Heuristic and exact algorithms for product configuration in software product lines

Juliana Alves Pereira\textsuperscript{a}, Lucas Maciel\textsuperscript{b}, Thiago F. Noronha\textsuperscript{b} and Eduardo Figueiredo\textsuperscript{b}

\textsuperscript{a}Department of Computer Science, University of Magdeburg, Magdeburg, Germany
\textsuperscript{b}Department of Computer Science, Federal University of Minas Gerais, Belo Horizonte, Brazil
E-mail: juliana.alves-pereira@ovgu.de [Pereira]; lucasmaciel82@dcc.ufmg.br [Maciel]; tfn@dcc.ufmg.br [Noronha]; figueiredo@dcc.ufmg.br [Figueiredo]

Received 3 February 2016; received in revised form 6 February 2017; accepted 15 March 2017

Abstract

Software product line (SPL) is a set of software applications that share a common set of features satisfying the specific needs of a particular market segment. SPL engineering is a paradigm to develop software applications that commonly use a feature model to capture and document common and variable features, and their relationships. A big challenge is to derive one product among all possible products in the SPL, which satisfies the business and customer requirements. This task is known as product configuration. Although product configuration has been extensively investigated in the literature, customer’s preferences are frequently neglected. In this paper, we propose a novel approach to configure a product that considers both qualitative and quantitative feature properties. We model the product configuration task as a combinatorial optimization problem, and heuristic and exact algorithms are proposed. As far as we are concerned, this proposal is the first work in the literature that considers feature properties in both leaf and nonleaf features. Computational experiments showed that the best of our heuristics found optimal solutions for all instances where those are known. For the instances where optimal solutions are not known, our heuristic outperformed the best solution obtained by a one-hour run of the exact algorithm by up to 67.89%.

Keywords: software product line; product configuration; search-based software engineering; combinatorial optimization; heuristic

1. Introduction

The growing need for developing larger and more complex software applications demands better support for reusable software artifacts (Pohl et al., 2005). In order to address these demands, software product line (SPL) has been increasingly adopted in the software industry (Clements and Northrop, 2001; Deelstra et al., 2004; Van der Linden et al., 2007; Apel et al., 2013). SPL is a set of software systems that share a common set of features satisfying the specific needs of a particular market segment (Pohl et al., 2005). It is built around a set of common software components (called
features) that allow product configuration (Clements and Northrop, 2001; van der Linden et al., 2007). A feature is an increment in functionality or a system property relevant to stakeholders (Kang et al., 1990; Batory, 2005). It may also refer to functional requirements, architecture, or design patterns (Bernardo et al., 2002). Feature model was proposed by Kang et al. (1990) as a part of the feature-oriented domain analysis (FODA) method. Since then, feature models have been applied in a number of domains, including mobile phones (Figueiredo et al., 2008; Czarnecki et al., 2012), telecom systems (Griss et al., 1998), smart houses (Cetina et al., 2009), network protocols (Barbeau and Bordeleau, 2002), Linux kernel (Lotufo et al., 2010), among others. Nowadays, they are the standard variability modeling technique in SPL.

In a feature model, there are common features found in all products of the product line (a.k.a. mandatory features) and variable features that allow the distinction between products in the product line (a.k.a. optional features). Optional features define specific points of variation, and their role is to allow the instantiation of different products by enabling or disabling specific SPL functionality according to some specific criteria, such as business and customer requirements (Goedicke et al., 2004; Pohl et al., 2005). Moreover, feature properties are classified as functional and nonfunctional. Functional properties (FPs) define software functional requirements. Non-FPs (NFPs) refer to software nonfunctional requirements. An NFP is said quantitative when the information is quantifiable (e.g., cost), and qualitative when the information is not quantifiable (e.g., customer’s degree of preference; Espinoza et al., 2006; Bagheri et al., 2010). Kang et al. (1998) were the first to suggest the use of NFP on feature models. Recently, a number of studies have been investigating SPL approaches that consider NFP (Benavides et al., 2005; Espinoza et al., 2006; White et al., 2009; Bagheri et al., 2010b; Olaechea et al., 2012; Siegmund et al., 2012; Asadi et al., 2014; Bagheri and Ensan, 2014; Zhang et al., 2014; Fédérle et al., 2015).

Feature models specify the products that can be generated by the respective SPL. They define the mandatory and optional features, as well as their properties and relationships (Kang et al., 1990; Czarnecki and Eisenecker, 2000). All products that can be obtained from the SPL are derived from the respective feature model and have to satisfy all their constraints. Feature models are frequently represented as a feature-tree structure (Kang et al., 2002), where nodes denote the features and edges illustrate the relationships between parent and child features (Batory, 2005; Czarnecki and Wasowski, 2007). These relationships define how features can be combined to obtain a valid product.

Figure 1a illustrates a simplified extended feature model of a smart home, using the notation inspired by Benavides et al. (2005) and Olaechea et al. (2012). Feature models can be extended by adding NFPs as feature attributes, such as cost and preference. Smart home is an SPL for a building equipped with a set of electrical sensors and actuators, in order to allow for an intelligent sensing and controlling of the building devices. The nodes are represented by boxes, and connections between them are represented by edges. The root node, SmartHome, represents the concept of the domain being modeled. It is assumed that the root feature is part of all valid product configurations.

In Fig. 1a, mandatory features are represented by filled circles, for example, illumination. They must be selected in a product configuration if their parent feature is selected. Optional features are represented by empty circles, for example, security. They can only be selected in a product configuration if their parent feature is also selected. XOR groups are represented by interlinked edges and connected by an empty arc, for example, manual and automatic. Exactly one feature must be selected whenever the XOR group’s parent feature is selected. OR-groups are represented by interlinked edges and connected by a filled arc, for example, fire, flood, and co.
in the OR group can be selected whenever the group’s parent feature is selected. XOR and OR groups are labeled as exclusive alternative and nonexclusive alternative, respectively. According to the feature model, all configurations must include support for either manual or automatic illumination, and may optionally include support for security and media.

In addition, features can be classified as leaf features (i.e., atomic features), which are features without any children situated at the leaf level of the feature model; otherwise, they are classified as nonleaf features (i.e., nonatomic features), which are features decomposed into subfeatures and used as grouping nodes. The smart-home feature model presents 16 leaf features and 10 nonleaf features. Note that a child feature can only appear in a product configuration if its parent feature does. Thus, each of the leaf features is a decision option related to the given parent feature, resulting in 16 decision options.

In addition to features and their relationships, feature models often contain additional composition rules (Czarnecki and Eisenecker, 2000). All constraints that cannot be expressed by the feature tree are called cross-tree constraints (Czarnecki et al., 2006). They are usually written as logical expressions formed by the binary operators $\land$ (conjunction), $\lor$ (disjunction), $\rightarrow$ (implication), $\leftrightarrow$ (biconditional), and the unary operator $\neg$ (negation), as well as Boolean values and the variables that represent the features (Czarnecki et al., 2006). For example, Fig. 1a shows a cross-tree constraint, which indicates that the features sensor or detection require the feature alarm ($\text{sensor} \lor \text{detection} \rightarrow \text{alarm}$).

A feature model represents all possible product configurations in an SPL. As an example, the smart-home feature model can generate up to 7144 different product variants. A sample product configuration for the smart-home product line is illustrated in Fig. 1b. This application has a set of basic features required by the customer: security of type fire sensor, detection inside, and silent and visual alarm; automatic illumination; and cellphone media.

The development lifecycle of an SPL consists of two main phases: the domain engineering phase and application engineering phase. The domain engineering phase is responsible for defining the
commonality and variability of the SPL, which are expressed in the form of a feature model (Pohl et al., 2005; Czarnecki et al., 2006). The application engineering phase is concerned with the configuration of the SPL into one specific product, based on business and customer requirements (Goedicke et al., 2004; Van der Linden et al., 2007). Requirements are specified based on the defined features’ NFPs. The problem of selecting a set of features that meet all business and customer requirements, while satisfying the customer preferences, is called the product configuration problem (PCP; Asadi et al., 2014; Pereira et al., 2014).

Manually solving PCP is challenging due to several reasons. First, feature models tend to be inherently big and complex, with several types of variability relations and cross-tree constraints overwhelming users to identify an appropriate configuration (Kang et al., 1990). Second, the number of possible configurations grows exponentially with the number of features (Benavides et al., 2005). Third, features are often subjective to the users and it is not clear which features selection fulfills their requirements better. Fourth, it may be very difficult to specify a valid configuration, since features of no interest can be needed to fulfill the feature model’s interdependencies (Pereira et al., 2016). Therefore, industrial-sized feature models with hundreds or thousands of features make the manual product configuration process impractical.

There are many different automatic approaches to tackle this problem in the literature (Ochoa et al., 2017). Most of these approaches consist in solving NP-complete problems (Benavides et al., 2005, 2006; White et al., 2009; Bagheri et al., 2010a, 2010b; Olaechea et al., 2012; Asadi et al., 2014; Zhang et al., 2014). Some of these works deal only with quantitative NFP (Benavides et al., 2005, 2006; White et al., 2009; Olaechea et al., 2012; Zhang et al., 2014), while others consider only qualitative NFP (Bagheri and Ensan, 2014). In fact, as far as we are concerned, Asadi et al. (2014) and Bagheri et al. (2010a, 2010b) are the only works in the literature that simultaneously tackle qualitative and quantitative NFP into account. However, those approaches only support NFP on the leaf features.

In this work, we propose a novel approach for the PCP that considers both qualitative (e.g., customer’s degree of preference) and quantitative (e.g., cost) NFPs. PCP is modeled as a combinatorial optimization problem, and heuristic and exact algorithms are proposed. It is the first work in the literature that considers feature properties in both leaf and nonleaf features. We assume that the leaf and nonleaf features in a feature model can have concrete implementations, and consequently both features need to be annotated with NFPs. Therefore, our proposal allows stakeholders to express their degree of preference for any feature in the feature model.

The remainder of this paper is organized as follows. Related works are discussed in Section 2. Section 3 describes the details of our approach, including a formal definition of PCP. This section also presents a preprocessing procedure, an exact algorithm, a greedy heuristic, and a biased random-key genetic algorithm (BRKGA) to solve this problem. Computational experiments are reported in Section 4. Finally, concluding remarks are drawn in Section 5.

2. Related work

Search-based software engineering (SBSE) is the field of research in which optimization techniques are used to address problems in software engineering (Harman and Jones, 2001). In this case, the term search refers to the search-based optimization algorithms that are used (Harman et al., 2012).
SBSE seeks to reformulate software engineering problems as search-based optimization problems. Therefore, a search problem is one in which optimal or near-optimal solutions are sought in a search space of candidate solutions, guided by a fitness function that distinguishes between better and worse solutions (Harman et al., 2012). SBSE has proved to be an applicable and successful field, with many studies across the software engineering life cycle, from requirements and project planning to maintenance and reengineering (Harman et al., 2012). There is a repository of publications on SBSE available on the Web.¹ This repository includes over 1000 relevant publications from 1976 to 2016, where a wide variety of different optimization and search techniques have been used. There is also increasing evidence of industrial interest in SBSE by many software-centric organizations including IBM (Yoo et al., 2011), Microsoft (Lakhotia et al., 2010), Motorola (Baker et al., 2006), and Nokia (Del Rosso, 2006).

SBSE is relevant for SPL because it offers a suite of adaptive, automated, and semiautomatic solutions in situations characterized by large and complex problems with multiple competing and conflicting objectives. A number of SBSE studies for solving problems in SPL have been documented in the literature (Féderle et al. 2015; Lopez-Herrejon et al., 2015; Mariani et al., 2015, 2016). Next, we present techniques for product configuration support in SPL. In addition, Table 1 points out some characteristics for each presented technique. Note that efficiency presented in this table corresponds to the feature model size (i.e., the number of features it provides support to) reported by the publication.

Batory (2005), Botterweck et al. (2007), Czarnecki and Wasowski (2007), Mannion (2002), Mendonça et al. (2009a), Sellier and Mannion (2007), and Thüm et al. (2014) applied propositional logic to automatically validate feature models, represent staged feature configurations, and provide support for manual feature selection. However, they provide a semiautomatic product configuration support, based only on FPs. Besides, these approaches depend on satisfiability (SAT) or binary decision diagrams exponential-time exact algorithms, which are not suitable for SPL applications due to their high computational times and tedious task.

Most of the works in the PCP literature that support NFP also rely on exponential-time exact algorithms. Benavides et al. (2005) have developed exponential-time constraint satisfaction (CS) algorithms to find optimal configurations based on a predefined objective function. Their approach showed a good performance on feature models with up to 25 features. In addition, Benavides et al. (2006) performed a comparative experiment between two off-the-shelf CSP solvers (JaCoP and Choco) and showed that they can solve instances whose feature models have up to 52 features. Asadi et al. (2014) applied hierarchical task network to PCP. They combine the analytical hierarchy process (AHP) and fuzzy cognitive maps to compute the NFP weights. Their approach returns an optimal product configuration for feature models with up to 200 features. Olaechea et al. (2012) and Antkiewicz et al. (2013) used an exact solver called Moolloy (Rayside et al., 2009). This approach uses type inheritance to modularize the quantitative NFP in feature model. The experiments show that this approach can handle feature models with up to a dozen features. However, as all other approaches above, it does not scale up efficiently to large industrial feature models with hundreds or thousands of features.

There are also works in the literature of PCP that rely on polynomial-time heuristic algorithms. White et al. (2009) proposed an approach based on filtered Cartesian flattening to approximate a

¹http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/ (last accessed December 2016).
Table 1
Comparative analysis of related work

<table>
<thead>
<tr>
<th>Related work</th>
<th>Extended feature model</th>
<th>Optimization algorithm</th>
<th>Efficiency (features)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benavides et al. (2005, 2006)</td>
<td>Leaf features with quantitative NFP</td>
<td>Exact</td>
<td>Up to 52</td>
</tr>
<tr>
<td>Antkiewicz et al. (2013) and Olaechea et al. (2012)</td>
<td>Leaf features with quantitative NFP</td>
<td>Exact</td>
<td>Up to 100</td>
</tr>
<tr>
<td>Asadi et al. (2014)</td>
<td>Leaf features with quantitative and qualitative NFP</td>
<td>Exact</td>
<td>Up to 200</td>
</tr>
<tr>
<td>White et al. (2009)</td>
<td>Leaf features with quantitative NFP</td>
<td>Approximation</td>
<td>Up to 140</td>
</tr>
<tr>
<td>Bagheri et al. (2010a, 2010b)</td>
<td>Leaf features with quantitative and qualitative NFP</td>
<td>Approximation</td>
<td>Up to 290</td>
</tr>
<tr>
<td>Zhang et al. (2014)</td>
<td>Leaf features with quantitative NFP</td>
<td>Approximation</td>
<td>No information</td>
</tr>
<tr>
<td>Henard et al. (2015)</td>
<td>Leaf features with quantitative and qualitative NFP</td>
<td>Approximation</td>
<td>Up to 6888</td>
</tr>
<tr>
<td>Bagheri and Ensan (2014)</td>
<td>Leaf features with qualitative NFP</td>
<td>Approximation</td>
<td>No information</td>
</tr>
<tr>
<td>Our approach</td>
<td>Leaf and nonleaf features with quantitative and qualitative NFP</td>
<td>Approximation</td>
<td>Up to 10,000</td>
</tr>
</tbody>
</table>

nearly optimal product configuration for large-scale feature models, considering only quantitative NFP. However, for performing this approach efficiently, multicore processors and parallel computing are required. Bagheri et al. (2010a, 2010b) applied a fuzzy logic approach to PCP. It uses a variant of propositional logic along with fuzzy logic to represent only qualitative NFP. Computational experiments, for two feature models with up to 290 features, showed that the proposed approach is efficient in both cases. Henard et al. (2015) introduced a search-based SPL feature selection algorithm, called SATIBEA, to address PCP in a large and highly constrained search space. However, computational experiments for feature models with up to 6888 features took approximately half an hour to be executed.

On the manual product configuration scenario, Bagheri and Ensan (2014) proposed dynamic decision models to guide the stakeholders through the configuration process. The authors’ approach predicts the utility of features for the stakeholders and provides a ranking of recommended features that are close to their preferences. In a similar scenario, Zhang et al. (2014) proposed an approach based on AHP. They employed AHP to compute the relative importance of features from qualitative NFP. However, it is not known how good the recommended solutions are.
In order to overcome the challenges faced by existing approaches, we have previously developed a decision support tool for the PCP (called SPLConfig\(^2\)) that adds support to extended feature model notation with quantitative and qualitative NFP, and automates the feature model configuration process (Machado et al., 2014). SPLConfig implements the approach for the PCP proposed in this work as an Eclipse plug-in, in order to quickly derive a product configuration that meets the business and customer requirements.

3. Search-based algorithms for product configuration

In this section, we model PCP as a combinatorial optimization problem and propose search-based algorithms for this problem. It aims at maximizing the customer satisfaction, subject to business and customer requirements. The qualitative NFP degree of preference is described using an ordinal scale consisting of a set of five predefined qualitative values: none, low, medium, high, and very high. Our approach uses this NFP only to rank the features and the corresponding solutions, so any objective function that maps this NFP to the customer satisfaction can be used. In this paper, these values were mapped to the real numbers 0, 1, 2, 3, and 4, respectively, and we maximize the sum of the customer’s degree of preference for each feature selected for the final product. Alternatively, one could use the AHP of Zhang et al. (2014) to compute the relative rank among the features and use this value to rank the final products. In addition, as in the works of Asadi et al. (2014), Bagheri et al. (2010a, 2010b), Benavides et al. (2005), White et al. (2009), and Zhang et al. (2014), business requirements can be defined by the cost of each feature, feature relations, and cross-tree constraints. Finally, in addition to the customer’s requirements, the sum of the cost of each feature selected must not exceed a given budget. This problem is clearly NP-hard as it contains a SAT (Cormen et al., 2009) as well as a knapsack (Gallo and Simeone, 1989) subproblem, both well-known NP-hard problems.

The following sections describe the new approach to the PCP proposed in this paper. The optimization problem is formally defined in Section 3.1. Next, a preprocessing algorithm to reduce the size of feature models is proposed in Section 3.2. Then, an exact backtracking algorithm is proposed in Section 3.3. As the worst-case complexity of the latter grows exponentially with the number of features, a greedy heuristic and a BRKGA are proposed in Sections 3.4 and 3.5, respectively.

3.1. Problem definition

Let the binary variables \( x_i \in \{0, 1\} \), such that \( x_i = 1 \), if feature \( i \) is selected for the final product, and \( x_i = 0 \) otherwise. Let also \( T = (V, A) \) be a feature tree, where \( V \) is a set of nodes, which represent features, and \( A \) is a set of arcs, such that \( (i, j) \in A \) denotes that the feature \( j \) can only be selected if feature \( i \) is also selected. Besides, let the following sets describe the FPs.

- \( M \subseteq V \) is the set of mandatory features, such that the root node \( r \in M \).
- \( O \subseteq V \) is the set of optional features, such that \( O \cup M = V \) and \( O \cap M = \emptyset \).
- \( XOR_i \subseteq 2^V \) is the set of all exclusive alternatives rooted in \( i \in V \), such that if \( \exists P \in XOR_i \) then \( (i, s) \in A \), for all \( s \in P \). Moreover, \( XOR_i = \emptyset \) if there is no exclusive alternative rooted on \( i \).

\(^2\)SPLConfig is available at http://homepages.dcc.ufmg.br/~figueiredo/spl/splconfig/
• \( OR_i \subset 2^V \) is the set of all nonexclusive alternatives rooted in \( i \in V \), such that if \( Q \in OR_i \) then \((i, s) \in A\), for all \( s \in Q \). Moreover, \( OR_i = \emptyset \) if there is no nonexclusive alternative rooted on \( i \).

Moreover, let the following constants describe the NFPs.

• \( c_i \in \mathbb{R} \) is the cost of the feature \( i \in V \).
• \( b_i \in \mathbb{R} \) is the degree of preference of the feature \( i \in V \), with \( b_i = 0 \) for all \( i \in M \), and \( b_j \leq b_i \), for all \((i, j) \in A\).
• \( D \in \mathbb{R} \) is the customer budget.

In addition, let \( E = (E_1(x), E_2(x), \ldots) \) be a set of cross-tree constraints, where \( E_j(x) \) is a logical expression over the variables \( x \).

Given the variables and constants described above, PCP is defined as follows:

maximize 

\[
F(x) = \sum_{i \in V} b_i x_i
\]

subject to

\[
\sum_{i \in V} c_i x_i \leq D
\]

\[
x_p = 1 \rightarrow x_i = 1 \quad (p, i) \in A : i \in M
\]

\[
x_p = 0 \rightarrow x_i = 0 \quad (p, i) \in A : i \in O
\]

\[
x_p = 1 \rightarrow \sum_{i \in P} x_i = 1 \quad \forall \ p \in V : XOR_p \neq \emptyset, \forall \ P \in XOR_p
\]

\[
x_p = 1 \rightarrow \sum_{i \in Q} x_i \geq 1 \quad \forall \ p \in V : OR_p \neq \emptyset, \forall \ Q \in OR_p
\]

\[
E_j(x) = true \quad \forall \ E_j(x) \in E.
\]

PCP consists in maximizing the customer satisfaction (1) subject to the budget constraint (2) required by the customer, as well as the composition constraints (3)–(6) and the cross-tree constraints (7), imposed by the feature model. Constraint (3) ensures that the mandatory feature \( i \in M \) is selected if its parent \( p \in V \) is selected. Constraint (4) ensures that the optional feature \( i \) is not selected if its parent \( p \in V \) is not selected. Constraint (5) ensures that, for each exclusive alternative set \( P \in XOR_p \), exactly one feature in \( P \) is selected if \( p \) is selected. Constraint (6) ensures that, for each nonexclusive alternative set \( Q \in OR_p \), at least one feature in \( Q \) is selected if \( p \) is selected. Finally, constraint (7) ensures that, all cross-tree constraints are satisfied. Previous research has shown that finding a feasible product configuration that conforms to the composite constraints and the cross-tree constraints, that is, (3)–(7), is NP-hard (Mendonça et al., 2009b; White et al., 2009). Besides, (1) and (2) consist of the classic knapsack problem (Gallo and Simeone, 1989), which is NP-hard. In addition, (1) and (7) can be reduced to the maximum satisfiability problem (MAX-SAT; Cormen et al., 2009), which is also NP-hard. Besides, PCP is also hard to approximate, since even to
find a feasible solution is NP-complete because a SAT problem must be solved in order to find any feasible solution. However, it is noteworthy that Mendonça et al. (2009b) showed that, for realistic feature models, the SAT instances induced by the cross-tree constraint (7) are easily solvable (which contributes to the efficiency of the results of the proposed algorithms).

3.2. Preprocessing procedure

The preprocessing procedure reduces the size of the feature model. It merges and removes nodes in the feature tree, reducing the number of features in $T$. The algorithm is divided into three parts. First, the mandatory features are merged with their parents. Second, it attempts to fix the value of variables $x_i$, for all $i \in O$, to 0 or to 1, by checking the cross-tree constraints and budget constraint. Then, for each set of exclusive and nonexclusive alternative set, it checks if after the two previous variables, there is only one feature left in this set. If so, this single feature is merged with its parent.

The pseudocode of the preprocessing procedure is shown in Algorithm 1. It takes as input the feature tree $T = (V, A)$, the sets $M$, $O$, $E$, $OR_v$, and $XOR_v$, for all $v \in V$, as well as the budget $D$, the cost $c_v$, and level of importance $b_v$ for each feature $v \in V$. The algorithm returns true if it successfully finishes or false if the instance turns out infeasible.

In the first part, the loop of lines 2–5 is repeated for each mandatory feature, except for the root of $T$. Let the feature $r \in V$ be the root of $T$ (line 1). For each mandatory feature $v \in M \setminus \{r\}$ (line 2), $v$ is merged with its parent $p \in V$ and $T$, $M$, $OR$, $XOR$, $c_p$, and $E$ are updated in line 4 as follows: (a) $v$ and $p$ are contracted in $T$; (b) $M = M \setminus \{v\}$; (c) $OR_p = OR_p \cup OR_v$; (d) $XOR_p = XOR_p \cup XOR_v$; (e) $c_p = c_p + c_v$; and (f) variable $x_v$ is replaced by variable $x_{p}$ in all expressions in $E$, if $p \neq r$, or by 1 otherwise. The value of $b_p$ does not need to be updated, because $b_v = 0$ as $v$ is a mandatory feature. After this update, the node $p$ represents two features. Variable $x_p = 1$ denotes that both features are selected, and $x_p = 0$ denotes that both features are not selected. For sake of simplicity, in the remainder text we do not detail the data structure that keeps track of which features are merged features or single features, and we assume that we have a function $features(v)$ that returns a set with all features from the original feature tree that were merged with $v \in V$. At the end of this part, all features in $V$ are optional, except the root feature.

In the second part (lines 6–18), the budget constraints (2) and cross-tree constraints (7) are checked for each feature $v \in V$, in order to (a) prove the instance is infeasible; (b) remove features that cannot be selected without violating (2) and (7); or (c) merge (with the respective parent) features that cannot be left unselected, if their parents are selected, without violating (2) and (7). We denote by $\hat{c}(v) = c_v + \hat{c}(p)$ a function that returns the sum of the cost of all features from $v \in O$ to the root $r$ of $T$, where $p$ is the parent of $v$ and $\hat{c}(r) = c_r$. Besides, we denote by $\hat{c}(v) = c_v + \sum_{x \in OR_v \cup XOR_v} (\min_{u \in x} \hat{c}(u))$ a function that returns the minimum cost for selecting a feature $v \in V$ without violating the composite constraints (3)–(6). In line 6, the procedure checks if $\hat{c}(r)$ fits the budget. If it does not, no feasible solution exists and the algorithm returns false in line 7. Following, the loop of lines 9–18 is repeated for each optional feature $v \in O$. Let $p \in V$ be the parent of $v$, and Impossible($E$, $v$, $\beta$) be a function that returns true if it is impossible to satisfy any expression $E_j(x) \in E$ by replacing the value of $x_v$ by $\beta \in \{0, 1\}$, and false otherwise. We note that this function does not solve a SAT problem, as it only checks the expressions in $E$ in which the
values of all variables are already fixed, except that of $x_v$. Three cases are checked in the loop of lines 9–18. In the first case (lines 11 and 12), if constraints (7) cannot be satisfied by fixing $x_v$ neither to 0 nor to 1, then the procedure returns \textit{false} in line 12, indicating that the original instance is infeasible. In the second case (lines 13 and 14), if making $x_v = 1$ violates the cross-tree constraints (7) or the budget constraint (2), then $v$ is removed from the feature model and $T$, $O$, $OR$, $XOR$, and $E$ are

updated in line 14 as follows: (a) the subtree rooted on $v$, denoted by $T[v] = (V[v], A[v])$, is removed from $T$; (b) $O = O \setminus V[v]$; (c) $Q = Q \setminus \{q\}$, for all $u \in V, Q \in OR_u \cup XOR_u, q \in V[v]$; and (d) for all $q \in V[v]$, variable $x_q$ is replaced by 0 in every expression in $E$. In the third case (lines 15 and 16), if making $x_v = 0$ violates constraints (7), then $v$ is merged with its parent $p \in V$, and $T$, $O$, $OR$, $XOR$, $E$, $c_p$, and $b_p$ are updated in line 16 as follows: (a) $v$ and $p$ are contracted in $T$; (b) $O = O \setminus \{v\}$; (c) $OR_p = OR_p \cup OR_v$; (d) $XOR_p = XOR_p \cup XOR_v$; (e) $c_p = c_p + c_v$; (f) $b_p = b_p + b_v$; and (g) variable $x_v$ is replaced by variable $x_p$ in all expressions in $E$, if $p \neq r$, or by 1 otherwise. Besides, (h) if there is a nonexclusive alternative set $Q \in OR_p$ such that $v \in Q$, then $OR_p = OR_p \setminus Q$ because this set is satisfied by $v$, and (i) if there is an exclusive alternative set $Q \in XOR_p$ such that $v \in Q$, then $XOR_p = XOR_p \setminus Q$ and each node $q \in Q$ is removed from the feature model the same way as in line 14. At the end of this part, all optional features can be selected individually without violating any constraint.

In the third part, the loop of lines 19–24 is repeated for all $Q \in OR_p \cup XOR_p$ with $p \in V$. If $|Q| = 1$ (line 20), then the single feature $q \in Q$ is merged with its parent $p$ and $T$, $O$, $OR$, $XOR$, $E$, $c_p$, and $b_p$ are updated in line 24 as follows: (a) the subtree rooted on $v$, denoted by $T[v] = (V[v], A[v])$, is removed from $T$; (b) $O = O \setminus V[v]$; (c) $Q = Q \setminus \{q\}$, for all $u \in V, Q \in OR_u \cup XOR_u, q \in V[v]$; and (d) for all $q \in V[v]$, variable $x_q$ is replaced by 0 in every expression in $E$. In the second case (lines 13 and 14), if making $x_v = 1$ violates the cross-tree constraints (7) or the budget constraint (2), then $v$ is removed from the feature model and $T$, $O$, $OR$, $XOR$, and $E$ are

Algorithm 1 \textit{Preprocess} ($T = (V, A), M, O, OR, XOR, E, D, c, b) \mapsto \{true, false\}$

1. let $r \leftarrow \text{root}(T)$
2. for each $v \in M \setminus \{r\}$ do
3.  let $p \leftarrow \text{parent}(v)$
4.  merge $v$ into $p$ and update $T$, $M$, $OR$, $XOR$, $E$, $c_p$
5. endfor
6. if $\tilde{c}(r) > D$ then
7.  return false
8. endif
9. for each $v \in O$ do
10.  let $p \leftarrow \text{parent}(v)$
11.  if Impossible($E$, $v$, 1) and Impossible($E$, $v$, 0) then
12.    return false
13. else if Impossible($E$, $v$, 1) or $\tilde{c}(v) + \tilde{c}(p) > D$ then
14.    remove $v$ from $T$ and update $T$, $O$, $OR$, $XOR$, $E$
15. else if Impossible($E$, $v$, 0) then
16.    merge $v$ into $p$ and update $T$, $O$, $OR$, $XOR$, $E$, $c_p$, $b_p$
17. endif
18. endfor
19. for each $X \in OR_p \cup XOR_p$ with $p \in V$ do
20.  if $|X| = 1$ then
21.    let $u$ be the only node in $X$
22.    merge $u$ into $p$ and update $T$, $O$, $OR$, $XOR$, $E$, $c_p$, $b_p$
23. endfor
24. endfor
25. return true

© 2017 The Authors.
International Transactions in Operational Research © 2017 International Federation of Operational Research Societies
c_v, b_v, and E are updated in line 22 (i.e., in the same way as they were in line 16). Finally, in line 25, the algorithm returns true meaning that the instance was successfully preprocessed.

3.3. Backtracking algorithm

Backtracking is an enumeration algorithm, where infeasible and suboptimal solutions are eliminated without being explicitly examined (Cormen et al., 2009). The fundamental principles of backtracking algorithms are the order of evaluating the variables and assigning the values to each variable. In the backtracking procedure proposed in this section, the features in V are sorted by the order they are visited by a breath first search in T started from its root feature. Thus, whenever a variable is evaluated, the value of the variable correspondent to its parent is already fixed. Variables are first assigned to the value 1, and then to the value 0. Thus, pruning by infeasibility regarding the budget constraint (2) occurs early in the decision tree.

The pseudocode of the backtracking procedure is shown in Algorithm 2. Let X ∈ {0, 1} | V | be an array representing a solution for PCP, where X[v] keeps the value of variable x_v. Let also F(X) be the value of X in the objective function (1), with F(X) < 0 if X is infeasible. The algorithm takes as input the feature tree T = (V, A), the sets M, O, E, OR_v, and XOR_v, for all v ∈ V, as well as the budget D, the cost c_v and the level of importance b_v of each feature v ∈ V. Moreover, it receives the permutation π that defines the order in which the variables correspondent to the features in V are assigned values, the index i that defines the next feature in π to be evaluated, the current (possibly infeasible) solution X, and the best-known solution X*. The backtracking procedure is first executed with i = 2, X[1] = 1 and X*[1] = 1, because π_1 is always the root feature. We note that this procedure can be executed after the preprocessing procedure of Algorithm 1.

In the base case (lines 1–6), all variables in X are assigned values. In this case, if the

```
Algorithm 2  Backtracking(T = (V, A), M, O, OR, XOR, E, D, c, b, π, i, X, X*) → X*
1. if i = |V| then
2. if X is feasible and F(X) > F(X*) then
3. return X
4. else
5. return X*
6. endif
7. else
8. let p ← parent(π_i)
9. if c_{π_i} ≤ D and X[p] = 1 and \exists Q ∈ XOR_p : v ∈ Q ∧ X[v] = 1 and \not NotInfeasible(E, π, 1)
10. X[π_i] ← 1
11. X* ← Backtracking(T, M, O, OR, XOR, E, D − c_{π_i}, c, b, π, i + 1, X, X*)
12. endif
13. if (π_i ∈ O or X[p] = 0) and \not NotInfeasible(E, π, 0) then
14. X[π_i] ← 0
15. X* ← Backtracking(T, M, O, OR, XOR, E, D, c, b, π, i + 1, X, X*)
16. endif
17. endif
```
current solution \( X \) is feasible and better than \( X^* \) (line 2), the procedure returns \( X \) in line 3, otherwise it returns \( X^* \) in line 5. The recursive case consists of lines 8–16. The parent \( p \) of the next feature \( \pi_i \) to be evaluated is identified in line 8. In line 9, the algorithm checks the case when \( \pi_i \in V \) is selected. That is, (a) if the cost of \( \pi_i \) fits the budget; (b) if its parent \( p \in V \) has been selected; (c) if there is no set \( Q \in XOR_p \) such that \( \pi_i \in Q \) and there is another feature \( v \in Q \) already selected in this exclusive alternative set; and (d) if the cross-tree constraints in \( E \) are not made infeasible by fixing the value of \( \pi_i \) to 1. If so, feature \( \pi_i \) is selected in line 10, and the backtracking procedure is recursively called in line 11. In line 12, the algorithm checks the case when \( \pi_i \in O \) is not selected. That is, (a) if \( \pi_i \) is an optional feature, or (b) if its parent was not selected, and (c) if the cross-tree constraints in \( E \) are not made infeasible by fixing the value of \( \pi_i \) to 0. If so, feature \( \pi_i \) is not selected in line 14, and the Backtracking procedure is recursively called in line 15.

### 3.4. Greedy heuristic algorithm

The greedy heuristic aims at finding a feasible solution that maximizes the objective function (1). Any subset \( S \in V \) of features that satisfies the constraints (2)–(7) is called a feasible solution. The heuristic proposed in this section for PCP is iterative. At each iteration, a locally optimal decision is made and one feature is selected, hoping that this choice leads to a near-optimal solution. Before each step, the feature tree is preprocessed, and one subfeature \( v \in O \) of the root feature \( r \) is selected and merged with the root. We note that after the preprocessing \( V = O \cup \{r\} \), and all features remaining in \( O \) can be selected without violating any constraint. The procedure stops when \( O = \emptyset \). At this point, all features selected have been merged with the root. We point out that it is not guaranteed that this heuristic returns a feasible solution. However, as mentioned in Section 3.1, it is NP-complete even to check if a feasible solution exists. Nevertheless, it can be seen from the experiments in Section 4 that the greedy heuristic found feasible solutions for all instances tested.

The pseudocode of the greedy heuristic is shown in Algorithm 3. It takes as input the feature tree \( T = (V, A) \), the sets \( M, O, E, OR_v, \) and \( XOR_v \), for all \( v \in V \), as well as the budget \( D \), the cost \( c_v \), and level of importance \( b_v \) of each feature \( v \in V \). It returns the subset \( S^* \) of features with a hopefully feasible product configuration. The root \( r \) of the feature tree is identified in line 1. \( S^* \) is initialized in line 2, indicating that no feasible solution is known so far. We note that making \( S^* = \text{features}(r) \) does not necessarily lead to a feasible solution, because it might not satisfy the composition constraints (5)–(6) or the cross-tree constraints (7). The loop in lines 3–12 is repeated while there are optional features in the preprocessed feature tree. First, the best-known solution is updated in line 4. Let \( \hat{O} \) be the set of all exclusive and nonexclusive alternative sets rooted in \( r \) (line 5). Next, if \( \hat{O} \) is not empty (line 6), then the alternative feature \( v \in \hat{O} \) with the best cost benefit ratio is selected in line 7. Features in \( \hat{O} \) are prioritized in order to select first features that satisfy the composition constraints (5) and (6). Otherwise, the subfeature \( v \in O \) of the root \( r \) with the best cost benefit ratio is selected in line 9. Feature \( v \) is merged with the root in order to guarantee it will be selected, and \( T, O, OR, XOR, c_r, b_r \), and \( E \) are updated in line 11 the same way as they were in line 16 of Algorithm 1. Finally, the best-known solution is updated in line 13 and returned in line 14.
Algorithm 3 GreedyHeuristic($T = (V, A)$, $M$, $O$, $OR$, $XOR$, $E$, $D$, $c$, $b$) $\mapsto S^*$

1. let $r \leftarrow \text{root}(T)$
2. $S^* \leftarrow \emptyset$
3. while Preprocess($T$, $M$, $O$, $OR$, $XOR$, $E$, $D$, $c$, $b$) and $O \neq \emptyset$ do
4. if features($r$) is feasible then $S^* \leftarrow$ features($r$)
5. let $\bar{O} \leftarrow \bigcup_{Q \in OR \cup XOR} Q$
6. if $\bar{O} \neq \emptyset$ then
7. Let $v \leftarrow \arg\max_{u \in \bar{O}} b_u / c_u$
8. else
9. Let $v \leftarrow \arg\max_{u \in O \setminus OR \cup XOR} b_u / c_u$
10. endif
11. merge $v$ into $r$ and update $T$, $O$, $OR$, $XOR$, $c_r$, $b_r$, $E$
12. endwhile
13. if features($r$) is feasible then $S^* \leftarrow$ features($r$)
14. return $S^*$

3.5. Biased random-key genetic algorithm

The random-key genetic algorithm (RKGA) were first introduced by Bean (1994) for combinatorial optimization problems for which solutions can be represented as a permutation vector. Solutions are represented as vectors of randomly generated real numbers called keys. A deterministic algorithm, called decoder, takes as input a vector of keys and associates with it a feasible solution of the combinatorial optimization problem, for which an objective value or fitness can be computed. Two parents are selected at random from the entire population to implement the crossover operation in the implementation of an RKGA. Parents are allowed to be selected for mating more than once in a given generation.

A biased RKGA (BRKGA) differs from an RKGA in the way parents are selected for crossover. In a BRKGA, each element is generated combining one element selected at random from the elite solutions in the current population, while the other is a nonelite solution. We say the selection is biased since one parent is always an elite individual and because this elite solution has a higher probability of passing its genes to the offsprings, that is, to the new generation. The choice of the BRKGA metaheuristic to develop a heuristic for PCP is motivated by its successful applications to many combinatorial optimization problems (for a survey, see Gonçalves and Resende, 2011).

In the BRKGA for PCP, each solution is represented by an $|O|$-vector, where each component is a real number in the range $[0, 1]$ associated with an optional feature. The decoding consists of two steps. First, the features are sorted in nondecreasing order of their key values. Then, the greedy heuristic of Section 3.4 considers the selection of each feature according to this order. The fitness of the chromosome is equal to its benefit. However, if the solution is not feasible, its fitness is set to 0.

The parametrized uniform crossover scheme proposed by Spears and DeJong (1991) is used to combine two parent solutions and produce an offspring solution. In this scheme, the offspring inherits each of its keys from the respective key from one of its two parents with probability 0.7.

This genetic algorithm does not make use of the standard mutation operator, where parts of the chromosomes are changed with small probability. Instead, the concept of mutants is used. In
each generation, a fixed number of mutant solutions are introduced in the population. They are generated in the same way as the initial population. As with mutation, mutants serve the role of helping the procedure escape from local optimal.

At each new generation, the population is partitioned into two sets: TOP and REST. Consequently, the size of the population is $|TOP| + |REST|$. The best solutions are kept in TOP while the others are placed in REST. As illustrated in Fig. 2, the chromosomes in TOP are copied, without change, to the population of the next generation. The new mutants are placed in a set called BOT. The remaining elements of the new population are obtained by crossover with one parent randomly chosen from TOP and the other from REST. Since a parent solution can be chosen for crossover more than once in a given generation, elite solutions have a higher probability of passing their random keys to the next generation. This algorithm stops after 10 generations without improving the best-known solution. The population size $p$ was set to 100, and the values of $|TOP|$, $|REST|$, and $|BOT|$ were set to $0.25 \times p$, $0.75 \times p$, and $5 \times p$ as suggest by Noronha et al. (2011) and Brandão et al. (2016).

4. Computational results

This section reports the results of empirical studies carried out to evaluate the algorithms proposed in Section 3. The preprocessing procedure, backtracking algorithm, greedy heuristic, and BRKGA were implemented in C++ and compiled with GNU GCC, version 4.4.3. The experiments were carried out on a single core of an Intel Core i7-4790k machine with 4.00 GHz of clock and 16 GB of RAM memory.

© 2017 The Authors.
International Transactions in Operational Research © 2017 International Federation of Operational Research Societies
Table 2
Characteristics of the instances with realistic feature models

| Feature model (FM) | |V| |E| |M| |O| #XOR | #OR | Height |
|--------------------|-----------------|-------|-------|-------|-------|-------|-------|-------|
| FM1—Mobile Media (Figueiredo et al., 2008) | |17| |1| |8| |9| |1| |1| |4| |
| FM2—Email System (Thüm and Benduhn, 2014) | |23| |0| |3| |20| |1| |0| |3| |
| FM3—Smart Home (Álvarez et al., 2010) | |28| |3| |3| |25| |1| |1| |4| |
| FM4—Devolution (Thüm and Benduhn, 2014) | |32| |0| |11| |21| |1| |4| |6| |
| FM5—Gasparc (Aranega et al., 2012) | |38| |0| |23| |15| |4| |1| |6| |
| FM6—Web Portal (Mendonça et al., 2008) | |43| |6| |9| |34| |3| |3| |5| |
| FM7—FraSCAti (Seinturier et al., 2012) | |63| |28| |19| |44| |2| |0| |5| |
| FM8—Model Transform (Czarnecki and Helsen, 2003) | |89| |0| |19| |70| |11| |14| |8| |
| FM9—Battle of Tanks (Thüm and Benduhn, 2014) | |144| |0| |8| |136| |9| |1| |4| |
| FM10—e-Shop (Lau, 2006) | |213| |32| |74| |139| |0| |43| |8| |

Two sets of benchmark instances were used in the experiments. The first set is based on 10 feature models (named from FM1 to FM10) with up to 213 features. The feature models were obtained from other works in the literature that used realistic feature models in their experiments (Czarnecki and Helsen, 2003; Lau, 2006; Figueiredo et al., 2008; Mendonça et al., 2008; Álvarez et al., 2010; Aranega et al., 2012; Seinturier et al., 2012; Thüm and Benduhn, 2014). The second set is based on 12 randomly generated feature models (named from FM11 to FM22) with up to 10,000 features. All feature models contain a set of cross-tree constraints whose size grows with the number of features. These feature models were generated using the Thüm’s method.3

A group of 10 instances was generated for each feature model, varying the value of NFPs $D$, $c_v$ and $b_v$, for all $v \in V$. The latter were randomly generated in the range $[0, 1000 \times |V|]$, $[0, 1000]$ and $[0, 5]$, respectively. Therefore, the first set is composed of 100 instances, while the second set is composed of 120 instances.

The characteristics of the feature models used in our experiments are presented in Table 2 (for the realistic instances) and Table 3 (for the randomly generated instances). The first column identifies the feature model. The number of features ($|V|$) and cross-tree constraints ($|E|$) are displayed in the second and third columns, respectively. The fourth and fifth columns show the number of mandatory ($|M|$) and optional ($|O|$) features, respectively. The number of exclusive (#XOR) and nonexclusive (#OR) alternative features is given in the sixth and seventh columns, respectively, while the height of the feature tree is depicted in the last column.

In the first experiment, we evaluate how much the preprocessing algorithm (i.e., Algorithm 1) reduces the size of the instances. The results are reported in Table 4. The first column gives the name of the feature model that identifies a group of 10 instances. Columns 2–8 show the average percentage of reduction for each instance characteristic reported in Tables 2 and 3. Moreover, the last column gives the average time in milliseconds spent by the preprocessing algorithm. It can be observed from Table 4 that after preprocessing the number of features ($|V|$) is reduced by up to 63% on average for the instances based on FM8. This reduction was at least 18% for the instances based on FM3. We note that the percentage reduction in mandatory features is not 100% because the root is the only mandatory feature that is not removed. The number of cross-tree constraints ($|E|$)
Table 3
Characteristics of the instances with randomly generated feature models

| FM   | $|V|$ | $|E|$ | $|M|$ | $|O|$ | #XOR | #OR | Height |
|------|-----|-----|-----|-----|------|-----|-------|
| FM11 | 200 | 20  | 60  | 140 | 7    | 9   | 9     |
| FM12 | 200 | 20  | 49  | 151 | 10   | 13  | 9     |
| FM13 | 500 | 50  | 147 | 353 | 18   | 29  | 10    |
| FM14 | 500 | 50  | 152 | 348 | 21   | 22  | 11    |
| FM15 | 1000| 100 | 306 | 694 | 38   | 52  | 14    |
| FM16 | 1000| 100 | 319 | 681 | 46   | 37  | 13    |
| FM17 | 2000| 200 | 604 | 1396| 81   | 93  | 15    |
| FM18 | 2000| 200 | 586 | 1414| 90   | 101 | 12    |
| FM19 | 5000| 500 | 1466| 3534| 213  | 231 | 18    |
| FM20 | 5000| 500 | 1486| 3514| 252  | 216 | 19    |
| FM21 | 10,000| 1000| 2918| 7082| 446  | 464 | 19    |
| FM22 | 10,000| 1000| 3038| 6962| 463  | 440 | 20    |

was reduced in 13 of the 17 instance groups with cross-tree constraints. The number of optional features in instances based on FM8 was reduced up to 54%. Furthermore, in seven (out of the 22) instance groups, the preprocessing algorithm was able to reduce the average height of the feature tree. Finally, the average time to preprocess the instances was never larger than 8.160 milliseconds.

Table 4
Characteristics of the preprocessed feature model instances

| FM   | $|V|$ (%) | $|E|$ (%) | $|M|$ (%) | $|O|$ (%) | #XOR (%) | #OR (%) | Height (%) | Time (milliseconds) |
|------|---------|---------|---------|---------|---------|---------|-----------|---------------------|
| FM1  | 47      | 0       | 88      | 11      | 0       | 0       | 50        | 0.04                |
| FM2  | 30      | –       | 67      | 25      | 0       | –       | 0         | 0.03                |
| FM3  | 18      | 33      | 67      | 12      | 0       | 0       | 0         | 0.16                |
| FM4  | 34      | –       | 91      | 5       | 0       | 25      | 33        | 0.23                |
| FM5  | 63      | –       | 96      | 13      | 0       | 0       | 50        | 0.09                |
| FM6  | 19      | 0       | 89      | 0       | 0       | 0       | 0         | 0.24                |
| FM7  | 29      | 0       | 95      | 0       | 0       | –       | 20        | 0.38                |
| FM8  | 63      | –       | 95      | 54      | 73      | 50      | 63        | 0.11                |
| FM9  | 34      | –       | 88      | 31      | 22      | 0       | 25        | 0.19                |
| FM10 | 54      | 13      | 99      | 31      | –       | 16      | 25        | 0.69                |
| FM11 | 37      | 10      | 98      | 11      | 22      | 30      | 30        | 0.47                |
| FM12 | 27      | 25      | 98      | 4       | 23      | 32      | 13        | 0.82                |
| FM13 | 35      | 12      | 99      | 9       | 0       | 3       | 0         | 0.92                |
| FM14 | 42      | 4       | 99      | 5       | 31      | 21      | 0         | 1.82                |
| FM15 | 32      | 29      | 99      | 2       | 42      | 25      | 14        | 2.01                |
| FM16 | 33      | 0       | 99      | 2       | 43      | 41      | 15        | 1.74                |
| FM17 | 31      | 34      | 99      | 2       | 43      | 32      | 13        | 3.58                |
| FM18 | 31      | 31      | 99      | 2       | 24      | 26      | 25        | 2.11                |
| FM19 | 30      | 22      | 99      | 1       | 26      | 27      | 17        | 5.26                |
| FM20 | 31      | 8       | 99      | 1       | 32      | 34      | 21        | 4.89                |
| FM21 | 30      | 16      | 99      | 1       | 28      | 31      | 21        | 7.53                |
| FM22 | 31      | 17      | 99      | 1       | 31      | 34      | 25        | 8.16                |
on the largest instances. In next experiment, one can observe that preprocessing the feature models has a significant impact in the performance of the backtracking algorithm.

In the second experiment, we evaluate the performance of the backtracking algorithm (i.e., Algorithm 2) with and without the preprocessing procedure. In both cases, the backtracking algorithm was made to stop after one hour of processing time. The results are presented in Table 5. The first column gives the name of the feature model that identifies a group of 10 instances. Columns 2–4 show, respectively, the average benefit, standard deviation, and average time (in seconds) to find the optimal solution by the backtracking algorithm running from the original instances. When the algorithm was not able to find the optimal solution within one hour of running time for at least one of the 10 instances in the group, the second and third columns report the average benefit and standard deviation (SD) of the best solution found by the algorithm, and the fourth column is filled with a dash. The same data are given for the backtracking algorithm running from the preprocessed instances in columns 5–7, respectively. The last column gives the average relative deviation \((\frac{b - a}{a})\) between the average benefit \(a\) of the solutions obtained by backtracking running from the original instances and the average benefit \(b\) of the solutions obtained by the same algorithm running from the preprocessed instances. It can be observed that the execution time of the backtracking algorithm for the preprocessed instances is always smaller than or equal to the execution time of the same algorithm for the original instances. For 13 (out of the 16) instance groups where the

<table>
<thead>
<tr>
<th>FM</th>
<th>Original Benefit</th>
<th>SD</th>
<th>Time (seconds)</th>
<th>Preprocessed Benefit</th>
<th>SD</th>
<th>Time (seconds)</th>
<th>Dev (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM1</td>
<td>35.50</td>
<td>7.60</td>
<td>0.004</td>
<td>35.50</td>
<td>7.60</td>
<td>0.002</td>
<td>0.00</td>
</tr>
<tr>
<td>FM2</td>
<td>65.80</td>
<td>21.10</td>
<td>0.001</td>
<td>65.80</td>
<td>21.10</td>
<td>0.001</td>
<td>0.00</td>
</tr>
<tr>
<td>FM3</td>
<td>81.00</td>
<td>11.10</td>
<td>4.487</td>
<td>81.00</td>
<td>11.10</td>
<td>1.249</td>
<td>0.00</td>
</tr>
<tr>
<td>FM4</td>
<td>77.30</td>
<td>33.16</td>
<td>0.025</td>
<td>77.30</td>
<td>33.16</td>
<td>0.010</td>
<td>0.00</td>
</tr>
<tr>
<td>FM5</td>
<td>102.00</td>
<td>25.72</td>
<td>0.001</td>
<td>102.00</td>
<td>25.72</td>
<td>0.001</td>
<td>0.00</td>
</tr>
<tr>
<td>FM6</td>
<td>142.50</td>
<td>17.08</td>
<td>0.317</td>
<td>142.50</td>
<td>17.08</td>
<td>0.215</td>
<td>0.00</td>
</tr>
<tr>
<td>FM7</td>
<td>134.50</td>
<td>65.79</td>
<td>–</td>
<td>163.30</td>
<td>108.68</td>
<td>21.42</td>
<td>21.42</td>
</tr>
<tr>
<td>FM8</td>
<td>221.00</td>
<td>38.10</td>
<td>–</td>
<td>241.10</td>
<td>57.60</td>
<td>9.09</td>
<td>9.09</td>
</tr>
<tr>
<td>FM9</td>
<td>313.80</td>
<td>65.35</td>
<td>–</td>
<td>313.80</td>
<td>65.35</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>FM10</td>
<td>381.70</td>
<td>114.89</td>
<td>–</td>
<td>400.50</td>
<td>102.04</td>
<td>4.91</td>
<td>4.91</td>
</tr>
<tr>
<td>FM11</td>
<td>423.90</td>
<td>174.76</td>
<td>–</td>
<td>424.10</td>
<td>178.89</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>FM12</td>
<td>340.10</td>
<td>113.82</td>
<td>–</td>
<td>365.70</td>
<td>122.09</td>
<td>7.53</td>
<td>7.53</td>
</tr>
<tr>
<td>FM13</td>
<td>1463.00</td>
<td>143.23</td>
<td>–</td>
<td>2012.10</td>
<td>335.36</td>
<td>37.53</td>
<td>37.53</td>
</tr>
<tr>
<td>FM14</td>
<td>1562.60</td>
<td>10.05</td>
<td>–</td>
<td>2081.90</td>
<td>110.67</td>
<td>33.23</td>
<td>33.23</td>
</tr>
<tr>
<td>FM15</td>
<td>977.10</td>
<td>27.80</td>
<td>–</td>
<td>1252.30</td>
<td>133.36</td>
<td>28.16</td>
<td>28.16</td>
</tr>
<tr>
<td>FM16</td>
<td>1248.90</td>
<td>18.74</td>
<td>–</td>
<td>1715.10</td>
<td>424.00</td>
<td>37.33</td>
<td>37.33</td>
</tr>
<tr>
<td>FM17</td>
<td>1847.10</td>
<td>22.38</td>
<td>–</td>
<td>2053.40</td>
<td>85.68</td>
<td>11.17</td>
<td>11.17</td>
</tr>
<tr>
<td>FM18</td>
<td>1844.10</td>
<td>1008.89</td>
<td>–</td>
<td>3175.40</td>
<td>571.11</td>
<td>11.65</td>
<td>11.65</td>
</tr>
<tr>
<td>FM19</td>
<td>7378.60</td>
<td>642.83</td>
<td>–</td>
<td>7238.20</td>
<td>522.60</td>
<td>–</td>
<td>–1.90</td>
</tr>
<tr>
<td>FM20</td>
<td>2888.30</td>
<td>39.62</td>
<td>–</td>
<td>4516.10</td>
<td>1020.28</td>
<td>56.36</td>
<td>56.36</td>
</tr>
<tr>
<td>FM21</td>
<td>10967.50</td>
<td>4194.68</td>
<td>–</td>
<td>10450.20</td>
<td>1630.84</td>
<td>–</td>
<td>–4.72</td>
</tr>
<tr>
<td>FM22</td>
<td>4425.30</td>
<td>520.79</td>
<td>–</td>
<td>6799.50</td>
<td>1745.36</td>
<td>53.65</td>
<td>53.65</td>
</tr>
</tbody>
</table>
backtracking algorithm could not find the optimal solution before one hour, the solution found by
the backtracking algorithm on the preprocessed instance was better than that found by the same
algorithm on the original instance.

In the next experiment, we evaluate the performance of the greedy and BRKGA heuristics on the
preprocessed instances. The results are displayed in Table 6. The first column identifies the instance
group. Columns 2–4 show, respectively, the average benefit, standard deviation, and average relative
deviation \((b - a)/a\) between the average benefit \(b\) of the solutions obtained by greedy heuristic and
the average benefit \(a\) of the best solutions obtained by the backtracking algorithm within one hour
of running time (see column 5 of Table 5). We note that positive values for the relative deviation
means that our heuristics outperform the backtracking algorithm. The fifth column shows the
average running time (in seconds) of the greedy heuristic.

The same data are provided for the BRKGA in columns 6–9, respectively. It can be observed that
the greedy heuristic finds solutions close to those obtained by the backtracking algorithm. For the
instances where optimal solutions are known (see FM1–FM6), the maximum relative optimality
gap of the greedy heuristic was only 2.27% (see instance FM3). Besides, BRKGA found optimal
solutions for all instances where those are known, and outperformed the backtracking algorithm
by up 67.89% on instance FM17. In the only two instance groups where BRKGA obtained worst
average results, the difference was at most 0.73%.

Table 6
Performance of the greedy and the BRKGA heuristic

<table>
<thead>
<tr>
<th></th>
<th>Greedy</th>
<th></th>
<th>BRKGA</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FM</td>
<td>Benefit</td>
<td>SD</td>
<td>Dev (%)</td>
<td>Time (seconds)</td>
</tr>
<tr>
<td>FM1</td>
<td>35.50</td>
<td>7.60</td>
<td>0.00</td>
<td>0.007</td>
</tr>
<tr>
<td>FM2</td>
<td>65.80</td>
<td>21.10</td>
<td>0.00</td>
<td>0.004</td>
</tr>
<tr>
<td>FM3</td>
<td>79.20</td>
<td>18.87</td>
<td>-2.27</td>
<td>0.028</td>
</tr>
<tr>
<td>FM4</td>
<td>77.30</td>
<td>33.16</td>
<td>0.00</td>
<td>0.031</td>
</tr>
<tr>
<td>FM5</td>
<td>142.50</td>
<td>17.08</td>
<td>0.00</td>
<td>0.039</td>
</tr>
<tr>
<td>FM6</td>
<td>175.30</td>
<td>112.56</td>
<td>7.35</td>
<td>0.035</td>
</tr>
<tr>
<td>FM8</td>
<td>241.10</td>
<td>57.60</td>
<td>0.00</td>
<td>0.019</td>
</tr>
<tr>
<td>FM9</td>
<td>356.35</td>
<td>72.19</td>
<td>13.55</td>
<td>0.033</td>
</tr>
<tr>
<td>FM10</td>
<td>413.05</td>
<td>112.76</td>
<td>3.13</td>
<td>0.022</td>
</tr>
<tr>
<td>FM11</td>
<td>420.00</td>
<td>182.19</td>
<td>-0.97</td>
<td>0.027</td>
</tr>
<tr>
<td>FM12</td>
<td>350.20</td>
<td>131.74</td>
<td>-4.24</td>
<td>0.033</td>
</tr>
<tr>
<td>FM13</td>
<td>1953.30</td>
<td>409.56</td>
<td>-2.92</td>
<td>0.034</td>
</tr>
<tr>
<td>FM14</td>
<td>2067.10</td>
<td>275.12</td>
<td>-0.71</td>
<td>0.042</td>
</tr>
<tr>
<td>FM15</td>
<td>1216.30</td>
<td>148.57</td>
<td>-2.87</td>
<td>0.136</td>
</tr>
<tr>
<td>FM16</td>
<td>1680.60</td>
<td>401.05</td>
<td>-2.01</td>
<td>0.165</td>
</tr>
<tr>
<td>FM17</td>
<td>2053.40</td>
<td>85.68</td>
<td>0.00</td>
<td>0.366</td>
</tr>
<tr>
<td>FM18</td>
<td>3170.50</td>
<td>578.45</td>
<td>-0.15</td>
<td>0.593</td>
</tr>
<tr>
<td>FM19</td>
<td>7220.20</td>
<td>524.61</td>
<td>-0.25</td>
<td>4.247</td>
</tr>
<tr>
<td>FM20</td>
<td>4466.90</td>
<td>962.02</td>
<td>-1.09</td>
<td>2.830</td>
</tr>
<tr>
<td>FM21</td>
<td>10417.40</td>
<td>1626.76</td>
<td>-0.31</td>
<td>12.238</td>
</tr>
<tr>
<td>FM22</td>
<td>6781.50</td>
<td>1746.03</td>
<td>-0.26</td>
<td>8.853</td>
</tr>
</tbody>
</table>

© 2017 The Authors.
International Transactions in Operational Research © 2017 International Federation of Operational Research Societies
5. Concluding remarks and future work

Customers often have a high-level view of their needs and it is hard for them to understand the feature structure of an SPL and to translate that into concrete feature requests. Therefore, our focus in this paper is to facilitate the application engineering phase by dynamically guiding the customers through the product configuration process. Thus, our approach enhances the quality of the final product and reduces the mental burden and cognitive complexity associated with the configuration process.

Computational experiments showed that our heuristic can quickly derive a high-quality feature selection. Therefore, our approach is able to positively influence the quality of the software products that are developed during the application engineering phase. Our empirical results point out to the fact that (a) the backtracking algorithm running from preprocessed instances was always faster than the same algorithm running from the original instances; (b) the greedy heuristic algorithm was competitive with the backtracking algorithm despite the much smaller computational cost; and (c) the BRKGA heuristic outperformed the backtracking algorithm in all but two instance groups, but with larger running times than the greedy heuristic (specially on the largest instances), which may make the greedy heuristic more appropriate to be deployed in a decision aid tool for SPL product configuration, where a recommendation must be done in a few seconds. Therefore, the SPLConfig tool provides open-source implementation of our greedy heuristic algorithm. The source code can be downloaded from the tool website: http://homepages.dcc.ufmg.br/~figueiredo/spl/splconfig/.

Future works may study other exact algorithms (e.g., based on CS) or heuristic algorithms (e.g., based on metaheuristics) for the PCP optimization problem proposed here. Besides, other objective functions that map the qualitative NFP degree of preference to the customer satisfaction could be experimented with. Moreover, future works can extend our approach in order to consider nonlinear relationships among NFP (i.e., feature interactions), NFP measurement, and collaborative product configuration.

Acknowledgments

This work was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq), the Foundation for Support of Research of the State of Minas Gerais, Brazil (FAPEMIG), and Coordination for the Improvement of Higher Education Personnel, Brazil (CAPES).

References


© 2017 The Authors.


© 2017 The Authors.
International Transactions in Operational Research © 2017 International Federation of Operational Research Societies


