A Feature-Based Personalized Recommender System for Product-Line Configuration

Juliana Alves Pereira Pawel Matuszyk Sebastian Krieter Myra Spiliopoulou Gunter Saake
University of Magdeburg, Germany
{juliana.alves-pereira, pawel.matuszyk, sebastian.krieter, myra.spiliopoulou, gunter.saake}@ovgu.de

Abstract

Today's competitive marketplace requires the industry to understand unique and particular needs of their customers. Product line practices enable companies to create individual products for every customer by providing an interdependent set of features. Users configure personalized products by consecutively selecting desired features based on their individual needs. However, as most features are interdependent, users must understand the impact of their gradual selections in order to make valid decisions. Thus, especially when dealing with large feature models, specialized assistance is needed to guide the users in configuring their product. Recently, recommender systems have proved to be an appropriate mean to assist users in finding information and making decisions. In this paper, we propose an advanced feature recommender system that provides personalized recommendations to users. In detail, we offer four main contributions: (i) We provide a recommender system that suggests relevant features to ease the decision-making process. (ii) Based on this system, we provide visual support to users that guides them through the decision-making process and allows them to focus on valid and relevant parts of the configuration space. (iii) We provide an interactive open-source configurator tool encompassing all those features. (iv) In order to demonstrate the performance of our approach, we compare three different recommender algorithms in two real case studies derived from business experience.

Categories and Subject Descriptors D.2.13 [*Software Engineering*]: Reusable Software.

Keywords Software Product Lines, Product-Line Configuration, Recommenders, Personalized Recommendations

GPCE '16, October 31-November 01, 2016, Amsterdam, Netherlands
© 2016 ACM. ISBN 978-1-4503-4446-3/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2993236.2993249

1. Introduction

In today's competitive market, mass production has given way to mass personalization, which means to satisfy particular needs of each specific customer. In product-line literature, mass personalization of products is known as product configuration. However, the actual process of configuration of large product lines with complex feature dependencies is a time-consuming, error-prone and tedious task. Thus, it is crucial for companies to have an easy and comprehensive product configuration process.

To achieve that goal we propose a product-line configuration approach encompassing a *recommender system* and *visualization mechanisms* that aid a user in the configuration process. The visualization mechanisms narrow the configuration space of possible features down to the permitted features and highlight selected features needed to finish a configuration. The remaining features are then scored with respect to their relevance by our recommender system. Consequently, a user is provided with a limited set of permitted, necessary and relevant choices.

Although there are approaches in the literature that recommend configurations to users by adopting recommender techniques [2, 3, 4, 5, 16, 30, 31, 32, 44], this is the first approach that uses an automated and personalized recommender system that learns about the relevant features from past configurations. Therefore, no intervention of human experts is necessary in the creation of the recommendations. Moreover, the aforementioned approaches do not guide users into a valid configuration using visualization mechanisms. While there are many specialized configurator tools that aim to provide this support (e.g., FeatureIDE [46] and SPLOT [34]), they only ensure that any partially configured product is in accordance with the product line constraints. This may lead to delays due to users' exploration of choices at each step of the configuration process. Thus, handling the product configuration process in large product lines is still a critical issue in many companies and the current literature still lacks a complete technique to support this process.

Based on the literature [10, 13, 36, 37, 39], we identify the following configuration challenges that arise from the industry needs when using configurators: (i) industry product lines often contain too many options and complex relationships; (ii) decision makers, e.g. requirement engineers and business analyst, are usually unsure about users needs when confronted with a set of choices; and (iii) it is difficult to define a valid configuration since often users specify requirements that are inconsistent with the feature model's constraints, and also features of no importance to the user need to be taken into account when it comes to fulfill the constraints. Thus, we address these challenges by providing the following contributions:

- We propose a collaborative feature recommender system that is based on previous users' configurations to generate personalized recommendations for a current user.
- We provide visual support that guides users through the decision-making process and allows them to focus on valid and relevant parts of the configuration space.
- We design an open-source configurator tool support for our approach[1] by extending a state-of-the-art tool [46].
- We empirically evaluate the performance of three different recommender algorithms on two real-world datasets of configurations derived from business experience.

Furthermore, from our experimental results on two datasets from our industry partners, we derive answers to the following research questions (*RQs*):

RQ1. Can a recommender system support product-line configuration in realistic configuration scenarios?

RQ2. In which phase of product configuration can a recommender system support its users?

RQ3. What is the impact of the selected algorithms on the quality of recommendations?

To address these questions, we conducted numerous experiments with three different recommendation algorithms on two datasets. To draw conclusions from the results, we compare them with the performance of a random algorithm. This algorithm recommends randomly chosen features and, therefore, it indicates the minimal performance level every algorithm should reach. It also simulates the performance of an uninformed user without any support from a recommender system. Our experiments show that two of the three proposed recommendation algorithms clearly and consistently outperform the random recommender in finding relevant features.

The remainder of this paper is structured as follows. Section 2 introduces the basic concepts of product lines. Section 3 clarifies the position of this paper in the literature, highlighting the gaps filled by our approach. Section 4 describes the proposed approach. Section 5 presents the results of our experiments. Finally, Section 6 concludes the paper and discusses future directions.

[1] The tool and the full code can be found in the Web supplementary material `http://wwwiti.cs.uni-magdeburg.de/~jualves/RS/`.

2. Background

In this section, concepts such as product-line engineering and recommender systems are briefly introduced.

2.1 Product-Line Engineering

Product-line engineering is a paradigm related to software engineering, used to define and derive a set of similar products from reusable assets [22]. The development life-cycle of a product-line encompasses two main phases: *domain engineering* and *application engineering* [40]. While the *domain engineering* phase focuses on establishing a reuse platform, the *application engineering* phase is concerned with the effective reuse of assets across multiple products.

Domain Engineering. The domain engineering phase is responsible for the capture and documentation of reusable assets through feature models. Feature models provide a formal notation to represent and manage the interdependencies among reusable common and variable assets, called features [25]. The term feature model was proposed by Kang et al. [21] in 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 network protocols [6], smart houses [11], mobile phones [14], telecom systems [17], the Linux kernel [28], etc.

Application Engineering. The application engineering phase is responsible for capturing product's requirements and derivation of a concrete product through a product configuration process. A *concrete configuration* defines a valid configuration (cf. DEFINITION 3 - Sec. 4.1) that covers as much as possible of the product's requirements. Thus, *product configuration* is a decision-making process involving selecting a concrete and complete (cf. DEFINITION 1 - Sec. 4.1) combination of features from a feature model. Our focus in this paper is to ease the application engineering phase by proposing mechanisms to support the product configuration process.

2.2 Recommender Systems

Recommender systems learn users' preferences and predict future items of interest to them. Their goal is to alleviate the problem of information overload that occurs also in product-line configuration, where the number of possible features and configurations is too high for a single user to handle. Therefore, a system that narrows the possible choices down to the relevant ones is indispensable. We need automated, learning algorithms that find relevant items from a large set of items in a personalized way. The most common classes of such algorithms are *content-based recommender systems* and *collaborative filtering algorithms*.

Content-Based Recommender Systems. Content-based recommender systems analyse the content of items. Then, given the history of a user, they search for items that are similar to the ones the user purchased before. For more information on those algorithms we refer to the extensive survey by Lops et al. [27]. This type of algorithm is applicable only when the content of items can be analysed efficiently. In our application scenario, features are the items to be analysed. Since features often do not have a simple representation (e.g. text descriptions) and are not structured, their analysis is not efficiently possible. Therefore, the class of content-based algorithms is not applicable to our scenario.

Collaborative Filtering Algorithms. In contrast to content-based algorithms, *collaborative filtering* (CF) algorithms do not analyse the content of items. Instead, they are based on relevance feedback from users. The feedback has the form of ratings that are stored in a user-item-rating matrix X (cf. Sec. 4.1). In this work we focus on this class of algorithms.

CF algorithms can be divided into two categories. The first of them is *k-Nearest Neighbor Collaborative Filtering* (kNN-CF). A formalization of this algorithm and our adaptation of it to the product-line configuration scenario are shown in Section 4.3.1. For a survey on these methods we refer to [15].

The second class of CF algorithms is *Matrix Factorization* (MF). MF algorithms have shown their great predictive power in numerous studies [24, 43, 23]. Nowadays, MF is considered state-of-the-art in recommender systems. Therefore, we use a representative of this class of algorithms, the BRISMF method (Biased Regularized Incremental Simultaneous Matrix Factorization) by Takacs et al. [43], and we adapt it to our scenario of product-line configuration (cf. Sec. 4.3.3).

Since there is no single recommendation algorithm that performs the best in all applications, in this work, we implemented both kNN-CF and BRIMS to investigate which of them performs better in the domain of product line configuration.

3. Related Work

Since the introduction of feature models in 1990 by Kang et al. [21], several interesting studies using recommender systems have been proposed in the product-line configuration literature [2, 3, 4, 5, 16, 30, 31, 32, 44]. These studies deal with the information overload resulting from interactive mechanisms to configure large and complex product lines. While some of them [2, 3, 4, 5, 32, 44] aim to predict the utility of each feature for the users, others [16, 30] aim to predict the utility of an entire set of features, which forms a valid product configuration. Moreover, considering a broader configuration scenario, there is much work that provides optimization and visualization support.

Feature Recommender System. On the feature recommender system scenario, Mazo et al. [32] present a collection of recommendation heuristics to prioritize choices and recommend a collection of candidate features to be configured. However, the authors do not propose any mechanism to guide the users choosing among the candidate features. Moreover, further investigation is needed to understand how much these recommender heuristics increase the success rate of the final configuration and how to combine the collection of heuristics in an interactive and incremental configuration process.

Bagheri & Ensan [4] present dynamic decision models for guiding users through the product configuration process. In their approach, users are constantly asked to choose between two competing features. However, this approach may introduce inconsistencies in the ranking. Moreover, if both features are of equal (or no) interest to the users, no support is provided to guide the selection process, leading the algorithm to make poor decisions.

In a similar scenario, Tan et al. [44] and Bagheri et al. [2] propose a feature ranking approach to support decision makers. In their approach, decision makers should compare a randomly selected pair of features and identify their relevance in terms of satisfying a given quality requirement. However, as one feature may contribute to many quality requirements, the amount and complexity of options presented by the recommender system can be overwhelming to users. Thus, there is no evidence that the use of this approach is faster than the usual configuration process.

Product Recommender System. On the product recommender system scenario, Martinez et al. [30] apply tailored data mining interpolation techniques to predict configuration likability. The authors' approach is based on users' votes for a dataset of configurations. However, in the real world situation, the large amount of features and relationships presented by the configurator exceeds the user's capability to vote confidently. Additionally, implicit users' votes are very often subjective.

Galindo et al. [16] propose an approach, named Invar, which provides to the users a decision model with a set of questions and a defined set of possible answers. Based on the users' implicit feedback, a product is configured using decision propagation strategies. However,some vague descriptions and even misleading information may often be introduced in the questionnaires. Moreover, there is no quantitative evidence that the use of questionnaires would enhance the desirability of the end product.

Configuration Optimization. In the optimization scenario, there is much recent work in the literature (e.g., [3, 5, 19, 20, 26, 29, 35, 45]) that support the selection of features addressing the use of quality requirements. However, considering the diversity of quality requirements, it is not easy for stakeholders to define an objective function, which matches their features of preference. Consequently, undesired features might be selected as well.

Visualization Support. There are also approaches that use visualizations to aid the users. Among them, Martinez et al. [31] present a visualization paradigm, called FRoGs (Feature Relations Graphs). FRoGs shows the impact, in terms of constraints, of the considered feature on all other features. The purpose of their approach is to support decision makers to obtain a better understanding of feature constraints, and serve as a recommendation system, during the product configuration process. However, as FRoGs is not integrated with the configuration process, the amount and complexity of information presented may exceed the capability of a user to identify an appropriate configuration. Although there are several configurator tools (e.g., FeatureIDE [46], SPLOT [34], FaMa [8], VariaMos [33], pure::variants [42], Feature Plug-in [1]), very few have implemented further configuration support to answer industry needs [10] (cf. Sec. 1). Therefore, the current literature still lacks a more automatic technique to support the configuration process.

To address the aforementioned issues, we propose a feature-based recommender system to guide users through the product configuration process by directing the order of selecting features and predicting which of them are more useful. In contrast to current literature, our approach benefits from a simplified view of the configuration space by dynamically predicting the importance of the features. Moreover, it requires just explicit information and contributes with automatic mechanisms that can be successfully integrated with the previous works.

4. The Proposed Approach

Our approach creates an interactive perspective for users and offers recommendations to maximize the chances to have an adequate configuration in the end. The workflow in Figure 1 presents an overview of the configuration process, where all steps are complementary to each other. The decision makers are engaged in all the steps, knowing which features are considered and their importance for the final product. The interaction ensures that the decision makers understand the configuration space and its limitations and are also comfortable with the decisions that are made during the whole process.

The configuration process is carried out by considering five main activities: *configure*, *propagate decisions*, *check validity*, *calculate recommendations*, and *visualize*. First, the process is started by collecting requirements of a product from users, customers, and other stakeholders. Second, based on the product' requirements, the decision maker select features of interest from a focused and highlighted view (cf. Sec 4.2) on the feature model. Each time a user (de)selects a particular feature, decision propagation strategies are applied to automatically validate feature models, which results in a non-conflicting configuration [38]. Next, algorithms are used to check the partial configuration validity and to compute predictions of features' relevance (cf. Sec. 4.3). The predictions are displayed for features on the focused view to guide the current decision maker through a step-wise selection of features. The predictions are constantly updated as new information is received based on (de)selected features. Therefore, if decision makers are not familiar with the features and cannot decide what is the best choice, suggestions are presented to them. Finally, if the decision maker wishes to finish the configuration process, the focused and highlighted view combined with the recommender system can support them to have a desired and valid configuration. Through the use of our approach, configuration update and upgrade can also be supported from a partial configuration.

In this section, we detail the proposed components shown in Figure 2 that centers around the above characteristics.

4.1 Formal Definition

In this section, we describe the formulation and typology of product configuration. We define the variables $f_i \in \{-1, 0, 1\}$, such that $f_i = 1$, if feature i is selected for the final product, $f_i = 0$ if feature i is deselected for the final product, and $f_i = -1$ if the state of feature i is undefined. Variables needed in feature models are defined in the following way:

- A feature model $\mathcal{FM} = (\mathbb{F}, \mathcal{R})$ is a tuple that consists of a feature space $\mathbb{F} = \{-1, 0, 1\}^h$, where h is the number of features in the feature model, and a set of constraints $\mathcal{R} = \{\vec{r}_1, \vec{r}_2, ..., \vec{r}_m\}$, where m is the number of constraints of the feature model.
- In our formalism, we consider a constraint $\vec{r}_i$ as a clause from the feature model's propositional formula in conjunctive normal form. We represent it as a vector in $\mathbb{F}$ (i.e., $\vec{r}_i \in \mathbb{F}$ for $i \in \mathbb{N}$ and $1 \leq i \leq m$) such that the component j of a constraint r_i specifies whether the feature j should be selected ($r_{i_j} = 1$), deselected ($r_{i_j} = 0$), or is not relevant ($r_{i_j} = -1$). As example, consider a feature space $\mathbb{F}$ with $h = 4$. For the clause $(\neg f_1 \vee f_3 \vee f_4)$ the corresponding vector would be $\vec{r}_1 = (0, -1, 1, 1)$.

Given a feature model $\mathcal{FM}$, $\mathcal{C} = \{\vec{c}_1, \vec{c}_2, ..., \vec{c}_k\}$ is the set of all possible configurations, such that $\vec{c}_i \in \mathbb{F}$ for $i \in \mathbb{N}$ and $1 \leq i \leq k$. Configurations can be classified as *partial* or *complete*, as well as *valid* or *invalid*.

Definition 1. (Complete Configuration) Given a feature model $\mathcal{FM}$, a configuration $\vec{c} \in \mathcal{C}$ is *complete* iff each component of $\vec{c}$ has a defined selection state (i.e., $\forall i \in \{1, .., h\} : c_i \neq -1$). We denote the set of all complete configurations with $\mathcal{CC}$.

Definition 2. (Partial Configuration) Given a feature model $\mathcal{FM}$, a configuration $\vec{c} \in \mathcal{C}$ is *partial* iff it is not complete (i.e., iff $\vec{c} \notin CC$). We denote the set of all partial configurations with $\mathcal{PC}$, which equals to $\mathcal{C} \setminus \mathcal{CC}$.

Definition 3. (Valid Configuration) Given a feature model $\mathcal{FM}$, a (partial) configuration $\vec{c} \in \mathcal{C}$ is *valid* iff it satisfies all constraints in $\mathcal{R}$ when considering all undefined features in $\vec{c}$ as deselected. More formally, $\vec{c}$ is valid iff $\forall \vec{r} \in \mathcal{R}, \exists i \in \{1, .., h\} : r_i \neq -1 \wedge complete(c_i) = r_i$, where the function *complete* is defined as:

$$complete(c) = \begin{cases} 0, & if\, c = -1 \\ c, & otherwise \end{cases}.$$

We denote the set of all valid configurations with $\mathcal{VC}$.

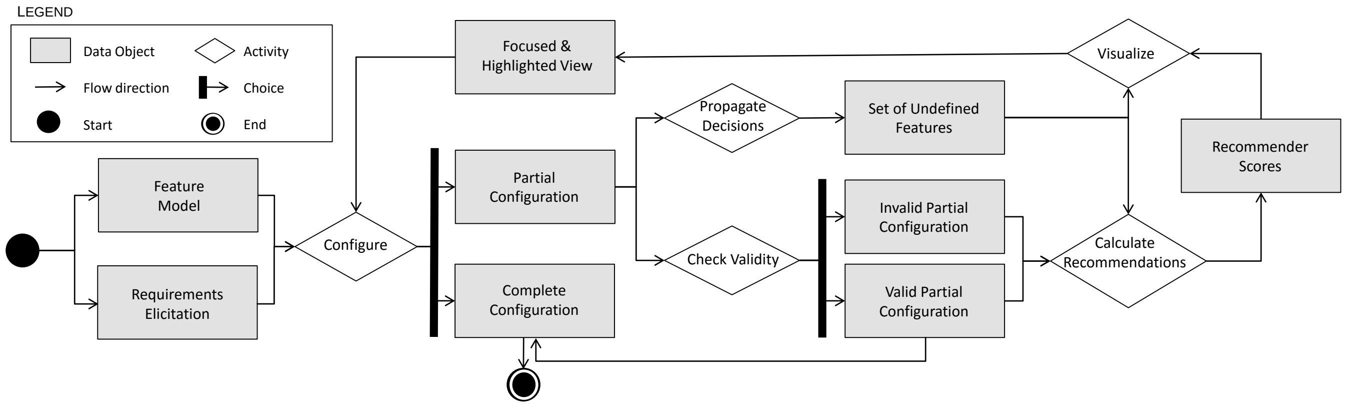

Figure 1. An overview of the configuration process.

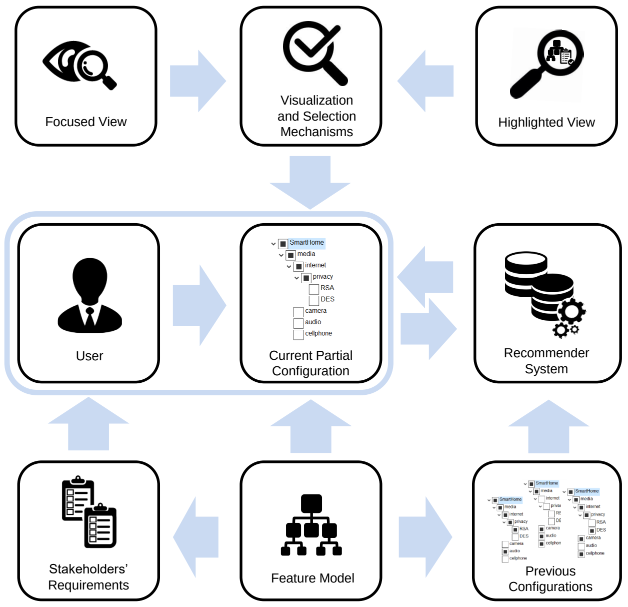
Focused View
Visualization and Selection Mechanisms
Highlighted View
User
Current Partial Configuration
Recommender System
Stakeholders' Requirements
Feature Model
Previous Configurations

Figure 2. Proposed components and their interplay.

Definition 4. (Invalid Configuration) Given a feature model $\mathcal{FM}$, a configuration $\vec{c} \in \mathcal{C}$ is *invalid* iff it is not valid (i.e., iff $\vec{c} \notin \mathcal{VC}$). We denote the set of all invalid configurations with $\mathcal{IC}$, which is equal to $\mathcal{C} \setminus \mathcal{VC}$.

In order to predict a valid and useful configuration to users, our recommender uses a *configuration matrix* X as input. In this context, given a feature model $\mathcal{FM}$ and a set of configurations $\{\vec{c}_1, ..., \vec{c}_n\} \subseteq \mathcal{CC}$, a *configuration matrix* X is defined as:

$$X = \begin{bmatrix} c_{1_1} & c_{1_2} & \cdots & c_{1_h} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n_1} & c_{n_2} & \cdots & c_{n_h} \end{bmatrix}$$

Given the variables and constants described above, our approach aims to predict the importance of features for each user in a way that they can make decisions more easily.

4.2 Visualization and Selection Mechanisms

Since our goal is to guide users in selecting features, we first focus on supporting the users' interaction with the feature model. To improve the configurator's interactivity, we propose two visualization mechanisms: (a) *focused view*, and (b) *highlighted view*.

Focused View. This mechanism focuses the user's view on the relevant configuration space. The essential idea of this mechanism is that a feature tree hierarchy represents the features degrees of abstraction. Thus, the higher a feature is located in the tree, the higher its level of abstraction is. In contrast, leaf features (i.e. features without any children) are the most detailed and technical features. Consequently, a user should first decide between the most abstract features before going into detail. To support this intuitive process, we implement a level-wise view on the feature tree. We assume that when the user selects a feature, they are probably interested in its sub-features. Thus, the view automatically expands and shows only the direct sub-features from selected features. With the focus on direct sub-features, we reduce the user's decision space in each configuration step. Thus, the user has a proper overview of the configuration process and is able to focus on one particular choice at each time.

Highlighted View. This mechanism guides the user to a valid configuration. It shows the user which decisions are necessary to finish the configuration process by highlighting the corresponding features. To find these features, we use an algorithm based on the propositional formula of a feature model in CNF. We chose this formalism because it can be easily implemented and applied to any kind of variability model [7]. Using the user's current configuration $\vec{c}$, the algorithm determines the set of all unsatisfied clauses $\mathcal{UR} \subseteq \mathcal{R}$ in the CNF. This set can be defined as $\mathcal{UR} = \{\vec{r} \in \mathcal{R} \mid \forall i \in \{1,..,|\mathbb{F}|\} : r_i = -1 \lor complete(c_i) \neq r_i\}$. Then, the algorithm is able to highlight every feature that is contained in an unsatisfied clause. More formally, for each $\vec{r} \in \mathcal{UR}$ the algorithm determines all features i with

$r_i \neq -1$. To provide optimized guidance for the user, the algorithm considers each unsatisfied clause separately and highlights its features. Naturally, (de)selecting one of those features automatically satisfies the corresponding clause $\vec{r}$. After a clause is satisfied by the user's (de)selection, the focus will automatically change to the next unsatisfied clause. Thus, with this mechanism, the user can efficiently finish the configuration process and simultaneously prevent undesired feature selections.

4.3 Recommender System

Even with visual mechanisms for reasoning on feature models, manually configuring a product can be a massive and difficult process. To ease this process, we adapt three personalized recommendation algorithms to the scenario of product-line configuration: (i) *neighbourhood-based CF*, (ii) *average similarity*, and (iii) *matrix factorization*. These algorithms use previous configurations for estimating and predicting relevance of features in order to guide a user through the process of feature selection.

4.3.1 Neighbourhood-based CF Recommender

In this section we present how neighbourhood-based CF algorithms (kNN-CF) can be adapted to make recommendations of features in the product-line configuration domain.

Given the configuration matrix X (cf. Sec. 4.1) and a new partial configuration $\overrightarrow{pc} \in \mathcal{PC}$ that is currently being processed by a user, a recommendation algorithm has the task of finding relevant features for this partial configuration. To achieve that, the kNN-CF algorithm calculates the relevance scores for the non-selected features. Once this calculation is completed, the most relevant features are recommended. The computation of the relevance scores is performed as follows.

For the partial configuration $\overrightarrow{pc}$ a set of neighbours is determined. A neighbour is a configuration $\overrightarrow{c}_x \in X$ (a row vector from X with $x \in \{1, ..., n\}$) that is similar to $\overrightarrow{pc}$ according to a similarity measure. A configuration qualifies as similar, if $sim(\overrightarrow{pc}, \overrightarrow{c}_x) > \tau$, where τ is a similarity threshold that is given as an input parameter (cf. Sec 5.2.1).

Since our configuration matrix X is binary, in our experiments we use similarity measures such as Jaccard Coefficient, Mean Hamming Similarity and Dice Coefficient. For explanation of those measures we refer to [12].

Once the neighbourhood $\mathcal{N}(\overrightarrow{pc}, \tau) \subseteq \mathcal{CC}$ has been determined, the relevance score Rel for a feature f from the set of all non-selected features is calculated as follows:

$$Rel(\overrightarrow{pc}, f) = \frac{1}{|\mathcal{N}(\overrightarrow{pc}, \tau)|} \sum_{\overrightarrow{c}_x \in \mathcal{N}(\overrightarrow{pc}, \tau)} sim(\overrightarrow{pc}, \overrightarrow{c}_x) \cdot c_{x_f} \tag{1}$$

In other words, if many similar configurations from the neighbourhood have the feature f, then the relevance score for this feature is high. Note that $c_{x_f} \in \{0, 1\}$, therefore our formula differs from typical CF algorithms working with non-binary ratings. If $c_{x_f} = 0$, i.e. the feature f was not selected in the configuration c_x, then its similarity does not influence the relevance score.

In the last step, the relevance scores $Rel(\overrightarrow{pc}, f)$ are sorted and the top-k of them are returned as recommendations.

4.3.2 Average Similarity Recommender

This algorithm uses the same principle as kNN-CF, but it does not use the notion of a neighbourhood. Consequently, the configurations of all users are considered for computing the relevance score of a feature. Accordingly, Equation 1 is changed to:

$$Rel(\overrightarrow{pc}, f) = \frac{1}{n} \sum_{\overrightarrow{c}_x \in X} sim(\overrightarrow{pc}, \overrightarrow{c}_x) \cdot c_{x_f} \tag{2}$$

Note that the sum iterates over all configurations from X, which contains n configurations. This means that the relevance score is an average similarity of $\overrightarrow{pc}$ over the configurations that have the feature f selected.

We conduct experiments with this algorithm to investigate if restricting the neighbourhood size has an influence on the quality of recommendations in our application.

4.3.3 Matrix Factorization Recommender

In contrast to neighbourhood-based methods, matrix factorization algorithms do not rely on similarity of configurations. Instead, they transform the configuration matrix X into a latent space (for readers not familiar with this concept we refer to [9, 43]). The transformation is obtained by incremental minimization of an error function.

In this work, we use the BRISMF algorithm by Takács et al. [43] and adapt it to our product-line configuration problem. Formally, this transformation is represented as follows:

$$X_{n \times h} = P_{n \times k} \cdot Q_{k \times h} \tag{3}$$

where P is a latent matrix of configurations and Q a latent matrix of features, n is the number of configurations in X (cf. Sec. 4.1), h the number of features, and k is the number of latent dimensions. k is an exogenous input parameter that needs to be optimized (cf. Sec. 5.2.1). Using this transformation, the relevance score for a feature f in a partial configuration is calculated using the following equation:

$$Rel(\overrightarrow{pc}_f) = \overrightarrow{p}_{\overrightarrow{pc}} \cdot \overrightarrow{q}_f \tag{4}$$

where $\overrightarrow{p}_{\overrightarrow{pc}} \in P$ is a row vector from the latent matrix P describing the configuration $\overrightarrow{pc}$ in the latent space. $\overrightarrow{q}_f \in Q$ is the corresponding latent vector of feature f, i.e. a column vector from the matrix Q. Both $\overrightarrow{p}_{\overrightarrow{pc}}$ and $\overrightarrow{q}_f$ have length k.

First, the matrices P and Q are initialized randomly. To improve them, *Stochastic Gradient Descent* (SGD) is used to iteratively update the matrices by minimizing an error function. The error function used in the transformation is a prediction error between the true value $c_{x_f} \in X$ (i.e. a value known from the data, e.g. one, if a feature was selected by a user) and the predicted relevance score $Rel(c_{x_f})$ on a training set Tr (cf. Sec. 5.2.2 for definition of a training set):

$$Error = \sum_{c_{x_f} \in Tr} (e_{c_{x_f}})^2 \tag{5}$$

$$e_{c_{x_f}} = c_{x_f} - Rel(c_{x_f}) \tag{6}$$

For the training, only the selected features are used, i.e. the entries, where $c_{x_f} = 1$, because the meaning of a zero is ambivalent. A zero in the configuration vector can mean deselecting a feature on purpose, or not selecting it, because the user did not know the feature, even though it was relevant. Using the zero entries as indication of irrelevance would be misleading.

To minimize the error function and to update the matrices P and Q, SGD uses the following formulas (cf. [43]):

$$\overrightarrow{p}_{c_x} := \overrightarrow{p}_{c_x} + \eta(e_{c_{x_f}} \cdot \overrightarrow{q}_f - \lambda \cdot \overrightarrow{p}_{c_x}) \tag{7}$$

$$\overrightarrow{q}_f := \overrightarrow{q}_f + \eta(e_{c_{x_f}} \cdot \overrightarrow{p}_{c_x} - \lambda \cdot \overrightarrow{q}_f) \tag{8}$$

where η is a learn rate and λ is a regularization parameter that prevents overfitting. Both of them are input parameters to be set in advance (cf. Sec. 5.2.1). For the derivation of these gradient formulas we refer to [43] for readers, who are not familiar with the concept of matrix factorization.

Once the training is completed (e.g. due to convergence or maximal iteration number in SGD) the matrices P and Q can be used to make relevance predictions, as presented in Equation 4.

5. Evaluation

This section describes the evaluation protocol used to evaluate three different recommendation algorithms introduced in Section 4.3. In addition, for the purpose of comparison we also experiment with a random recommender system and report our results.

5.1 Target Feature Models and Datasets

In order to address the research questions (*RQ1—3*) introduced in Section 1, we use two real-world datasets of configurations from our industry partner as a configuration matrix[2] (DEFINITION 5 - Sec. 4.1). Table 1 summarizes the properties from both datasets used in the evaluation. For each dataset, we present four properties including the number of features ($\#f$), percentage of cross-tree constraints ($\mathcal{R}$), number of all possible configurations ($\#\mathcal{C}$), and number of historic configurations ($\#\overrightarrow{c}_x$). We give an upper bound on the number of possible configurations for the models since it is not feasible to determine the number of products for such a large model.

These models cover a range of sizes in terms of features and historic configurations. While the ERP System dataset provides a high-level representation of a product line in the business management content, the E-Agribusiness dataset represents variability in the e-commerce agribusiness domain. To the best of our knowledge, both of the feature models are the largest real-world datasets of configurations already cited in the literature. They have a very high degree of variability which would be hard for a user to go through without any additional support. These characteristics from both target models make them ideal to be employed in our experiments and explore our research questions.

5.2 Experiment Design

To evaluate our algorithms, we performed an offline evaluation that encompasses three components: (i) parameter optimization, (ii) splitting into training and test datasets, and (iii) evaluation metrics.

5.2.1 Parameter Optimization

Parameter optimization is essential for comparing algorithms that require parameter tuning. Consider a scenario, in which two algorithms A and B should be compared with respect to a quality measure q_A and q_B. If the parameters of those algorithms were tuned manually by a human expert, a conclusion from comparing q_A and q_B might be biased. Assuming, without loss of generality, that $q_A > q_B$, then it is not known, if the difference in the quality is due to the algorithm A being better, or because the human expert tuned A better. Therefore, an objective parameter optimization is necessary. Only then, results with the approximately optimal parameter settings q_A* and q_B* can be compared.

Recommender systems also require setting of parameters. Neighbourhood-based CF, for instance, requires a similarity measure and a threshold value τ, above which users are considered neighbours. Matrix factorization algorithms also have parameters to be set, e.g. the number of latent dimensions k, learn rate η and regularization λ.

In our optimization, we used a genetic algorithm. In the parameter optimization phase, we used a random hold-out sample of 30% of all configurations from our dataset, i.e. this set was held out from further training and evaluation for the sake of reliable conclusions. On this hold-out dataset the aforementioned genetic algorithm was used to optimize the F-Measure of different recommendation algorithms (cf. Sec. 5.2.3). Every experiment in this phase was validated using a 10-fold cross validation. The optimal parameter setting was then taken over into our main validation and applied onto the remaining 70% of configurations. The values of optimal parameter are shown in Table 2. The results presented in the following sections were achieved using those optimal parameter settings.

5.2.2 Splitting into Training and Test Datasets

Once the optimal parameter settings were found, we performed our main evaluation on the remaining 70% of configurations. For the sake of a fair evaluation splitting of the dataset

[2] The configuration dataset can be found at `http://wwwiti.cs.uni-magdeburg.de/~jualves/RS/`.

Dataset	**Domain**	$\#f$	$\mathcal{R}$	$\#\mathcal{C}$	$\#\overrightarrow{c}_x$
ERP System	Business Management	1653	$\approx 7\%$	$> 10^9$	171
E-Agribusiness	E-Commerce	2008	none	$> 10^9$	5749

Table 1. Main properties of the datasets.

		Dataset	
Method	**Parameter**	**ERP System**	**E-Agribusiness**
Avg. Sim.	$Sim.Measure$	Jaccard Coeff.	Jaccard Coeff.
kNN-CF	τ	0.000001	0,757576
	$Sim.Measure$	Jaccard Coeff.	Jaccard Coeff.
BRISMF	k	40	80
	η	0.0166	0.0948
	λ	0.0062	0.1

Table 2. Optimal parameter values.

into disjoint training and test dataset is necessary. In this main evaluation phase, we use the leave-one-out evaluation protocol to create realistic evaluation conditions. According to this protocol one configuration is left out from the training set and used for testing. The remaining ones are given to the algorithm as training data, i.e. the configuration matrix X. This simulates the behaviour of a real system, where a user logs in and carries out a new configuration. The data of the past configurations are available to the system and only the new configuration should be predicted. To perform well, a recommender system has to recommend the features that were used in the left-out test configuration based on the training data, i.e. all other configurations.

Formally, a test configuration is a partial configuration $\overrightarrow{pc} \in \mathcal{PC}$ that was left out from training set, i.e. it is not contained in the configuration matrix $\overrightarrow{pc} \notin X$. The recommender system returns an estimated configuration $\widehat{pc} = \{0,1\}^h$. Then, the degree of overlapping between the real configuration $\overrightarrow{pc}$ (known from the data, but held out from the recommendation algorithm) and the predicted one $\widehat{pc}$ is calculated using an evaluation measure (cf. Sec. 5.2.3; note that it is not a similarity measure from Sec. 4.3.1).

To further simulate the progress of a user in the configuration process, we gradually give parts of the configuration to the recommender system as training data, e.g. 10% of a complete configuration. Then, a new prediction $\widehat{pc}_{@10\%}$ is made and its quality is estimated using the aforementioned quality measure. The quality of a prediction is good, if the predicted configuration overlaps with the remaining 90% of the configuration that is not known to the system. A good recommender system should learn from the given parts of the configuration and improve the quality of recommendations of the remaining features as it obtains more information about the current configuration.

The entire process is repeated for all configurations and all recommendations algorithms separately. The final quality measure of a recommendation algorithm is the average quality over all configurations.

5.2.3 Evaluation Metrics

In our evaluation we use precision, recall and F-measure at w, where w is the number of permitted recommendations (cf. [41] for more details). Given a set Rec of features recommended by an algorithm and a set of truly relevant features Rel known from a test configuration, precision is calculated as follows: $Precision = \frac{|Rec \cap Rel|}{w}$. Analogously, recall is calculated using the formula: $Recall = \frac{|Rec \cap Rel|}{|Rel|}$.

As precision and recall should not be considered separately, for our plots we use a measure that combines them, i.e. the F-Measure.

$$F\text{-}Measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \qquad (9)$$

In our experiments we use a typical value of $w = 10$.

Evaluation metrics that are typical to recommender systems, such as RMSE of MAE, are in this case not applicable, since in our scenario we have binary data in the matrix X (cf. Sec. 4.1).

5.2.4 Comparison Baseline

As a comparison baseline we use a random recommender system. It returns a randomly ordered list of non-selected features as recommendations. It is important to compare with this algorithm, because it indicates a basic performance level every algorithm should reach. If a method does not outperform the random recommender, then it should not be used in the given application scenario.

Furthermore, the performance of this algorithm is equivalent to the performance of a hypothetical, fully uninformed user without any support from a recommender system.

5.3 Analysis of Results and Discussion

In our evaluation we performed more than 149,000 experiments (cf. Sec. 5.2.2) on a cluster running the (Neuro)Debian operating system [18]. In Figure 3 and Table 3 we present the results on the *dataset from the ERP domain.* The figure represents the F-Measure achieved by four different recommendation algorithms. F-Measure combines recall and precision, i.e. higher values are better. On the horizontal axis of the figure, we present the completeness of a configuration, i.e. the percentage features that are given to the algorithm as training data, where only the remaining part of the configuration needs to be predicted.

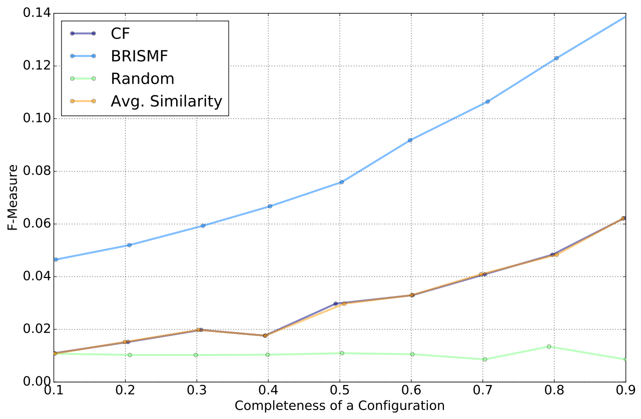

Figure 3. F-Measure achieved by four different recommendation algorithms on the ERP dataset (higher values are better). The horizontal axis shows, how much of the current configuration has been completed. The performance is calculated on the remaining part of a configuration.

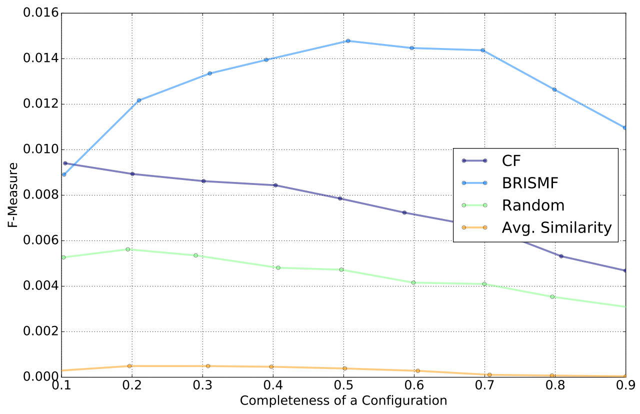

Figure 4. F-Measure achieved by four different recommendation algorithms on the E-Agribusiness dataset.

We observe that the BRISMF algorithm outperforms all other recommendation algorithms at all stages of the configuration process. We also observe an increase of performance as the configuration becomes more complete, i.e. as the recommendation algorithm receives more data for training.

The CF algorithm and average similarity algorithm perform nearly the same (their curves overlap). They dominate the random recommender at nearly all stages of configuration process, except for the initial part, when only little information about the current configuration is available.

The corresponding numeric values of F-Measure and also precision and recall are shown in Table 3. Also in the table, we see that the BRISMF algorithm dominates the other algorithms not only with respect to F-Measure, but also with respect to precision and recall.

In Fig. 4 and Tab. 4 we present the analogous results on the *E-Agribusiness dataset*. Also on this dataset, the BRISMF algorithm performed the best, except for the initial part of a configuration. In this part, the CF algorithm yielded a better recall and F-Measure. At all other stages of the configuration BRISMF clearly outperformed the remaining algorithms.

Different than on the ERP dataset, here the CF and average similarity algorithms perform much differently. While CF is the second best algorithm, the average similarity algorithm performs worse than random. Consequently, this algorithm should not be used in the E-Agribusiness scenario.

In comparison to the ERP dataset the absolute values of the quality measure are lower on the E-Agribusiness dataset. Since this dataset is more demanding, good predictions with a random recommender are unlikely, and the performance difference between the random recommender and the other algorithms, especially BRISMF, is larger.

The difficulty of the dataset also explains the different tendency of the curves. While the F-Measure curves on the ERP dataset show an increasing tendency (except for the random algorithm), here such a tendency cannot be observed. When a configuration becomes more complete, then there are fewer and fewer features that are relevant, i.e. their relative proportion in the set of all features drops. This results from the fact that features used by a user cannot be recommended any more. Thus, the number of remaining relevant features decreases as the configuration becomes more complete. Therefore, precision that influences the F-measure also decreases. On this dataset this effect outweighs the learning effect and, therefore, the curves are not monotonically increasing. Nevertheless, the curve of the BRISMF algorithm increases initially and reaches optimum around 50% of a configuration.

Considering the results of our experiments, we answer the research questions as follows:

RQ1. Yes, we have shown that our recommender system finds relevant features better than our comparison baseline and by recommending them it can support product-line configuration.

RQ2. Our results show that the BRISMF and CF algorithms provide better recommendations than a random recommender already at the initial 10% of a configuration. The optimal percentage differs and depends both, on the application domain and on the recommendation algorithm.

RQ3. The choice of the algorithm has a big impact onto the quality of recommendations. Due to the best performance in our experiments we recommend the usage of the BRISMF algorithm.

6. Conclusion and Future Work

In this paper, we targeted an open research question in the product-line configuration domain: *How to predict a suitable set of features from a feature model based on explicit information from users?* We answer this question by providing an advanced feature-based personalized recommender system. Our system guides the product configuration process by delivering capabilities to effectively communicate with the decision makers and understand users' needs and preferences. In-

Measure	Method	Completeness of configuration								
		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
F-Measure	CF	0.0109	0.0152	0.0198	0.0176	0.0297	0.0330	0.0409	0.0483	0.0622
	BRISMF	0.0465	0.0520	0.0593	0.0667	0.0759	0.0918	0.1065	0.1230	0.1396
	Random	0.0108	0.0103	0.0103	0.0104	0.0110	0.0106	0.0086	0.0135	0.0086
	Avg. Similarity	0.0108	0.0152	0.0198	0.0176	0.0297	0.0330	0.0409	0.0483	0.0621
Precision	CF	0.0525	0.0576	0.0644	0.0551	0.0737	0.0771	0.0814	0.0788	0.0856
	BRISMF	0.4449	0.4331	0.4195	0.4119	0.3975	0.3729	0.3347	0.3034	0.2432
	Random	0.1153	0.1008	0.0805	0.0805	0.0788	0.0602	0.0492	0.0500	0.0263
	Avg. Similarity	0.0517	0.0576	0.0644	0.0551	0.0737	0.0771	0.0814	0.0788	0.0847
Recall	CF	0.0069	0.0099	0.0133	0.0123	0.0217	0.0251	0.0324	0.0436	0.0588
	BRISMF	0.0247	0.0279	0.0323	0.0368	0.0426	0.0535	0.0652	0.0799	0.1026
	Random	0.0060	0.0058	0.0064	0.0059	0.0064	0.0066	0.0052	0.0092	0.0078
	Avg. Similarity	0.0069	0.0099	0.0133	0.0123	0.0217	0.0251	0.0324	0.0436	0.0587

Table 3. Performance of four recommendation algorithms w.r.t F-Measure, precision and recall on the ERP dataset. The BRISMF algorithm performs the best w.r.t. all three measures.

Measure	Method	Completeness of configuration								
		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
F-Measure	CF	0.0094	0.0089	0.0086	0.0084	0.0079	0.0072	0.0065	0.0053	0.0047
	BRISMF	0.0089	0.0122	0.0134	0.0139	0.0148	0.0145	0.0144	0.0126	0.0110
	Random	0.0053	0.0056	0.0054	0.0048	0.0047	0.0042	0.0041	0.0035	0.0030
	Avg. Similarity	0.0003	0.0005	0.0005	0.0005	0.0004	0.0003	0.0001	0.0001	0.0000
Precision	CF	0.0401	0.0337	0.0286	0.0250	0.0207	0.0157	0.0115	0.0078	0.0052
	BRISMF	0.0584	0.0585	0.0529	0.0471	0.0413	0.0326	0.0259	0.0191	0.0127
	Random	0.0234	0.0208	0.0177	0.0146	0.0125	0.0091	0.0075	0.0051	0.0035
	Avg. Similarity	0.0018	0.0017	0.0014	0.0012	0.0009	0.0005	0.0002	0.0001	0.0000
Recall	CF	0.0062	0.0061	0.0060	0.0061	0.0061	0.0059	0.0057	0.0054	0.0055
	BRISMF	0.0049	0.0071	0.0082	0.0091	0.0108	0.0111	0.0125	0.0121	0.0119
	Random	0.0035	0.0039	0.0039	0.0035	0.0037	0.0033	0.0039	0.0036	0.0038
	Avg. Similarity	0.0002	0.0004	0.0004	0.0004	0.0003	0.0003	0.0001	0.0001	0.0001

Table 4. Performance of four recommendation algorithms w.r.t F-Measure, precision and recall on the E-Agribusiness dataset. Also here, the BRISMF algorithm performs the best except for the initial part of a configuration.

stead of making decisions over the whole configuration space, decision makers of our personalized recommender only go through a small number of features to configure a product. It ensures a valid and complete product configuration while simultaneously interacting with the decision maker, both to learn their preferences and provide new recommendations.

Our experimental results show that the proposed approach is very useful as: (i) it provides feature predictions that are in accordance with the preferences of users and constraints over the feature model, and (ii) it has a good performance on current partial configurations with just 10% of selected features. Furthermore, our approach can be automated. It requires only one manual selection of a single feature to create an initial partial configuration. Thus, we can further facilitate the adoption of product-line practices and increase their benefits, such as mass personalization.

Since there are no other publicly available datasets with real configurations and obtaining them from companies is problematic, we could not test our work on further datasets. However, as future work, we plan to conduct a user study of our approach and tool, in order to investigate the gain in terms of acceptance and usability of the proposed interactive configuration process. Moreover, we will extend this work to take into account non-functional properties. Thus, we can make recommendations even when there is no historical data.

Acknowledgments

This work was partially supported by the CNPq grant (202368/2014-9). Moreover, the authors would like to thank the Institute of Psychology II at the University of Magdeburg for making their computational cluster available for our experiments.

References

[1] M. Antkiewicz and K. Czarnecki. FeaturePlugin: Feature Modeling Plug-in for Eclipse. In *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, pp. 67–72. ACM, 2004.

[2] E. Bagheri, M. Asadi, D. Gasevic, and S. Soltani. *Stratified Analytic Hierarchy Process: Prioritization and Selection of Software Features*. Springer, 2010.

[3] E. Bagheri, T. Di Noia, A. Ragone, and D. Gasevic. Configuring Software Product Line Feature Models based on Stakeholders' Soft and Hard Requirements. In *Software Product Lines: Going Beyond*, pp. 16–31. Springer, 2010.

[4] E. Bagheri and F. Ensan. Dynamic Decision Models for Staged Software Product Line Configuration. *Requirements Engineering*, 19(2):187–212, 2014.

[5] E. Bagheri, T. D. Noia, D. Gasevic, and A. Ragone. Formalizing Interactive Staged Feature Model Configuration. *Journal of Software: Evolution and Process*, 24(4):375–400, 2012.

[6] M. Barbeau and F. Bordeleau. A Protocol Stack Development Tool Using Generative Programming. In *Generative Programming and Component Engineering*, pp. 93–109. Springer, 2002.

[7] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems*, 35(6):615–708, 2010.

[8] D. Benavides, S. Segura, P. Trinidad, and A. R. Cortés. FAMA: Tooling a Framework for the Automated Analysis of Feature Models. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pp. 129–134, 2007.

[9] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[10] J. Bosch, R. Capilla, and R. Hilliard. Trends in Systems and Software Variability. *IEEE Software*, 32(3):44–51, 2015.

[11] C. Cetina, P. Giner, J. Fons, and V. Pelechano. Autonomic Computing Through Reuse of Variability Models at Runtime: The Case of Smart Homes. *Computer*, 42(10):37–43, 2009.

[12] S. Choi, S. Cha, and C. C. Tappert. A Survey of Binary Similarity and Distance Measures. *Journal of Systemics, Cybernetics and Informatics*, 8(1):43–48, 2010.

[13] K. Constantino, J. A. Pereira, J. Padilha, P. Vasconcelos, and E. Figueiredo. An Empirical Study of Two Software Product Line Tools. In *Proceedings of the International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, 2016.

[14] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wąsowski. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pp. 173–182. ACM, 2012.

[15] C. Desrosiers and G. Karypis. A Comprehensive Survey of Neighborhood-based Recommendation Methods. In *Recommender Systems Handbook*, pp. 107–144. Springer, 2011.

[16] J. A. Galindo, D. Dhungana, R. Rabiser, D. Benavides, G. Botterweck, and P. Grünbacher. Supporting Distributed Product Configuration by Integrating Heterogeneous Variability Modeling Approaches. *Information and Software Technology*, 62:78–100, 2015.

[17] M. L. Griss, J. Favaro, and M. D. Alessandro. Integrating Feature Modeling with the RSEB. In *Proceedings of the International Conference on Software Reuse (ICSR)*, pp. 76–85. IEEE, 1998.

[18] Y. O. Halchenko and M. Hanke. Open is Not Enough. Let's Take the Next Step: An Integrated, Community-Driven Computing Platform for Neuroscience. *Frontiers in Neuroinformatics*, 2012, 2012.

[19] C. Henard, M. Papadakis, M. Harman, and Y. Le Traon. Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pp. 517–528. IEEE, 2015.

[20] R. M. Hierons, M. Li, X. Liu, S. Segura, and W. Zheng. SIP: Optimal Product Selection from Feature Models Using Many-Objective Evolutionary Optimization. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25(2):17, 2016.

[21] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, 1990.

[22] K. C. Kang, J. Lee, and P. Donohoe. Feature-Oriented Product Line Engineering. *IEEE Software*, 19(4):58–65, 2002.

[23] Y. Koren. Collaborative Filtering with Temporal Dynamics. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 447–456. ACM, 2009.

[24] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42:30–37, 2009.

[25] K. Lee, K. C. Kang, and J. Lee. Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In *Software Reuse: Methods, Techniques, and Tools*, pp. 62–77. Springer, 2002.

[26] X. Lian and L. Zhang. Optimized Feature Selection Towards Functional and Non-Functional Requirements in Software Product Lines. In *Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pp. 191–200. IEEE, 2015.

[27] P. Lops, M. de Gemmis, and G. Semeraro. Content-based Recommender Systems: State of the Art and Trends. In *Recommender Systems Handbook*, pp. 73–105. Springer, 2011.

[28] R. Lotufo, S. She, T. Berger, K. Czarnecki, and A. Wąsowski. Evolution of the Linux Kernel Variability Model. In *Software Product Lines: Going Beyond*, pp. 136–150. Springer, 2010.

[29] L. Machado, J. Pereira, L. Garcia, and E. Figueiredo. SPLConfig: Product Configuration in Software Product Line. In *Proceedings of the Brazilian Conference on Software: Theory and Practice (CBSoft)*, pp. 1–8, 2014.

[30] J. Martinez, G. Rossi, T. Ziadi, T. F. D. A. Bissyandé, J. Klein, and Y. Le Traon. Estimating and Predicting Average Likability on Computer-Generated Artwork Variants. In *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*, pp. 1431–1432. ACM, 2015.

[31] J. L. Martinez, T. Ziadi, R. Mazo, T. F. Bissyandé, J. Klein, and Y. Le Traon. Feature Relations Graphs: A Visualisation Paradigm for Feature Constraints in Software Product Lines. In *Proceedings of the IEEE Working Conference on Software Visualization (VISSOFT)*, pp. 50–59. IEEE, 2014.

[32] R. Mazo, C. Dumitrescu, C. Salinesi, and D. Diaz. Recommendation Heuristics for Improving Product Line Configuration Processes. In *Recommendation Systems in Software Engineering*, pp. 511–537. Springer, 2014.

[33] R. Mazo, C. Salinesi, and D. Diaz. VariaMos: a tool for product line driven systems engineering with a constraint based approach. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, pp. 147–154. CEUR-WS.org, 2012.

[34] M. Mendonça, M. Branco, and D. Cowan. S.P.L.O.T.: Software Product Lines Online Tools. In *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, pp. 761–762. ACM, 2009.

[35] G. G. Pascual, R. E. Lopez-Herrejon, M. Pinto, L. Fuentes, and A. Egyed. Applying Multiobjective Evolutionary Algorithms to Dynamic Software Product Lines for Reconfiguring Mobile Applications. *Journal of Systems and Software (JSS)*, 103:392–411, 2015.

[36] J. W. Payne, J. R. Bettman, and E. J. Johnson. *The Aadaptive Decision Maker*. Cambridge University Press, 1993.

[37] J. A. Pereira, K. Constantino, and E. Figueiredo. In *Proceedings of the International Conference on Software Reuse (ICSR)*. Springer.

[38] J. A. Pereira, S. Krieter, J. Meinicke, R. Schröter, G. Saake, and T. Leich. FeatureIDE: Scalable Product Configuration of Variable Systems. In *Software Reuse: Bridging with Social-Awareness*, pp. 397–401. Springer, 2016.

[39] J. A. Pereira, C. Souza, E. Figueiredo, R. Abilio, G. Vale, and H. A. X. Costa. In *Proceedings of the Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*. IEEE.

[40] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[41] G. Shani and A. Gunawardana. Evaluating Recommendation Systems. In *Recommender Systems Handbook*. Springer, 2011.

[42] O. Spinczyk and D. Beuche. Modeling and Building Software Product Lines with Eclipse. In *Companion to the ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pp. 18–19. ACM, 2004.

[43] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable Collaborative Filtering Approaches for Large Recommender Systems. *Journal of Machine Learning Research*, 10:623–656, 2009.

[44] L. Tan, Y. Lin, and L. Liu. Quality Ranking of Features in Software Product Line Engineering. In *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC)*, volume 2, pp. 57–62. IEEE, 2014.

[45] T. H. Tan, Y. Xue, M. Chen, J. Sun, Y. Liu, and J. S. Dong. Optimizing Selection of Competing Features via Feedback-Directed Evolutionary Algorithms. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pp. 246–256. ACM, 2015.

[46] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich. FeatureIDE: An Extensible Framework for Feature-Oriented Software Development. *Science of Computer Programming (SCP)*, 79(0):70–85, 2014.