ARTHUR HENRIQUE SOUSA CRUZ

PCP-R: A NEW VARIANT OF PRODUCT CONFIGURATION IN SOFTWARE PRODUCT LINES

LAVRAS – MG

2019

ARTHUR HENRIQUE SOUSA CRUZ

PCP-R: A NEW VARIANT OF PRODUCT CONFIGURATION IN SOFTWARE PRODUCT LINES

Trabalho apresentado à Universidade Federal de Lavras a fim de cumprir os requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Prof. Dr. Mayron César de Oliveira Moreira
Orientador

Prof. Dr. Heitor Augustus Xavier Costa
Coorientador

LAVRAS – MG

2019

Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos

da Biblioteca Universitária da UFLA

Cruz, Arthur Henrique Sousa.

PCP-R: A new variant of product configuration in software product lines / Arthur Henrique Sousa Cruz. – Lavras : UFLA, 2019.

40 p. :

TCC (graduação)–Universidade Federal de Lavras, 2019.
Orientador: Prof. Dr. Mayron César de Oliveira Moreira.
Coorientador: Prof. Dr. Heitor Augustus Xavier Costa.
Bibliografia.

1. Linha de Produto de Software. 2. Problema de Configuração de Produto. 3. Otimização. I. Moreira, Mayron César de Oliveira. II. Costa, Heitor Augustus Xavier. III. Título.

ARTHUR HENRIQUE SOUSA CRUZ

PCP-R: A NEW VARIANT OF PRODUCT CONFIGURATION IN SOFTWARE PRODUCT LINES

Trabalho apresentado à Universidade Federal de Lavras a fim de cumprir os requisitos para a obtenção do título de Bacharel em Ciência da Computação.

APROVADA em 02 de Dezembro de 2019.

Prof. Dilson Lucas Pereira UFLA
Prof. Paulo Afonso Parreira Junior UFLA

Mayron César de O. Moreira

Prof. Dr. Mayron César de Oliveira Moreira
Orientador

Prof. Dr. Heitor Augustus Xavier Costa
Co-Orientador

LAVRAS – MG
2019

Dedico este trabalho primeiramente a Deus e a minha família. Também a todos os meus amigos que me acompanharam durante meus anos de graduação.

AGRADECIMENTOS

Agradeço ao Professor Mayron, não somente pela sua dedicação como orientador, mas também por ser um ótimo amigo e conselheiro durante minha graduação.

Agradeço ao Professor Heitor, que me orientou em conteúdos nos quais eu não tinha conhecimento.

Agradeço à Universidade Federal de Lavras por disponibilizar recursos para a realização do trabalho.

"Devia ter amado mais

Ter chorado mais

Ter visto o sol nascer"

- Titãs

RESUMO

A crescente necessidade de sistemas de software maiores e mais complexos trouxe a ideia do reuso de partes de produtos com características (features) comuns em novos produtos. Linhas de Produto de Software (LPS) têm sido cada vez mais adotadas para atender essas demandas da indústria, e geralmente são representadas com modelos de características (feature models) estruturados em árvores. Contudo, escolher a melhor configuração de features que satisfaça o cliente pode ser difícil, e este é um problema de otimização chamado de Problema de Configuração de Produtos (PCP). Neste trabalho, nós propomos uma variante do PCP, o Problema de Configuração de Produtos com Requisitos (PCP-R). Essa nova variação considera a preferência do cliente em relação aos requisitos de software como o critério de escolha de features. Apresentamos também um modelo inteiro para este problema junto a experimentos com instâncias de teste geradas tendo base o repositório SPLOT. Os resultados mostram que a formulação resolve todo os grupos de instâncias na otimalidade em menos de 0,1 segundos e, cerca de 36% dos grupos testados, têm mais de 50% dos requisitos de maior preferência como parte da solução.

Palavras-chave: Linha de Produto de Software. Problema de Configuração de Produto. Modelo de Característica. Otimização. Modelo Matemático.

ABSTRACT

The growing need for larger and more complex software systems leads to better support to reuse product parts with commons features in a new product. Software Product Line (SPL) has been increasingly adopted to attend these demands in the software industry and it is generally represented by a feature model as a tree structure. However, choosing the best features configuration that satisfies the client can be difficult, and this is an optimization problem called Product Configuration Problem (PCP). In this work, we propose a variant of PCP, the Product Configuration Problem with Requirements (PCP-R). This new variation considers the client's preferred software requirements as the features' choice criteria. We also present a linear model for this problem along with experimental tests using instances generated inspired on the SPLOT repository. The results shows the formulation solves optimally all instances' groups in less than 0.1 seconds and, about 36% of the tested groups of instances, have 50% of the most preferred requirements as part of its solution.

Keywords: Software Product Line. Product Configuration Problem. Feature Model. Optimization. Mathematical Model.

LISTA DE ABREVIATURAS SIGLAS

BRKGA	Biased random-key genetic algorithm
FP	Functional Property
LP	Linear Programming
NFP	Non-Functional Property
PCP	Product Configuration Problem
PCP-R	Product Configuration Problem with Requirements
SPL	Software Product Line

SUMÁRIO

1 Introdução Geral 19

REFERÊNCIAS 21

ANEXO A – Trabalho seguindo as normas do periódico Empirical Software Engineering 23

1 INTRODUÇÃO GERAL

O Problema de Configuração de Produto (ou *Product Configuration Problem* - PCP) é um problema de otimização que visa escolher a melhor combinação de elementos em um modelo de características (*feature model*) de uma Linha de Produto de Software. Tal modelo é comumente representando através de uma árvore, onde os nós representam as features e as arestas suas relações. Para a solução do PCP, o trabalho de (PEREIRA et al., 2017) considera a satisfação do cliente em relação à *feature* como critério de escolha. Neste trabalho, apresenta-se o Problema de Configuração de Produto com Requisitos (PCP-R), que considera a preferência do cliente em relação aos requisitos de *software* como o critério de escolha. Essa nova abordagem traz como maior vantagem uma facilitação para o cliente, já que este passará a avaliar os requisitos de *software* que melhor atendem suas necessidades e não as *features* que, muitas vezes, não preenchem um requisito por completo.

Foi considerado que cada *feature* implementa, parcial ou integralmente, um ou mais requisitos e a escolha das *features* é limitada pelo orçamento do usuário, estabelecendo restrições do tipo mochila (GALLO; SIMEONE, 1989). Além disso, as features selecionadas devem respeitar: (i) as relações (XOR ou OR) com suas features pais; (ii) restrições lógicas entre items de ramos distintos da árvore do modelo de features; (iii) integralidade da escolha de requisitos, ou seja, um requisito não pode ser parcialmente atendido. Ao que consta, este é o primeiro trabalho que faz essa consideração. O objetivo dessa pesquisa é responder a seguinte pergunta: "existe um modelo matemático inteiro misto para o PCP-R, que possa ser resolvido através de um solver de maneira eficiente?".

A formulação proposta maximiza a combinação de features que implementam a satisfação de um cliente, baseado nos requisitos pelos quais ele tem maior preferência, respeitando as demais características do problema.

O título escolhido para este trabalho foi "PCP-R: A new variant of product configuration in software product lines", seguindo as normas do periódico Empirical Software Engineering, para o qual será submetido.

REFERÊNCIAS

GALLO, G.; SIMEONE, B. On the supermodular knapsack problem. **Mathematical Programming**, Springer, v. 45, n. 1-3, p. 295–309, 1989.

PEREIRA, J. A. et al. Heuristic and exact algorithms for product configuration in software product lines. **International Transactions in Operational Research**, Wiley Online Library, v. 24, n. 6, p. 1285–1306, 2017.

ANEXO A – Trabalho seguindo as normas do periódico Empirical Software Engineering

Noname manuscript No.
(will be inserted by the editor)

PCP-R: A new variant of product configuration in software product lines

Arthur Henrique Sousa Cruz · Mayron César de Oliveira Moreira · Heitor Augustus Xavier Costa

Received: date / Accepted: date

Abstract The growing need for larger and more complex software systems leads to better support to reuse product parts with commons features in a new product. Software Product Line (SPL) has been increasingly adopted to address these demands in the software industry and it is generally represented by a feature model as a tree structure. However, choosing the best features configuration that satisfies the client can be difficult, and this is an optimization problem called Product Configuration Problem (PCP). In this work, we propose a variant of the PCP, the Product Configuration Problem with Requirements (PCP-R). This new variation considers the client's preferred software requirements as the features' choice criteria. We also present a integer model for this problem along with experimental tests using instances generated inspired on the SPLOT repository. The results show the formulation solves optimally all instances' groups in less than 0.1 seconds and, about 36% of the tested groups of instances, have 50% of the most preferred requirements as part of its solution.

Keywords Software Product Line · Product Configuration Problem · Feature Model · Optimization · Mathematical Model

Arthur Henrique Sousa Cruz
Universidade Federal de Lavras
E-mail: arthur.cruz@estudante.ufla.br

Mayron César de Oliveira Moreira
Universidade Federal de Lavras
E-mail: mayron.moreira@ufla.br

Heitor Augustus Xavier Costa
Universidade Federal de Lavras
E-mail: heitor@ufla.br

1 Introduction

The growing need for larger and more complex software systems leads to better support to reuse product parts with commons features Pohl et al. (2005). Software Product Line (SPL) has been increasingly adopted to attend these demands in the software industry Clements and Northrop (2001). As Thüm et al. (2014) affirms, the SPL "is a family of software products that share a common set of features". Pereira et al. (2017) indicate that an SPL needs to satisfy the specific necessities of a particular segment, besides having a standard set of features that allow product configuration.

A feature can be an increment in functionality, a system property, functional requirements, an architecture or design pattern Kang et al. (1990); Bernardo et al. (2002); Batory (2005). Since a feature can represent different characteristics of software, Software Product Line can be applied in different contexts, such as mobile phones Figueiredo et al. (2008) and smart houses Cetina et al. (2009).

Feature modeling has become the standard method to represent an SPL in the research community since it can express the variability in product lines Myllärniemi et al. (2016). All of the possibles products of the SPL are derived from the feature model, and they need to satisfy its constraints Pereira et al. (2017). Choosing the best product configuration of features is called Product Configuration Problem (PCP) Asadi et al. (2014).

This study introduces the Product Configuration Problem with Requirements (PCP-R), a new variant of the PCP that considers the software requirements as a solution's quality criteria. This approach makes the process of choosing a feature easier for clients since they will evaluate the software requirements that best meet their necessities, not the features, which often do not fulfill a requirement. Our goal is to answer the following question: "Is there a mixed-integer mathematical formulation that can model the PRP-R such that accurate solutions are found in practical computational times by a commercial black-box solver?".

The remainder of the document is structured as follows. Section 2 presents a theoretical reference of feature models and mathematical formulations. Section 3 presents a brief literature review. Section 4 shows the mathematical model for PCP-R. Section 5 presents the database and the numerical experiments performed. In Section 6, we highlight the threats to validate this proposal. Conclusions and future works are presented in Section 7.

2 Theorical Reference

This section presents the basis of the proposed methodology for the PCP-R. In Section 2.1, we describe the main concepts concerning feature models. Then, Section 2.2 defines some aspects related to linear and integer formulations of combinatorial optimization problems.

2.1 Feature Model

The SPL is generally represented by a feature model as a tree structure. Nodes are the features, and their relationships are the parent-child edges Batory (2005).

Features can be classified into two types: the mandatory and optional ones. The mandatory features represent the common features among all products of the product line, while optional features indicate the variability among the products Pereira et al. (2017). It is possible to enable or disable SPL functionality according to different criteria, such as business and customer requirements. This characteristic is due to the optional features which allow variables configurations of products since they define specific points of variation Pohl et al. (2005); Goedicke et al. (2004). Furthermore, features can have *functional properties* (FPs), and *non-functional properties* (NFPs) Pereira et al. (2017).

Each feature can be classified as a *leaf feature* or a *non-leaf feature*. The leaf features have no children, whereas the non-leaf features have at least one child Pereira et al. (2017), and are used as nodes for grouping its children. The grouping nodes can be classified as one or zero of two kinds of groups: *exclusive alternative* (XOR) and *nonexclusive alternative* (OR), where $XOR \cap OR = \emptyset$. At least one of the children from the OR group must be selected, whereas the XOR one states that exactly one of them must be chosen Pereira et al. (2017); Soltani et al. (2012). Furthermore, a group may not be classified, making it possible to select zero, one, or more features.

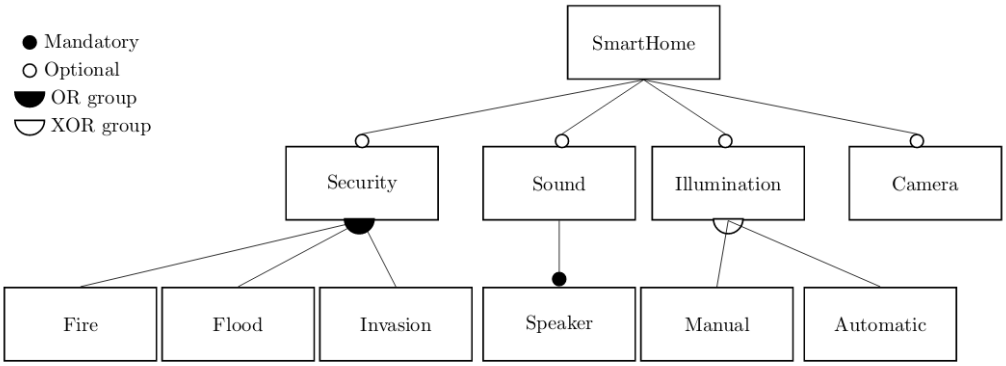
Some features have composition rules that cannot be represented in the feature tree. These characteristics define dependencies between them Pereira et al. (2018). These rules are called *cross-tree constraints*, and are logical expressions composed by binary operators such as $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication), $\leftrightarrow$ (biconditional), and the unary operator $\neg$ (negation) Pereira et al. (2017). These operators are used together with *boolean* variables that represent whether a feature was chosen or not.

Figure 1 is an example of a feature model. In this case, the feature *Speaker* is mandatory. *Sound*, *Illumination* and *Camera* are optional features. *Fire*, *Flood* and *Invasion* belongs to an alternative group (*OR-group*), while *Mannual* and *Automatic* belongs to an exclusive alternative group (*XOR-group*). The cross-tree constraint $Camera \rightarrow Sound \wedge Illumination$ makes necessarily choose *Sound* and *Illumination* if *Camera* is chosen. The cross-tree constraint $Security \rightarrow Camera$ makes necessarily chosen *Camera* if *Security* is chosen.

As defined by Soltani et al. (2012), a feature model FM can be represented as a sextuple, such that, $FM = (F, O, M, OR, XOR, C)$, where:

- F is the set of features;
- O is the set of optional parent-child relationships ($O \subseteq F \times F$);
- M is the set of mandatory parent-child relationships ($M \subseteq F \times F$);
- OR is the set that represent same parents children into alternative groups ($OR \subseteq F \times P(F)$, where $P(F)$ is the set of the features which are parents of other features);

Fig. 1 Example of a feature model. Based on Smart-home made by Pereira et al. (2017), which was adapted from Cetina et al. (2009).



$$\text{Camera} \rightarrow \text{Sound} \wedge \text{Illumination}$$

$$\text{Security} \rightarrow \text{Camera}$$

- XOR is the set that represent same parents children into exclusive alternative groups ($XOR \subseteq F \times P(F)$);
- C is a set of cross-tree constraints.

Finding the best product configuration is a difficult problem since we have two difficult subproblems. First, we have the boolean satisfiability problem (SAT), related to cross-tree constraints. Moreover, there are binary knapsack inequalities, due to the incurred costs of the implementation of features and the limited budget Gallo and Simeone (1989); Cormen et al. (2009). These characteristics make the PCP an NP-hard problem Pereira et al. (2017).

2.2 Linear Programming

Linear programming (LP) is a technique to solve optimization problems with linear functions and linear constraints Bazaraa et al. (2011). We define $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ as vectors of constants, and $A \in \mathbb{R}^{m \times n}$ as a full row rank m coefficient matrix. Let $x \in \mathbb{R}^n$ be a vector of variables. A linear problem can be written as:

$$\min(\text{or } \max) f(x) = c^T x \tag{1}$$

subject to:

$$Ax \oplus b \tag{2}$$

$$x \in \mathbb{R}^n \tag{3}$$

where $Ax \oplus b$ is a set of linear constraints, $\oplus \in \{\leq, \geq, =\}$, and $x \in \mathbb{R}^n$ define the domain of variables. The standard formulation of an LP model considers $\oplus = \{=\}$, and $x \in \mathbb{R}_+$.

A feasible solution for an optimization problem is a combination of values for all the variables, satisfying the constraints. A solution is optimal if there is no feasible solution with better objective value. An optimization problem can have a single optimal solution, multiple optimal solutions, infinite optimal solutions, or even the model can be infeasible Bazaraa et al. (2011).

George B. Dantzig proposed the "Simplex Method" which can optimally solve linear programs Bazaraa et al. (2011). However, finding the best solution for a problem may demand too much time since some problems have a large number of possible solutions. Even so, linear modeling is a valuable resource and can solve a variability of problems optimally.

The PCP-R formulation Section (4.2) has a set of binary variables. In this case, the solving strategy needs to include Integer Programming techniques, such as Branch & Bound based algorithms, Cutting Planes, and Column Generation. For more detailed examples of problems that can be solved via Linear and Integer Programming, we refer the book of Bazaraa et al. (2011); Wolsey (1998).

3 Related Works

There are many approaches proposed in the literature to solve the PCP Ochoa et al. (2017) with automatic methods. Bagheri et al. (2010) applied a variant of propositional logic along with fuzzy logic to represent and solve PCP with qualitative NFP. The proposed approach proved to be efficient for two feature models with up 290 features. Henard et al. (2015) proposed a search-based SPL feature selection algorithm to address PCP in larger search spaces. Their approach took approximately half an hour for feature models with up to 6888 features in their computational experiments.

Junior and Costa (2016) introduced a plug-in for the IDE Eclipse platform to help the management of SPL. The proposed plug-in allows CRUD operations in a feature model. Further, it can generate different combinations of products, limited by a maximum budget.

Xiang et al. (2018) proposed the SATVaEA, a combination of a multi-objective evolutionary algorithm called VaEA Xiang et al. (2016) with two SAT solvers. The algorithm was tested in 21 feature models with up 62.482 features. SATVaEA returned valid products for almost all feature models and needed only a few minutes for instances with up 10000 features.

Pereira et al. (2017) introduced a preprocessing algorithm to remove unfeasible feature combinations. The authors also proposed a backtracking algorithm and a biased random-key genetic algorithm (BRKGA) Gonçalves and Resende (2011) with a greedy heuristic for the constructive phase. The algorithms are tested in artificial and classical feature models from the literature. The performance of backtracking showed more efficient when the preprocessing algorithm is executed. The BRKGA constructive heuristic alone presented competitive results compared to the backtracking algorithm, but with shorter execution time. Further, BRKGA outperformed the backtracking algorithm in

all but two groups of instances. A desirable decision aid tool for SPL products must run in a few seconds. Thus, the authors concluded that the greedy heuristic is the most appropriate choice in their context.

Although PCP includes the customer features preferences, as far as its known, no work considers the software requirements as part of quality criteria. This work differs from the previous ones due to the addition of software requirements as the feature choosing criteria, the PCP-R, instead of the client satisfaction with a feature. Also this work propose a mixed-integer formulation for it.

4 Methodology

This section presents the Product Configuration Problem with Requirements (PCP-R) through a set of notations and definitions. In the following, we introduce a mathematical model to solve this problem.

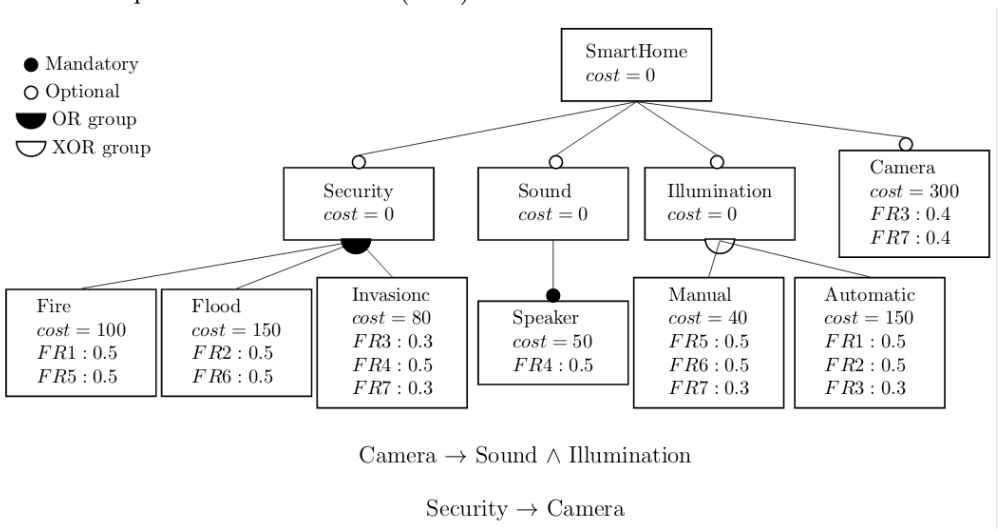
4.1 Formal definitions

Let $T = (V, A)$ be a feature tree, where V is the set of nodes (features) and A is the set of edges such that (i, j) means that feature j is selected if, and only if, task i is also selected. We define $M \subseteq V$ and $O \subset V$ as a set of mandatory and optional features, respectively, such that $M \cap O = \varnothing$, $M \cup O = V$. Take $L \subset V$ as a set of leaves of T. Each feature $v \in V$ has a cost l_v. In this paper, we establish that only leaves are concrete features. Therefore, the inner nodes do not have real implementation cost since they are abstract features, that is, $l_v = 0$, $\forall v \in V \backslash L$. A customer budget of B limits the choice of features.

Consider P as a set of features that have at least one child, and $D_p \in D$ as the set of children for $p \in P$. A feature can belong to an alternative group (OR) or an exclusive alternative group (XOR). ORP is the set of parents from groups OR. For all $p \in ORP$, there is a set ORC_p containing all the children of p. We also defined the sets $XORP$ and $XORC_p$ with the same rationale. The union of all ORC_p, for each $p \in ORP$, gives the set ORC, and the union of all $XORC_p$, for each $p \in XORP$, provides the set $XORC$. Since the tree T represents a feature model, we defined the root as $\gamma \in V$.

The main difference between PCP and PCP-R is the objective: in PCP, features are selected based on customer satisfaction with it. On the other hand, in PCP-R, features are chosen based on the customer's preferred requirements. The set of requirements R are divided in weighted groups such that the group $W_i \subseteq W$ is the i^{th} preferred group, and its elements have a weight w_i, where $w_1 > w_2 > ... > w_{|W|}$. If the requirement $r \in R$ has a preference weight w_i, then $r \in W_i$ and r has a higher preference than all the requirements that belong to W_j, with $j > i$. Note that $\bigcup_{i=1}^{|W|} W_i = R$ and $\bigcap_{i=1}^{|W|} W_i = \emptyset$.

Fig. 2 Example of a feature model. Based on Smart-home made by Pereira et al. (2017), which was adapted from Cetina et al. (2009).



Each requirement $r \in R$ is implemented by a set of features $V_r \subseteq V$. For each $\tilde{v} \in V_r$, we denote $\alpha_{\tilde{v}}$ as the percentage of implementation of requirement r by $\tilde{v}$, where $\sum_{\tilde{v}} \alpha_{\tilde{v}} = 1$. We highlight that a requirement $r \in R$ must preserve its atomic nature. Thus, the selected features must implement their corresponding requirements completely.

Let C be a set of cross-tree constraints C. We state that $c_i \in C$ is a $p \rightarrow q$ formula, where p and q are logical operations composed by $\wedge$, $\vee$ or $\neg$, such that the allowed number of boolean variables in p and q are 1 or 2.

Consider the following software functional requirements (FR):

- FR1: The lights must be automatically turned off in case of fire
- FR2: The lights must be automatically turned off in case of flood
- FR3: Invasions must be recorded in video (with automatic illumination)
- FR4: An alarm must be sounded in case of invasion
- FR5: The lights can be manually turned off in case of fire
- FR6: The lights can be manually turned off in case of flood
- FR7: Invasions must be recorded in video (with manual illumination)

Figure 2 is a variation of Figure 1 to exemplify the PCP-R with the FRs listed above. The feature model follows the previous specifications, which means only leaves are concrete and can implement a requirement. It's impossible to choose *Manual* and *Automatic* illumination in the same product due to the XOR grouping, meaning that there will be no product which implements the requirements FR1, FR2, FR3 and FR5, FR6, FR7. The best product configuration will depend on the input budget and the client's preference for the requirements.

4.2 Mathematical modeling

The mathematical modeling of the PCP-R is composed of the following set of variables:

$x_v \in \{0,1\}$	equal to one only if feature $v \in V$ is selected;
$g_r \in \{0,1\}$	equal to one only if the solution implements all features with requirement r;
$k_i \in \mathbb{Z}_+$	number of requirements of weight w_i selected.

The PCP-R formulation reads:

$$\max f(x) = \sum_{i=1}^{|W|} w_i k_i \tag{4}$$

subject to:

$$\sum_{v \in V} l_v x_v \leq B \tag{5}$$

$$x_\gamma = 1 \tag{6}$$

$$x_c \leq x_p, \quad \forall p \in P \ , \ \forall c \in D_p \ \mid c \in O \tag{7}$$

$$x_c = x_p, \quad \forall p \in P \ , \ \forall c \in D_p \ \mid c \in M \tag{8}$$

$$\sum_{c \in XORC_p} x_c = x_p, \quad \forall p \ \in \ XORP \tag{9}$$

$$\sum_{c \in ORC_p} x_c \geq x_p, \quad \forall p \ \in \ ORP \tag{10}$$

$$g_r \leq \frac{\sum\limits_{v \ \in \ V_r} x_v}{\mid V_r \mid}, \quad \forall r \in R \tag{11}$$

$$k_i = \sum_{r \in W_i} g_r, \quad \forall i \in W \tag{12}$$

$$\text{(Cross-tree constraints)} \tag{13}$$

$$x_v \in \{0,1\}, \quad \forall v \in V \tag{14}$$

$$k_i \in \mathbb{Z}_+, \quad i = 1, 2, ..., \mid W \mid \tag{15}$$

$$g_r \in \{0,1\}, \quad \forall r \in R. \tag{16}$$

The objective function (4) maximizes the weighted sum of the selected requirements, taking into account that the higher the weight, the more preferable the requirement. Constraint (5) establlish the choice of features limited by the budget B, while Constraint (6) forces the root feature to be chosen.

Constraint (7) indicate that an optional feature can only be chosen if its parent is selected. Similarly, Constraints (8) guarantee that a mandatory feature must be chosen if its parent is chosen. Constraints (9) state that at most, one child is picked in an XOR group. On the other hand, Constraints (10) ensure that at least one child is selected from OR groups.

Constraints (11) define an auxiliary variable g_r, while Constraints (12) count the number of selected requirements from each weighted group. Constraints (13) represent linearized logic cross-tree, explained with details in the sequence. Finally, Constraints (14), (15) and (16) estabilish the domain of variables.

In this study, we work with a limited number of cross-tree constraints. To linearize them, it is possible to deal with each case separately. Table 1 shows all the possibilities of cross-tree constraints and its equivalent linear form. Then, for each $c \in C$, we create one constraint according to the Table 1. Since the negation ($\neg$) operator can be represented as $1 - a$, where a is a boolean variable, it can be also represented with the same rules. Further, according to De Morgan Theorem, $\neg(a \vee b)$ and $\neg(a \wedge b)$ are the same as $\neg a \wedge \neg b$ and $\neg a \vee \neg b$, respectively, where a and b are boolean variables. Therefore, it is possible to represent the negation of disjunctions and conjunctions with the constraints presented in Table 1.

Table 1 Possible cross-tree constraints rules, and the equivalent model constraint, where $a, b, c, d \in \{0, 1\}$.

Cross-tree constraint	**Linear constraint**
$a \rightarrow c$	$c - a \geq 0$
$a \rightarrow c \vee d$	$c + d - a \geq 0$
$a \rightarrow c \wedge d$	$c + d - 2a \geq 0$
$a \vee b \rightarrow c$	$a + b - 2c \leq 0$
$a \vee b \rightarrow c \vee d$	$a + b - 2(c + d) \leq 0$
$a \vee b \rightarrow c \wedge d$	$a + b - 2c \leq 0$ $a + b - 2d \leq 0$
$a \wedge b \rightarrow c$	$a + b - c \leq 1$
$a \wedge b \rightarrow c \vee d$	$a + b - (c + d) \leq 1$
$a \wedge b \rightarrow c \wedge d$	$a + b - c \leq 1$ $a + b - d \leq 1$

5 Results and Discussion

In this section, we evaluate the quality of the solutions obtained by solving the proposed PCP-R formulation. Section 5.1 presents the characteristics of the database adopted in the computational experiment phase. Section 5.2 shows the results regarding different quality criteria.

5.1 Test instances

The literature related to PCP variants does not provide a set of instances concerning the PCP-R. For that reason, we introduce a benchmark for this problem, inspired in examples present int the SPLOT repository Mendonca et al. (2009). The instances were collected using the code present in Cruz (2019). In total, there are $1,145$ instances, but 8 of them were not discarded since they presented characteristics such as duplicate feature names. The parameters adopted to determine the range of values randomly generated were based on the instance generation scheme described in Pereira et al. (2017).

We converted all the cross-tree constraints from the conjunctive normal form (CNF) clause to the pattern described in 4.2. The converted constraints with no match with the previous definitions were excluded from its corresponding instance. The costs of the leaves were randomly generated according to $U[0, 100]$, $U[0, 1000]$, and $U[0, 10000]$, for instances with up 100, 1000, and greater than 1000 features, respectively.

We calculate a lower bound of the number of requirements, $\underline{r}$, such that all leaves can implement at least 10% of a single requirement. Then, the number of requirements was given by $\underline{r}$ plus a random integer in $U[0, 10]$, $U[0, 20]$, and $U[0, 30]$, for instances with up 100, 1000, and greater than 1000 features, respectively.

We divided the number of requirements into the following number of weighted groups: (i) $|W| = 3$, for instances with up 100 features; (ii) $|W| = 7$, for instances with up 1000 features; (iii) $|W| = 10$, for instances with number of features greater than 1000. We set $|R| = |W|$ whenever $|R| < |W|$, during the generation phase. The weights of the groups ranged from $[10, 20, ..., |W| * 10]$.

The generated instances are available in Cruz et al. (2019). The model was tested in an *Intel core i*7 - 7^{th} *generation* and $16GB$ computer, codified in *python3* using the Gurobi Solver 8.1.1 with 6 threads.

5.2 Results and discussion

We grouped the generated instances by groups with up 20, 50, 100, 200, 500 and 1000 features. Table 2 shows some of its characteristics. The first column is the maximum number of features ($|V|_{max}$); the second is the number of instances in the group ($Inst$); the third is the average number of cross-tree constraints of the instances ($\overline{C}$); the fourth one is the average number of requirements ($\overline{R}$); finally, Imp is the number of infeasible instances according to two criteria, described below:

1. If a requirement r is implemented by two features which are descendants of the same feature $v \in XORP$;
2. If a requirement r is implemented by a mandatory feature v with a mandatory sibling u, and u is a child of a feature $p \in XORP$.

For each instance, the model was executed with the following budgets: 100, 200, 500, 1000, 2000, 5000, 10000, 20000 and 50000. Table 3 shows the results

Table 2 Characteristics of the generated instances.

$\|V\|_{max}$	$Inst$	$\overline{C}$	$\overline{R}$	Imp (%)
20	480	1.192	6.373	31.14
50	456	2.814	7.605	35.22
100	168	10.834	9.315	42.41
200	24	11.042	19	36.61
500	8	69.75	30.25	20.45
1000	1	0	52	9.61

obtained after the tests. The new information in the columns are, from left to right:

- B: the budget tested;
- Opt: the number of optimal solutions found;
- $\underline{z}$: the number of optimal solutions with objective function value equal to zero, which means the model could not choose any complete requirement. Then, solutions are composed of abstract features and can be considered infeasible. We allowed this situation to avoid the addition of constraints for the non-leave features;
- Inf: the number of instances with infeasible solutions.

The results of groups that presented small deviations for different budgets were placed in the same line. For instance, see the fourth line of the Table 5, where the group with up 20 features showed no difference for the input of 1000, 2000, 5000, 10000, 20000 and 50000 of budget.

According to the experiments, the number of optimal solutions found increases as the input budget increases. This is somehow expected since we give more slack to knapsack constraints. On the other hand, the number of instances reported in column $\underline{z}$ reduces as the budget increases due to the possibility of choosing more concrete features, resulting in implemented requirements. The number of infeasible solutions also reduce with larger values of budget, explained by the same reason we observed with $\underline{z}$ values.

Table 4 includes the other two quality criteria to compare the results:

- $G\downarrow$: the average percentage of features that were chosen over the least half preferred groups. Note that if $G\downarrow = 100\%$, all features of these groups were chosen;
- $G\uparrow$: the average percentage of possibles features that were chosen over the most preferred group. Note that if $G\uparrow = 100\%$, all features of the group W_1 were selected.

We can see that part of the requirements were impossible to achieve due to the random generation. Moreover, about 36% of the groups tested have 50% of the high priority requirements, and only a single group ($V = 20$ and $B > 1000$) obtained more than 50% of the requirements from the worst half weight classes.

Table 5 reports the effectiveness of the PCP-R model solved by Gurobi. In this set of results, we present:

Table 3 Number of optimal and infeasible solutions found.

$\|V\|_{max}$	B	Opt	$\underline{z}$	Inf
20	100	301	264	179
20	200	398	202	82
20	500	470	80	10
20	[1000,2000, 5000,10000, 20000,50000]	476	74	4
50	100	235	227	221
50	200	309	240	147
50	500	399	94	57
50	1000	438	67	18
50	2000	444	62	12
50	[5000,10000, 20000,50000]	444	62	12
100	100	115	115	53
100	200	125	108	43
100	500	148	54	20
100	1000	159	39	9
100	2000	164	35	4
100	[5000,10000, 20000,50000]	164	35	4
200	[100,200,500]	11	11	13
200	1000	12	12	12
200	2000	14	12	10
200	5000	17	6	7
200	10000	22	7	2
200	20000	22	4	2
200	50000	22	4	2
500	[100,200, 500,1000]	5	5	3
500	2000	6	6	2
500	5000	7	4	1
500	10000	8	1	0
500	20000	8	0	0
500	50000	8	0	0
1000	[100,200,500, 1000,5000]	0	0	1
1000	10000	1	0	0
1000	20000	1	0	0
1000	50000	1	0	0

- $t(s)$: the average computational time spent by Gurobi to solve the model;
- $f^*(x)$: the average of objective function;
- $\#Nodes$: the average number of nodes explored in the Branch & Bound based algorithm by Gurobi;
- $\#Splx$: the average number of Simplex iterations performed by Gurobi.

The mathematical model performed accurately for the test problems since all instances were solved to optimality. The computational time was low, with an average of less than 0.1 seconds, in all budget scenarios. We also highlight the reduced number of nodes explored in the Branch & Bound based algorithm,

Table 4 Average percentages of best and worst chosen requirements and impossibles requirements.

$\|V\|_{max}$	B	$G\downarrow$ (%)	$G\uparrow$ (%)
20	100	2.3	5.9
20	200	12.2	26
20	500	43.3	75.1
20	[1000,2000, 5000, 10000, 20000,50000]	51.6	80.8
50	100	0.2	2.4
50	200	2.6	8.8
50	500	15.9	49.5
50	1000	37.1	74.8
50	2000	45.4	80
50	[5000,10000, 20000,50000]	45.6	80.1
100	100	0	0
100	200	1	3.5
100	500	4.2	42.5
100	100	12.8	58.8
100	2000	26.8	70
100	[5000,10000, 20000,50000]	29.3	71
200	[100,200,500]	0	0
200	1000	0	0
200	2000	0	8.3
200	5000	1.5	52.4
200	10000	4.7	32.9
200	20000	7.4	43
200	50000	18.7	50.4
500	[100,200, 500,1000]	0	0
500	2000	0	0
500	5000	0.5	4.8
500	10000	0.6	17.1
500	20000	4.5	38.3
500	50000	12.1	44.6
1000	[100,200,500, 1000,5000]	-	-
1000	10000	0	16.7
1000	20000	0	16.7
1000	50000	10	0

as well as the number of simplex iterations. Possible reasons that can explain these results are:

1. the limitations constraints facilitated the solution of the instances;
2. the atomicity of a requirement (11) constrained the solutions' search space;
3. the high number of impossible requirements created some instances of easy-solving;
4. having only the leaves as concrete features reduced the difficulty of finding the optimal solution.

6 Threats to validity

Some characteristics of this work may threaten its validity. As follows, we enumerate them.

Table 5 Results of the PCP-R model according to the execution of Goro

$\|V\|_{max}$	B	$t(s)$	$f*(x)$	$\#Nodes$	$\#Splx$
20	100	0.003	4.319	0.000	0.000
20	200	0.003	18.794	0.071	0.510
20	500	0.003	64.170	0.0896	1.221
20	[1000,2000, 5000,10000, 20000,50000]	0.003	71.723	0.000	0.015
50	100	0.004	0.766	0.000	0.000
50	200	0.004	5.405	0.0329	0.213
50	500	0.005	34.236	0.421	6.956
50	1000	0.004	65.434	0.336	4.999
50	2000	0.003	76.351	0.011	0.281
50	[5000,10000, 20000,50000]	0.004	76.667	0.009	0.270
100	100	0.005	0.000	0.000	0.000
100	200	0.005	3.040	0.024	0.060
100	500	0.006	21.216	0.673	13.107
100	1000	0.007	40.000	1.512	24.649
100	2000	0.005	62.805	0.143	3.512
100	[5000,10000, 20000,50000]	0.005	66.463	0.030	0.506
200	[100,200,500]	0.007	0.000	0.000	0.000
200	1000	0.007	0.000	0.000	0.000
200	2000	0.007	8.571	0.0417	0.000
200	5000	0.009	45.882	0.333	12.333
200	10000	0.012	71.818	1.833	66.125
200	20000	0.013	122.727	5.917	112.500
200	50000	0.009	190.909	0.417	26.167
500	[100,200, 500,1000]	0.014	0.000	0.000	0.000
500	2000	0.015	0.000	0.000	0.000
500	5000	0.017	22.857	0.125	1.375
500	10000	0.028	75.000	3.500	135.625
500	20000	0.035	166.250	1.000	293.125
500	50000	0.040	243.750	5.625	333.250
1000	[100,200,500, 1000,5000]	0.020	-	0.000	0.000
1000	10000	0.030	70.000	0.000	5.000
1000	20000	0.041	120.000	1.000	273.000
1000	50000	0.037	140.000	1.000	255.000

– Cross-tree constraints: the limitations added to the cross-tree constraints may have a considerable influence on the results obtained by the model. Regarding that, we have at most four binary variables in each clause, the SAT problem could be easily solved. Thus, the time spent to find feasible solutions was reduced.
– Abstract features: different from PCP, the non-leaves features can not implement any requirements in PCP-R. Thus, the number of possible requirements combinations could be reduced, which decreases the time to find optimal solutions.

– Test instances: the impossible combination of requirements generated by the random distribution might facilitate the search for the optimal solution. Furthermore, the majority of the instances that are available in SPLOT Mendonca et al. (2009) have less than 100 features, making it difficult to prove the scalability of the proposed mathematical formulation.

7 Conclusions and future avenues

The product configuration problem (PCP) is a common optimization problem in SPL engineering. It is difficult to choose the best configuration of an SPL among all the possibles configurations that can be represented as a feature model.

The PCP considers customer satisfaction as the solution quality criteria. In this work, we presented a variation for the Product Configuration Problem, which includes the client's preferred requirements as the main criteria for choosing a feature, since it's possible to add an implementation value for a feature.

The PCP-R was formally and mathematically described. To answer the question "Is there a mixed-integer mathematical formulation that can model the PRP-R such that accurate solutions are found in practical computational times by a commercial black-box solver?", we propose a mixed-integer model with some limitations concerning cross-tree constraints. According to the tests, the implemented model could optimally solve a set of instances created based on the SPLOT repository Mendonca et al. (2009). For all the instances, the model execution time was less than one second, and the results show that the most preferred requirements were prioritized compared with the least preferred. Finally, we evaluate criteria related to the results obtained by Gurobi. The goal is to understand better the model behavior to explain the execution time and the possible threats of the validity of this work.

We intend to linearize more general cross-tree constraints, to remove the cross-tree pattern limitation. For future avenues of this works, it is possible to reformulate the model to accept leaves and non-leaves as concrete features. Also, more complex instances can be tested to verify the efficiency of the proposed formulation.

References

Asadi M, Soltani S, Gasevic D, Hatala M, Bagheri E (2014) Toward automated feature model configuration with optimizing non-functional requirements. Information and Software Technology 56(9):1144–1165

Bagheri E, Di Noia T, Ragone A, Gasevic D (2010) Configuring software product line feature models based on stakeholders' soft and hard requirements. In: International Conference on Software Product Lines, Springer, pp 16–31

Batory D (2005) Feature models, grammars, and propositional formulas. In: International Conference on Software Product Lines, Springer, pp 7–20

Bazaraa MS, Jarvis JJ, Sherali HD (2011) Linear programming and network flows. John Wiley & Sons

Bernardo M, Ciancarini P, Donatiello L (2002) Architecting families of software systems with process algebras. ACM Transactions on Software Engineering and Methodology (TOSEM) 11(4):386–426

Cetina C, Giner P, Fons J, Pelechano V (2009) Autonomic computing through reuse of variability models at runtime: The case of smart homes. Computer 42(10):37–43

Clements P, Northrop L (2001) Software product lines: Patterns and practice. Boston, MA, EUA: Addison Wesley Longman Publishing Co

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms. MIT press

Cruz AHS (2019) Spl repository feature model scrapper. URL `https://github.com/thuzax/dowloader-splot-instances`

Cruz AHS, Moreira MCdO, Costa HAX (2019) Test instances generated based on splot repository. URL `https://drive.google.com/file/d/1odKcfyfoUPbowf8jL2nI6Bmp6NAcTwSl/view?usp=sharing`

Figueiredo E, Cacho N, Sant'Anna C, Monteiro M, Kulesza U, Garcia A, Soares S, Ferrari F, Khan S, Castor Filho F, et al. (2008) Evolving software product lines with aspects. In: 2008 ACM/IEEE 30th International Conference on Software Engineering, IEEE, pp 261–270

Gallo G, Simeone B (1989) On the supermodular knapsack problem. Mathematical Programming 45(1-3):295–309

Goedicke M, Köllmann C, Zdun U (2004) Designing runtime variation points in product line architectures: three cases. Science of Computer Programming 53(3):353–380

Gonçalves JF, Resende MG (2011) Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics 17(5):487–525

Henard C, Papadakis M, Harman M, Le Traon Y (2015) Combining multi-objective search and constraint solving for configuring large software product lines. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press, pp 517–528

Junior CDM, Costa HAX (2016) Customização de Produtos em Linhas de Produto de Software por Algoritmos de Busca e Otimização Mediante Demanda de Custo

Kang KC, Cohen SG, Hess JA, Novak WE, Peterson AS (1990) Feature-oriented domain analysis (foda) feasibility study. Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst

Mendonca M, Branco M, Cowan D (2009) Splot: software product lines online tools. In: Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications, ACM, pp 761–762

Myllärniemi V, Savolainen J, Raatikainen M, Männistö T (2016) Performance variability in software product lines: proposing theories from a case study. Empirical Software Engineering 21(4):1623–1669

Ochoa L, Pereira JA, González-Rojas O, Castro H, Saake G (2017) A survey on scalability and performance concerns in extended product lines configuration. In: Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems, ACM, pp 5–12

Pereira JA, Maciel L, Noronha TF, Figueiredo E (2017) Heuristic and exact algorithms for product configuration in software product lines. International Transactions in Operational Research 24(6):1285–1306

Pereira JA, Martinez J, Gurudu HK, Krieter S, Saake G (2018) Visual guidance for product line configuration using recommendations and non-functional properties. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, ACM, pp 2058–2065

Pohl K, Böckle G, van Der Linden FJ (2005) Software product line engineering: foundations, principles and techniques, vol 1, 1st edn. Springer Science & Business Media

Soltani S, Asadi M, Gašević D, Hatala M, Bagheri E (2012) Automated planning for feature model configuration based on functional and non-functional requirements. In: Proceedings of the 16th International Software Product Line Conference-Volume 1, ACM, pp 56–65

Thüm T, Apel S, Kästner C, Schaefer I, Saake G (2014) A classification and survey of analysis strategies for software product lines. ACM Computing Surveys (CSUR) 47(1):6

Wolsey LA (1998) Integer programming. Wiley

Xiang Y, Zhou Y, Li M, Chen Z (2016) A vector angle-based evolutionary algorithm for unconstrained many-objective optimization. IEEE Transactions on Evolutionary Computation 21(1):131–152

Xiang Y, Zhou Y, Zheng Z, Li M (2018) Configuring software product lines by combining many-objective optimization and sat solvers. ACM Trans Softw Eng Methodol 26(4):14:1–14:46, DOI 10.1145/3176644, URL `http://doi.acm.org/10.1145/3176644`