Applied Soft Computing xxx (2016) xxx–xxx

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

# IBED: Combining IBEA and DE for optimal feature selection in software product line engineering

Yinxing Xue [a], Jinghui Zhong [a,b,*], Tian Huat Tan [c], Yang Liu [a], Wentong Cai [a], Manman Chen [c], Jun Sun [c]

[a] Nanyang Technological University, 50 Nanyang Avenue, 639798, Singapore
[b] School of Computer Science and Engineering, South China University of Technology, China
[c] Singapore University of Technology & Design, 8 Somapah Road, 487372, Singapore

ARTICLE INFO

*Article history:*
Received 9 December 2015
Received in revised form 9 July 2016
Accepted 24 July 2016
Available online xxx

*Keywords:*
Optimal feature selection
Indicator-based evolutionary algorithm (IBEA)
Differential evolutionary algorithm (DE)
Software product line engineering

ABSTRACT

Software configuration, which aims to customize the software for different users (e.g., Linux kernel configuration), is an important and complicated task. In software product line engineering (SPLE), feature oriented domain analysis is adopted and feature model is used to guide the configuration of new product variants. In SPLE, product configuration is an optimal feature selection problem, which needs to find a set of features that have no conflicts and meanwhile achieve multiple design objectives (e.g., minimizing cost and maximizing the number of features). In previous studies, several multi-objective evolutionary algorithms (MOEAs) were used for the optimal feature selection problem and indicator-based evolutionary algorithm (IBEA) was proven to be the best MOEA for this problem. However, IBEA still suffers from the issues of correctness and diversity of found solutions. In this paper, we propose a dual-population evolutionary algorithm, named IBED, to achieve both correctness and diversity of solutions. In IBED, two populations are individually evolved with two different types of evolutionary operators, i.e., IBEA operators and differential evolution (DE) operators. Furthermore, we propose two enhancement techniques for existing MOEAs, namely the feedback-directed mechanism to fast find the correct solutions (e.g., solutions that satisfy the feature model constraints) and the preprocessing method to reduce the search space. Our empirical results have shown that IBED with the enhancement techniques can outperform several state-of-the-art MOEAs on most case studies in terms of correctness and diversity of found solutions.


## 1. Introduction

Software product line engineering (SPLE) is a new software development paradigm for a set or a family of similar products [1–3]. Relying on much similarity existing among software products [4,5], SPLE improves software productivity and quality. The basic idea of SPLE is to manage the similar products based on software reuse. In last two decades, SPLE has been an active research area in software engineering [6–9]. The motivation of SPLE originates from the various user requirements, as most of the time companies need to manage multiple variants of the same software for different users. All these product variants have the commonality but each of them meanwhile still owns some variability that is specific to user preference [10]. This fact motivates the reuse of features. Systematic reuse avoids the wasteful duplication of effort, reduces the development and maintenance costs, shortens the time-to-market, and improves the overall quality of software [11].

In software family or software product line (SPL), *feature model* is widely used to model the common and different parts among various product variants [1]. From the perspective of the internal developers, a *feature* is often defined as a program function that realizes a group of individual relevant requirements [12]. In contrast, for external users, a feature is usually defined as a visible value, quality, or characteristic of software [13,14]. Based on the feature model, different features are carefully selected to meet the requirements of users and to avoid possible feature conflict problems. In the era of a thriving market of mobile and service-based applications, vendors are required to configure their applications promptly for new users, to expand the market share. Therefore, it is desirable to automatically configure products that satisfy various users and avoid feature conflicts.

A *feature model* provides a representation of features in an SPL, which could be used to facilitate the reasoning and configuration of

* Corresponding author at: School of Computer Science and Engineering, South China University of Technology, China.
*E-mail address:* jinghuizhong@gmail.com (J. Zhong).

http://dx.doi.org/10.1016/j.asoc.2016.07.040
1568-4946/

the SPL [14]. Common SPLs consist of hundreds or even thousands of features. For instance, as reported in [15], the Linux X86 kernel contains 6888 features, and 343,944 constraints. In addition, the features are usually associated with quality attributes such as cost and reliability. It is hard for the vendor to select a set of features that comply with the feature model, and meanwhile optimize the quality attributes according to user preferences. This complexity provides challenges for the reasoning and configuration of feature models. This is called the *optimal feature selection* problem [16].

Existing works [10,16–18] have adopted evolutionary algorithms (EAs) for feature selection with resource constraints and generate the product based on the feature property value of user preferences, respectively. Guo et al. [16] proposed a genetic algorithm (GA) approach for tackling the optimal feature selection problem. In their work, a repair operator is used to fix each candidate solution, so that it is fully compatible with the feature model after each round of crossover and mutation operations. This approach might be non-terminating, and furthermore, it does not take advantage of the automatic correction that is brought by the GA. In addition, GA combines all objectives into a single fitness function with respective weights. This only gives users a solution that is specific to the weights used in the objective formula.

To address this problem, Sayyad et al. [10,17] proposed an approach that uses EAs to support multi-objective optimization. EAs usually return a range of optimal solutions (i.e., a Pareto front) to the user as a result. They investigated seven EAs and discovered that the indicator-based evolutionary algorithm (IBEA) [19] yields the best results among the seven tested EAs in terms of time, correctness (i.e., the validity of the product in satisfying requirement constraints) and satisfaction to user preferences. After that, several approaches have been proposed to enhance the IBEA. Sayyad et al. [18] also used a static method of feature pruning before execution of IBEA to reduce the search space. They further introduced a "seeding method", which feeds the IBEA with a pre-computed correct solution, to help IBEA produce more correct solutions by mutating the given seed.

In our previous work [20], we introduce a novel feedback-directed mechanism for existing EAs to improve the correctness of the found solutions. Feedback-directed mechanism is inspired by the testing strategy that generates new test cases by the feedback (i.e., results) of the tested ones. In our feedback-directed mechanism, during each round of executing EA, the violated constraints are analyzed. The analyzed results are used as feedback to guide evolutionary operators (i.e., crossover and mutation) for producing solutions for the next round. Besides, to improve the efficiency of searching, before the execution of an EA, the feature model is first preprocessed via propositional satisfiability (SAT) solving to remove the core (or dead) features that must be selected (or deselected) [9]. We have shown that we always prune more features than the pruning method proposed by Sayyad et al. [18].

Existing EA-based approaches mainly focus on finding correct solutions and IBEA is one of the most powerful approaches that can obtain solutions with high correctness ratio [20]. However, IBEA may produce many duplicated solutions in the evolving population. To address this issue, in this paper, we extend IBEA with the idea of differential evolution (DE) for the purpose of finding more *unique* and *non-dominated* correct solutions. A correct solution is called non-dominated or Pareto optimal, if none of the objective functions can be improved in value without degrading any other objective values.

The key idea of our approach, named IBED (to indicate the meaning of the combination of **IBE**A and **D**E), is to evolve dual populations with different evolution mechanisms. The first population is evolved by using the traditional IBEA operators that were also used in our previous work [20]. The population focuses on obtaining enough correct solutions as well as maintaining population diversity. Meanwhile, the second population is evolved by using operators extended from the differential evolution (DE). We choose DE operators to evolve the second population because these operators have been shown quite effective to maintain population diversity and very efficient to search the optimal solution [21]. The offspring (i.e., the newly created solutions) generated by one population are shared with the other population in order to improve the search efficiency. To evaluate our proposed method, both SPLOT [22] and LVAT [23] repositories are utilized. SPLOT is a repository of feature models used by many researchers as a benchmark, and LVAT contains the real-world feature models which have large feature sizes, including the aforementioned Linux X86 kernel model that contains 6888 features.

Our main contributions are summarized below.

1. We introduce IBED – a dual-population evolutionary algorithm. The basic idea is to combine one IBEA population and one DE population to achieve both the correct and non-dominated solutions.
2. We propose the enhancement techniques – the preprocessing method and the feedback-directed mechanism for existing EAs. The preprocessing method is used to prune the features and reduce the problem space. The feedback-directed mechanism helps to find correct solutions. In a feedback-directed EA, solutions are analyzed by their violated constraints. The analysis result is used as feedback for EA operators to produce offspring that are more likely to satisfy more constraints.
3. We use the seeding method proposed by Sayyad et al. [18] to find valid solutions on the feature model *Linux X86*, which has 6888 features. We find combining IBED with the seeding method significantly shortens the search time, compared with the original seeding method proposed by Sayyad et al. [18].
4. We evaluate IBED with feature models that are available publicly. Our IBED is able to find more unique and non-dominated solutions than the state-of-the-art algorithm IBEA [19], meanwhile retaining a comparable correctness rate with IBEA.

Among the claimed contributions, we firstly extend the work [20] by introducing the novel IBED and comparing it with IBEA. Secondly, we present the enhancement techniques, which has been evaluated for existing EAs in previous work [20], and evaluate how they can improve IBED. Thirdly, for the scalability, we combine IBED and the seeding method to find correct solutions for the largest feature model *Linux X86*. Last, to the best knowledge of ours, for the SPL problem, it is the first attempt to check the diversity via the dominance relation between the solutions found by two EAs (i.e., IBED and IBEA), respectively. Knowing the dominance relation of found solutions (see Table 6) is more informative than merely using the common quality indicators such as hypervolume and spread [10,17,18]. Hence, except the second contribution from [20], the other three are original contributions of this paper. We have published the code and the experimental data for public review[1].

Section 2 introduces the background of this work. Section 3 explains the steps of IBED. Section 4 presents the preprocessing method for feature pruning and the feedback-directed mechanism in detail. Section 5 provides the evaluation of our approach. Section 6.4 reviews the related work. Finally, Section 7 concludes and outlines future work.

[1] The code and data can be downloaded from this link: https://sites.google.com/site/yinxingxue/home/projects/ibed.

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040

## 2. Background

In this section, we provide the background knowledge on SPL, feature model, and multi-objective optimization problem.

### 2.1. Software product line

SPLE is feature-oriented software development paradigm, as SPLE adopts feature-oriented domain analysis for requirements analysis and builds the architecture for reuse [4,14]. SPLE is a two-phase approach composed of domain engineering and application engineering [24]. The task of domain engineering is to build the SPL architecture consisting of the codebase and the variant features, while the application engineering focuses on derivation of new products by different customizations of variant features applied onto the codebase [25]. The *codebase* refers to the same code shared by all the product variants, which is the implementation of the basic functionality of a software family (a set of similar products) [2]. *Variant features*, which are different extra functions, are used to satisfy the needs of various users.

To satisfy the feature constraints and user preferences, it is impossible to enumerate all product variants and analyze them individually due to the large search space of feature selection [26]. Hence, automation of processing and verification of product configuration (the configuration for feature selection) is a fundamental problem in SPLE because of its importance and complexity [16]. Exploring an efficient and scalable approach for the optimal feature selection problem, which helps to reduce the development costs and shorten the time-to-market of products, is important to the success of SPLE [18].

### 2.2. Feature model and product configuration

The concept of feature model in domain engineering is to represent the features and their relationships [14,27]. Since the proposal of SPL, feature model has been characterized as "the greatest contribution of domain engineering to software engineering" [28].

A feature model is a tree-like hierarchy of features [1,29,30]. The structural and semantic relationships between a parent feature (or compound) and its child features (or sub-features) can be specified as [1,29]:

- *Alternative* – if the parent feature is selected, exactly one among the exclusive sub-features must be selected.
- *Or* – if the parent feature is selected, at least one of the sub-features must be selected.
- *Mandatory* – a mandatory feature must be selected if its parent is selected.
- *Optional* – an optional feature is optional to be selected.

Besides the above structure or parental relationships between features, cross-tree constraints (CTCs) are also often adopted to represent the mutual relationship for features across the feature model. There are three types of common CTCs [29]:

- $f_a$ *requires* $f_b$ – the inclusion of feature $f_a$ implies the inclusion of feature $f_b$ in the same product.
- $f_a$ *excludes* $f_b$ – the inclusion of feature $f_a$ implies the exclusion of feature $f_b$ in the same product, and vice versa.
- $f_a$ *iff* $f_b$ – the inclusion of feature $f_a$ implies the inclusion of feature $f_b$ in the same product, and vice versa.

In Fig. 1, the feature model of a Java Chat System (*JCS*) is illustrated. The root feature of the feature model is *Chat*, which has a mandatory sub-feature (i.e., *Output*) and several optional sub-features (e.g., *Encryption*). Since the feature *Output* is mandatory, exactly one of its sub-features (i.e., *GUI*, *CMD*, and *GUI2*) must be selected. In addition, if the *Encryption* feature is selected, at least one of its sub-features (i.e., *Caesar* and *Reverse*) needs to be selected. There is a CTC for *JCS* which is of the form $f_a$ *iff* $f_b$ – *Encryption_OR* is selected if and only if *Caesar* or *Reverse* is selected.

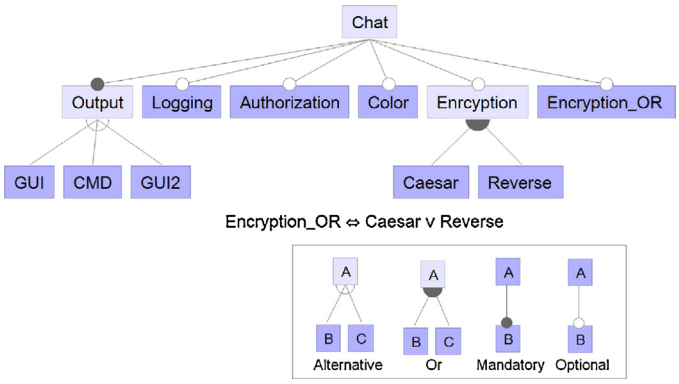


**Fig. 1.** The feature model of *JCS*.

The feature model listed in Fig. 1 can be represented by the constraints that are listed in Table 1. The constraints are specified according to the semantics of feature model. Constraint c(1) specifies that the root feature must be present, to prevent a trivial feature model with no selected feature. Constraint c(2) specifies the mandatory feature *Output*, and constraints c(3)–c(7) specify constraints on the other five optional sub-features. The sub-features of *Output* are in an *Alternative* relationship. Constraints c(8)–c(11) together denote that exactly one of the sub-features must be selected if *Output* is selected. Constraint c(8) states that *Output* is selected, if and only if at least one of *CMD*, *GUI* and *GUI2* is selected; but constraints c(9)–c(11) additionally define the mutual exclusiveness of *CMD*, *GUI* and *GUI2*, that is, at most one of them could be chosen. The sub-features of *Encryption* are in *Or* relationship. The constraint c(12) denotes if the feature *Encryption* is selected, then at least one feature from *Caesar* and *Reverse* needs to be selected, and vice versa. The only CTC of *JCS* is captured in the constraint c(13). Constraints c(1)–c(12) are called *tree constraints*, since they are related to the tree structure of the feature model. Hence, given a feature model $M$, we refer tree constraints and CTCs of the $M$, as the *constraints* of $M$. We denote the conjunction of constraints of $M$ as $conj(M)$. We use $Fea(M)$ to denote the set of entire features of the feature model $M$. For the *JCS* example, $Fea(JCS) = \{Chat, \ldots\}$ and $|Fea(JCS)|$=12.

**Definition 1** *(Valid product configuration).* Given a feature model $M$, a valid product configuration (or valid product) for $M$ is a non-empty feature set $F \subseteq Fea(M)$, such that $F$ satisfies the constraints of $M$.

**Table 1**
Constraints of *JCS*.

| | |
|---|---|
| $Chat$ | c(1) |
| $Output \Leftrightarrow Chat$ | c(2) |
| $Logging \Leftrightarrow Chat$ | c(3) |
| $Authorization \Leftrightarrow Chat$ | c(4) |
| $Color \Leftrightarrow Chat$ | c(5) |
| $Encryption \Leftrightarrow Chat$ | c(6) |
| $Encryption\_OR \Leftrightarrow Chat$ | c(7) |
| $(GUI \vee CMD \vee GUI2) \Leftrightarrow Output$ | c(8) |
| $\neg(GUI \wedge CMD)$ | c(9) |
| $\neg(GUI \wedge GUI2)$ | c(10) |
| $\neg(CMD \wedge GUI2)$ | c(11) |
| $(Caesar \vee Reverse) \Leftrightarrow Encryption$ | c(12) |
| $Encryption\_OR \Leftrightarrow (Caesar \vee Reverse)$ | c(13) |

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040

We write $F \models M$ if $F \subseteq Fea(M)$ is a valid product of the feature model $M$.

Consider the feature model *JCS* as an example, the feature set {*Chat*, *Output*, *GUI*} (denoted as $F$) is a valid product of *JCS* ($F \models JCS$).

FODA [14] already proposed the inclusion of some additional information of features in feature models. For instance, relationships between features and feature attributes were introduced. Later, Kang et al. [31] explicitly proposed the concept of extended, advanced or attributed feature models and referred to feature attributes as "non-functional" properties. Benavides et al. [32] also suggested that a feature attribute should consist of a name, a domain and a value. Extended feature models can also include complex constraints among feature attributes [9]. In this paper, we mainly consider feature attributes such as cost, defect and the times for which the feature has been used (see Section 5.1.4).

### 2.3. Multi-objective optimization problem

Many real-world problems have multiple objectives that need to be optimized simultaneously. However, these objectives usually conflict with each other, which prevents optimizing all objectives simultaneously. A remedy is to have a set of optimal trade-offs between the conflicting objectives.

A $k$-objective optimization problem could be written in the following form:

$$\begin{aligned} &Minimize \quad Obj(F) = (Obj_1(F), Obj_2(F), \ldots, Obj_k(F)) \\ &subject \quad to \quad F \models M \end{aligned} \tag{1}$$

where $Obj(F)$ is a $k$-dimensional objective vector for $F$ and $Obj_i(F)$ is the value of $F$ for the $i$th objective.

Given $F_1, F_2 \models M$, $F_1$ can be viewed as better than $F_2$ for the minimization problem in Eq. (1), if Eq. (2) holds.

$$\forall i : Obj_i(F_1) \leq Obj_i(F_2) \wedge \exists j : Obj_j(F_1) < Obj_j(F_2) \tag{2}$$

where $i, j \in \{1, \ldots, k\}$.

In this case, we say that $F_1$ *dominates* $F_2$. $F_1$ is called a *Pareto-optimal solution* if $F_1$ is not dominated by any other $F \models M$. We denote all Pareto-optimal solutions as the *Pareto front*.

Many evolutionary algorithms (e.g., IBEA [19], NSGA-II [33], ssNSGA-II [34], MOCell [35]) are proposed to find a set of *non-dominated solutions* that approximate the Pareto front for solving the multi-objective optimization problem.

For the optimal feature selection problem, Sayyad used MOEAs for the first time to provide solutions [36]. Our work also addresses this problem and aims at finding correct solutions that approximate the optimal Pareto front via MOEAs. Our goal is to improve the correctness ratio of solutions found by MOEAs, meanwhile achieving a good diversity of correct non-dominated solutions.

## 3. Our dual-population evolutionary algorithm combining IBEA and DE Operators

This section describes the proposed dual-population EA (i.e., IBED) to address the optimal feature selection problem. First, a brief introduction on EA and IBEA is given. Then, the general framework of IBED and the implementation of each step of the framework are presented.

### 3.1. Preliminaries of evolutionary algorithm

EAs, inspired by the "survival of the fittest" principle of the Darwinian theory of natural evolution, are stochastic search methods based on principles of the biological evolution [37]. By applying the EA, a problem is encoded into a simple chromosome-like data structure, and then evolutionary operators (e.g., selection, crossover, and mutation) are applied on these data structures to preserve "the fittest" information, which is analogous to "survival of the fittest" in the natural world [37]. EAs often perform well in approximating solutions, and therefore EAs are typically suitable for the optimization problems especially if the search space of the problem is large and complex [37].

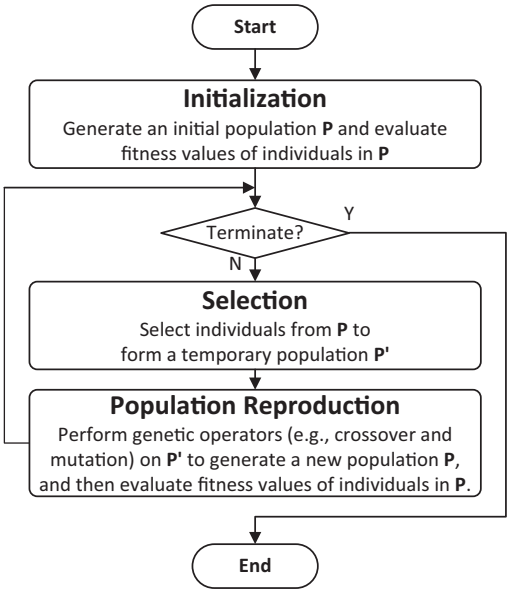


**Fig. 2.** The general procedure of EA.

The common procedure of EAs is illustrated in Fig. 2, which consists of three main steps. The first step is to generate an initial population of chromosomes (labeled as $P$). Each individual in the population represents a candidate solution of the problem and is randomly generated. After the random initial individuals are generated, their fitness values are evaluated. Then, in the second step, a specific selection strategy (e.g., tournament selection) is used to select chromosomes from $P$ to form a temporary population $P'$. After that, the third step applies genetic operators such as crossover and mutation on $P'$ to generate a new population $P$. The fitness values of individuals in $P$ are also evaluated in this step. The second step and the third step are repeated until the termination condition is met. The termination conditions are set specifically according to the encountered problems and the preferences of the users. An example of the termination condition can be that the number of generations exceeds a predefined upper bound $n \in \mathbb{Z}_{>0}$.

IBEA is an EA paradigm for solving multi-objective optimization problems (MOPs). Unlike traditional MOEAs that use Pareto-based fitness assignment scheme, in IBEA, a new indicator-based measure is proposed to assign fitness values of individuals [19]. The indicator-based fitness assignment scheme can flexibly integrate preferences of decision maker to search for trade-off solutions. Specifically, given a population $P$, the indicator-based measure of $x_1 \in P$ is defined as follows:

$$F(x_1) = \sum_{x_2 \in P/\{x_1\}} -e^{-I(\{x_2\},\{x_1\})/\kappa} \tag{3}$$

where $\kappa$ is a scaling factor, and $I$ is a binary quality indicator that describes the preferences of decision maker. The binary quality indicator, a function that maps $m$ Pareto set approximations to a real number, is used to compare the quality of two Pareto set approximations relatively to each other. The commonly used binary quality indicator is the $\epsilon-$indicator $I_{\epsilon+}$ which is defined as:

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040

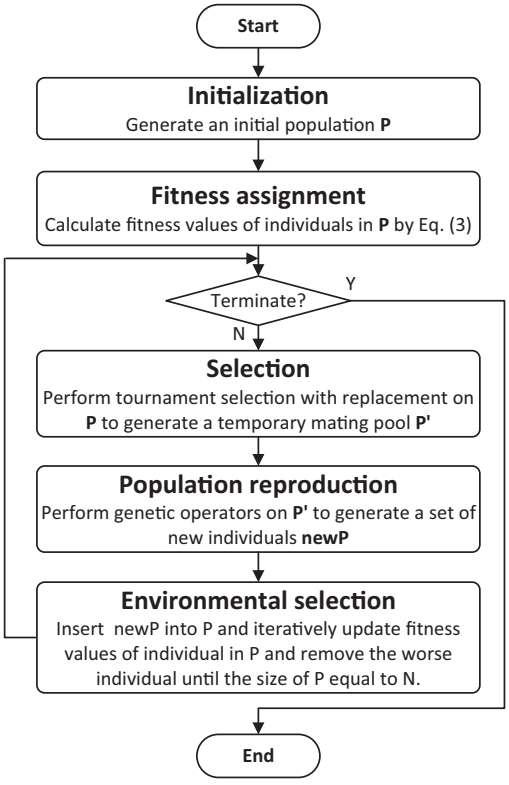


**Fig. 3.** The general procedure of IBEA.

$$I(A,B) = \min_{\epsilon}\{\forall x_2 \in B, \quad \exists x_1 \in A : f_i(x_1) - \epsilon \leq f_i(x_2) \quad for i = \{1,\ldots,n\}\} \tag{4}$$

where $A$ and $B$ are two Pareto set approximations, $f_i$ is the $i$th objective function, and $n$ is the total number of objectives. Let us assume that the task is to minimize all objective values. Based on the above indicator-based fitness assignment scheme, the basic procedure of IBEA is described in Fig. 3. First of all, a random initial population $P$ is generated. The fitness values of individuals in $P$ are evaluated by Eq. (3). Then, the selection, population reproduction, and environmental selection operations are performed iteratively until the termination condition (e.g., reaching the maximum generations) is met. In the selection operation, better individuals in $P$ are selected to form a temporary mating pool $P'$. In the population reproduction operation, genetic operators such as crossover and mutation are utilized to create a new population *newP* based on $P'$. Finally, in the environmental selection operation, the *newP* is inserted into $P$ and fitness values of all individuals in $P$ are evaluated. The worse individuals are removed from $P$ so as to ensure that the size of $P$ equals to $N$.

### 3.2. The proposed algorithm

IBEA has shown great potential to find high quality solutions for the SPLE problem [10]. However, a major drawback of IBEA is that it often provides many duplicated non-dominated solutions at the end of the algorithm. In order to find better and more distinct non-dominated solutions, we propose an enhanced algorithm, namely IBED for solving the SPLE problem. The procedure of the proposed algorithm is illustrated in Fig. 4. The key idea is to incorporate multiple genetic operators to evolve dual populations simultaneously. One population is evolved by IBEA operators, while the other is evolved by DE operators. In this way, the population diversity can be maintained, which is useful for finding more non-dominated solutions. The implementations of the major steps are described as follows.

#### 3.2.1. Initialize population

The first step is to generate two random populations, namely *population*1 and *population*2. Each population contains $N$ random chromosomes and each chromosome represents a candidate solution (i.e., a candidate feature set to be selected). Specifically, the selected features of a feature model are encoded using an array-based chromosome. Given a chromosome of length $n$, array indices are numbered from 0 to $n-1$. Each feature is assigned with an array index starting from 0. Each value on the chromosome ranges over $\{0, 1\}$, where 0 and 1 represents the absence and presence of the feature respectively. Given a feature model $M$, we define a function $f_M : Fea(M) \rightarrow \{\mathbb{Z}, \perp\}$ that maps each feature $f$ of the feature model $M$ to an array index. $f_M(f_1) = \perp$ denotes that there is no array index assigned for the feature $f_1$. Similarly, we define $f_M{}^{-1} : \mathbb{Z} \rightarrow Fea(M)$ as a function that maps a given array index to the feature it represents. In this way, a chromosome can be represented by a vector of integers:

$$X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}] \tag{5}$$

where $X_i$ represents the $i$-th chromosome in the first (or second) population, and $n$ is the number of features under consideration. The value of $x_{i,j}$ can be 0 or 1. If $x_{i,j} = 1$, the $j$th feature is selected. Otherwise, the $j$-th feature is ignored. The value of $x_{i,j}$ is randomly initialized in the first generation.

#### 3.2.2. Update population using IBEA operators

This step updates *population*1 using the operators of IBEA [20]. Generally, there are three sub-steps to achieve the goal of this step. In the first sub-step, the selection, crossover, and mutation operators in IBEA are utilized to generate $N/2$ offspring, where $N$ is the size of population. Then, in the second sub-step, the offspring generated by the IBEA operators and those generated by the *population*2 are inserted into *population*1 to form a new population. Note that the size of the new population now may be larger than $N$. In the third sub-step, the *environmental selection* is utilized

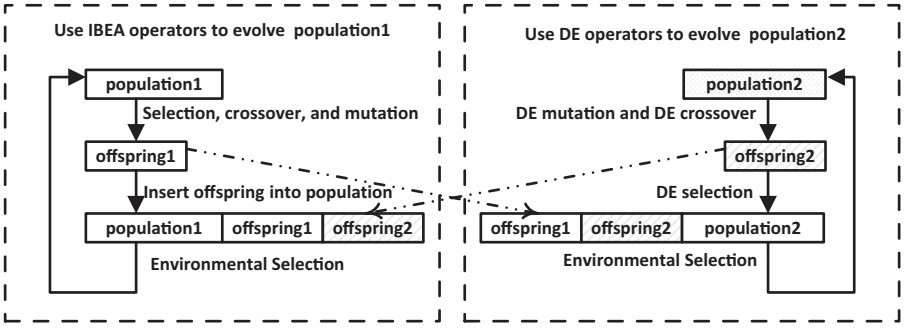


**Fig. 4.** The general procedure of the proposed IBED.

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040

to remove some individuals in new population ($P_1$). In the *environmental selection*, the fitness values of chromosomes in the new population are evaluated using the indicator-based approach. Then, the worse chromosome in $P_1$ are removed one-by-one until the size of the new population does not exceed $N$, as done in traditional IBEA. The revised $P_1$ is then used to replace *population*1 for the next generation.

*3.2.3. Update population using DE operators*

In this step, the DE operators are utilized to update *population*2. This step contains four sub-steps:

1. DE mutation: this sub-step aims to perform DE mutations on each chromosome in *population*2 to generate a set of mutation vectors. We adopt the "DE/current-to-rand/2" mutation scheme to accomplish this task [38]. The "DE/current-to-rand/2" mutation can be expressed by

$$Y_i = X_{r1} + F \cdot (X_{r2} - X_{r3}) + F \cdot (X_{r4} - X_{r5}). \quad (6)$$

where $r1$, $r2$, $r3$, $r4$, and $r5$ are five distinct random individual indices, and $F$ is a scaling factor (i.e., a user defined constant to scale the differential). Since the traditional "DE/current-to-rand/2" mutation is only applicable to real coded chromosomes, we adopt the strategy proposed by Zhong et.al. [39] to extend the 'DE/current-to-rand/2″ mutation to integer-coded chromosomes. Specifically, for each element ($x_{i,j}$) of the $i$-th parent chromosome, a mutation probability is calculated by:

$$\varphi = F \cdot \psi(x_{r1,j}, x_{r2,j})) + F \cdot \psi(x_{r3,j}, x_{r4,j})) - F \cdot \psi(x_{r1,j}, x_{r2,j})) * F \cdot \psi(x_{r3,j}, x_{r4,j})) \quad (7)$$

where $\psi(a, b)$ is defined as

$$\psi(a, b) = \begin{cases} 1, \text{if} \quad a \neq b \\ 0, \text{otherwise} \end{cases} \quad (8)$$

The element $x_{i,j}$ has a probability of $\varphi$ to be mutated to a new value. When a mutation is required, the new value to assign $x_{i,j}$ is randomly selected from $\{0, 1\}$. Otherwise, its value is set equal to $x_{r1,j}$.

In order to improve the probability of generating valid solutions, we adopt a tournament selection (with tournament size = $T$) to select $i$ and $r1$, so that correct solutions (i.e., those where no constraint is violated) is more likely to be selected. *Tournament selection* means executing several tournaments among a few individuals (or chromosomes) chosen at random from the population. The rationale is that partial product configurations of the correct solutions should be more likely to build new correct solutions. In addition, to improve the robustness of the algorithm, the values of $F$ is randomly set by:

$$F = rand(0, 1) \quad (9)$$

where $rand(a, b)$ returns a random value uniformly distributed within $[a, b]$.

2. DE crossover: in this sub-step, each selected parent chromosome $X_i$ is crossover with its mutation vector $Y_i$ to generate a trial vector $U_i$:

$$u_{i,j} = \begin{cases} y_{i,j}, \text{if} \quad rand(0, 1) < CR \quad \text{or} \quad j = k \\ x_{i,j}, \text{otherwise} \end{cases} \quad (10)$$

where $CR$ is the crossover rate, $k$ is a random integer between 1 and $n$, $u_{i,j}$, $y_{i,j}$ and $x_{i,j}$ are the $j$-th variables of $U_i$, $Y_i$ and $X_i$ respectively. Similar to $F$, the value of $CR$ is set to be $rand(0, 1)$ for the purpose of the robustness of the algorithm.
3. DE selection: this step adopts the one-to-one selection strategy in DE to insert newly created trial vectors into *population*2. Then, for each $X_i$ in *population*2, if $X_i$ is dominated by $U_i$, $X_i$ is replaced by $U_i$. Otherwise, if $X_i$ and $U_i$ are non-dominated with each other, then $U_i$ is inserted into *population*2. The offspring generated by the first population are also inserted into *population*2.
4. Environmental selection: the goal of this sub-step is to remove some worse individuals in *population*2 to form a new *population*2 for the next generation. Firstly, the fitness values of all individuals in *population*2 are evaluated by the Eq. (3). Then the worse chromosome in *population*2 are removed one-by-one until the size of *population*2 does not exceed $N$, as done in traditional IBEA.

The *population*1 and *population*2 are updated iteratively by using the operators of IBEA and DE until the termination condition is met. When the termination condition is met, the two populations are joint to form a temporary population. Then, the *environmental selection* is performed on the temporary population to select $N$ best individuals as the final approximation Pareto optimal solutions (i.e., the alternative trade-off solutions). It should be noted that DE is a popular and powerful EA for global optimization. The operators of DE have been shown quite effective to maintain population diversity and find global (or near global) optimal solutions [21]. Meanwhile, the operators of IBEA have been shown to be capable of generating high-quality correct solutions [10]. In the proposed algorithm, both the operators of DE and IBEA are integrated to generate offspring simultaneously – in such a way, we hope to improve the population diversity and meanwhile obtain more correct solutions.

## 4. Enhancement techniques for EAs

In this section, we propose two enhancement techniques for existing MOEAs to address the optimal feature selection problem. First, we introduce a preprocessing method to filter out prunable features before the execution of an EA, in order to reduce the search space. Second, we illustrate feedback-directed evolutionary operators that are used in this work to guide an EA for the optimal feature selection.

**Algorithm 2.** PrunableFeatures

```
input  : Fea(M)
output : F_c, F_d ⊆ Fea(M)
1 F_c ← ∅;
2 F_d ← ∅;
3 foreach f ∈ Fea(M) do
4     if ¬SAT(conj(M) ∧ ¬f) then
5         F_c = F_c ∪ f;
6     else if ¬SAT(conj(M) ∧ f) then
7         F_d = F_d ∪ f;
8 return (F_c, F_d);
```

*4.1. Preprocessing of feature model*

In the following, we introduce the features that could be pruned from $Fea(M)$ before the execution of an EA. With the pruning of features, the search space of the EA would be reduced, which could make the optimal feature selection more efficient.

Our approach of preprocessing is by exploiting the *core features* and *dead features* [9] of the products. It is observed that some features must be present in all products derived from $M$. For example, the feature set {*Chat*, *Output*} is shared by all derived products, and we call these features as *core features*. Similarly, we call the set of features that must not be used in all derived products as *dead features*. Dead features do not exist in *JCS* but they are common in feature models of real systems (e.g., Linux X86 kernel and

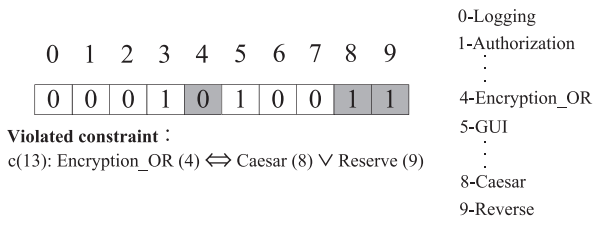


**Fig. 5.** Feedback-directed mutation operator.

eCos operating system). Henceforth, we denote core features and dead features as $F_c$ and $F_d$ respectively, where $F_c, F_d \subseteq Fea(M)$, and $F_c \cap F_d = \emptyset$. The preprocessed features that are passed to the execution of EAs are $Fea(M) \setminus (F_c \cup F_d)$, and we denote $F_c \cup F_d$ as *prunable features*, the union of core features and dead features.

The function *PrunableFeatures* (Algorithm 2) is used to find core and dead features. Recall that $conj(M)$ represents the conjunction of all tree constraints and CTCs of feature model $M$, and *SAT* is a function that is used at line 4 and 6 in Algorithm 2 to check the satisfiability of the constraints. Note that *SAT* function is readily provided by many off-the-shelf SAT solvers (e.g., SAT4J [40]). We assume that $conj(M)$ is satisfiable (i.e., there is at least a valid product from $Fea(M)$). If $conj(M) \wedge \neg f$ is unsatisfiable (line 4), it implies that feature $f$ must exist in all derived products of $M$. Therefore, feature $f$ is added to the set of core features $F_c$ (line 5). The detection of dead features $F_d$ at line 6 and 7 is similar.

As an example, we show how a feature set on the *JCS* is encoded. Note that features *Chat* and *Output* have been pruned by the preprocessing algorithm in Algorithm 2. Therefore, they are not contained in the chromosome (i.e., $f_M(Chat) = f_M(Output) = \perp$). The features are indexed level by level, and their indexes have been listed in Fig. 5 (e.g., $f_M(Logging) = 0$). The chromosome in Fig. 5 represents the feature set {*Encryption*, *GUI*, *Caesar*, *Reverse*}.

### 4.2. Feedback-directed evolutionary operators

The violated constraints of a chromosome $C_i$ provide an important hint on which features $C_i$ need to be modified. If we focus on these features, we may converge faster on the optimal feature selection.

We incorporate this feedback into the crossover and mutation operations, which are the main evolutionary operators common for almost all EAs. The feedback-directed crossover and mutation operators provide an effective guidance for EAs to perform the optimal feature selection.

#### 4.2.1. Feedback-directed mutation

The objective of *mutation* operator is to change some values in a selected chromosome leading to additional genetic diversity to help the search process escape from local optimal traps. In detail, we introduce how the *feedback-directed mutation* operator works. Before the mutation, the feedback-directed mutation analyzes the violated constraints on the selected chromosome. We denote the corresponding positions on the chromosomes for the features that are contained in the violated constraints as *error positions*.

We illustrate the feedback-directed mutation operator using the *JCS* example shown in Fig. 5. Given the values of the chromosome as shown in Fig. 5, we can easily check that it violates the constraint $c(13)$. The constraint $c(13)$ contains three features, which are *Encryption_OR*, *Caesar*, and *Reverse*. The corresponding array positions of these three features are shaded on the chromosome in Fig. 5. These shaded positions are the *error positions*.

**Algorithm 3.** FMutation

```
input  : P, P_emut and P_mut
output : C
1  C ← P;
2  n ← |P|;
3  Err ← ErrPos(C);
4  for i = 0 to n − 1 do
5      if (i ∈ Err ∧ rand(0, 1) < P_emut) ∨
6      (i ∉ Err ∧ rand(0, 1) < P_mut) then
7          C[i] ← randInt(0, 1);
8  return C;
```

The algorithm *FMutation* for feedback-directed mutation is given in Algorithm 3. An offspring chromosome $C$ is initialized with values in the chromosome $P$ (line 1), and $n \in \mathbb{Z}$ is initialized with the length of the chromosome $P$ (line 2). At line 3, $Err \in \mathcal{P}(\mathbb{Z})$ is assigned with the set of integers that is returned from $ErrPos(C)$. The set of integers returned by $ErrPos(C)$ represents the error positions on the chromosome $C$. Each position on the chromosome is iterated (line 4). The function $rand(a, b)$ (or $randInt(a, b)$), with $a > b$, chooses a real (or integer) number between numbers $a$ and $b$. At line 5, if the current position $i$ is an error position, and the random number is less than the error mutation probability $P_{emut}$, then the value in the position $i$ on the chromosome is mutated by randomly choosing an integer between 0 and 1 (line 7). Otherwise, if the position is not an error position and the random number is less than $P_{mut}$ (line 6), then the value in the position $i$ is mutated. Note that the probability $P_{emut}$ should be set with a value that is far larger than $P_{mut}$, so that the mutation occurs more frequently on error positions. For $P_{emut}$ and $P_{mut}$, example values could be 1.0 and 0.0000001. Note that we set $P_{mut}$ lower than classic mutation probability (e.g. 0.001–0.05 [41]). This is because lower $P_{mut}$ with higher $P_{emut}$ would lead to faster convergence, since it allows faster correction of constraint violations by minimizing the changes of non-error positions and focusing on the changes of error positions. This is demonstrated in Section 5.3.

**Algorithm 4.** ErrPos

```
input  : C, T and Fea(M)
output : ePos
1  ePos ← ∅;
2  n ← |C|;
3  Π ← ∅;
4  for i = 0 to n − 1 do
5      Π ← Π ∪ {f_M^{-1}(i) ↦ (C[i]≠0)};
6  foreach feature ∈ F_c do
7      Π ← Π ∪ {F_c ↦ true};
8  foreach feature ∈ F_d do
9      Π ← Π ∪ {F_d ↦ false};
10 foreach t ∈ T do
11     if Π ⊭ t then
12         ePos ← ePos ∪ getFeatures(t);
13 return ePos;
```

We now introduce the *ErrPos* function that is referenced in Algorithm 3 and defined in Algorithm 4. At line 1, *ePos* is initialized with an empty set. At line 2, variable $n$ is initialized with the length of chromosome $C$. The valuation function $\Pi : Fea(M) \to \{true, false\}$ (line 3) maps each feature $f$ of the feature model $M$ to a Boolean value that denotes whether the corresponding feature is selected or not. The mappings in $\Pi$ are populated according to the values on the chromosome (line 5). Subsequently, core and dead features

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040

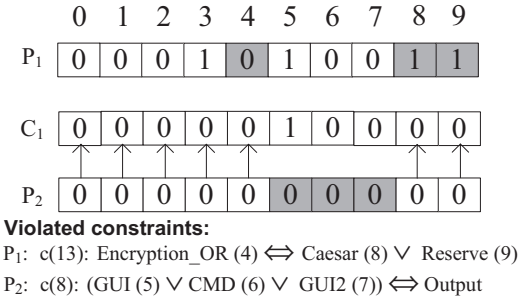


**Fig. 6.** Feedback-directed crossover operator.

are added to the mappings in Π with values *true* and *false* respectively (lines 7 and 13). The reason is that core (or dead) features must (or must not) belong to any valid product of feature model *M* as explained in Section 4.1. At line 8, *T* represents the constraints (including tree constraints and CTCs) of feature model *M*, and each constraint $t \in T$ are iterated for operations in lines 9 and 10. At line 9, $\Pi \not\models t$ holds if replacing each feature *f* contained in the constraint *t* with $\Pi(f)$ evaluates to false. In other words, $\Pi \not\models t$ means that the selection represented by chromosome *C* violates the constraint *t*. In this case, the function *getFeatures*(*t*) is used to get the features that are contained in the constraint *t* (line 10). For example, given the constraint *c*(13) in Table 1 for *JCS* as an input, the function *getFeatures* returns {4, 8, 9}. These array indexes that represent the error positions are included in *ePos*.

#### 4.2.2. Feedback-directed crossover

The crossover operation is used to generate a new chromosome by exchanging values in a pair of chromosomes chosen from the population. It happens with a crossover probability $P_{cross}$. The feedback-directed crossover operator uses values in the non-error positions to crossover. The objective for using values from non-error positions is to pass the "good genes" to the offspring.

We demonstrate the feedback-directed crossover operator, using the *JCS* example shown in Fig. 6. Suppose the chromosomes $P_1$ and $P_2$ have violated constraints *c*(13) and *c*(8) respectively. The offspring chromosome $C_1$ is first initialized as the same values with the chromosome $P_1$. The values from non-error positions of the chromosome $P_2$ are copied to the chromosome $C_1$ (shown by the arrows). This results in the chromosome $C_1$. The production of the chromosome $C_2$ (not shown in the graph) is symmetric to the production of the chromosome $C_1$.

The algorithm *FCrossover* from the feedback-directed crossover operator is given in Algorithm 5. The chromosomes $C_1$ and $C_2$ are initialized with the values from chromosomes $P_1$ and $P_2$ respectively (lines 1 and 2). Variable *n* is initialized with the length of chromosome $P_1$ (line 3). If the generated random number is smaller than the crossover probability $P_{cross}$ (line 4), then it performs the crossover operation. First, the operation verifies whether there exists any error position in chromosomes $P_1$ and $P_2$, by checking whether the size of their error positions is greater than 0 (line 5). If so, then the feedback-directed crossover is performed. The algorithm iterates through the chromosome (line 6). It copies only the values of non-error positions from chromosome $P_1$ to the same positions in chromosome $C_2$ (lines 7 and 8). Similarly, the values of non-error positions from chromosome $P_2$ are copied to chromosome $C_1$ (lines 9 and 10).

Otherwise, if both chromosomes $P_1$ and $P_2$ do not have any error position, the classic single point crossover operator is applied. First, an array index, $crossIndex \in \{0, \ldots, n-1\}$, is randomly selected. Subsequently, all values starting from the position *crossIndex* are copied from chromosome $P_2$ (or $P_1$) to chromosome $C_1$ (or $C_2$) (lines 12–15).

#### 4.2.3. Complexity of algorithms

In this subsection, we analyze the complexity of algorithms. For Algorithm 2, the complexity is $O(N \cdot k)$, where *N* is the number of features in *Fea*(*M*) and *k* represents the overhead of SAT solver for solving the constraints at line 4 and 6. For Algorithm 3 and Algorithm 5, the complexities are both $O(|\#1|C)$, where $|\#1|C$ is the length of a chromosome *C*. For Algorithm 4, the complexity is $O(|\#1|T \cdot F)$, where $|\#1|T$ is the number of constraints of a feature model *M*, and *F* is the maximum number of features used in a constraint $t \in T$. In *JCS*, *F* = 4, which is from constraint c(8).

**Algorithm 5.** FCrossover

```
input  : P1, P2 and Pcross
output : C1 and C2
1  C1 ← P1;
2  C2 ← P2;
3  n ← |P1|;
4  if rand(0, 1) < Pcross then
5      if |ErrPos(P1)| > 0 ∧ |ErrPos(P2)| > 0 then
6          for i = 0 to n − 1 do
7              if i ∉ ErrPos(P1) then
8                  C2[i] ← P1[i];
9              if i ∉ ErrPos(P2) then
10                 C1[i] ← P2[i];
11     else
12         crossIndex ← randInt(0, n−1);
13         for i = crossIndex to n−1 do
14             C1[i] ← P2[i];
15             C2[i] ← P1[i];
16 return (C1, C2);
```

## 5. Evaluation

We conduct experiments to evaluate IBED and also the effectiveness of the enhancement techniques for different EAs (including IBED). Specifically, we aim to answer the following research questions (RQs):

**RQ1.** Is IBED comparable with the best EA algorithm for this problem, namely IBEA, in terms of the correctness of the solutions?
**RQ2.** What is the runtime of IBED compared to the existing state-of-the-art EAs?
**RQ3.** Can IBED find more non-dominated solutions than IBEA?
**RQ4.** How do the enhancement techniques improve on the existing state-of-the-art EAs in terms of the correctness and the runtime?
**RQ5.** Can the enhancement techniques be generalized to different EAs?
**RQ6.** How scalable are IBED and the enhancement techniques in terms of the size of feature model?

Note that RQ1 to RQ3 examine IBED in terms of correctness, performance and diversity, respectively. RQ4 examines the usefulness of the enhancement techniques on improving correctness and performance of EAs. RQ5 examines the generality of applying the enhancement techniques on different EAs. RQ6 examines the scalability of IBED (enhanced by our preprocessing method and feedback-directed mechanism) with the large-size academic feature model and industrial models.

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040

**Table 2**
Brief overview of the tested EAs.

| Algorithm | Population | Operators | Criteria for domination | Objective of the criteria |
|---|---|---|---|---|
| IBED | Main and archive for DE; main and archive for IBEA | DE mutation, DE crossover, DE selection, crossover, mutation, environmental selection | The amount of domination is calculated based on quality indicator | Suitable for user preferences and absolute domination |
| IBEA | Main and archive | Crossover, mutation, environmental selection | | Suitable for user preferences |
| NSGA-II | Main | Crossover, mutation, tournament selection | Distances to closest point of each objective are calculated; suitable for the point with greater distance from other objectives | Suitable for more spread out solutions and absolute domination |
| ssNSGA-II | | | Similar to NSGA, with the exception that only one new individual inserted into population at a time | |
| MOCell | Main and archive | Crossover, mutation, tournament selection, random feedback | Similar to NSGA, a ranking and a crowding distance estimator is used, but bigger distance values are favored | |

### 5.1. Implementation and experimental setup

To conduct a fair comparison between different EAs, we use the existing EAs from the same third party library, jMetal [42]. For IBED, we also implement it in the same way as other EAs in jMetal. In Section 5.1.2, we introduce the quality indicators to evaluate EAs. From Sections 5.1.3–5.1.5, we introduce the six feature models, their feature attributes, and the multiple objective functions for the optimal feature selection problem. Last, in Sections 5.1.6 and 5.1.7, we explain the configurations of EAs and the parameters they used.

The experiments were conducted on an Intel Core I7 4600U CPU with 8 GB RAM, running on Windows 7.

#### 5.1.1. Implementation of EAs

We have implemented our approach based on jMetal [42], which is a Java-based open source framework that supports multi-objective optimization with EAs. Sayyad et al. [10,17] have made an extensive experiment to test how different EAs implemented in jMetal could contribute to the optimal feature selection. In addition to those existing EAs [10], we also implement our IBED and evaluate it with the enhancement techniques (i.e., the preprocessing step and feedback-directed mechanism).

1. **IBED**: our dual-population evolutionary algorithm combining IBEA and DE operators
2. **IBEA**: indicator-based evolutionary algorithm [19]
3. **NSGA-II**: non-dominated sorting genetic algorithm [33]
4. **ssNSGA-II**: steady-state NSGA-II [34]
5. **MOCell**: A cellular genetic algorithm for multi-objective optimization [35]

A brief overview of these EAs is provided in Table 2. For each EA, we list the used population strategy, operators, its criteria for domination (to check non-dominated solutions) and its suitable objectives. As reported in previous work [43], IBEA is an indicator-based method, but NGSA-II, ssNSGA-II and MOCell are Pareto-based ones. In indicator-based methods, binary performance metrics that map an ordered pair of individuals to a scalar value are suggested as indicator functions [43]. The indicator values returned by the indicator functions are then used as the selection criteria in the environmental selection operation. In contrast, Pareto-based methods use certain selection strategies (e.g., tournament selection) that are mainly based on the qualitative information of Pareto-dominance. In Table 2, IBED adopts both criteria owing to the dual-population design.

#### 5.1.2. Quality indicators

To measure the quality of Pareto front, we make use of two indicators: (a) hypervolume [44], (b) percentage of correctness [33].

a) *Hypervolume (HV)*: hypervolume of the solution set $S=(x_1, \ldots, x_n)$ is the volume of the region that is dominated by $S$ in the objective space. In jMetal, although all objectives are minimized, but the Pareto front is inverted before the hypervolume is calculated. Therefore, the preferred Pareto front would be with the most hypervolume.
b) *Percentage of correctness (%Correct)*: there might be solutions that violate some constraints in the Pareto front, since the correctness is an optimization objective that evolves over time. Correct solutions (i.e., valid product configurations from definition 1) are more useful to the user. Therefore, we are interested in the percentage of solutions that are correct in the Pareto front.

#### 5.1.3. Feature models used for evaluation

The details of feature models used in the experiment are summarized in Table 3, with the repository information (*Repo.*), feature model name (*Model*), number of features (*Fea.*), number of constraints (*Cons.*), number of prunable features with the preprocessing method in Algorithm 2 ($F_p$), number of prunable features with the preprocessing method in [18] ($F'_p$), and literatures (*Ref.*) associated with each feature model.

*JCS* feature model, which we have used throughout this paper, has been included in SPLOT feature model repository [22] – a benchmark used by many researchers. Two feature models, *Web Portal* and *E-Shop*, are also from SPLOT. The *Web Portal* feature model captures the various configuration options for a Web portal system. The *E-Shop* model, which is one of the largest feature models in SPLOT, describes the functions of an online B2C system that sells fixed-price products. These two models are chosen to facilitate the comparison with the algorithm proposed by [10]. In addition, to further evaluate the scalability of our methods, we make use of feature models from the Linux variability analysis tools (LVAT) repository

**Table 3**
Feature models used for evaluation.

| Repo. | Model | Fea. | Cons. | $F_p$ | $F'_p$ | Ref. |
|---|---|---|---|---|---|---|
| SPLOT | JCS | 12 | 13 | 2 | – | [20] |
| | Web Portal | 43 | 36 | 4 | – | [45] |
| | E-Shop[a] | 290 | 186 | 28 | – | [30] |
| LVAT | eCos | 1244 | 3146 | 54 | 19 | [15,30] |
| | uClinux | 1850 | 2468 | 1244 | 1244 | [46] |
| | Linux X86 | 6888 | 343944 | 156 | 94 | [15] |

[a] The full name of "E-Shop" on SPLOT is "Electronic Shopping".

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040

[23]. The models in LVAT were reversed-engineered by making use of source code, comments and documentations of big projects such as Linux kernel and eCos operating system. Compared to the feature models in SPLOT, the feature models in LVAT contain a significantly larger number of features and constraints, and have higher branching factors, but they have lower ratios of feature groups, and hence shallower tree structures in general.

As shown in Table 3, $F_p$ always contains the same number or more features than $F'_p$ – this shows that our preprocessing method with Algorithm 2 has found more prunable features than the method proposed by Sayyad et al. [18]. Their preprocessing method is based on static analysis. In particular, they detect always true disjunctions (rules) with only one feature, which means the feature is either a core feature or a dead feature. In addition, they investigate the disjunctions (rules) that include two features, if one of them is prunable in the first round, and the other one could be prunable as well. Hence, our method based on SAT solving could detect all features that could be found by their preprocessing method [18], and it shows that the number of features preprocessed by our method ($F_p$) is always higher or equal than that preprocessed by their method ($F'_p$).

#### 5.1.4. Feature attributes

Each feature in the feature models has the following attributes, which are the same as the attributes used by Sayyad et al. [10]:

1. *Cost* $\in \mathbb{R}$, records the costs of selecting the feature. For each feature, the *Cost* value is assigned with a real number that is normally distributed between 5.0 and 15.0.
2. *Used_Before* $\in \{true, false\}$, indicates whether this feature was used before. The value of *Used_Before* is *true* if the feature has been used before, otherwise it is *false*. For each feature, the *Used_Before* value is assigned with a Boolean value that is distributed uniformly.
3. *Defects* $\in \mathbb{Z}$, records the number of defects known in the feature. For each feature, the *Defects* value is assigned with an integer number that is normally distributed between 0 and 10. However, if the feature has not been used before, the *Defects* value is set to 0.

#### 5.1.5. Objective functions

We introduce the five optimization objectives that we use in the experiments. Since jMetal requires minimization of the objectives, all objectives listed here are objectives to be minimized.

**Obj1.** *Correctness*: minimize the number of violated constraints of the feature model.
**Obj2.** *Richness of features:* minimize the number of features that are not selected.
**Obj3.** *Cost:* minimize the total cost.
**Obj4.** *Feature used before:* minimize the number of features that have not been used before.
**Obj5.** *Defects:* minimize the number of known defects.

We specify correctness as an objective, rather than a constraint. The reason is that this allows EA to gradually guide the search toward solutions that contain fewer violated constraints, which eventually leads to correct solutions (i.e., valid product configurations). Furthermore, some objectives are conflicting, e.g., *Obj*2 and *Obj*3, because a larger number of features would imply a higher cost, but meanwhile the cost needs to be minimized.

#### 5.1.6. Configurations of EAs

Given an EA (including our IBED), we introduce the configurations for comparison. Note that we refer to the guided (or unguided) version of the EA as the EA with (or without) the feedback-directed mechanism.

1. **F+P**: this is the EA that makes use of feedback-directed crossover and mutation (Section 4.2), and our preprocessing method (Section 4.1).
2. **U+P**: the unguided version of EA with preprocessing (Section 4.1) applied before the execution of the EA. We have shown that, our method has found more prunable features than the preprocessing method used by [18] (see Section 5.1.3). Thus, $U+P$ can be seen as an improved version of the method used by [18] with results in a smaller search space.
3. **U**: the unguided version of EA without preprocessing, which is used by [10,17].

#### 5.1.7. Parameter settings

For *U*, the same EA used by [17], single-point crossover and bit-flip mutation are used as crossover and mutation operators, with crossover and mutation probabilities set to 0.1 and 0.01 respectively (the default setting used in [17]). *Bit-flip mutation* means random negation of a bit position within the chromosomes of an individual in the case of binary encoding. These operators and probabilities also apply to $U+P$. For $F+P$, the feedback-directed crossover (Algorithm 5) and feedback-directed mutation (Algorithm 3) operators are used. The error mutation probability $P_{emut}$, mutation probability $P_{mut}$, and crossover probability $P_{cross}$ are set to 1.0, 0.0000001, and 0.1 respectively. As mentioned in Section 4.2.1, $P_{mut}$ is set to 0.0000001 in order to retain the non-error positions of the chromosome, while $P_{emut}$ is set to 1.0 to change the error positions. In Table 4, we test configurations $F+P$, $U+P$ and $U$ of each EA with the above parameter values. In Table 5, to prove the effectiveness of a small value of $P_{mut}$ (i.e., 0.0000001), we test another configuration $F'+P$, which is the same as $F+P$ but with the mutation probability $P_{mut}$ is set to 0.01.

For all the EAs, the population size is set to 100. As our IBED adopts a design of dual-population in Section 3.2, each of these two populations (i.e., one evolved by IBEA operators and the other one evolved by DE operators) has 50 instances. All other parameter settings for each EA are default settings of jMetal (e.g., population size is set to 100), and therefore are omitted here.

For SPLOT case study, we make use of 25,000 evaluations using five EAs (IBED, IBEA, NSGAII, ssNSGAII, and MoCell). For the larger LVAT case study, we make use of 100000 evaluations for IBED and IBEA. For both case studies, we generate 10 sets of attributes. For each set of attributes, we run each EA repeatedly for 30 times, and report the median values of the metrics. The evaluation results for SPLOT and LVAT are reported in Tables 4 and 5. They are detailed in Section 5.2 and 5.3.

As pointed by Sayyad et al. [18], IBEA (similarly for IBED) takes five times longer than NSGAII to perform the same number of evaluations. For a fair comparison, they found that giving NSGAII more time to evolve more generations did not help it to achieve significant better *%Correct* and *HV*. We also try 100000 evaluations for NSGAII on SPLOT models, but the results comply with the observation from Sayyad et al. [18].

We make use of Mann–Whitney *U*-test [47] to test the statistical significance of *%Correct*, *Time*(*s*), *HV* indicators. We highlight the corresponding indicators in **bold** for $F+P$, if the confidence level exceeds 95% when $F+P$ is better than $U+P$ (i.e., the lower value for *Time*(*s*), and the higher value for *%Correct* and *HV*).

### 5.2. Evaluation with SPLOT

Table 4 shows the results with SPLOT case study, where *Time*(*s*), *HV*, and *%Correct* represent execution time in seconds, hypervolume and percentage of correct solutions in the Pareto front. Table 6 also

**Table 4**
Evaluation of the five EAs with/without the feedback-directed mechanism and the preprocessing method on SPLOT.

| Model | Measure | IBED | | | IBEA | | | NSGAII | | | ssNSGAII | | | MOCell | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | F+P | U+P | U | F+P | U+P | U | F+P | U+P | U | F+P | U+P | U | F+P | U+P | U |
| E-Shop | Time(s) | **6.7** | 7 | 7.5 | 7 | 6.4 | 7.4 | **1.9** | 2.2 | 2.5 | **15.2** | 16.9 | 17.5 | **2.8** | 4 | 4.5 |
| | HV | **0.3** | 0.28 | 0.21 | **0.3** | 0.18 | 0.19 | **0.26** | 0.2 | 0.17 | **0.24** | 0.22 | 0.22 | **0.24** | 0.19 | 0.22 |
| | %Correct | **99** | 87 | 11 | **100** | 0 | 0 | **12** | 0 | 0 | **15** | 0 | 0 | **14** | 0 | 0 |
| Web Portal | Time(s) | 5.6 | 5.7 | 6 | 5.7 | 4.6 | 4.6 | **0.4** | 0.5 | 0.5 | 8.3 | 8.3 | 8.2 | **1** | 1.8 | 1.9 |
| | HV | 0.31 | 0.31 | 0.27 | **0.32** | 0.2 | 0.23 | **0.3** | 0.24 | 0.21 | **0.3** | 0.21 | 0.24 | **0.31** | 0.21 | 0.22 |
| | %Correct | 97 | 98 | 82 | **100** | 1 | 0 | **28** | 0 | 0 | **20** | 1 | 0 | **41** | 0 | 0 |
| JCS | Time(s) | 5.1 | 5.2 | 5.3 | 4.7 | 4.3 | 4.7 | 0.3 | 0.3 | 0.3 | 7.3 | 6.8 | 6.9 | **0.3** | 0.4 | 0.6 |
| | HV | 0.28 | 0.28 | 0.24 | 0.29 | 0.3 | 0.28 | 0.3 | 0.29 | 0.28 | 0.29 | 0.29 | 0.29 | **0.33** | 0.31 | 0.3 |
| | %Correct | 85 | 85 | 64 | **86** | 78 | 54 | **27** | 22 | 16 | **31** | 24 | 14 | **34** | 21 | 18 |

shows the number of non-dominated solutions found by IBED and IBEA for two sets of feature attribute values.

### 5.2.1. Answer to RQ1

RQ1 examines the effectiveness of IBED in terms of percentage of correct solutions. We notice that the IBED and IBEA have significantly outperformed other EAs in terms of *%Correct*, regardless of enhancement techniques or parameter settings. This conforms to the observation reported by Sayyad et al. [10]. According to their explanation [10], the reason is that all EAs used in this case study (other than IBEA and IBED) use diversity-based selection criteria, which favor higher distances between solutions. Thus, non-indicator based EAs tend to remove solutions crowded toward the zero-violation point, achieving the lower value for *%Correct*.

IBED strikes a balance between correctness and diversity of solutions, as it is a combination of indicator-based method and diversity-based selection criteria. In Table 4, we denote the three configurations of an EA (i.e., the guided version with preprocessing, the unguided version with preprocessing and the unguided version without preprocessing) as *F+P*, *U+P* and *U* (see Section 5.1.6), respectively. The results show that IBED *U* (i.e., the unguided version of IBED without preprocessing) outperforms IBEA *U* in terms of *%Correct* on the three models from SPLOT. For the *U+P* configuration (i.e., the unguided version with preprocessing), IBED is also much better than IBEA in terms of *%Correct*, especially for the two large models *Web Portal* and *E-Shop*.

However, if we consider both preprocessing and the feedback directed mechanisms, IBEA *F+P* (i.e., the guided version of IBEA with preprocessing) achieves the best correctness rate among all the configuration of all EAs. The correctness rate of IBED *F+P* is less than IBEA *F+P* (5% on average), and its HV is less than IBEA *F+P* (0.01 on average).

To summarize, IBED *U* is significantly better than IBEA *U* in terms of *%Correct* and HV. With preprocessing and the feedback-directed mechanism, IBED *F+P* is almost close to IBEA *F+P* in terms of *%Correct* and HV. We attribute the high correctness rate of IBED to the design of adopting one sub-population evolved by IBEA operators (see Section 5.1.7). Despite IBED (a sub-population evolved by IBEA operators) is slightly worse than IBEA (the whole population evolved by IBEA operators) for correctness, more non-dominated solutions are found due to the diversity of the DE sub-population. Thus, the benefit in diversity is proved to be worthy of the compromise at correctness (see Section 5.2.3).

### 5.2.2. Answer to RQ2

RQ2 examines the efficiency of IBED, compared with other EAs. Among all the EAs, although NSGAII and MOCell are extremely fast (less than 30% of runtime of the IBEA), they produce many invalid solutions, resulting in low correctness ratio. We mainly consider the runtime of IBEA as the benchmark to test the performance of IBED on SPLOT.

*U+P and U configurations:* For these unguided configurations *U* and *U+P*, IBED takes more runtime than the corresponding configuration of IBEA. This reason is that IBED performs two environmental selection processes, which require iteratively calculating the fitness values (i.e., the indicators calculated in Eq. (3) in Section 3.1).

*F+P configuration:* on the small-size model *JCS*, IBED is less efficient than IBEA, taking 8% more time. On the medium-size model *Web Portal*, these two EAs are quite close in performance. We attribute this to the reason of the two environmental selection processes. On the large-size model *E-Shop*, IBED takes less time (5% less) than IBEA. The rationale is that IBED adopts DE operators that performs well with a small population size on the problem with a large solution space. For the larger solution space, DE may converge faster than IBEA. Hence, we conclude that IBED is more stable and more tolerant to the impact of feature model size, showing the similar runtime for different SPLOT models. Finally, IBED may be

**Table 5**
Evaluation IBED and IBEA with/without the feedback-directed mechanism and the preprocessing method on LVAT.

| Model | Measure | IBED | | | IBEA | | |
|---|---|---|---|---|---|---|---|
| | | F+P | F'+P | U+P | F+P | F'+P | U+P |
| eCos | Time (s) | **47.5** | 58.2 | 58.1 | **35.6** | 51.3 | 58.6 |
| | HV | **0.24** | 0.22 | 0.21 | **0.24** | 0.21 | 0.18 |
| | %Correct | **89** | 17.9 | 6.0 | **91** | 61 | 0.0 |
| uClinux | Time (s) | 45 | 44.8 | 41.9 | 50.7 | 43.9 | 47 |
| | HV | 0.3 | 0.28 | 0.29 | **0.31** | 0.29 | 0.28 |
| | %Correct | 100 | 98 | 100 | **100** | 100 | 0.0 |
| Linux X86 | Time (s) | 2713 | 2808 | 15457[a] | 2613 | 3277 | 12804[b] |
| | HV | 0.21 | 0.21 | 0.2 | 0.2 | 0.22 | 0.18 |
| | %Correct | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

[a] For Linux X86, the jMetal configuration of IBED *U+P* that uses cached results throws runtime exceptions of memory limit with 4G memory for JVM. We have to disable cache for IBED *U+P*, and it greatly increases the time.
[b] Due to the memory limit exception, we have to disable cache to avoid the exception for IBEA *U+P*.

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040

**Table 6**
Non-dominated solutions found by IBED F+P (A) and IBEA F+P (B) on SPLOT and LVAT.

| Model | A %Correct | B %Correct | A Time (s) | B Time (s) | A HV | B HV | $\|A\|$ | $\|B\|$ | $\|A \cap B\|$ | $\|N_A \cup N_B\|$ | $\|N_A\|$ | $\|N_B\|$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| JCS 1 | 92.4 | 92.9 | 5.0 | 3.5 | 0.30 | 0.30 | 19 | 21 | 19 | 21 | 19 | 21 |
| JCS 2 | 66.9 | 68.4 | 5.8 | 4.3 | 0.23 | 0.23 | 12 | 16 | 12 | 12 | 12 | 12 |
| Web Portal 1 | 99.2 | 100.0 | 5.7 | 5.4 | 0.31 | 0.29 | 272 | 508 | 20 | 213 | 141 | 77 |
| Web Portal 2 | 99.5 | 100.0 | 5.6 | 5.8 | 0.28 | 0.28 | 273 | 414 | 23 | 262 | 148 | 129 |
| E-shop 1 | 99.1 | 100.0 | 6.8 | 7.8 | 0.29 | 0.29 | 1491 | 1355 | 0 | 873 | 713 | 160 |
| E-shop 2 | 98.7 | 99.9 | 6.7 | 9.1 | 0.30 | 0.30 | 1441 | 1378 | 0 | 930 | 829 | 101 |
| eCos 1 | 87.6 | 92.3 | 48.2 | 33.9 | 0.25 | 0.25 | 1306 | 1533 | 0 | 1514 | 742 | 772 |
| eCos 2 | 88.8 | 92.6 | 47.9 | 32.8 | 0.25 | 0.25 | 1337 | 1621 | 0 | 1520 | 669 | 851 |
| uClinux 1 | 100.0 | 100.0 | 44.9 | 54.8 | 0.30 | 0.30 | 1929 | 1511 | 0 | 968 | 894 | 74 |
| uClinux 2 | 100.0 | 100.0 | 45.6 | 53.7 | 0.30 | 0.30 | 1875 | 1620 | 0 | 1074 | 875 | 199 |

less efficient than IBEA due to the two extra selection processes, otherwise it may converge faster in some cases.

#### 5.2.3. Answer to RQ3

To evaluate the diversity of found correct solutions (RQ3), we compare IBED with the best configuration of IBEA on the same sets of feature attribute values. For each feature model, we randomly generate two sets of feature attribute values, on which both the configuration $F+P$ of IBED and IBEA are applied. For each set of feature attribute values, we execute 30 times and record the results in Table 6. For simplicity, we denote the results of IBED $F+P$ as $A$ and results of IBEA $F+P$ as $B$. We also record the *mean* correctness rate, time and hypervolume of 30 executions of IBED $F+P$ (or IBEA $F+P$) in column *A %Correct*, *A Time(s)* and *A HV* (or in column *B %Correct*, *B Time(s)* and *B HV*).

To evaluate the searching capability of IBED and IBEA in diversity, we compare them based on the union of found non-dominated solutions of 30 executions. We record the *correct* solutions found by IBED in column $|A|$, and those by IBEA in column $|B|$. Column $|A \cap B|$ shows the number of correct solutions commonly found by IBED and IBEA, while column $|N_A \cup N_B|$ lists the number of non-dominated solution existing in all solutions found by both EAs. We also list the number of non-dominated solutions found by IBED (or IBEA) in column $|N_A|$ (or column $|N_B|$).

*Results on JCS:* IBED is more suitable for the problem with large search space. For the small model, IBED and IBEA show the similarity capability in searching for non-dominated solutions. As model *JCS* has a small number of features, the corresponding solution space is also small. For the first set of feature attribute values, only 19 and 21 correct solutions are found by IBED and IBEA, respectively. The solutions found by IBED are all found by IBEA, as $|A \cap B| = 19$. Due to $|N_A \cup N_B| = 21$, all these solutions are non-dominated. Thus, for the first value set, IBEA finds 2 more non-dominated ones. For the second value set, although IBEA finds 4 more correct solutions than IBED (12 solutions), these 4 solutions are dominated by others. Hence, for the second value set, IBED finds the same non-dominated solutions as IBEA.

*Results on Web Portal and E-Shop:* as the size of feature model increases, IBED can find more non-dominated solutions. As the solution space *Web Portal* is not as small as that of *JCS*, IBED and IBEA share less commonly found solutions (see $|A \cap B|$). Although IBEA finds more correct solutions (508 and 414 for two sets of feature attribute values), most of them are dominated by other solutions, especially those that IBED finds. On *E-Shop*, as the feature size is 290, the solution space can be very big. Two EAs share no common correct solutions, and they all report a similar number of correct solutions (1355–1491) after 30 executions. However, it is surprising to see that most of solutions (about 90%) found by IBEA are dominated by the ones found by IBED. These results indicate that DE operators can work more effectively than the traditional GA operators at finding high-quality non-dominated solutions in the *E-Shop* case.

#### 5.2.4. Answer to RQ4

RQ4 examines the effectiveness of the two enhancement techniques (i.e., the preprocessing method and the feedback-directed mechanism).

*Enhancement and correctness rate:* in Table 4, we notice that for each EA, the configuration $U+P$ outperforms the configuration $U$ on the percentage of correctness. This is because the preprocessing method has filtered away the prunable features, which makes the search space smaller. Hence, the preprocessing method makes EAs more effective for optimal feature selection. We also observe that $F+P$ outperforms $U+P$ constantly on the percentage of correctness. This is attributed to the feedback-directed crossover and mutation, which have effectively guided EAs to explore more promising region of the solution space for locating the optimal feature selection. The average improvement for the configuration $F+P$ over $U+P$ is summarized in Table 7, where the values are calculated by summing up the differences of *%Correct* between $F+P$ and $U+P$ for all tested EAs, and divided by three (the number of test cases). Positive values mean improvements, while negative values mean the opposite. This has proven that the feedback-directed mechanism provides an improvement on the percentage of correctness for all case studies using different EAs, especially for IBEA it increases *%Correct* by 69%.

*Enhancement and runtime*: as can be seen from Table 4, the runtime of configurations $F+P$, $U+P$, and $U$ are comparable. There is not a configuration that has a clear advantage over the others in terms of the runtime. The reason is that all configurations go through the same number of evaluations. One concern of using optimization is that the configuration $F+P$ requires an extra calculation of the error position using Algorithm 4. In fact, when checking the validity of solutions in each generation, the constraints also need to be enumerated for configurations $U+P$ and $U$ during each generation of evolution. Note that checking the validity of a specific solution just requires constant time when the number of features is fixed, and no SAT solving is needed. So the extra operation of $F+P$ is only the *getFeatures* function used at line 10 of Algorithm 4, which has a low complexity. Due to the preprocessing method, $F+P$ and $U+P$ have shorter chromosome than $U$ as prunable features are not included in chromosome. To summarize, the feedback-directed mechanism requires extra time on calculation of the error positions, while the preprocessing method reduces the search space. Hence, combining both techniques (i.e., the $F+P$ configuration) has no significant differences in runtime with $U$ and $U+P$.

**Table 7**
Improvement of F+P over U+P for EAs on SPLOT.

| Algo. | $-\Delta$Time (s) | $+\Delta$HV | $+\Delta$%Correct |
|---|---|---|---|
| IBED | 0.2 | 0.01 | 4% |
| IBEA | −0.7 | 0.08 | 69% |
| NSGA-II | 0.1 | 0.05 | 15% |
| ssNSGA-II | 0.4 | 0.04 | 14% |
| MOCell | 0.7 | 0.06 | 23% |

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040

#### 5.2.5. Answer to RQ5

RQ5 examines the generality of the preprocessing method and the feedback-directed mechanism. We notice that the percentage of correctness of all tested EAs (IBED, IBEA, NSGA-II, ssNSGA-II and MOCell) have been improved by using these two enhancement techniques in configuration $F+P$. These results confirm that, the preprocessing method, and the feedback-directed crossover and mutation have provided an advantage on the percentage of correctness and HV, regardless of the underlying EAs. The reason is that the preprocessing method effectively prunes the search space, and the feedback-directed crossover and mutation allow underlying EAs to use the feedback for faster finding of valid solutions. This shows that the preprocessing method and the feedback-directed mechanism are general methods that could be applied to different EAs.

#### 5.2.6. Answer to RQ6

RQ6 examines the scalability of IBED together with the enhancement techniques. In Table 4, the results show that, for the *E-Shop* model, with $U+P$ and $U$, none of the EAs (except IBED) could locate a correct solution. On the other hand, with $F+P$, IBEA and IBED have achieved 100% and 99% of correctness respectively, while NSGA-II, ssNSGA-II and MOCell have achieved 12–14% of correctness. We have further conducted the experiment of using *50 million* generation as the termination criteria for IBEA $U+P$. It has only achieved 46% of correctness after 50 million generations which takes *3.25 h*. In contrast, the configuration IBEA $F+P$ achieves 100% of correctness by just *7 s*. Our IBED $F+P$ takes even less time 6.7 s to get 99% of correctness, and IBED $U+P$ takes *7 s* to get 87% of correctness with the default 25,000 generations (see Section 5.1.7). After we relax the termination criteria to 50,000 generations, IBED $U+P$ also gets 99% of correctness. As IBED $F+P$ can handle hundreds of features (e.g., the *E-Shop* model) with above 99% of correctness within 10 s, IBED $F+P$ enables the evaluation with the larger industrial models (see Section 5.3).

### 5.3. Evaluation with LVAT

To confirm the scalability of the IBED and the enhancement techniques, we conduct the evaluation using LVAT. As reported by Sayyad et al. [18], IBEA achieves the best results for this problem than other EAs used in Table 4. The results of our experiments with SPLOT models also comply with their observation. We also tried standard EAs (e.g., NSGA-II and MOCell) for the three LVAT models, but they found about 0% correct solutions using the parameter setting in Section 5.1.7. Besides, Sayyad et al. [18] also tried the standard NSGA-II for the LVAT models and reported the results in Table IV of their paper. Their results showed NSGA-II can only achieve around 0-1% correctness. As other EAs hardly find correct solutions on industrial models that are much larger than *E-Shop*, we just show the results of IBED and IBEA on LVAT models in Table 5.

To answer RQ1 to RQ3 with LVAT, we evaluate the effectiveness of IBED in terms of correctness, performance and diversity in Section 5.3.1. To answer RQ4 to RQ5 with LVAT, we evaluate the enhancement techniques (our preprocessing method and feedback-directed mechanism) in Section 5.3.2. The successful application of IBED with the enhancement techniques on LVAT exhibits the scalability of our approach, which answers RQ6. However, as EAs on their own fail to find any correct solution on *Linux X86*, we propose to use the seeding method for IBED and IBEA in Section 5.3.3.

#### 5.3.1. Evaluating IBED with LVAT

We evaluate IBED from the following aspects:

*Correctness and performance of IBED*: as can be observed in Table 5, for the configuration $F+P$, IBED performs quite similarly with IBEA in terms of *Time*(*s*) (execution time in seconds), *HV* (hypervolume), and *%Correct* (percentage of correct solutions in the Pareto front). For the configuration $F'+P$, the same situation is observed, except on *eCos*. For the instance *eCos*, the correctness of IBED $F'+P$ (17.9%) is lower than that of IBEA $F'+P$ (61%), as IBED may mutate too greatly to keep searching near the correct solutions. However, an interesting finding is that the diversity of IBED helps to work better than IBEA, when the feedback-directed mechanism is disable. Thus, for the configuration of $U+P$, IBED shows better correctness rate except on *Linux X86* where both IBED and IBEA have 0% correctness. For the instance *uClinux*, IBED $U+P$ achieves 100% correctness. The reason is that 1244 prunable features in *uClinux* lead to a smaller solution space (after pruning these 1244 features in $U+P$) that can contain many correct solutions. Hence, some extent of diversity (a certain probability of mutation) is preferred, especially when the feedback directed mechanism is disabled and no correct solutions have been found yet.

**Table 8**
Improvement of F+P over U+P for IBED and IBEA on LVAT.

| Algo. | −ΔTime (s) | +ΔHV | +Δ% Correct |
|---|---|---|---|
| IBED | 3.8[a] | 0.02 | 28% |
| IBEA | 10.9[b] | 0.03 | 64% |

[a] The time measurement is based on using cache results. We ignore the Linux X86 case, as IBED $U+P$ causes runtime exceptions of memory limit.
[b] We ignore the Linux X86 case, as IBEA $U+P$ using cache throws runtime exceptions of memory limit.

*Non-dominated solutions found by IBED*: neither IBED nor IBEA can find correct solutions on *Linux X86*. In Table 6, we show the results for two sets of feature attribute values on *eCos* and *uClinux*. For the model *eCos*, IBEA $F+P$ find more correct ($|B| > |A|$) and also more non-dominated solutions ($|N_B| > |N_A|$) than IBED $F+P$ for both two value sets (*ecos*1 and *ecos*2). The reason is that *eCos* has very few prunable features, 54 out of 1244 (see Table 3), but as many as 3146 constrains. DE operations in IBED does not help much, as the Pareto front for non-dominated solutions is small and more mutations may reduce the correctness of the results. In contrast, *uClinux* has more prunable features but less constrains. The Pareto front for *uClinux* is large enough to tolerate some extent of mutations without significantly reducing the correctness. So on *uClinux*, IBED achieves more correct solutions (1925 and 1875 for the two value sets *uClinux*1 and *uClinux*2, respectively) than IBEA (1511 and 1620, respectively). Among the union of solutions found by IBED and IBEA, we calculate the total non-dominated ones as $N_A \cup N_B$. The results also show that IBED finds most of the non-dominated solutions (894/968 and 875/1074 for *uClinux*1 and *uClinux*2, respectively).

#### 5.3.2. Evaluating enhancement techniques with LVAT

Table 5 demonstrates how the enhancement techniques improve IBED and IBEA on LVAT models. Configuration $F'+P$ is the same as $F+P$, with the exception that the mutation probability $P_{mut}$ is set to 0.01 (for $F+P$, $P_{mut} = 0.0000001$). The average improvement for the configuration $F+P$ over $U+P$ is summarized in Table 8. In general, the finding from Table 7 complies with that from Table 8 – the feedback-directed mechanism improves more for IBEA. The reason is that IBEA needs more for the diversity that feedback-directed mutation and crossover provide, while IBED by itself prefers solutions of diversity due to differential evolution.

In Table 5, we notice that for *eCos* and *uClinux*, IBEA $F+P$ achieves 90% above correctness for all cases, while IBEA $U+P$ does not find any correct solution after 100000 executions. Although $F+P$ achieves overall better runtime, it does not have clear advantage over $U+P$ for all models. These results have confirmed for the better percentage of correctness and comparable runtime of $F+P$ over $U+P$. For *Linux X86*, none of the configurations ($F+P$, $F'+P$, and $U+P$)

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040

**Table 9**
Non-dominated solutions found by IBED F+P (A) and IBEA F+P (B) that both use 3 common seeds on Linux X86.

| Model | A %Correct | B %Correct | A Time (s) | B Time (s) | A HV | B HV | $\|A\|$ | $\|B\|$ | $\|A \cap B\|$ | $\|N_A \cup N_B\|$ | $\|N_A\|$ | $\|N_B\|$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Linux X86 1 | 28.6 | 30.2 | 2495.4 | 2311.4 | 0.23 | 0.23 | 464 | 251 | 0 | 632 | 421 | 213 |
| Linux X86 2 | 20.3 | 25.4 | 2611.3 | 2549.4 | 0.23 | 0.23 | 344 | 164 | 0 | 447 | 315 | 134 |

for IBED or IBEA has found a correct solution. Therefore, we make use of the "seeding method" proposed by [18].

#### 5.3.3. Seeding method for Linux X86

To find a "seed", which is a correct solution that leads to more correct solutions, Sayyad et al. [18] made use of two methods, namely SMT (abbr. for satisfiability modulo theories) solver and IBEA with two objectives to find correct solutions. Then they planted the seed in the initial population of IBEA with the hope to find more valid solutions. In this paper, we adopt three seeding methods, namely the two methods used by Sayyad et al. [18] and also IBED with two objectives. For each seeding method, we generate one seed. We run these three seeds for both IBED *F+P* and IBEA *F+P*, and compare the results with the seeding method with IBEA proposed by [18] (the unguided version of IBEA).

*Seed 1 produced by SMT solver:* Microsoft Z3 SMT solver [48] is used to find a seed solution. In our case, Z3 successfully finds a valid solution in around 3 s (we repeat for 30 times, and medium of the number of selected features is 1455). With the seed, IBEA *F+P* successfully finds 34 correct solutions using no more than 30 s, but IBEA *U+P* finds no new solution in 30 min.

*Seed 2 produced by IBEA:* IBEA with two objectives is used to generate seed 2. Using seed 2 to get solutions for 5 objectives, IBEA *F+P* takes around *40 s* to get more than 30 correct solutions. While for IBEA *U+P*, it spends a total of *3.5 h* of execution time for 30 correct solutions. *F+P* has shortened the search time of *U+P* for more than 200 times. In particular, *U+P* spends 3 h to generate the seed, and spends half an hour to obtain 30 correct solutions. Even given another half hour, *U+P* finally obtains 36 correct solutions.

*Seed 3 produced by IBED:* we adopt the same way of using two objectives in IBEA to generate seed 3 in IBED. With seed 3 for all the 5 objectives, IBED *F+P* takes around *50 s* to get more than 20 correct solutions. IBED *U+P*, in medium case, spends around *4 h* of execution time for getting more than 20 correct solutions, which takes longer time than IBEA *F+P*. We also observe that in seed generation, the solutions found by IBED exhibit more variability due to the diversity based selection criteria.

*The quality of three seeds:* we observe that the seeds generated by IBED and IBEA with two objectives, are better than the seed generated by the Z3 SMT solver. These results have shown that *F+P* outperformed *U+P* with all the three seeds. Both configurations *F+P* and *U+P* of IBED or IBEA find more solutions using seed 2 or seed 3, compared with seed 1, according to our experiments. For example, using seed 2 for IBEA *F+P* or seed 3 for IBED *F+P*, they can find around 30 correct solutions on average in 30 executions, while for seed 1 either IBED *F+P* or IBEA *F+P* can find only less than 20 solutions on average. This conforms to the observation in [18]. According to [18], it is because the seed generated by IBEA with two objectives has more selected features, and the "feature-rich" seed allows the effective search of other valid solutions.

*Evaluating with three common seeds:* to avoid the bias due to different seeds, we include all the three seeds in the initial populations. We run IBEA *F+P* and IBED *F+P* 30 times with random values for feature attributes, and use the medium value for comparison. IBED gets 20% as correctness, taking 2566 s (for 100,000 generations) and achieving HV of 0.23. In contrast, IBEA gets 27% as correctness, taking 2452 s and achieving the same HV of 0.23. Although IBEA exhibits a slightly higher correctness, it does not mean IBEA can find more unique and non-dominated solutions. Thus, we randomly choose two sets of feature attribute values, and compare the specific solutions found by the two algorithms.

In Table 9, we show the non-dominated solutions found by IBED *F+P* and IBEA *F+P* using the three common seeds. Given two sets of random values for feature attributes, on average, IBED has found 97.7% and 135.1% more unique and non-dominated solutions than IBEA on both sets. Hence, IBEA has a slightly higher correctness, but find less unique correct solutions ($|B|$ is much smaller than $|A|$ for both value sets of feature attribute). The reason is that the solutions found by IBEA according to the three seeds are quite duplicated in 30 executions. In contrast, for the same seeds, solutions found by IBED are more diverse. The lower diversity of IBEA solutions also leads to the smaller number of non-dominated solutions, compared with IBED, for both sets of feature attribute values.

### 5.4. Summary and discussion

From the results reported in Sections 5.2 and 5.3, we can conclude that, with the feedback-directed mechanism, IBED *F+P* is comparable with IBEA *F+P* in terms of execution time, hypervolume and correctness ratio on the models except *Linux X86*. Without feedback-directed mechanism (e.g., the *U+P* and *U* configurations), IBED can achieve higher correctness ratio than the corresponding version of IBEA on the models except *Linux X86*. For the diversity, compared with the best configuration of IBEA (IBEA *F+P*), IBED *F+P* also find more non-dominated solutions on the models except *eCos* and *Linux X86*.

For the *Linux X86* model, using EAs alone cannot find correct solutions in the default parameter setting. When using the same seeding methods, IBED *F+P* can also find more non-dominated solutions than IBEA *F+P*. Hence, on all models except *eCos*, IBED has exhibited the capability in finding more non-dominated solutions while retaining the correctness ratio (or just slightly reducing it).

After the inspection on *eCos*, we find that *eCos* is a distinct model. First, as it contains few prunable features (only 54 out of 1244), the preprocessing method does not help much so that the problem space is larger than 2 to the power of 1000. Second, according to the report [49], many features (about 30% in the *eCos* model) declare cross-tree dependencies (a.k.a. CTCs in this paper). As can be seen from Table 5, these characteristics make it difficult for EAs to directly find many correct solutions (in the *U+P* configuration). This deactivates the diversity of differential evolutionary operations of IBED, as mutating one correct solution may not lead to another correct solution due to CTCs.

All these experiments shed light on how to apply EAs for the optimal feature selection problem. Under all scenarios, the preprocessing method and feedback-directed mechanism are helpful. IBED suits well for many feature models with correct solutions that are distributed continuously, but not densely gathered. In contrast, IBEA suits better for models with correct solutions gathered near zero-violation point in solution space (e.g., the *eCos* model) [10]. For *Linux X86* that has thousands of features after preprocessing and a noticeable coverage of CRCs, combining IBED with the seeding methods is recommended. It is effective to derive better solutions from a good seed solution via differential evolution operations.

In addition, we further perform experiments to investigate how the mutation parameter $P_{mut}$ affects the feedback-directed

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040

mechanism. To better observe the effect, we examine the number of executions required to obtain 50% of correct solutions in the Pareto front. For LVAT models except *Linux X*86, *F+P* only needs a small number of generations to reach 50% of correct solutions (6300 and 600 generations for *eCos* and *uClinux*, respectively). In contrast, *F'+P* needs 62,400 and 2100 generations to reach 50% of correct solutions for *eCos* and *uClinux*. This observation indicates that smaller $P_{mut}$ leads to faster convergence of correct solutions in Pareto front. This is because smaller $P_{mut}$ minimizes the modification of non-error positions; therefore, it allows IBEA to focus more on the correction of constraint violations.

### 5.5. Threats to validity

There are several threats to validity. The first threat to external validity is due to the fact that values for the feature attributes (i.e., *Cost*, *Defects*, and *Used_Before*) were randomly generated. This is due to difficulty in obtaining the attributes that are associated with real-world products since many of them are proprietary. To mitigate the effect of randomness, we generate 10 sets of attributes for each case study. Furthermore, for each set of attributes, we run each EA repeatedly for 30 times, and report the medium values of the metrics. Our current work is conducted on the feature models commonly used by other researchers [10,17,18] for fair comparison. Future work should involve the use of real data for the evaluation, e.g. the *EC2* feature model used in the study [50], the *Drupal* feature model introduced in the study [51] and the *SAS* architecture model used in the work [52].

The second threat to external validity comes from the incomplete synthetic feature attributes and objective functions. Currently, we only use 3 feature attributes and 5 objective functions mentioned in Sections 5.1.4 and 5.1.5. And now only *Obj2* (richness of features) is logically an objective to be maximized, as selecting more features leads to the larger number of possible violated constraints (*Obj1*), more costs (*Obj3*), more unused features (*Obj4*) and more defects (*Obj5*). Thus, *Obj2* is logically competing with other objectives. In future, we should test using a different number of objective functions and feature attributes. We could also try other objective functions to be logically maximized, other stakeholders' restrictions and other attributes.

The threat to internal validity stems from our choice of using an exemplar parameter set (e.g., for crossover and mutation probability), which comes with the default setting of jMetal, in order to cope with the combinatorial explosion of options. To address these threats, it is clear that more experimentations with different feature models and experimental parameters are required, so that we could investigate effects that have not been made explicit by our dataset and experimental parameters.

## 6. Related work

### 6.1. Optimal feature selection for SPL

White et al. [53] reduced the feature selection problem in SPL to a multidimensional multi-choice knapsack problem (MMKP). They proposed a polynomial time approximation algorithm, called filtered Cartesian flattening (FCF), to derive an optimal feature configuration subject to resource constraints. Their evaluation showed that FCF can stably achieve the optimality above 90% even when the number of resources increases up to 91, while the optimality of constraint satisfaction problem (CSP) based feature selection proposed by [32] drops down to 30% when there are 91 resources.

Although FCF can achieve a highly optimal solution, it requires significant computing time. To address the problem of scalability, Guo et al. [16] proposed GAFES (a genetic algorithm based approach). The rationale is that GAs are quite suitable for the highly constrained problems, such as the feature selection (product configuration) problem. GAFES can integrate a new *repair* operator for feature selection and also define a *penalty* function for resource constraints. The evaluation showed GAFES may not beat the FCF and CSP in optimality, but it scaled up to large-scale models with a reasonable optimality.

GAs used by Guo et al. [16] only allow single objective function. Further, GAFES repairs each solution explicitly, and does not take advantage of the evolution of GA for repairing. To address this problem, Sayyad et al. [10] investigated the use of different types of EAs that support multi-objective function for the optimal feature selection. They adopted 7 types of EAs, such as IBEA, NSGA-II and MOCell, to search for the optimal product. The results have shown that IBEA performs much better than other 6 EAs in terms of time, correctness and satisfaction to user preferences. Sayyad et al. [17] improved their previous work [10] by turning down the crossover probability from 0.9 to 0.1 and mutation probability from 0.05 to 0.01, and they reported HV-mean and spread mean may increase by 5–10% in most cases. To further make the searching scalable, Sayyad et al. [18] proposed the use of EA with simple heuristic in larger product lines from LVAT repository. They proposed the use of static analysis to identify prunable features for reducing search space, and the use of the seeding method to find more correct products for Linux X86 Kernel. Olaechea et al. [54] reported that it is feasible to use the exact algorithm, the Guided Improvement Algorithm (GIA) [55], to find optimal solutions for small SPL models. They also reported that IBEA is more efficient for the large model (*E-Shop*), with using less than 20 min, but it requires much effort in setting up the best parameter for acceptable sub-optimal solutions. To speed up the performance of MOEAs, Guo et al. [56] introduced five novel parallel algorithms for Multi-Objective Combinatorial Optimization (MOCO) to allow parallel processing.

Our method has improved the method proposed in [10,17,18] by incorporating feedback-directed mechanism for EAs (see Section 5). We also show that our method for finding prunable feature with Algorithm 2 is always not fewer than the method used by the work [18] (see Section 5.1.3). Compared with the work [56], our work complements with them by considering feedback-directed mechanism for MOCO problem.

To save the extra time due to the seeding method [18] or new SAT-based replacement and mutation operators [57], Hierons *et al.* [58] proposed the SIP (ShrInk Prioritize) approach to prioritize the objective of correctness, which is different from the traditional MOEAs that treat all objectives equally. Besides, SIP also adopts a novel representation that always satisfies two types of constraints: constraints relevant to core features, and constraints relevant to a mandatory feature that is meanwhile a parent feature of a group of sub-features. The evaluations showed that SIP can return a population with only correct solutions in all executions.

In this paper, although we still use the traditional MOEAs that treat all objectives equally, we adopt the feedback-directed mechanism to find more correct solutions than using the standard MOEAs. Similar to the idea of the novel representation used in the work [58], we also apply the pruning method to reduce the search space. Compared with the work [58] that focuses on the correctness of the solutions, we focus on the diversity of correct solutions even with some compromise at the correctness ratio of the final population. In future, the idea of adopting a DE population can be combined with the SIP method to help the diversity of the correct solutions.

In addition to the optimal feature selection problem, Cruz et al. [59] combined fuzzy inference systems and MOEAs to select best products of various users and group these products in a portfolio. Hence, they solved the product portfolio scoping problem (the scope of product costs, line of code, complexity, coupling, etc.) using computational intelligence techniques.

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040

### 6.2. Constraint solving for SPL

In the previous work [60], an experiment for measuring the efficiency of BDD (abbr. for binary decision diagrams), SAT and CSP solvers is conducted using feature models from SPLOT repository. They reported the long runtime for certain operations and an exponential runtime increase with the number of features for non-BDD solvers on the "valid" operations. Later, the state-of-art solvers (e.g., JavaBDD BDD, JaCoP CSP, and SAT4J SAT) were used to answer the questions such as "derive one valid product from a feature model" and "number of products" [61]. They found that CSP and SAT solvers have exponential time complexity as the feature size of feature model increases, while BDD just requires a maximum of 28 s to derive one valid product for web-portal, without considering the quality of feature attributes. Thus, these automated reasoning techniques can be precise, but generally not scalable for large feature models. Our work complements with their work by using enhanced EAs that scale well for large feature models.

Recently, Henard et al. [57] proposed to combine IBEA with constraint solving. Essentially, they still adopted SAT to provide the correct solutions. They permuted different SAT parameters to maximize the diversity of SAT solutions in a cheap way by calling SAT solver hundreds of times. Hence, they got hundreds of correct solutions from SAT for new mutation and replacement operations on the invalid ones in the IBEA population during evolution. In this paper, to clearly show the differences between IBED and IBEA, we compare them without using solutions from SAT solving (except for *Linux X86* where traditional MOEAs all fail to find a correct solution). For *Linux X86*, we just use 3 seed solutions to eliminate the bias due to the different sources of seed solutions. In future work, it is interesting to investigate whether further benefits can be obtained by combining IBED and SAT solver, using more seeds from SAT or new SAT-based replacement and mutation operators.

### 6.3. Feedback-directed method

Pacheco et al. [62] proposed RANDOOP, a feedback-directed mechanism for performing random test. It uses erroneous results of previous method invocation to generate a better random test. Clarke et al. [63] proposed CEGAR, which uses spurious counterexamples as a feedback to guide the refinement process. Our method is on feedback-directed methods in EAs for the optimal feature selection. This work is also related to using EAs and SMT solvers in tackling software engineering problems. In the previous work [64], we make use of genetic algorithm in calculating the optimal recovery plan during service failure. In our related work on time requirement management [65,66], we calculate the local time requirements of individual components given the global time requirement of the system with Z3 SMT solver [48]. In this work, our focus on making use of EAs in tackling optimal feature selection.

### 6.4. Search-based software engineering

In addition to the SPL domain, MOEAs have also been applied to various software engineering problems. Harman et al. [67] proposed the term search-based software engineering (SBSE), and reported that the surveyed and proposed optimization techniques for SE problems by 2001 were all single-objective based. Seeing the potential of using multi-objective optimization, Harman [68] discussed about the possible usage of the meta-heuristic search techniques such as: simulated annealing and genetic algorithm. Harman considered it insensible combination of multiple metrics into an aggregate fitness in the way of assigning coefficients, and suggested to use Pareto optimality rather than aggregate fitness.

## 7. Conclusion and future work

Due to the large and highly constrained search space, the product configuration (i.e., optimal feature selection) is a difficult task. In this study, the major contribution is to propose a novel algorithm – IBED to achieve both correctness and diversity of the found solutions. IBED maintains dual populations, one IBEA population for correctness and one DE population for diversity of results. The key of designing IBED lies in how to design the DE operators for evolving solutions of the specific application and meanwhile achieving the population diversity. Further, we combine two enhancement techniques for EAs, i.e., the feedback-directed mechanism and the preprocessing. For the feedback-directed mechanism, the key is how to analyze violated constraints of an invalid solution, and use the information as a feedback to guide the search toward more correct solutions. In addition, we also introduce the preprocessing step to reduce the search space, by removing the prunable features in valid products. Our evaluation shows that IBED can find more unique and non-dominated solutions than IBEA on most cases. Generally, IBED compromises slightly at correctness, but find more non-dominated solutions in return.

Furthermore, the preprocessing technique and the feedback-directed mechanism have both improved over existing unguided EAs on the optimal feature selection. The feedback-directed IBEA successfully found 69% and 64% more correct solutions for case studies in SPLOT and LVAT repositories, compared to the unguided IBEA. Due to its own diversity of IBED, the feedback mechanism only improves the correctness by 4% and 28%. In addition, with "seeding method" proposed by Sayyad et al. [18] and feedback-directed IBEA or IBED, we have reduced the running time from about *3.5–4 h* to about *40–50 s* to find more than 30 correct solutions on *Linux X86*.

As future works, we plan to find other types of feedback that could be incorporated in EAs, to address the scalability problem of large feature models, such as *Linux X86*. Moreover, we would also investigate extensibility of our method to other software engineering problems. Meanwhile, we plan to further evaluate the method using different case studies (e.g., the *EC2* feature model [50], the *Drupal* feature model [51] and the *SAS* architecture model [52]). Lastly, it is interesting to combine constraint solving with IBED, and compare the result with that of using IBEA and constraint solving [57].

## References

[1] K.C. Kang, J. Lee, P. Donohoe, Feature-oriented product line engineering, IEEE Softw. 19 (4) (2002) 58–65.
[2] K. Pohl, G. Böckle, F. van der Linden, Software Product Line Engineering – Foundations, Principles, and Techniques, Springer, 2005, http://dx.doi.org/10.1007/3-540-28901-1.
[3] S. Apel, C. Kästner, An overview of feature-oriented software development, J. Object Technol. 8 (5) (2009) 49–84, http://dx.doi.org/10.5381/jot.2009.8.5.c5.
[4] P. Clements, L. Northrop, L.M. Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley Professional, 2001.
[5] Y. Xue, Reengineering legacy software products into software product line based on automatic variability analysis, in: Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu, HI, USA, May 21–28, 2011, 2011, pp. 1114–1117, http://dx.doi.org/10.1145/1985793.1986009.
[6] M. Sinnema, S. Deelstra, Classifying variability modeling techniques, Inf. Softw. Technol. 49 (7) (2007) 717–739.
[7] L. Chen, M.A. Babar, N. Ali, Variability management in software product lines: a systematic review, in: Software Product Lines, 13th International Conference, SPLC 2009, San Francisco, CA, USA, August 24–28, 2009, 2009, pp. 81–90.
[8] T. Thüm, S. Apel, C. Kästner, I. Schaefer, G. Saake, A classification and survey of analysis strategies for software product lines, ACM Comput. Surv. 47 (1) (2014) 6:1–6:45, http://dx.doi.org/10.1145/2580950.
[9] D. Benavides, S. Segura, A.R. Cortés, Automated analysis of feature models 20 years later: a literature review, Inf. Syst. 35 (6) (2010) 615–636, http://dx.doi.org/10.1016/j.is.2010.01.001.

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040

[10] A.S. Sayyad, T. Menzies, H. Ammar, On the value of user preferences in search-based software engineering: a case study in software product lines, in: ICSE, 2013, pp. 492–501.
[11] I. Jacobson, M.L. Griss, P. Jonsson, Software Reuse – Architecture, Process and Organization for Business, Addison-Wesley-Longman, 1997.
[12] J. Karlsson, S. Olsson, K. Ryan, Improving practical support for large-scale requirement prioritising, Requir. Eng. 2 (1) (1997) 51–60.
[13] M.L. Griss, J. Favaro, M.d. Alessandro, Integrating feature modeling with the RSEB, in: Proceedings of the 5th International Conference on Software Reuse, ICSR '98, 1998, 76.
[14] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, A.S. Peterson, Feature-oriented domain analysis (foda) feasibility study, in: Tech. Rep. CMU/SEI-90-TR-21, Carnegie Mellon University, November 1990.
[15] S. She, R. Lotufo, T. Berger, A. Wasowski, K. Czarnecki, Reverse engineering feature models, in: ICSE, 2011, pp. 461–470.
[16] J. Guo, J. White, G. Wang, J. Li, Y. Wang, A genetic algorithm for optimized feature selection with resource constraints in software product lines, J. Syst. Softw. 84 (12) (2011) 2208–2221.
[17] A.S. Sayyad, J. Ingram, T. Menzies, H. Ammar, Optimum feature selection in software product lines: let your model and valuesguide your search, in: CMSBSE, 2013, pp. 22–27.
[18] A.S. Sayyad, J. Ingram, T. Menzies, H. Ammar, Scalable product line configuration: a straw to break the camel's back, in: ASE, 2013.
[19] E. Zitzler, S. Künzli, Indicator-based selection in multiobjective search, in: PPSN, 2004, pp. 832–842.
[20] T.H. Tan, Y. Xue, M. Chen, J. Sun, Y. Liu, J.S. Dong, Optimizing selection of competing features via feedback-directed evolutionary algorithms, in: Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, Baltimore, MD, USA, July 12–17, 2015, 2015, pp. 246–256.
[21] S. Das, P.N. Suganthan, Differential evolution: a survey of the state-of-the-art, IEEE Trans. Evol. Comput. 15 (1) (2011) 4–31.
[22] M. Mendonça, M. Branco, D.D. Cowan, S.P.L.O.T.: software product lines online tools, in: OOPSLA Companion, 2009, pp. 761–762.
[23] Linux Variability Analysis Tools (LVAT) Repository, https://code.google.com/p/linux-variability-analysis-tools/source/browse/?repo=formulas.
[24] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, J.H. Obbink, K. Pohl, Variability issues in software product lines, in: Software Product-Family Engineering, 4th International Workshop, PFE 2001, Bilbao, Spain, October 3–5, 2001, Revised Papers, 2001, pp. 13–21.
[25] J.D. McGregor, D. Muthig, K. Yoshimura, P. Jensen, Guest editors' introduction: successful software product line practices, IEEE Softw. 27 (3) (2010) 16–21, http://dx.doi.org/10.1109/MS.2010.74.
[26] H. Post, C. Sinz, Configuration lifting: Verification meets software configuration, in: 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15–19 September 2008, L'Aquila, Italy, 2008, pp. 347–350, http://dx.doi.org/10.1109/ASE.2008.45.
[27] K. Czarnecki, S. Helsen, U.W. Eisenecker, Staged configuration using feature models, in: Software Product Lines, Third International Conference, SPLC, Boston, MA, USA, August 30–September 2, 2004, 2004, pp. 266–283.
[28] K. Czarnecki, U.W. Eisenecker, Generative Programming – Methods, Tools and Applications, Addison-Wesley, 2000.
[29] D.S. Batory, Feature models, grammars, and propositional formulas, in: 9th International Conference Software Product Lines, SPLC, Rennes, France, September 26–29, 2005, 2005, pp. 7–20.
[30] Y. Xue, Z. Xing, S. Jarzabek, Understanding feature evolution in a family of product variants, in: WCRE, 2010, pp. 109–118.
[31] K.C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, M. Huh, FORM: A feature-oriented reuse method with domain-specific reference architectures, Ann. Softw. Eng. 5 (1998) 143–168, http://dx.doi.org/10.1023/A:1018980625587.
[32] D. Benavides, P.T. Martín-Arroyo, A.R. Cortés, Automated reasoning on feature models, in: CAiSE, 2005, pp. 491–503.
[33] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[34] J.J. Durillo, A.J. Nebro, F. Luna, E. Alba, On the effect of the steady-state selection scheme in multi-objective genetic algorithms, in: EMO, 2009, pp. 183–197.
[35] A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, E. Alba, Mocell: a cellular genetic algorithm for multiobjective optimization, Int. J. Intell. Syst. 24 (7) (2009) 726–746.
[36] A.S. Sayyad, Evolutionary Search Techniques with Strong Heuristics for Multi-objective Feature Selection in Software Product Lines, (Ph.D. thesis), West Virginia University, 2014.
[37] Z. Michalewicz, Genetic Algorithms+ Data Structures= Evolution Programs, Springer Science & Business Media, 2013.
[38] K. Price, An Introduction to Differential Evolution, New Ideas in Optimization, McGraw-Hill Ltd., UK, Maidenhead, UK, 1999.
[39] J. Zhong, Y.-S. Ong, W. Cai, Self-learning gene expression programming, IEEE Trans. Evol. Comput. 99 (2015) 1.
[40] SAT4J – The Boolean Satisfaction and Optimization Library in Java, http://www.sat4j.org/.
[41] M. Srinivas, L.M. Patnaik, Adaptive probabilities of crossover and mutation in genetic algorithms, IEEE Trans. Syst. Man Cybern. 24 (4) (1994) 656–667.
[42] J.J. Durillo, A.J. Nebro, jmetal: a java framework for multi-objective optimization, Adv. Eng. Softw. 42 (10) (2011) 760–771.
[43] T. Wagner, N. Beume, B. Naujoks, Pareto-, aggregation-, and indicator-based methods in many-objective optimization, in: 4th International Conference Evolutionary Multi-Criterion Optimization, EMO 2007, Matsushima, Japan, March 5–8, 2007, 2006, pp. 742–756.
[44] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach, IEEE Trans. Evol. Comput. 3 (4) (1999) 257–271.
[45] M. Mendonça, T.T. Bartolomei, D.D. Cowan, Decision-making coordination in collaborative product configuration, in: SAC, 2008, pp. 108–113.
[46] T. Berger, S. She, R. Lotufo, K. Czarnecki, T. Berger, S. She, R. Lotufo, A. Wasowski, K. Czarnecki, Variability Modeling in the Systems Software Domain, Tech. Rep., University of Waterloo, 2012.
[47] A. Arcuri, L.C. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, in: ICSE, 2011, pp. 1–10.
[48] L.M. de Moura, N. Bjørner, Z3: an efficient SMT solver, in: TACAS, 2008, pp. 337–340.
[49] T. Berger, R. Pfeiffer, R. Tartler, S. Dienst, K. Czarnecki, A. Wasowski, S. She, Variability mechanisms in software ecosystems, Inf. Softw. Technol. 56 (11) (2014) 1520–1535, http://dx.doi.org/10.1016/j.infsof.2014.05.005.
[50] J. García-Galán, P. Trinidad, O.F. Rana, A.R. Cortés, Automated configuration support for infrastructure migration to the cloud, Future Gen. Comp. Syst. 55 (2016) 200–212.
[51] A.B. Sánchez, S. Segura, J.A. Parejo, A. Ruiz-Cortés, Variability testing in the wild: the Drupal case study, Software & Systems Modeling (2015) 1–22, http://dx.doi.org/10.1007/s10270-015-0459-z. URL http://dx.doi.org/10.1007/s10270-015-0459-z.
[52] N. Esfahani, S. Malek, K. Razavi, Guidearch: guiding the exploration of architectural solution space under uncertainty, in: 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18–26, 2013, 2013, pp. 43–52, URL http://dl.acm.org/citation.cfm?id=2486795.
[53] J. White, B. Dougherty, D.C. Schmidt, Selecting highly optimal architectural feature sets with filtered Cartesian flattening, J. Syst. Softw. 82 (8) (2009) 1268–1284.
[54] R. Olaechea, D. Rayside, J. Guo, K. Czarnecki, Comparison of exact and approximate multi-objective optimization for software product lines, in: 18th International Software Product Line Conference, SPLC '14, Florence, Italy, September 15–19, 2014, 2014, pp. 92–101, http://dx.doi.org/10.1145/2648511.2648521.
[55] D. Rayside, H.-C. Estler, D. Jackson, The guided Improvement Algorithm for Exact, General-purpose, Many-objective Combinatorial Optimization, Tech. Rep. MIT-CSAIL-TR-2009-033, MIT CSAIL, 2009.
[56] J. Guo, E. Zulkoski, R. Olaechea, D. Rayside, K. Czarnecki, S. Apel, J.M. Atlee, Scaling exact multi-objective combinatorial optimization by parallelization, in: ASE, 2014.
[57] C. Henard, M. Papadakis, M. Harman, Y.L. Traon, Combining multi-objective search and constraint solving for configuring large software product lines, in: 37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16–24, 2015, Vol. 1, 2015, pp. 517–528, http://dx.doi.org/10.1109/ICSE.2015.69.
[58] R.M. Hierons, M. Li, X. Liu, S. Segura, W. Zheng, Sip: Optimal product selection from feature models using many-objective evolutionary optimization, ACM Trans. Softw. Eng. Methodol. 25 (2) (2016), 17:1-17:39.
[59] J. Cruz, P. de Alcântara dos Santos Neto, R. Britto, R.A.L. Rabelo, W. Ayala, T. Soares, M. Mota, Toward a hybrid approach to generate software product line portfolios, in: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20–23, 2013, 2013, pp. 2229–2236.
[60] R. Pohl, K. Lauenroth, K. Pohl, A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models, in: ASE, 2011, pp. 313–322.
[61] R. Pohl, V. Stricker, K. Pohl, Measuring the structural complexity of feature models, in: ASE, 2013, pp. 454–464.
[62] C. Pacheco, S.K. Lahiri, M.D. Ernst, T. Ball, Feedback-directed random test generation, in: ICSE, IEEE Computer Society, 2007, pp. 75–84.
[63] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement, in: CAV, 2000, pp. 154–169.
[64] T.H. Tan, M. Chen, É. André, J. Sun, Y. Liu, J.S. Dong, Automated runtime recovery for QOS-based service composition, in: WWW, 2014, pp. 563–574.
[65] T.H. Tan, É. André, J. Sun, Y. Liu, J.S. Dong, M. Chen, Dynamic synthesis of local time requirement for service composition, in: ICSE, 2013, pp. 542–551.
[66] Y. Li, T.H. Tan, M. Chechik, Management of time requirements in component-based systems, in: FM, 2014, pp. 399–415.
[67] M. Harman, B.F. Jones, Search-based software engineering, Inf. Softw. Technol. 43 (14) (2001) 833–839.
[68] M. Harman, The current state and future of search based software engineering, in: FOSE, 2007, pp. 342–357.

Please cite this article in press as: Y. Xue, et al., IBED: Combining IBEA and DE for optimal feature selection in software product line engineering, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.040