 6,888. The authors' empirical study demonstrated that SATIBEA is a scalable and significant improvement over approaches proposed in Sayyad et al. [74]. In addition, the proposed algorithm does not require any seed.

Hierons et al. [35] proposed the SIP approach for selecting optimal products from SPLs. In SIP, a novel encoding (that shrinks the representation of the problem) and a $1 + n$ approach (that optimizes first on the number of violated constraints and only then on the other objectives) were used to enhance the performance of EMO algorithms. The authors evaluated the SIP method on FMs with realistic attributes, and they found that the proposed method could return valid products on six published FMs and a randomly generated FM with 10,000 features. In addition, their experimental results demonstrated that the performance of the SIP framework was insensitive to which EMO algorithm was used but was highly related to the used encoding methods (i.e., direct encoding, core encoding, or novel encoding) and the adopted optimisation approaches (i.e., $1 + n$ approach or $n + 1$ approach).

Xue et al. [94] proposed a dual-population evolutionary algorithm named *IBED* for the optimal feature selection problem by combining IBEA with the differential evolution (DE) [67] operator. In IBED, the two populations were separately evolved with two different types of evolutionary operators: IBEA operators and DE operators. In addition, two enhancements were employed to improve the performance of the existing EMO algorithm. One is the feedback-directed mechanism and the other is the preprocessing method, which were designed to quickly find valid solutions and to reduce the search space, respectively. By planting three common seeds in the initial population, IBED aided by the preceding two enhancements could obtain 28.6% nondominated valid solutions for the 2.6.28.6-icse11 model in about 40 minutes.

Other search-based techniques used in SPL development phases, such as architectural design, testing, and feature selection, can be found in the latest surveys [31, 54, 62]. It should be noted here that removing mandatory and dead features was widely adopted by many authors [32, 35, 74, 82, 94]. In this article, we employ a similar technique. However, the difference between our method and the existing ones is that the FMs are simplified concerning not only the search space

Table 16. Summary of Representative Works in the SPLE Domain

Work	Year	SAT Solver Used	Style of SAT solver	Largest FM (#)
This article	2017	WalkSAT [10] + Sat4j [8]$^{a+b}$	SLS + DPLL/CDCL	62,482
Hierons et al. [35]	2016	—	—	10,000
Xue et al. [94]	2016	Sat4j [8]^{c}	DPLL/CDCL	6,888
Henard et al. [32]	2015	Sat4j [8]$^{a+b}$	DPLL/CDCL	6,888
Tan et al. [82]	2015	Sat4j [8]^{c}	DPLL/CDCL	6,888
Lian and Zhang [52]	2015	—	—	290
Liang et al. [53]	2015	Sat4j [8]^{a}	DPLL/CDCL	62,482
Olaechea et al. [63]	2014	Z3 SMT [17]^{a}	DPLL/CDCL	290
Guo et al. [30]	2014	Z3 SMT [17]^{a}	DPLL/CDCL	290
Henard et al. [34]	2013	Sat4j [8]^{a}	DPLL/CDCL	94
Sayyad et al. [75]	2013	—	—	290
Sayyad et al. [74]	2013	Z3 SMT [17]^{d}	DPLL/CDCL	6,888
Henard et al. [33]	2013	Sat4j [8]^{a}	DPLL/CDCL	6,888
Johansen et al. [39]	2012	Sat4j [8]^{c}	DPLL/CDCL	6,888
Guo et al. [29]	2011	—	—	10,000
Pohl et al. [66]	2011	PicoSAT [9], Sat4j [8], etc.^{a}	DPLL/CDCL	287
Perrouin et al. [65]	2010	MiniSAT [22] or Zchaff2004 [56]^{a}	DPLL/CDCL	19
Mendonca et al. [59]	2009	Sat4j [8]^{a}	DPLL/CDCL	10,000
White et al. [88]	2008	—	—	5,000
Batory [4]	2005	MiniSAT [22]^{a}	DPLL/CDCL	21

Note: For each work, the table lists the name and style of the used SAT solver (if any), and the number of features for the largest FM evaluated. The purpose of the SAT solver is indicated by four superscript characters—*a*, *b*, *c*, and *d*—which stand for generating (random) solutions, repairing solutions, checking the satisfiability of the constraints, and generating a feature-rich seed, respectively.

(by removing mandatory and dead features) but also the number of constraints. According to Section 5.1, the number of constraints after simplification decreases by 5.3% and 75.4% in the worst and best cases, respectively. On average, it decreases by 27.8%. The decline of constraints is helpful to save time when calculating the number of violated constraints for a given solution.

In the software engineering community, SAT solvers are a common technique used in several works [4, 32, 34, 53, 65, 66, 82, 94]. In these works, the most popular solvers are Sat4j [8] and MiniSAT [22]. However, both of them are based on DPLL/CDCL procedure and are used to design mutation or replacement operators [32], check the satisfiability of the constraints [82, 94], or generate random solutions [34, 65]. A more detailed summary on the use of SAT solvers can be found in Table 16. It is clear from this table that SAT solvers are frequently used in SPLE. In practice, the SAT solvers are found to be easy when analyzing FMs [5, 59] and even for large real-world ones [53]. The authors in Mendonca et al. [59] concluded that the previously reported high efficiency of SAT solvers is not incidental in practice, and SAT instances induced from FMs are easy throughout the spectrum of realistic models. Further, Liang et al. [53] well explained this phenomenon with the discovery that most variables in large real-world FMs are unrestricted—that is, the models are satisfiable for both true and false assignments to such variables under the current partial assignment. Given these, SAT solvers are encouraged to be adopted by researchers to analyze FMs [59] and to configure optimal products for SPLs (e.g., [32, 33, 39]). Actually, there are two mainstreams of high-performance algorithms for solving SAT problems. One is the DPLL/CDCL-style solvers [16], including Chaff [60] and GRASP [57], MiniSAT [22], BerkMin [27], and PicoSAT

[9]. The other is SLS-style solvers, such as GSAT [77], WalkSAT [76], GASAT [45], probSAT [3], and CCEHC [55]. On one hand, DPLL/CDCL-style SAT solvers methodically traverse the search space such that when the procedure terminates, either a satisfiable assignment is found or all possible branches are considered with the conclusion that the problem is unsatisfiable. The SLS-style solvers, on the other hand, are typically greedy algorithms that try to quickly satisfy as many clauses as possible. However, they have no guarantee that a satisfying solution can be found. The SLS-style solvers are also incapable of proving that an instance is unsatisfiable. The DPLL/CDCL-style solvers have exponential worst-case time complexity, whereas SLS-style solvers are usually computationally efficient. Complementary advantages of these two kinds of SAT solvers promote practical applications of them in many fields [80]. According to Table 16, all previous works use DPLL/CDCL-style SAT solvers, and SLS-style solvers catch little attention from the researchers in the SPLE community. This article first proposes using an SLS-style SAT solver to repair invalid solutions when conducting a search for selecting optimal products in SPLs. The evaluations in Section 5.3 demonstrate the effectiveness of this SAT solver. In addition, the SLS-style solver works cooperatively with another DPLL/CDCL-style solver that is introduced for DP. The simultaneous use of two different SAT solvers aims at searching for a set of diversified products as quickly as possible.

Finally, this article evaluates the proposed approach on 14 real-world FMs (and 7 of them have more than 10,000 features), where the largest model has 62,482 features and 273,799 constraints. Concerning the number of both features and constraints, these FMs are significantly larger than previously used ones [32, 35, 75] (see Table 16). In addition, the proposed SATVaEA is tested and compared to other state-of-the-art algorithms on two models with realistic values for feature attributes. Previously, values for feature attributes were mainly generated randomly following the first work of Sayyad et al. [75]. Another merit of SATVaEA is related to its high efficiency. SATVaEA is able to find 100% valid products for small FMs in seconds. Even for extremely large FMs (e.g., with more than 10,000 features), the search takes only a few minutes.

7 CONCLUSIONS AND FUTURE WORK

An FM compactly represents all possible products from an SPL. The product configuration for an SPL involves the selection of a set of optimal features from the corresponding FM [2, 29, 61, 75, 79, 85, 87–89]. Recently, the many-objective feature selection problem has been widely studied in the software engineering community [31, 32, 35, 42, 54, 63, 71–75, 82, 94]. EMO algorithms, such as NSGA-II, IBEA, and SPEA2, have been used to handle the preceding problem. However, as demonstrated in Sayyad et al. [74, 75], pure EMO algorithms did not scale and tended to find inadequate valid products. Therefore, they were enhanced by either a seeding technique [74], or the removal of both mandatory and dead features [74], or SAT-based replacement and mutation operators [32], or a novel encoding scheme combined with a $1 + n$ optimization approach [35].

Inspired by the pioneer works of Sayyad et al. [75], Henard et al. [32], and Hierons [35], this article proposes a new method called *SATVaEA* for the same optimal feature selection problem by combining our previously suggested many-objective optimization algorithm VaEA [92] and two different SAT solvers [8, 10]. In SPLE, although SAT solvers were widely used in many works [4, 32, 34, 53, 65, 66, 82, 94], they were all DPLL/CDCL-style solvers, such as Sat4j [8] and MiniSAT [22]. This article first proposes using an SLS SAT solver to repair invalid configurations when dealing with the many-objective optimal feature selection problem. Moreover, a DPLL/CDCL-style SAT solver (i.e., Sat4j [8]) is introduced to promote diversity of solutions. To the best of our knowledge, the simultaneous use of both SLS- and DPLL/CDCL-style SAT solvers has not been done previously in SPLE.

In prior works on many-objective optimal feature selection [32, 35, 74, 82], an FM was simplified by removing both mandatory and dead features to reduce the search space. In addition, we use the BCP procedure [98] to simplify constraints in this article. Although BCP is a standard step in DPLL/CDCL-style SAT solvers, it is not commonly used inside SLS-style SAT solvers. With the simplification method proposed in this work, both the number of features and the number of constraints have been reduced significantly, which makes sense for the subsequent applications of the two different styles of SAT solvers. On average, according to our results in Section 5.1, the number of features and that of constraints are decreased by 42.5% and 27.8%, respectively.

In SATVaEA, a product is represented by a direct encoding with *true* a feature being selected and *false* deselected. This encoding scheme is easy and can be directly used without considering the structure of different FMs. After an offspring population is generated by applying genetic operations, the parent and offspring populations are merged into a union population. An SLS-style SAT solver is adopted to quickly repair an infeasible solution in the union population (if any), whereas a DPLL/CDLS-style SAT solver is introduced to produce dissimilar solutions by stochastically permuting control parameters of the SAT solver. The use of the two solvers is controlled by a parameter θ. Once the population has been updated by SAT solvers, the environmental selection is invoked to select diversified individuals for the next generation. In this phase, individuals in the union population are first divided into different layers according to the number of validated constraints and then into different sublayers according to Pareto dominance. This ranking method emphasizes individuals with small number of validated constraints and individuals performing well in terms of Pareto dominance. Finally, the critical sublayer can be detected, and individuals in this sublayer are selected one by one according to the maximum-vector-angle-first principle introduced in VaEA [92].

The proposed SATVaEA is evaluated on 14 real-world FMs taken from the LVAT repository, with features ranging from 544 to 62,482. Seven of them have more than 10,000 features, where the largest model has 62,482 features and 273,799 constraints. Concerning the number of features and that of constraints, FMs used in this study are significantly larger than previously used ones [32, 35, 75]. We first show that VaEA is unable to handle the many-objective SPL feature selection problem well, leading to the enhancements based on SAT solvers. The experimental results demonstrate that the fast local search with an SLS-style SAT solver and DP with a DPLL/CDLS-style solver indeed improve the convergence and diversity of the proposed SATVaEA. In addition, the effect of the parameter θ is investigated experimentally, indicating the use of the DPLL/CDLS-style solver should be limited. Since a DPLL/CDLS-style solver is generally more time expensive than an SLS-style solver, more computational resources should be given to the latter to quickly return a large number of valid products. Empirically, the optimal value of θ could be around 0.9. A prominent benefit of SATVaEA over other related approaches is that it does not necessarily require seeds.

Finally, SATVaEA is comprehensively compared to the state-of-the-art SATIBEA [32]. Compared to SATIBEA, experimental results demonstrate that SATVaEA is able to return much more valid products in less execution time. Moreover, the high quality of solutions found by SATVaEA is reflected by performance metrics HV and IGD. The proposed algorithm is further compared to SATIBEA and two SIP-based algorithms [35] on seven FMs taken from Hierons et al. [35], including two models with realistic values for feature attributes. SATVaEA outperforms SATIBEA with respect to the ratio of valid solutions and outperforms SIP-based algorithms with regard to the quality of solutions, especially the diversity. Compared to solutions found by SIP-based algorithms, those obtained by SATVaEA are more diversified. A set of diversified solutions would be easy to meet the various preferences of software engineers and end users. Another advantage of SATVaEA over other state-of-the-art approaches is its high efficiency. SATVaEA could find 100%

valid products for small FMs in seconds and for large models (e.g., with more than 10,000 features) in a few minutes.

There are several research directions for future studies. First, it is possible to combine EMO algorithms with other SAT solvers. An important lesson that can be learned from this work is that SAT solvers are effective when handling the many-objective optimal feature selection problem, and they are encouraged to be used in the future. Currently, Sat4j [8] has widely been used in SPLE. However, the performance of existing approaches using Sat4j is expected to be improved by switching to more competitive SAT solvers, such as PicoSAT [9] and probSAT [3]. It would be also interesting to compare the performance of different SAT solvers. Second, SATVaEA currently uses normal crossover and mutation operators from the evolutionary computation community. In the future, one could use and design problem-specific operators as in Tan et al. [82] and investigate whether or not problem-specific operators outperform the normal ones in the framework of SAT-VaEA. Third, at present, our proposed algorithm handles preferences of software engineers and end users in an a posteriori manner (selection after search). In fact, the preference information (e.g., the budget of a user) can be integrated during the optimization process, which would be helpful to direct the search to products of interest. Finally, feature attributes in this study are all quantitative. In practice, qualitative feature attributes [41] (e.g., the customer's degree of preference [23]) are also important when configuring products from an SPL [64]. As one of our subsequent studies, we will consider both qualitative and quantitative feature properties when solving the optimal feature selection problem in the context of many-objective optimization.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for providing valuable comments to improve this article. We also would like to thank Professor Robert M. Hierons (Brunel University London, UK) and Dr. Sergio Segura (University of Seville, Spain) for their helpful discussions and constructive suggestions.

REFERENCES

[1] Johannes Bader and Eckart Zitzler. 2011. HypE: An algorithm for fast hypervolume-based many-objective optimization. *Evolutionary Computation* 19, 1, 45–76.

[2] Ebrahim Bagheri and Faezeh Ensan. 2014. Reliability estimation for component-based software product lines. *Canadian Journal of Electrical and Computer Engineering* 37, 2, 94–112. DOI: http://dx.doi.org/10.1109/CJECE.2014.2323958

[3] Adrian Balint and Uwe Schöning. 2012. Choosing probability distributions for stochastic local search and the role of make versus break. In *Theory and Applications of Satisfiability Testing—SAT 2012.* Lecture Notes in Computer Science, Vol. 7317. Springer, 16–29

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

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

[6] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wasowski, and Krzysztof Czarnecki. 2010. Variability modeling in the real: A perspective from the operating systems domain. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE'10).* ACM, New York, NY, 73–82. DOI: http://dx.doi.org/10.1145/1858996.1859010

[7] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wasowski, and Krzysztof Czarnecki. 2012. *Variability Modeling in the Systems Software Domain.* Technical Report. Generative Software Development Laboratory, University of Waterloo, Waterloo, Canada.

[8] Daniel Le Berre and Anne Parrain. 2010. The Sat4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation* 7, 59–64.

[9] Armin Biere. 2008. PicoSAT essentials. *Journal on Satisfiability, Boolean Modeling and Computation* 4, 75–97.

[10] Shaowei Cai. 2013. *Faster Implementation for WalkSAT.* Technical Report. Queensland Research Lab, NICTA, Australia.

[11] Francisco Chicano, Darrell Whitley, and Enrique Alba. 2014. Exact computation of the expectation surfaces for uniform crossover along with bit-flip mutation. *Theoretical Computer Science* 545, 76–93.

[12] P. Clements and L. Northrop. 2001. *Software Product Lines: Practices and Patterns.* Addison Wesley Longman.

[13] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. 2007. *Evolutionary Algorithms for Solving Multi-Objective Problems* (2nd ed.). Springer Science + Business Media LLC, New York, NY.

[14] Carlos A. Coello Coello, Gregorio Toscano Pulido, and M. Salazar Lechuga. 2004. Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation* 8, 3, 256–279.

[15] Personal communication with Sergio Segura and Robert M. Hierons. 2017.

[16] Martin Davis, George Logemann, and Donald Loveland. 1962. A machine program for theorem-proving. *Communications of the ACM* 5, 5, 394–397.

[17] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Proceedings of Theory and Practice of Software, the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08/ETAPS'08).* 337–340.

[18] Kalyanmoy Deb. 2001. Multi-objective optimization. In *Multi-Objective Optimization Using Evolutionary Algorithms.* Wiley, 13–46.

[19] Kalyanmoy Deb and Himanshu Jain. 2014. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part I: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation* 18, 4, 577–601.

[20] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and Tamt Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2, 182–197.

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

[22] Niklas Eén and Niklas Sörensson. 2003. An extensible SAT-solver. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing.* 502–518.

[23] Huáscar Espinoza, Hubert Dubois, Sébastien Gérard, Julio Medina, Dorina C. Petriu, and Murray Woodside. 2006. *Annotating UML Models With Non-Functional Properties for Quantitative Analysis.* Springer, Berlin, Germany, 79–90. DOI: http://dx.doi.org/10.1007/11663430_9

[24] M. Farina and P. Amato. 2002. On the optimal solution definition for many-criteria optimization problems. In *Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society.* IEEE, Los Alamitos, CA, 233–238. DOI: http://dx.doi.org/10.1109/NAFIPS.2002.1018061

[25] M. Fleischer. 2003. *The Measure of Pareto Optima Applications to Multi-Objective Metaheuristics.* Springer, Berlin, Germany, 519–533.

[26] Jesús García-Galán, Pablo Trinidad, Omer F. Rana, and Antonio Ruiz-Cortés. 2016. Automated configuration support for infrastructure migration to the cloud. *Future Generation Computer Systems* 55, C, 200–212. DOI: http://dx.doi.org/10.1016/j.future.2015.03.006

[27] Eugene Goldberg and Yakov Novikov. 2007. BerkMin: A fast and robust SAT-solver. *Discrete Applied Mathematics* 155, 12, 1549–1561.

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

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

[30] Jianmei Guo, Edward Zulkoski, Rafael Olaechea, Derek Rayside, Krzysztof Czarnecki, Sven Apel, and Joanne M. Atlee. 2014. Scaling exact multi-objective combinatorial optimization by parallelization. In *Proceedings of the ACM/IEEE International Conference on Automated Software Engineering (ASE'14).* 409–420. DOI: http://dx.doi.org/10.1145/2642937.2642971

[31] Mark Harman, Yue Jia, Jens Krinke, Bill Langdon, Justyna Petke, and Yuanyuan Zhang. 2014. Search based software engineering for software product line engineering: A survey and directions for future work. In *Proceedings of the 18th International Software Product Line Conference—Volume 1 (SPLC'14).* ACM, New York, NY, 5–18. DOI: http://dx.doi.org/10.1145/2648511.2648513

[32] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. 2015. Combining multi-objective search and constraint solving for configuring large software product lines. In *Proceedings of the 37th International Conference on Software Engineering*, Vol. 1. 517–528. DOI: http://dx.doi.org/10.1109/ICSE.2015.69

[33] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. 2013. Assessing software product line testing via model-based mutation: An application to similarity testing. In *Proceedings of the 2013 IEEE*

6th International Conference on Software Testing, Verification and Validation Workshops. 188–197. DOI: http://dx.doi.org/10.1109/ICSTW.2013.30

[34] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. 2013. Multi-objective test generation for software product lines. In *Proceedings of the 17th International Software Product Line Conference (SPLC'13)*. ACM, New York, NY, 62–71. DOI: http://dx.doi.org/10.1145/2491627.2491635

[35] Robert M. Hierons, Miqing Li, Xiaohui Liu, Sergio Segura, and Wei Zheng. 2016. SIP: Optimal product selection from feature models using many-objective evolutionary optimization. *ACM Transactions on Software Engineering and Methodology* 25, 2, Article 17, 39 pages. DOI: http://dx.doi.org/10.1145/2897760

[36] C. A. R. Hoare. 1962. Quicksort. *Computer Journal* 5, 1, 10–16.

[37] Hisao Ishibuchi, Noritaka Tsukamoto, and Yusuke Nojima. 2008. Evolutionary many-objective optimization. In *Proceedings of the 2008 3rd International Workshop on Genetic and Evolving Systems*. 47–52. DOI: http://dx.doi.org/10.1109/GEFS.2008.4484566

[38] Himanshu Jain and Kalyanmoy Deb. 2014. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: Handling constraints and extending to an adaptive approach. *IEEE Transactions on Evolutionary Computation* 18, 4, 602–622.

[39] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. 2012. An algorithm for generating *t*-wise covering arrays from large feature models. In *Proceedings of the 16th International Software Product Line Conference—Volume 1*. ACM, New York, NY, 46–55.

[40] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. CMU/SEI-90-TR-21. Georgetown University.

[41] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. 1998. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering* 5, 1, 143. DOI: http://dx.doi.org/10.1023/A:1018980625587

[42] Ahmet Serkan Karataş, Halit Oğuztüzün, and Ali Doğru. 2013. From extended feature models to constraint logic programming. *Science of Computer Programming* 78, 12, 2295–2312.

[43] P. Knauber, J. Bermejo, G. Böckle, J. C. S. do Prado Leite, F. van der Linden, L. Northrop, M. Stark, and D. M. Weiss. 2002. Quantifying product line benefits. In *Software Product-Family Engineering*. Lecture Notes in Computer Science, Vol. 2290. Springer, 155–163. DOI: http://dx.doi.org/10.1007/3-540-47833-7_15

[44] Joshua Knowles and David Corne. 1999. The Pareto archived evolution strategy: A new baseline algorithm for Pareto multiobjective optimisation. In *Proceedings of the Congress on Evolutionary Computation—Volume 1 (CEC'99)*. IEEE, Los Alamitos, CA, 98–105.

[45] Frédéric Lardeux, Frédéric Saubion, and Jin Kao Hao. 2006. GASAT: A genetic local search algorithm for the satisfiability problem. *Evolutionary Computation* 14, 2, 223–253.

[46] Bingdong Li, Jinlong Li, Ke Tang, and Xin Yao. 2015. Many-objective evolutionary algorithms: A survey. *ACM Computing Surveys* 48, 1, 1–35.

[47] Ke Li, Kalyanmoy Deb, Qingfu Zhang, and Sam Kwong. 2015. An evolutionary many-objective optimization algorithm based on dominance and decomposition. *IEEE Transactions on Evolutionary Computation* 19, 5, 694–716.

[48] Miqing Li, Shengxiang Yang, and Xiaohui Liu. 2014. Diversity comparison of Pareto front approximations in many-objective optimization. *IEEE Transactions on Cybernetics* 44, 12, 2568–2584.

[49] Miqing Li, Shengxiang Yang, and Xiaohui Liu. 2014. Shift-based density estimation for Pareto-based algorithms in many-objective optimization. *IEEE Transactions on Evolutionary Computation* 18, 3, 348–365

[50] Miqing Li, Shengxiang Yang, and Xiaohui Liu. 2015. Bi-goal evolution for many-objective optimization problems. *Artificial Intelligence* 228, 45–65.

[51] Miqing Li, Liangli Zhen, and Xin Yao. 2017. How to read many-objective solution sets in parallel coordinates. *IEEE Computational Intelligence Magazine* 12, 4, 88–100.

[52] Xiaoli Lian and Li Zhang. 2015. Optimized feature selection towards functional and non-functional requirements in software product lines. In *Proceedings of the 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 191–200. DOI: http://dx.doi.org/10.1109/SANER.2015.7081829

[53] Jia Hui Liang, Vijay Ganesh, Krzysztof Czarnecki, and Venkatesh Raman. 2015. SAT-based analysis of large real-world feature models is easy. In *Proceedings of the 19th International Conference on Software Product Line (SPLC'15)*. ACM, New York, NY, 91–100. DOI: http://dx.doi.org/10.1145/2791060.2791070

[54] Roberto E. Lopez-Herrejon, Lukas Linsbauer, and Alexander Egyed. 2015. A systematic mapping study of search-based software engineering for software product lines. *Information and Software Technology* 61, 33–51.

[55] Chuan Luo, Shaowei Cai, Kaile Su, and Wenxuan Huang. 2017. CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence* 243, 26–44.

[56] Yogesh S. Mahajan, Zhaohui Fu, and Sharad Malik. 2005. Zchaff2004: An efficient SAT solver. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT'04).* 360–375. DOI: http://dx.doi.org/10.1007/11527695_27

[57] J. P. Marques-Silva and K. A. Sakallah. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers* 48, 5, 506–521.

[58] Marcilio Mendonca, Moises Branco, and Donald Cowan. 2009. SPLOT: Software product lines online tools. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications.* ACM, New York, NY, 761–762.

[59] Marcilio Mendonca, Andrzej Wasowski, and Krzysztof Czarnecki. 2009. SAT-based analysis of feature models is easy. In *Proceedings of the 13th International Software Product Line Conference (SPLC'09).* 231–240.

[60] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Conference on Design Automation (DAC'01).* 530–535.

[61] Johannes Müller. 2011. Value-based portfolio optimization for software product lines. In *Proceedings of the 2011 15th International Software Product Line Conference.* 15–24. DOI: http://dx.doi.org/10.1109/SPLC.2011.18

[62] Lina Ochoa, Juliana Alves Pereira, Oscar González-Rojas, Harold Castro, and Gunter Saake. 2017. A survey on scalability and performance concerns in extended product lines configuration. In *Proceedings of the 11th International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS'17).* ACM, New York, NY, 5–12. DOI: http://dx.doi.org/10.1145/3023956.3023959

[63] Rafael Olaechea, Derek Rayside, Jianmei Guo, and Krzysztof Czarnecki. 2014. Comparison of exact and approximate multi-objective optimization for software product lines. In *Proceedings of the International Software Product Line Conference.* 92–101.

[64] Juliana Alves Pereira, Lucas Maciel, Thiago F. Noronha, and Eduardo Figueiredo. 2017. Heuristic and exact algorithms for product configuration in software product lines. *International Transactions in Operational Research* 24, 6, 1285–1306.

[65] Gilles Perrouin, Sagar Sen, Jacques Klein Benoit Baudry, and Yves le Traon. 2010. Automated and scalable T-wise test case generation strategies for software product lines. In *Proceedings of the IEEE 6th International Conference on Software Testing, Verification, and Validation.* 459–468. DOI: http://dx.doi.org/doi.ieeecomputersociety.org/10.1109/ICST.2010.43

[66] Richard Pohl, Kim Lauenroth, and Klaus Pohl. 2011. A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering.* 313–322.

[67] Kenneth Price, Rainer M. Storn, and Jouni A. Lampinen. 2005. *Differential Evolution: A Practical Approach to Global Optimization.* Natural Computing Series. Springer-Verlag, Secaucus, NJ.

[68] Derek Rayside, H. Christian Estler, and Daniel Jackson. 2009. *The Guided Improvement Algorithm for Exact, General-Purpose, Many-Objective Combinatorial Optimization.* Technical Report MIT-CSAIL-TR-2009-033. Massachusetts Institute of Technology, Cambridge, MA.

[69] Ana B. Sánchez, Sergio Segura, José A. Parejo, and Antonio Ruiz-Cortés. 2017. Variability testing in the wild: The Drupal case study. *Software and Systems Modeling* 16, 1, 173–194.

[70] D. K. Saxena, J. A. Duro, A. Tiwari, K. Deb, and Q. Zhang. 2013. Objective reduction in many-objective optimization: Linear and nonlinear algorithms. *IEEE Transactions on Evolutionary Computation* 17, 1, 77–99.

[71] Abdel Salam Sayyad. 2014. *Evolutionary Search Techniques With Strong Heuristics for Multi-Objective Feature Selection in Software Product Lines.* Ph.D. Dissertation. West Virginia University.

[72] Abdel Salam Sayyad, Katerina Goseva-Popstojanova, Tim Menzies, and Hany Ammar. 2013. On parameter tuning in search based software engineering: A replicated empirical study. In *Proceedings of the International Workshop on Replication in Empirical Software Engineering Research.* 84–90.

[73] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. 2013. Optimum feature selection in software product lines: Let your model and values guide your search. In *Proceedings of the International Workshop on Combining Modelling and Search-Based Software Engineering.* 22–27.

[74] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. 2013. Scalable product line configuration: A straw to break the camel's back. In *Proceedings of the 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE'13).* 465–474. DOI: http://dx.doi.org/10.1109/ASE.2013.6693104

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

[76] Bart Selman, Henry A. Kautz, and Bram Cohen. 1994. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Intelligence—Volume 1 (AAAI'94).* 337–343. http://dl.acm.org/citation.cfm?id=199288.178090

[77] Bart Selman, Hector Levesque, and David Mitchell. 1992. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92)*. 440–446.

[78] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki. 2011. Reverse engineering feature models. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*. ACM, New York, NY, 461–470. DOI: http://dx.doi.org/10.1145/1985793.1985856

[79] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. 2012. SPL conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal* 20, 3, 487–517. DOI: http://dx.doi.org/10.1007/s11219-011-9152-9

[80] Ali Asgar Sohanghpurwala, Mohamed W. Hassan, and Peter Athanas. 2017. Hardware accelerated SAT solvers—a survey. *Journal of Parallel and Distributed Computing* 106, 170–184. DOI: http://dx.doi.org/10.1016/j.jpdc.2016.12.014

[81] M. Srinivas and Lalit M. Patnaik. 1994. Genetic algorithms: A survey. *Computer* 27, 6, 17–26.

[82] Tian Huat Tan, Yinxing Xue, Manman Chen, Jun Sun, Yang Liu, and Jin Song Dong. 2015. Optimizing selection of competing features via feedback-directed evolutionary algorithms. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA'15)*. 246–256.

[83] Tobias Wagner, Nicola Beume, and Boris Naujoks. 2007. Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In *Evolutionary Multi-Criterion Optimization*. Lecture Notes in Computer Science, Vol. 4403. Springer, 742–756.

[84] Handing Wang, Yaochu Jin, and Xin Yao. 2016. Diversity assessment in many-objective optimization. *IEEE Transactions on Cybernetics* PP, 99, 1–13. DOI: http://dx.doi.org/10.1109/TCYB.2016.2550502

[85] Yinglin Wang and Jinwei Pang. 2014. Ant colony optimization for feature selection in software product lines. *Journal of Shanghai Jiaotong University (Science)* 19, 1, 50–58.

[86] Lyndon While, Lucas Bradstreet, and Luigi Barone. 2012. A fast way of calculating exact hypervolumes. *IEEE Transactions on Evolutionary Computation* 16, 1, 86–95.

[87] Jules White, Brian Dougherty, and Douglas C. Schmidt. 2009. Selecting highly optimal architectural feature sets with filtered Cartesian flattening. *Journal of Systems and Software* 82, 8, 1268–1284. DOI: http://dx.doi.org/10.1016/j.jss.2009.02.011

[88] Jules White, Brian Doughtery, and Douglas C. Schmidt. 2008. Filtered Cartesian flattening: An approximation technique for optimally selecting features while adhering to resource constraints. In *Proceedings of the 12th International Software Product Lines Conference—Volume 2 (SPLC'08)*. 209–216.

[89] Jules White, José A. Galindo, Tripti Saxena, Brian Dougherty, David Benavides, and Douglas C. Schmidt. 2014. Evolving feature model configurations in software product lines. *Journal of Systems and Software* 87, 119–136. DOI: http://dx.doi.org/10.1016/j.jss.2013.10.010

[90] Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics Bulletin* 1, 6, 80–83.

[91] Yi Xiang, Jing Peng, Yuren Zhou, Miqing Li, and Zefeng Chen. 2017. An angle based constrained many-objective evolutionary algorithm. *Applied Intelligence* 47, 3, 705–720. DOI: http://dx.doi.org/10.1007/s10489-017-0929-9

[92] Yi Xiang, Yuren Zhou, Miqing Li, and Zefeng Chen. 2017. A vector angle based evolutionary algorithm for unconstrained many-objective problems. *IEEE Transactions on Evolutionary Computation* 21, 1, 131–152. DOI: http://dx.doi.org/10.1109/TEVC.2016.2587808

[93] Yi Xiang, Yuren Zhou, and Hailin Liu. 2015. An elitism based multi-objective artificial bee colony algorithm. *European Journal of Operational Research* 245, 1, 168–193.

[94] Yinxing Xue, Jinghui Zhong, Tian Huat Tan, Yang Liu, Wentong Cai, Manman Chen, and Jun Sun. 2016. IBED: Combining IBEA and DE for optimal feature selection in software product line engineering. *Applied Soft Computing* 49, 1215–1231. DOI: http://dx.doi.org/10.1016/j.asoc.2016.07.040

[95] Shengxiang Yang, Miqing Li, Xiaohui Liu, and Jinhua Zheng. 2013. A grid-based evolutionary algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation* 17, 5, 721–736.

[96] Jinn-Yi Yeh and Tai-Hsi Wu. 2005. Solutions for product configuration management: An empirical study. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 19, 1, 39–47.

[97] Z. Michalewicz. 2013. *Genetic Algorithms+ Data Structures = Evolution Programs*. Springer Science & Business Media.

[98] Ramin Zabih and David McAllester. 1988. A rearrangement search strategy for determining propositional satisfiability. In *Proceedings of the 7th AAAI National Conference on Artificial Intelligence (AAAI'88)*. 155–160.

[99] Qingfu Zhang and Hui Li. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation* 11, 6, 712–731.

[100] Qingfu Zhang, Aimin Zhou, Shizheng Zhao, Ponnuthurai Nagaratnam Suganthan, Wudong Liu, and Santosh Tiwari. 2009. Multiobjective optimization test instances for the CEC 2009 special session and competition. In *Proceedings of the Special Session on Performance Assessment of Multi-Objective Optimization Algorithms*.

[101] Y. Zhou, Z. Chen, and J. Zhang. 2017. Ranking vectors by means of the dominance degree matrix. *IEEE Transactions on Evolutionary Computation* 21, 1, 34–51.

[102] Eckart Zitzler and Simon Künzli. 2004. Indicator-based selection in multiobjective search. In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*. 832–842.

[103] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. 2001. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm.* Technical Report. Computer Engineering and Networks Laboratory, Department of Electrical Engineering, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.

[104] Eckart Zitzler and Lothar Thiele. 1999. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation* 3, 4, 257–271.

Received March 2017; revised November 2017; accepted December 2017