N-dimensional Tensor Factorization for Self-Configuration of Software Product Lines at Runtime

Juliana Alves Pereira[1], Sandro Schulze[1], Eduardo Figueiredo[2], Gunter Saake[1]
[1]University of Magdeburg, Germany; [2]Federal University of Minas Gerais, Brazil
{juliana.alves-pereira,sandro.schulze,gunter.saake}@ovgu.de;figueiredo@dcc.ufmg.br

ABSTRACT

Dynamic software product lines demand self-adaptation of their behavior to deal with runtime contextual changes in their environment and offer a personalized product to the user. However, taking user preferences and context into account impedes the manual configuration process, and thus, an efficient and automated procedure is required. To automate the configuration process, *context-aware recommendation techniques* have been acknowledged as an effective mean to provide suggestions to a user based on their recognized context. In this work, we propose a collaborative filtering method based on tensor factorization that allows an integration of contextual data by modeling an N-dimensional tensor *User-Feature-Context* instead of the traditional two-dimensional *User-Feature* matrix. In the proposed approach, different types of non-functional properties are considered as additional contextual dimensions. Moreover, we show how to self-configure software product lines by applying our N-dimensional tensor factorization recommendation approach. We evaluate our approach by means of an empirical study using two datasets of configurations derived for medium-sized product lines. Our results reveal significant improvements in the predictive accuracy of the configuration over a state-of-the-art non-contextual matrix factorization approach. Moreover, it can scale up to a 7-dimensional tensor containing hundred of configurations in a couple of milliseconds.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; *Software functional properties*; *Extra-functional properties*; • **Information systems → Recommender systems**; *Collaborative filtering*; *Similarity measures*;

KEYWORDS

Software Product Lines, Runtime Decision-Making, Self-Configuration, Recommender Systems.

ACM Reference Format:
Juliana Alves Pereira[1], Sandro Schulze[1], Eduardo Figueiredo[2], Gunter Saake[1]. 2018. N-dimensional Tensor Factorization for Self-Configuration, of Software Product Lines at Runtime. In *22nd International Systems and Software Product Line Conference (SPLC '18), September 10–14, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3233027.3233039

SPLC '18, September 10–14, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-6464-5/18/09…$15.00
https://doi.org/10.1145/3233027.3233039

1 INTRODUCTION

Dynamic *Software Product Lines* (SPLs) provide configuration options to adjust a software system at runtime to deal with changes in the users' context [16]. To enable such a dynamic configuration, several *semi-automatic* and *automatic* approaches have been proposed in previous work [13]. Nevertheless, the applicability of these approaches is still limited. In particular, for SPLs with a huge exponential configuration space they have shown to be infeasible. On the one hand, *semi-automatic* approaches require many tasks to be carried out manually by decision makers. Consequently, decision makers must know a lot of detailed, technical information about the features and the context. However, decision-makers usually lack such knowledge, which often leads them to invalid configurations and an increase in configuration time. On the other hand, although *automatic* approaches do not require any user intervention, checking the consistency of a large set of configurations can also be non-trivial. Therefore, due to the computational complexity of the task, scalability and performance concerns are an issue when facing runtime environments. To overcome these issues, suboptimal *automatic* approaches have been proposed that perform well for large SPLs. However, these approaches may generate a set of resulting suboptimal configurations. Thus, as a complementary solution, we propose a context-aware recommendation technique to automatically prioritize features and self-configure SPLs at runtime.

Recommendation techniques have become essential to efficiently filter the huge amount of SPL variants and support the configuration of personalized products [34–37]. In recommender systems, user preferences may be inferred from consumption patterns from other users (a technique known as *Collaborative Filtering* (CF)). However, most model-based CF techniques such as *Matrix Factorization* (MF) [20] do not provide a straightforward way of integrating context data (*i.e.*, features' *non-functional properties* (NFPs) [5]) into the model. The use of MF turns the SPL configuration problem into a sparse two-dimensional matrix in which we have no contextual information and very few selected features from previous users that must be used to compute feature predictions [35]. Although this approach works fine in static contexts, in dynamic contexts the system needs to be constantly reconfigured to deal with changes in the environment. Therefore, in this paper, we present an N-dimensional *Tensor Factorization* (TF) [18] approach that dynamically and proactively adapts the feature selection according to the context of the running applications (*e.g.*, availability of resources and current user needs) while avoiding unexpected behavior.

In particular, we are interested in whether and how good such a technique can be to support the SPL self-configuration at runtime.

To this end, we formulate the following three research questions that guide us in evaluating our approach.

- *RQ1.* How effective does an N-dimensional TF recommender system support the SPL self-configuration compared to others state-of-the-art contextual recommender techniques?
- *RQ2.* How accurate is a context-aware TF recommender system compared to a state-of-the-art non-contextual recommender?
- *RQ3.* How long does it take, on average, for a TF recommender system to self-configure a complete valid product?

To answer these questions and prove the applicability of our approach, we present an empirical study on two subject systems: a laptop and a library product lines[1]. Both product lines take various contextual information into account, such as by *whom*, *when*, and *where* the laptop and library are used. To address *RQ1*, we demonstrate the effectiveness of our approach based on interactive recommendation updates against four state-of-the-art context-aware reduction-based approaches [37] and a random configuration of features as baseline. To address *RQ2*, we compare the results from our proposed context-aware TF approach with a state-of-the-art non-contextual MF approach [35]. Finally, since recommender systems are frequently intended to work on very large datasets, the performance of the recommender is essential. Thus, for *RQ3*, we investigate how fast is our proposed technique.

Our results reveal that the quality of our contextual TF-based approach improves against the non-contextual MF-based approach up to 37% in terms of the *F-Measure* metric. We also compare our approach to other state-of-the-art context-aware approaches and show that they present quite similar results regarding the quality of recommendations. Finally, we show that our proposed approach is able to guarantee a fast response time.

Overall, we make the following three contributions:

(1) We adopt a tensor-based recommender system tailored for the SPL configuration scenario in which we explicitly take the user's context into account.
(2) We target a challenge in the field of dynamic SPLs, which is the efficient self-configuration by interactive recommendation updating.
(3) We conduct extensive experiments on two medium-sized product lines to evaluate our proposed approach.

The remaining paper is structured as follows: Section 2 presents the relevant preliminary background. Section 3 gives an overview of related work. Section 4 presents our approach. Subsequently, Section 5 describes the design of the performed experiments and Section 6 discusses the experimental results. In addition, Section 7 further discusses threats to the validity of our results. Finally, Section 8 concludes the paper and outline directions for future work.

2 BACKGROUND

In this section, we provide basic information about *dynamic configuration of SPLs* and *recommender systems*, as they play a pivotal role for our approach, proposed in Section 4.

[1]The product lines and the configuration datasets can be found at http://wwwiti.cs.uni-magdeburg.de/ jualves/PROFilE/.

Dynamic Configuration of SPLs. An SPL is a set of software systems that share a common set of components (called *features*) [38]. A feature is an increment in functionality or a system property relevant to stakeholders [17]. SPL techniques allow to explicitly configure a variable set of features to build personalized products (*a.k.a.* variants). The automatic process of building personalized products from an SPL is known as self-configuration. *Dynamic SPLs* are software systems in which self-configurations occur at runtime.

Extended Feature Models (EFM) describe variability in terms of system's functional and *non-functional properties* (NFP) [5]. Specifically, it defines mandatory and optional features, as well as their NFPs and relationships [17]. As an example, consider the simplified EFM in Figure 1. While mandatory features (*e.g.*, `windows`) are present in all products of the product line, optional features (*e.g.*, `automation`) define specific points of variation, thus, allowing the instantiation of different products. Note that, in our example, some features constitute states (e.g., *open*) to easily exemplify our approach. However, in our case study, we use real system features in their actual sense. We rely on EFM to describe adaptation rules. Adaptation rules explicitly define under which circumstances a reconfiguration should take place. Contexts can be either numeric (*e.g.*, `Temperature`) or categorical (*e.g.*, `Weather`). Furthermore, multiple contexts can be associated to the same feature. In addition, *Cross-Tree Constraints* (CTC) add further feature and contextual interdependencies to the EFM, thus, restricting the selection of non-directly connected optional features and (or) contexts, *e.g.*, `access-rights` $\rightarrow$ `close`. Overall, all relationships in the model define how features can be combined to obtain a valid configuration.

Recommender Systems. Recommender systems deal with challenging issues such as the personalization of products [11]. Their goal is to alleviate the problem of information overload that also occurs when configuring products of an SPL. In this paper, we focus on the most common class of latent, factor-based CF algorithms, *i.e.* matrix and tensor factorization, that gained popularity due to their good accuracy and scalability [18, 19]. Basically, with these algorithms, products are personalized based on previous consumption patterns from past users. For instance, for the dynamic configuration of SPLs, we use data from previously selected features to self-configure a product over changes in the environment.

In MF, a configuration matrix X is factorized into a product of two other matrices [20]. Formally, given a feature model and a set of valid configurations, a configuration matrix X and its factorization is represented as follows:

$$X = \begin{bmatrix} c_{1_1} & c_{1_2} & \cdots & c_{1_m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n_1} & c_{n_2} & \cdots & c_{n_m} \end{bmatrix} = P_{n \times h} \cdot Q_{h \times m}$$

where X is defined as the set of configurations $X = \{\vec{c_1}, ..., \vec{c_n}\}$ (*i.e.*, the previous configurations plus the current partial configuration); and selected features are encoded as 1, deselected as 0, and undefined features as -1. In addition, P is a latent matrix of configurations and Q a latent matrix of features, n is the number of configurations in X, m the number of features, and h is the number of latent dimensions. The matrices P and Q are used then to make

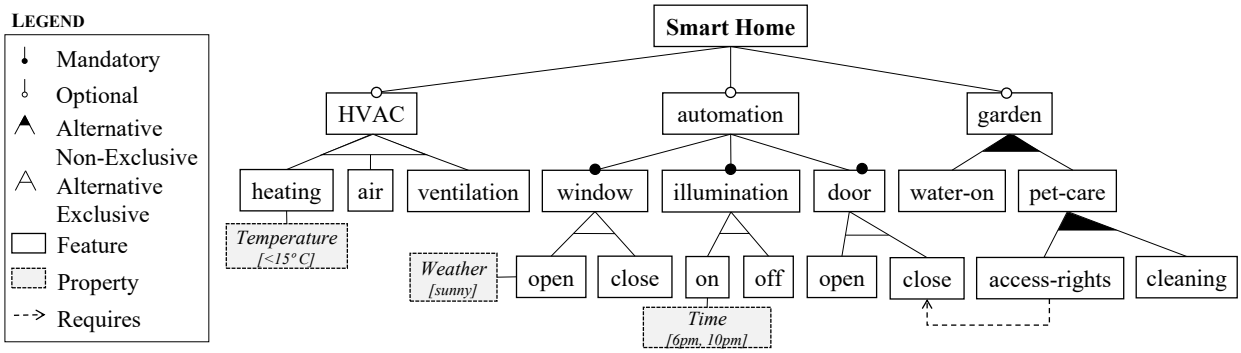


Figure 1: A sample of an EFM for a smart home product line (adapted from Cetina et al. [9]).

predictions for undefined features. In summary, this algorithm factorizes a matrix X into two matrices such that when you multiply them you will get back the original matrix.

Although MF can be used successfully in many static environments, in dynamic environments additional contextual variables come into play. For instance, for our example in Figure 1 the contextual variables: *weather* conditions, *temperature* and *time* play an important role in defining a configuration. Therefore, the system should dynamically self-reconfigure the features at runtime to meet the current user's context. To achieve this goal, the two-dimensional matrix is turned into a multidimensional tensor and feature predictions are done by using TF, an N-dimensional extension of MF.

The use of TF can provide recommendations based on multiple dimensions that go beyond the typical two dimensions (*i.e.*, users and features) used in MF [18]. The order of a tensor is the number of dimensions, *i.e.* the number of relevant contextual information. In the example given in Figure 2, the usual two-dimensional $User \times Feature$ matrix is converted into a three-order tensor $User \times Feature \times Weather$. The intuition behind using TF to support the self-configuration of dynamic SPLs is that there should be some latent features that determine how to configure a product at runtime. Hence, if we can discover the set of latent features from a current configuration through the user-specified context, we should be able to predict the configuration for a specific user (*i.e.*, the characteristics associated with the user, the feature, and the context should match among them). TF takes advantage of most of the benefits of MF, such as fast prediction computations as well as simple and efficient optimization techniques. For more information of this algorithm, we refer to Section 4.2.

3 RELATED WORK

In previous work, we studied six different CF recommendation algorithms to support the SPL configuration process and showed that MF outperformed the others [35, 36]. Similar to our approach, the previous approach computes feature predictions for a current configuration from an active user based on a set of previous configurations from past users. While in previous work we proposed the single use of binary data from previous configurations to generate personalized recommendations [35, 36], in this work we use a generalization of MF (known as TF) to address contextual information, and thus, self-configure a product at runtime. Our aim is to take advantage of the same principles behind MF to deal with N-dimensional contextual information. Moreover, in this paper, we treat missing entries (unknown features interest) as -1, instead of assuming these entries as 0 (deselected features), which would introduce a bias against unobserved features. Then, the approach of regularized TF proposed in this paper follows by optimizing the observed values in the configuration tensor.

We previously proposed a context-aware reduction-based approach [37] to reduce the problem of N-dimensional $User \times Feature \times Context$ recommendations to the traditional two-dimensional $User \times Feature$ recommendation. It uses traditional CF and *average similarity* (AS) approaches to predict features' interest based only on previous configurations data related to the current user-specified non-functional requirements. For example, to recommend a laptop to a gamer, this approach uses only the data from previous users which has high processing laptops. Although reduction approaches lead to more relevant data for calculating unknown features interest, they also lead to fewer data used in this calculation based only on the configurations with the same or similar context. Thus, to improve the quality of the recommendations, we incorporate a multidimensional recommendation technique (*i.e.*, TF) so that it allows the specification of contextual data in the form of a tensor. Therefore, the TF approach we propose here has three main advantages compared to our previous reduction-based approach. First, there is no need for pre-filtering or post-filtering of the data based on context since TF uses all the available data to model users and features. Second, it provides a computational simplicity. Instead of relying on a sequence of techniques, TF relies on a single and less computationally expensive model. Third, it provides capabilities to handle N-dimensional data. The TF approach generalizes well to an arbitrary amount of contextual information. In Section 6.1, we empirically study the tradeoff between a reduction-based approach and a TF approach.

Several authors have proposed exact and approximate optimization approaches to automatically support the statical and dynamic self-configuration of SPLs [1, 13, 22, 29, 30]. *Statical approaches* have focused on techniques to derive product configurations in a single step (*e.g.*, [4, 14, 15, 21, 23, 31, 33, 42, 46, 48]). In static approaches, features are selected based only on product requirements and human desires. Nowadays, this may not be enough due to context changes that without reconfiguration would lead to context violations. Thus, *dynamic approaches* monitors the environment and

when context changes, it dynamically adapts (self-reconfigure) its behaviour to the current situation to keep fulfilling its requirements (*e.g.*, [2, 3, 7, 8, 28, 32, 39, 41]). However, quite often it requires the optimization of multiple and sometimes contradicting objectives. Consequently, there may be a set of resulting valid configurations instead of a single one, *i.e.*, a wide variety of feature combinations may meet the requirements. Therefore, those configurations should be prioritized before the configuration process continues. The manual prioritization of features may result in an overwhelming task. Thus, developing an automatic and effective set of prioritization is a challenging task for developers due to the complexity and dynamicity of the context. Furthermore, as the size and complexity of a software system increases, not only exact techniques but also approximated ones present scalability issues. Consequently, improvements related to effectiveness, scalability and performance are still needed. To achieve this, we propose a TF recommender system to add contextual data and thus exploit additional information to reduce the search effort to self-configure SPLs at runtime. Moreover, our approach that aims at self-configuring a single product can be complementary to these existing techniques.

To the best of our knowledge, there is no previous work on the use of an N-dimensional context-aware TF algorithm in the SPL configuration domain, which is the main contribution of this paper as presented in details in the next section.

4 TENSOR-BASED APPROACH

In this section, we explain the details of how we have adapted the 2-dimensional MF for an N-dimensional TF. First, we introduce how context is specified and modeled and how we relate context to features. Afterwards, we describe details of the proposed TF approach and illustrate it by means of an example.

4.1 Modeling Features & Context

As an important preliminary step, we integrate context information with (de)selected features. As mentioned in Section 2, a feature model is a common way to model features and their dependencies. Moreover, extended feature models (EFM) have been proposed to assign NFPs to features. Our approach is based on the idea to make use of such EFMs in order to integrate contextual information as a specific kind of NFPs (similar to [40] and [25]). This way, we make the context information explicit and relate it to features of the SPL. It makes possible to easily assess the context information and reason about corresponding feature selections in the configuration.

As an example, we introduce three different contexts in the feature model in Figure 1: *Temperature*, *Weather*, and *Time*. Each context is associated with a particular feature, for instance, weather is associated with the `window` feature, thus, may influence whether the window is open or closed. Furthermore, each context may encompass an abitrary number of numeric or categorical values, *e.g.*, the weather can be *sunny* or *rainy*.

For our approach, we consider each context as a separate dimension in our recommendation model (note that also features and the configuration itself constitute separate dimensions), with each dimension encompassing a number of predefined numeric or categorical values. Since this may lead to a high number of dimension, we aim at minimizing our recommendation model by omitting contexts that are not relevant (*i.e.*, which have no influence on the feature (de)selection).

To determine which properties are relevant, we apply the *binary operations* of union and intersection over the set of NFP values. For example, consider a simple case of a single-attribute dimension *Weather* which has only two possible qualitative values *Sunny* and *Rainy*. To determine the relevance of context *Weather*, we first split the dataset of configurations into two sets, one containing the set of *Sunny* configurations and another containing the set of *Rainy* configurations. Then, for each set we perform an union operation over all selected features across all configurations (*i.e.*, the transitive closure of all features selected in at least one of the configurations) followed by an intersection operation between the sets. Consequently, if the distributions of selected features for sunny and rainy days are the same (*i.e.*, $Config(Sunny) \cap Config(Rainy) = 1$), then the dimension *Weather* would not matter for recommendation purposes. Hence, the weather context does not affect the configured product, and thus, the *Weather* dimension can be omitted from the TF model.

Next, we explain details of our TF model and how we make use of it to automate the configuration process.

4.2 Using TF for Self-Configuration of SPLs

Our basic idea for applying an N-dimensional TF approach for context-aware self-configuration is to model the relevant product context by taking the interactions between users, features, and context into account. The proposed tensor-based technique is actually a context optimization method based on the set of previous configurations. For the sake of simplicity, we describe our approach for a single contextual variable C, and therefore the tensor X, containing the previous configurations, is a 3-dimensional tensor[2] $X \in x^{n \times m \times c}$, where n is the number of configurations, m the number of features, and c the number of numeric or categorical values from a relevant contextual variable.

A configuration is given as $X \in \{0, 1, -1\}^{n \times m \times c}$, where the values 0 and 1 indicate that a user deselected and selected a feature, respectively. In addition, -1 indicates a lack of knowledge from the user regarding the feature relevance (*i.e.*, the feature is *undefined)*. The list of recommended features can then be learned from the set of previous configurations and contexts observed in X. To this end, we apply *High Order Singular Value Decomposition* (HOSVD) as shown in Figure 2, to factorize the 3-dimensional tensor into three matrices $User$: $U \in \mathbb{R}^{n \times d_U}$, $Feature$: $F \in \mathbb{R}^{m \times d_F}$, and $Context$: $C \in \mathbb{R}^{c \times d_C}$, and one central core tensor $S \in \mathbb{R}^{d_U \times d_F \times d_C}$. This factorization allows to predict which features are most likely to be selected for a given context. In particular, the prediction function for a single user i, feature j, and context k is:

$$Y_{i,j,k} = S \times_U U_i \times_F F_j \times_C C_k \tag{1}$$

such that Y approximates X, *i.e.* minimizes a loss function $L(Y, X)$ between the real and the predicted values. Additionally, we use a tensor-matrix multiplication operator denoted by $\times_U$ where the index U shows the direction to multiply the matrix, *e.g.* $T = X \times_U U$ is $T_{i,j,k} = \sum_{i=1}^{n} X_{i,j,k} U_{i,j}$.

[2]We only consider third order tensors for explaining the concepts, though the generalization to an N-dimensional tensor is trivial.

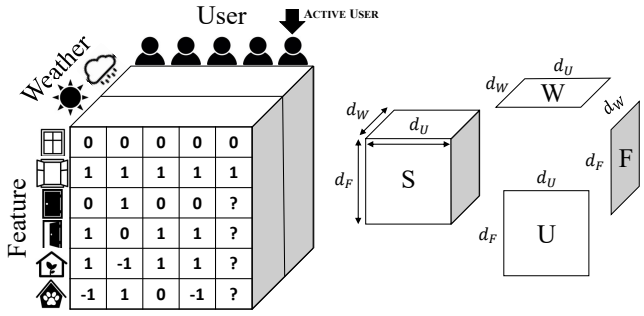


Figure 2: A 3-dimensional example of a tensor factorization model derived from the SPL in Fig 1. Selected features are encoded as 1 and deselected as 0. All other entries (-1 and ?) are unknown features' interests.

However, minimizing the loss function for models with a large number of parameters will lead to overfitting [6]. A common way to prevent overfitting is to regularize the optimization criterion. Therefore, we add to the loss function a regularization term $\Omega(Y)$. Thus, the objective function for the minimization problem is:

$$R[U, F, C, S] = min(L(Y, X) + \Omega(Y)) \tag{2}$$

Loss Function. We define the loss function as:

$$L(Y, X) = \frac{1}{||S||_1} \sum_{i,j,k} D_{i,j,k} * l(Y_{i,j,k}, X_{i,j,k}) \tag{3}$$

where $D \in \{0; 1\}_{n \times m \times c}$ is a binary tensor $D_{i,j,k}$ whenever $X_{i,j,k}$ is observed; and $l(Y_{i,j,k}, X_{i,j,k})$ is computed by the least squares loss function $l(Y_{i,j,k}, X_{i,j,k}) = \frac{1}{2}(Y_{i,j,k} - X_{i,j,k})^2$.

Regularization. We define the regularization function as:

$$\Omega(Y) = \Omega[U, F, C] + \Omega[S] \tag{4}$$

where $\Omega[U, F, C]$ and $\Omega[S]$ are computed by the Frobenius norm [12]:

$$\Omega[U, F, C] = \frac{1}{2}[\lambda||U||^2_{Frob} + \lambda||F||^2_{Frob} + \lambda||C||^2_{Frob}] \tag{5}$$

$$\Omega[S] = \frac{1}{2}[\lambda_S||S||^2_{Frob}] \tag{6}$$

with λ and λ_S being the regularization parameters for the matrices and core tensor respectively, and $||.||^2_{Frob}$ represents the *Frobenius norm* [12]. The Frobenius norm $||U||$ of a matrix U is given by: $||U||_{Frob} = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} u^2_{i,j}}$.

We use the simplest algorithm to solve the optimization problem of Equation 2 which performs *Stochastic Gradient Descent* (SGD) in the factors U_i, F_j, C_k and S for a given tensor $X_{i,j,k}$ (see Algorithm 1). SGD is a standard algorithm for training a wide range of models in machine learning. This algorithm uses a stochastic update approach, that means we need to compute the gradients of the loss function and the objective function with respect to the individual components of the model:

$$\partial_{U_i} l(Y_{i,j,k}, X_{i,j,k}) = \partial_{Y_{i,j,k}} l(Y_{i,j,k}, X_{i,j,k}) S \times_F F_j \times_C C_k$$
$$\partial_{F_j} l(Y_{i,j,k}, X_{i,j,k}) = \partial_{Y_{i,j,k}} l(Y_{i,j,k}, X_{i,j,k}) S \times_U U_i \times_C C_k$$
$$\partial_{C_k} l(Y_{i,j,k}, X_{i,j,k}) = \partial_{Y_{i,j,k}} l(Y_{i,j,k}, X_{i,j,k}) S \times_U U_i \times_F F_j$$
$$\partial_S l(Y_{i,j,k}, X_{i,j,k}) = \partial_{Y_{i,j,k}} l(Y_{i,j,k}, X_{i,j,k}) U_i \times F_j \times C_k$$

Algorithm 1 Tensor Factorization (X)

Initialize U, F, C, and S with small random values.
set $t = t_0$
while (i, j, k) in observations Y **do**
  $\eta \leftarrow \frac{1}{\sqrt{t}}$ and $t \leftarrow t + 1$
  $Y_{i,j,k} = S \times_U U_i \times_F F_j \times_C C_k$
  $U_i = U_i - \eta\lambda U_i - \eta\partial_{U_i} l(Y_{i,j,k}, X_{i,j,k})$
  $F_j = F_j - \eta\lambda F_j - \eta\partial_{F_j} l(Y_{i,j,k}, X_{i,j,k})$
  $C_k = C_k - \eta\lambda C_k - \eta\partial_{C_k} l(Y_{i,j,k}, X_{i,j,k})$
  $S = S - \eta\lambda_S S - \eta\partial_S l(Y_{i,j,k}, X_{i,j,k})$
end while
return U, F, C, S

As an example, consider the three-dimensional cube $User \times Feature \times Weather$ shown in Figure 2 for the smart home product line in Figure 1. Assume that we want to automatically and intelligently self-configure features to an active user. As for the standard two-dimensional case, we start with an initial set of previous configurations. It has the following dimensions:

- *User*: represents all the people for whom features are self-configured in an application.
- *Feature*: represents all the features that can be self-configured in a given application.
- *Weather*: represents the weather forecast when the application is dynamically self-configured, *e.g.* $Weather\{Sunny, Rainy\}$.

Then, we define a function $Y_{i,j,k}$ on the recommendation space $User \times Feature \times Weather$ specifying how much feature $j \in Feature$ is important for a user $i \in User$ in weather $k \in Weather$. For example, the first user configured the features `open window`, `open door`, and `water-on` for a *sunny day*. A second user configured the features `open window`, `close door`, and `pet-care` also for a *sunny day*. Overall, we assume that we have the historical configuration data for the first four users in the multidimensional cube as described in Figure 2. Then, we monitor the context to detect changes that require the system to adapt. Consequently, when adaptation is required, we select and deselect all hard features that are in accordance with the adaptation rules described in the EFM. Next, the remaining features are prioritized by our algorithm and predicted until we self-configure a valid complete configuration.

Suppose that the current context for the active user Elizabeth is sunny, and thus, the relevance of the interest on features `close` and `open door`, `water-on`, and `pet-care` need to be computed. To self-configure these features for Elizabeth, the decisions should be made at runtime based in how often the specific unknown features **"?"** are configured for the specific context *Sunny*. So, we use the Algorithm 1 to compute $Y_{i,j,k} = S \times_U U_i \times_F F_j \times_W W_k$, where $i =$ Elizabeth; $j = \{$`close` and `open door`, `water-on`, and `pet-care`$\}$; and $k =$ *Sunny*. $U \in \mathbb{R}^{5\times6}$, $F \in \mathbb{R}^{6\times2}$, $W \in \mathbb{R}^{2\times5}$, and $S \in \mathbb{R}^{6\times2\times5}$.

Finally, the system should self-configure the best N features that make the configuration valid for the specified context. Given a predictor Y, the list of the top N highest scoring features for a given user u and context c can be calculated by:

$$Top(u, c, N) = max^N_{f \in F}(Y_{u,f,c}) \tag{7}$$

where N denotes the number of features to get a valid and complete configuration. A *valid and complete configuration* is a configuration where each feature is defined (*i.e.*, (de)selected) and it satisfies all functional and non-functional product line constraints. Therefore, as each top feature is selected, decision propagation strategies are applied to automatically validate the configuration where all implied and excluded features are automatically (de)selected and the remaining ones stay undefined, until we have a complete configuration (*i.e.*, all features are defined). Due to the application of decision propagation the user will eventually come to a valid configuration. Furthermore, a complete configuration created using this process will alway be valid.

5 EXPERIMENTAL DESIGN

To evaluate the benefits and drawbacks of our approach, and thus, to answer our research questions, we conduct an empirical study. In this section, we provide details about the datasets used, the experimental protocol, and the state-of-the-art approaches we compare our proposal with. All material (*e.g.*, EFM, datasets, results) of our evaluation is accessible at our complementary webpage[3].

5.1 Target Software Product Lines and Contexts

For our study, we make use of two state-of-the-art product lines [27, 45]. In Table 1, we present five characteristics for each product line, including the number of features, number of cross-tree-constraints (CTC), an upper bound estimation of the number of valid configurations by FeatureIDE statistics [26], number of previous configurations, and number of context dimensions. Using these subject product lines, we evaluate the *effectiveness* of our approach. The effectiveness evaluates how well the proposed approach is capable of understanding the context of the users and self-configure a product at runtime.

Dell Laptop Product Line. The first subject system constitutes a publicly available dataset of laptop configurations [27]. The dataset is a dense tensor containing laptop configurations, encompassing 42 products (*i.e.*, previous configurations) and 68 features. It is constructed as a 6-dimensional tensor representing *User* × *Feature* × *Usage* × *Line* × *Price* × *Performance*. It delivers an application scenario where the four contexts (*i.e.*, *Usage*, *Line*, *Price*, and *Performance*) are described as relations having the following attributes:

- Usage [Game, Play, Program, Study, Work]
- Line [Personal, Professional, Gamer]
- Price [Cheap, Medium, Expensive]
- Performance [Low, Medium, High]

By counting the occurrence of each entry, a tensor of size $42 \times 68 \times 5 \times 3 \times 3 \times 3$ was created.

A Dell laptop is available in all different shapes, prices, and configurations. In order to evaluate our approach, we have to analyze the influence of a varying context on the user configuration. To this end, we defined three target contexts: *gamer*, *programmer*, and *kid* laptops. The contextual information consists of the following specifications:

Gamer laptop: The aim of the final product is to serve as a *portable gaming laptop*. We assume the following (informal) requirements for this context:

- *lightning-fast gaming* installs and loads
- able to play *intensive games* (*e.g.*, Battlefield 4, Watch Dogs, Assassin's Creed IV, etc.) as well as *online games* (via wifi)
- as such games require a lot of power, this laptop needs more *longevity out* (*i.e.*, should not run hot all the time)
- *portable* yet powerful
- high *Memory* (RAM) is also crucial
- games are storage-intensive, *i.e.* this laptop should be able to store 20+ games (5-10 GB each)
- should support *multiple applications* running at once and *streaming* games

Programmer laptop: For this context, the customer needs a laptop with a great combination of performance and power, thus, assuming the following requirements:

- adequate support for all programming language compilers, interpreters, local servers, and code editors
- *Speed* is important (*e.g.*, to program several intensive game applications), thus, a very good processor is required.
- good amount of memory with additional storage to efficiently run local servers, compilers, code editor, and a web browser simultaneously
- battery life is not a priority (as the laptop is supposed to be used in the office)
- as users are supposed to spent lots of time in programming, *comfortable features* are crucial, *i.e.* this laptop should provide comfortable keyboard and a large and high-resolution display (*e.g.*, to reduce/prevent eye strain)

Kids laptop: This product should serve as an entertainment laptop for kids, assuming the following requirements:

- good wireless connection (for playing games and watching videos online)
- lightweight and highly portable (as it is for kids)
- longer battery life (assuming that kids use it away from standard power source)

Library SPL. For our second subject system, we derived the dataset from a library SPL [45]. The scope and purpose of this SPL is to have a library system equipped with all operations and facilities needed to provide services to its users. In general, a library has its own management system, operating environment, payment methods, network and security system. In addition, the library offers several services to its users, both, offline and online.

The library SPL consists of 74 previous configurations and 135 features. From the configurations, we constructed a 7-dimensional, dense tensor constituting *User* × *Feature* × *Resource Access* × *Device* × *Internet Connection* × *Environment* × *Age Range*. By counting the occurrence of each entry, a tensor of size $74 \times 135 \times 2 \times 4 \times 2 \times 6 \times 4$ was created. It delivers an application scenario where the five contexts (*i.e.*, *Resource Access*, *Device*, *Internet Connection*, *Environment*, and *Age Range*) are described as relations having the following attributes:

- Resource Access [Digital, Phisical]
- Device [Mobile, Computer, Tablet, None]

[3] http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/

Dataset	Features	CTC	Valid Configurations	Previous Configurations	Context Dimensions
Dell laptop [27]	68	7	>154,832	42	4
Library [45]	135	none	>273,534	74	5

Table 1: Main properties of the datasets.

- Internet Connection [Yes, No]
- Environment [City, Company, University, Farm, Prison, Any]
- Age Range [<10, 10-20, 20-60, >60]

To obtain a dataset of configurations, we conducted an *a priori experiment* with 37 Software Engineering and Database Master and PhD students from two universities (University of Magdeburg in Germany and Federal University of Minas Gerais in Brazil). The students were asked to solve a given configuration task[4], consisting of three subtasks: (1) analyze the library feature model, (2) configure two products based on the library feature model, and (3) briefly describe the requirements specification for the created configuration.

For our evaluation, we use the whole set of generated configurations based on four contextual target environments derived from our apriori experiment: *digital scientific library*, *digital company e-book library*, *software engineering research group library*, and *farm library*. The contextual information consists of the following specifications:

Digital Scientific Library. It constitutes a digital scientific library of research articles and books.

- access is purely online (via web)
- allows to search by classification and keywords
- registration via email.
- enables notification about newly published titles
- no loan and renewal, as items are downloaded in PDF format
- usage is free of charge for university's employees and students (from university network); for other users an annual fee applies
- password authentication and digital certificates for security
- reasonable security for payment data and transactions.

Digital Company E-book Library. It constitutes a digital library for a small company that allows to manage e-books.

- entirely free of charge with e-books used via mobile interface
- all interactions (*e.g.*, registration, searching the library, etc.) must be possible via mobile devices
- no overdue policies or reservations (due to e-books only)
- email notification about new resources
- no specific network security measures

Software Engineering Research Group Library. It serves as a small information system to track lending of books, periodicals, and monographs between members of a research group at a university.

- access for members by a specific website
- notifications on new arrivals
- web interface for log in, search & reserve resources, view account, etc.
- policy for renewing reservations ⇒ not possible if already reserved by someone else
- catalog should state fees for loss and damage of items
- fees can only be paid cash

Farm Library. It constitutes a traditional (public) library system.

- should be able to operate offline
- physical items, fees for damage and loss as well as for returning items late
- reminders (for returning items) as notification via email and mobile phone message
- comprehensive user profile (including borrowing history) to be used for reading campaigns

5.2 Evaluation Protocol

We assess the effectiveness of our approach by conducting a leave-one-out cross-validation study. Leave-one-out cross-validation involves using one configuration as the validation set and the remaining configurations ($n-1$ where n is the total number of configurations) as the training set. For each validation set (*i.e.*, target configurations defined in Section 5.1), we select $\alpha * 100\%$ features from the dataset that are subject to recommendation. For example, if $\alpha = 0.8$, we randomly give 80% of the configured features to the recommender system, while we hide the remaining features (20%). We considered different percentages, that is, $\alpha = \{0; 0.2; 0.4; 0.6; 0.8\}$. Subsequently, the system automatically conduct a self-configuration of a product based on the target context defined in Section 5.1 by using the tensor-based recommendation algorithm, as well the recommendation methods introduced in Section 5.3. This is repeated 1,000 times for each dataset and target context to ensure different combinations for each validation set. Therefore, we carried out $1,000 \times 5 \times 3$ runs on the Dell laptops dataset and $1,000 \times 5 \times 4$ runs on the Library dataset. All experiments were conducted for each of our seven methods separately. Consequently, they give us 105,000 experiments on the Dell laptops dataset and 140,000 experiments on the Library dataset. Note that a part of the configurations from the datasets have been held out for the purpose of parameter optimization. Every parameter in this phase was validated using a 10-fold cross validation over the use of other four target contexts (*i.e.*, office and study laptops, and city and prison libraries). Those additional experiments for parameter optimization are not counted here.

We used the *Precision* and *Recall* metrics to make a comparison between the set *Rec* of self-configured features by the algorithm and a set of relevant features *Rel* known from the oracle containing the desired configuration. Precision is calculated as follows: $Precision = \frac{|Rec \cap Rel|}{w}$. where w represents the number of self-configured features by the algorithm to have a valid complete configuration. Analogously, recall is calculated using the formula: $Recall = \frac{|Rec \cap Rel|}{|Rel|}$. *Precision* states how many of the self-configured features were relevant. *Recall* measures what percentage of relevant features has been self-configured. Since our goal is to maximize both precision and recall, in our evaluation we use a measure that combines both *i.e.* the F-Measure, defined as follows:

$$F\text{-}Measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \quad (8)$$

[4]Further details about the configuration task carried out by the participants are available at http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFilE/.

All the experiments were performed on a 2 sockets Intel Xeon E5620 @2.40GHz with 4 cores per socket and 20GB of RAM. All reported results constitute the F-Measure average of all runs.

5.3 Comparison Approaches

We evaluate the effectiveness of our approach by comparing it with a random baseline recommender that simulates the performance of an uninformed user without any support from a recommender system. This method recommends randomly chosen non-selected features, indicating the minimal performance level every algorithm should reach (*i.e.* it serves only as a comparison baseline). Additionally, we also compare our approach to a context-aware reduction-based approach, which is based on traditional CF methods [37]. This approach computes recommendations using only the configurations made in the same context as the target one according to the contextual information introduced in Section 5.1. For the Dell laptops product line, we use the context *Usage* as pre-filtering data, *Performance* as prediction modeling data, and *Price* as post-filtering data. For the Library SPL, we use the contexts *Resource Access*, *Environment* and *Age Range* as pre-filtering data; and *Device* and *Internet Conection* as post-filtering data. For more information on those stages, we refer to our previous work [37].

Finally, to demonstrate the improved effectiveness of a context-aware approach, we compare our TF approach with our previously proposed two-dimensional, non-contextual MF approach [35]. This approach uses the BRISMF algorithm by Takács et al. [44] to transform a two-dimensional configuration matrix into a latent space by incremental minimization of an error function. In addition, to make a fair comparison, we encoded unknown feature interests from past configurations as -1, instead of using zero entries as in [35, 36].

6 EXPERIMENTAL RESULTS

In this section, we investigate how efficient and effective our proposed N-dimensional context-aware TF algorithm. To this end, we answer the research questions stated in Section 1 based on the results of our evaluation. First, in Section 6.1, we evaluate the efficiency of our approach by comparing it with a random selection of features and four reduction-based approaches. Second, in Section 6.2, we assess the impact of using contextual information by comparing the proposed context-aware TF approach to the non-context-aware MF approach [35]. Finally, in Section 6.3, we demonstrate how fast our approach can self-configure an SPL.

6.1 RQ1: Approach Effectiveness

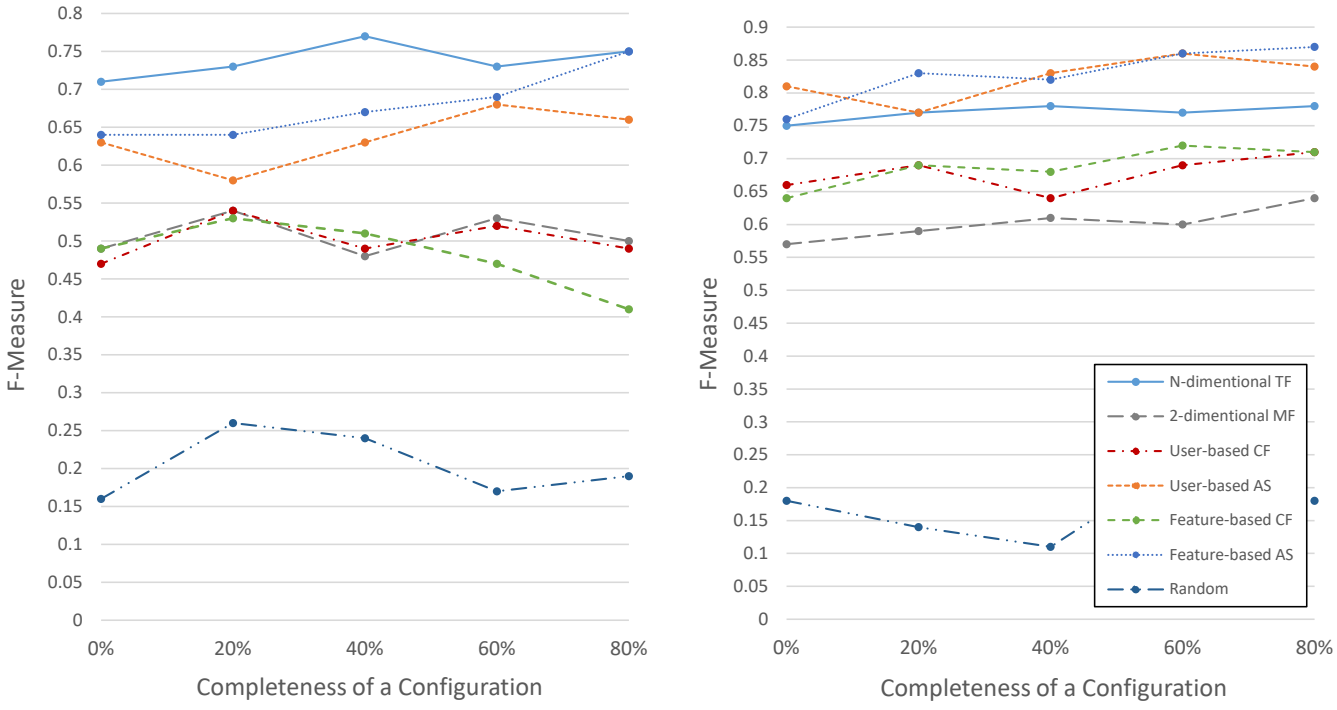
Figures 3a and 3b present the *F-Measure* achieved by the six methods: *TF*, four *reduction-based algorithms* (user-based CF and AS, and feature-based CF and AS), and a *random* baseline method. F-Measure combines recall and precision, *i.e.* higher values are better. On the horizontal axis of the figure, we present the completeness of a configuration, *i.e.* the percentage of (de)selected features that were given to the method as training data, where only the remaining part of the configuration needed to be self-configured. In practice, it simulates the hard requirements imposed by the context. The reported results for each dataset are averaged over all F-Measures computed for the set of target contexts introduced in Section 5.1, which we use as validation set. For example, for the Dell laptops dataset, we average all F-Measure values from an exhaustive cross-validation in each of the three target contexts (cf. Section5.1). Moreover, note that the F-Measure value cannot be directly compared to the previous computed F-Measure values in [35–37], since here we do not compute precision based on the list of the top-10 recommended features. Instead, we run the algorithm each time to self-configure each top-1 feature at once until we have a valid and complete configuration.

In summary, our data reveal that, on average, the TF method outperforms the reduction-based method on the Dell laptop dataset, while it slightly underperforms on the library dataset for feature-based and user-based AS algorithms. Experimentally, when comparing the prediction quality of the reduction-based and the tensor-based approaches, it is clear that the reduction-based approaches work better on larger datasets. On the Library dataset with 74 configurations, the results are even better than on the Dell laptop dataset with 42 configurations. This is because reduction-based approaches [37] provide recommendations on a particular segment and build a local prediction model for this segment. Consequently, these recommendations are based on a small number of configurations limited to the same or similar context. This tradeoff between having fewer yet more relevant data for calculating predictions explains why the reduction-based method underperforms the tensor-based method on some segments and outperforms on others.

Moreover, we observe that all algorithms outperform the baseline random recommender. However, for some algorithms, we observe a decrease of the F-Measure as the configuration becomes more complete, *i.e.* when we use more than 80% of selected features as training data. This is because the precision considers w as the number of self-configured features to automatically generate a valid and complete configuration. However, for most of the target products, the data validation does not represent a maximum valid configuration (*e.g.*, the relevance of the feature `Additional_Warranty` was not specified by gamer and programmer Dell laptops). Nevertheless, it does not mean that the self-selected features are not relevant since they are represented as unknown relevance (-1) in the validation set. Overall, the N-dimensional TF and the AS reduction-based algorithms achieved the best performance over all other algorithms at nearly all stages of the configuration process. The main reason is the benefit of having more historical data for calculating unknown features' relevance. However, reduction-based approaches are a rather computationally expensive operation as for each combination of contextual requirements, a CF model needs to be trained and tested [37]. Therefore, in the given scenario, the use of a TF algorithm seems more appropriate.

6.2 RQ2: Contextual vs. Non-Contextual

Since it is still not clear if context matters in our scenario, in this section we conducted some experiments to assess the relevance of contextual information. We compare our context-aware multidimensional TF approach with the non-context-aware two-dimensional MF approach introduced in previous work [35] and we analyze the tradeoffs between them. We choose this algorithm instead of others presented in [35] because it is usually more effective since it allows us to discover the latent configurations underlying the interactions between users and features. We conduct a comparison analysis between the algorithms from both approaches on our target datasets (see Section 5.1). It is worth mentioning that we do not use the



a) Dell Laptops dataset.

b) Library dataset.

Figure 3: F-Measure achieved by seven different recommendation methods on the Dell Laptop and the Library datasets (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.

datasets of configurations from [35], since these datasets do not work with NFPs. In Figures 3a and 3b, we present the *F-Measure* results achieved by both approaches for each considered dataset.

Our data reveal that the performance of the tensor-based approach outperforms the matrix-based approach on both datasets disregarding the completeness of the configuration, which indicates that context matters. The main reason is the benefit of having contextual data for calculating feature predictions, instead of only having binary information. However, the extent to which the contextual approach can outperform the non-contextual approach may depend on many different factors, such as the application domain and the specifics of the contextual available data. Consequently, pure matrix-based approaches work best on static environments where context does not matter.

Overall, our tensor-based approach is more efficient than a pure matrix-based approach via singular value decomposition, whereas on the library SPL dataset the F-Measure values clearly outperform the values achieved in the Dell laptop dataset. This may be because the library SPL consists of a larger dataset with a higher number of context dimensions. Therefore, we believe that for most of the applications with context information available, context-aware recommendation techniques are supposed to outperform non-contextual recommendation techniques. Still, conducting experiments with other SPLs to prove this claim remains part of our future work.

6.3 RQ3: Approach Performance

In a last experiment, we evaluate how fast our approach is by measuring its response time for self-configuring a valid and complete configuration. We record the average response time during the effectiveness experiment presented in Section 6.1 for each target context being analyzed.

Both datasets lead to a reasonable runtime (up to 112.5ms). The prediction runtime of our tensor-based approach is independent of the size of the dataset (*i.e.*, number of previous configurations) and is dominated by the factorization dimensions. For the larger factorization of the Library SPL (7 dimensions), the runtime of our approach is worse than that for the smaller factorization of the Dell laptop (6 dimensions), resulting into 112.5ms and 51.6ms, respectively. The runtime in the Dell laptop dataset for 1-4 context dimensions are 15.6ms, 27.7ms, 39.5ms, and 51.6ms respectively; and in the library dataset for 1-5 context dimensions are 30.2ms, 47.5ms, 71.3ms, 88.8ms, and 112.5ms respectively. Hence, we conclude that our approach scales linearly to the dimensionalities d_U, d_F, and d_C of the factors $User \times Feature \times Context$.

While we found it to be very sensitive to the number of contextual dimensions (*i.e.*, the more context dimensions exist, the more time is required for training), we observed that by increasing the completeness of configurations, we obtain better results in a faster time. To improve the results, on larger datasets with large context dimensions the training phase may be done offline. Moreover, we

aim at optimizing Algorithm 1, as it is trivial to parallelize the algorithm's execution by performing several updates independently, because it accesses only one row of U, F, and C at a time. Therefore, we may use a low level language such as C++; and use parallel concurrency to exploit all processor's cores and add the results of each different thread. Also, the tradeoff between quality and speed can be minimized by controlling the number of context dimensions. That means depending on the application we can statistically measure each contextual relevance and thus reduce the number of context dimensions. There are several approaches, *e.g.* from machine learning or data mining, to determine the relevance of a given type of contextual information. This, however, goes beyond the scope of this paper and remains as an important next step, which is part of our future work. Thus, further efforts can be taken in future to improve the current implementation regarding performance.

We conclude that, although the training phase posed additional challenges for our algorithms, the tensor-based approach works efficiently (within milliseconds) for both datasets. In addition, the number of NFPs for these product lines reaches a threshold of five which is an usual number of context dimensions used in real-world applications (*cf.* Mairiza et al.[24] and Sommerville et al. [43]).

7 THREATS TO VALIDITY

Even though our experiments provide evidence that our approach is feasible, a key issue when performing these experiments are assumptions that may affect the validity of the results. Therefore, in this section, we describe some concerns related to the validity of our experiments. We have followed the guidelines proposed by Wohlin et al. [47] in order to identify and discuss how the main validity threats to our approach were addressed. We discuss the four groups of common validity threats: *internal validity*, *external validity*, *construct validity*, and *conclusion validity* [47].

An *internal validity* threat concerns the specification of NFPs. In this work, we have investigated and created suitable NFPs based on experts opinion and general characteristics of NFPs found in the literature [10, 24] (*i.e.*, our focus is in the recommender system). In our approach, we assume that a particular NFP has been already measured or specified and we use it in our N-dimensional model. We are aware that measuring NFPs might influence the scalability and time performance of our approach. This, however, goes beyond the scope of this paper. In addition, our approach does not consider aggregation measures of NFPs. This is a complex problem, and its general solution also lies outside of the scope of this paper.

We have identified two *external validity* threats. The first validity threat is the characteristics of the product lines and datasets used for evaluating our approach. We have addressed this validity threat by reporting the characteristics of the product lines and datasets, such as the product line size, the variability degree, the number of cross-tree constraints and previous configurations (see Table 1). Note that in our experiments we used different product lines from [35–37]. These product lines were more suitable for our experiments due to their easily understandable domain and an available real medium-sized dataset of configurations. However, we are aware that the use of other datasets of configurations with different characteristics could have impacted our results. Still, conducting experiments with other product lines remains part of our future work.

The second validity threat is the selection of the comparison approaches. To address this threat, we have included all known feature-based recommendation algorithms. However, since multi-objective optimization algorithms may also support the process of self-configuration of SPLs, we plan to extend our experiments with optimization algorithms as part of our future work.

Construct validity threats have been addressed by using the same evaluation process for all algorithms and using a standard evaluation metric. To simulate the practical configuration task while measuring the effectiveness and performance of our approach, the set of self-configured features were chosen randomly from the whole set of relevant features. Moreover, to ensure significance of the results, the reported F-Measure values are averaged over 105,000 and 140,000 runs, respectively. However, it is important to mention that our approach is designed to work most effectively when a large dataset of previous configurations and context are available. This is a requirement for our recommender system that allows us to build a set of more reliable configurations.

Finally, to address *conclusion validity* threats every effort has been made to eliminate machine dependencies and minimize the influence of the environment (*i.e.*, platform, coding, compiling, caching, etc.). Moreover, all tests were performed using the same machine and the same compiler. Furthermore, all the data used to run these experiments is publicly available for replication.

8 CONCLUSION AND FUTURE WORK

In this paper, we propose an efficient multidimensional context-aware recommender approach to handle the self-configuration of SPLs at runtime. Our approach adopts a TF recommender algorithm to predict unknown features interest on undefined features from a set of previous configurations according to product requirements. To assess the effectiveness of our approach, we conducted a series of experiments on state-of-the-art product lines. In our experiments, we empirically demonstrated that the proposed context-aware TF approach significantly outperforms the non-contextual MF approach [35, 36] in terms of quality of recommendations. In addition, it has a good performance already at the initial configuration stage. Also, we have shown that although the proposed approach presents a similar behavior to reduction-based recommendation approaches [37], tensor-based approaches seem to be more appropriate in dynamic contexts.

As future work, we aim to explore how the number of previous configurations affect the efficiency of our approach (*i.e.*, at which point they start to be useful). We also aim at exploring various types of statistical tests to identify which of the contextual NFPs are truly significant in the sense that they indeed affect the recommendations. Moreover, we plan to compare our approach with state-of-the-art optimization approaches. Finally, we aim at extending a state-of-the-art SPL configuration tool with our approach.

ACKNOWLEDGMENT

We gratefully acknowledge the financial support of the *Brazilian National Council for Scientific and Technological Development* (CNPq) grant 202368/2014-9 and the project EXPLANT of the *German Research Foundation* (DFG) grant SA 465/49-1.

REFERENCES

[1] Uzma Afzal, Tariq Mahmood, and Zubair Shaikh. 2016. Intelligent software product line configurations: A literature review. *Computer Standards & Interfaces* 48 (2016), 30–48.

[2] Germán H Alférez, Vicente Pelechano, Raúl Mazo, Camille Salinesi, and Daniel Diaz. 2014. Dynamic adaptation of service compositions with variability models. *Journal of Systems and Software* 91 (2014), 24–47.

[3] André Almeida, Everton Cavalcante, Thais Batista, Nelio Cacho, and Frederico Lopes. 2014. A component-based adaptation approach for multi-cloud applications. In *Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 49–54.

[4] Ebrahim Bagheri, Tommaso Di Noia, Dragan Gasevic, and Azzurra Ragone. 2012. Formalizing interactive staged feature model configuration. *Journal of Software: Evolution and Process* 24, 4 (2012), 375–400.

[5] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: a literature review. *Information Systems* 35, 6 (2010), 615–708.

[6] Christopher M. Bishop. 2007. *Pattern recognition and Machine Learning (Information Science and Statistics)* (1 ed.). Springer.

[7] Johannes Bürdek, Sascha Lity, Malte Lochau, Markus Berens, Ursula Goltz, and Andy Schürr. 2014. Staged configuration of dynamic software product lines with complex binding time constraints. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*. ACM, 16.

[8] Carlos Cetina, Joan Fons, and Vicente Pelechano. 2008. Applying software product lines to build autonomic pervasive systems. In *Proceedings of the International Software Product Line Conference (SPLC)*. 117–126.

[9] Carlos Cetina, Pau Giner, Joan Fons, and Vicente Pelechano. 2009. Autonomic computing through reuse of variability models at runtime: The case of smart homes. *Computer* 42, 10 (2009), 37–43.

[10] International Organization For Standardization/International Electrotechnical Commission et al. 2001. Software engineering–Product quality–Part 1: Quality model. *ISO/IEC* 9126 (2001), 2001.

[11] Andreas Falkner, Alexander Felfernig, and Albert Haag. 2011. Recommendation technologies for configurable products. *Ai Magazine* 32, 3 (2011), 99–108.

[12] Gene H Golub and Charles F Van Loan. 2012. *Matrix computations*. Vol. 3. JHU Press.

[13] Gabriela Guedes, Carla Silva, Monique Soares, and Jaelson Castro. 2015. Variability management in dynamic software product lines: A systematic mapping. In *Components, Architectures and Reuse Software (SBCARS), 2015 IX Brazilian Symposium on*. IEEE, 90–99.

[14] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. 2015. Combining multi-objective search and constraint solving for configuring large software product lines. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 517–528.

[15] Robert M Hierons, Miqing Li, XiaoHui Liu, Sergio Segura, and Wei Zheng. 2016. SIP: optimal product selection from feature models using many-objective evolutionary optimization. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 25, 2 (2016), 17.

[16] Mike Hinchey, Sooyong Park, and Klaus Schmid. 2012. Building Dynamic Software Product Lines. *IEEE Computer* 45, 10 (October 2012), 22–26. https://doi.org/10.1109/MC.2012.332

[17] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report CMU/SEI-90-TR-21. Software Engineering Institute.

[18] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. 2010. Multiverse Recommendation: N-Dimensional Tensor Factorization for Context-Aware Collaborative Filtering. In *ACM Recommender Systems Conference (ACM RecSys)*. ACM, 79–86.

[19] Yehuda Koren and Robert Bell. 2015. Advances in collaborative filtering. In *Recommender systems handbook*. Springer, 77–118.

[20] Y. Koren, R. Bell, and C. Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42 (Aug. 2009), 30–37.

[21] Xiaoli Lian and Li Zhang. 2015. 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)*. IEEE, 191–200.

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

[23] Lucas Machado, Juliana Pereira, Lucas Garcia, and Eduardo Figueiredo. 2014. SPLConfig: product configuration in software product line. In *Brazilian Congress on Software Engineering (CBSoft)*. 1–8.

[24] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. 2010. An investigation into the notion of non-functional requirements. In *ACM Symposium on Applied Computing (SAC)*. ACM, 311–317.

[25] Jacopo Mauro, Michael Nieke, Christoph Seidl, and Ingrid Chieh Yu. 2016. Context Aware Reconfiguration in Software Product Lines. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*. ACM, 41–48.

[26] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE*. Springer.

[27] Marcílio Mendonça, Moises Branco, and Donald Cowan. 2009. S.P.L.O.T.: software product lines online tools. In *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. ACM, 761–762.

[28] Rabeb Mizouni, Mohammad Abu Matar, Zaid Al Mahmoud, Salwa Alzahmi, and Aziz Salah. 2014. A framework for context-aware self-adaptive mobile applications SPL. *Expert Systems with Applications* 41, 16 (2014), 7549–7564.

[29] Lina Ochoa, Oscar Gonzalez-Rojasa, Juliana Alves Pereira, Harold Castro, and Gunter Saake. 2018. A Systematic Literature Review on the Semi-Automatic Configuration of Extended Product Lines. *Journal of Systems and Software (JSS)* 144 (2018), 511–532.

[30] Lina Ochoa, Juliana Alves Pereira, Oscar González-Rojas, Harold Castro, and Gunter Saake. 2017. A survey on scalability and performance concerns in extended product lines configuration. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*. ACM, 5–12.

[31] Gustavo G Pascual, Roberto E Lopez-Herrejon, Mónica Pinto, Lidia Fuentes, and Alexander Egyed. 2015. Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications. *Journal of Systems and Software (JSS)* 103 (2015), 392–411.

[32] Gustavo G Pascual, Roberto E Lopez-Herrejon, Mónica Pinto, Lidia Fuentes, and Alexander Egyed. 2015. Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications. *Journal of Systems and Software* 103 (2015), 392–411.

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

[34] Juliana Alves Pereira, Jabier Martinez, Hari Kumar Gurudu, Sebastian Krieter, and Gunter Saake. 2018. Visual Guidance for Product Line Configuration Using Recommendations and Non-Functional Properties. In *ACM Symposium on Applied Computing (SAC)*. ACM, 2058–2065.

[35] Juliana Alves Pereira, Pawel Matuszyk, Sebastian Krieter, Myra Spiliopoulou, and Gunter Saake. 2016. A feature-based personalized recommender system for product-line configuration. In *Proceedings of the International Conference on Generative Programming and Component Engineering (GPCE)*. ACM, 120–131.

[36] Juliana Alves Pereira, Pawel Matuszyk, Sebastian Krieter, Myra Spiliopoulou, and Gunter Saake. 2018. Personalized Recommender Systems for Product-line Configuration Processes. *Computer Languages, Systems & Structures (COMLAN)* (2018).

[37] Juliana Alves Pereira, Sandro Schulze, Sebastian Krieter, Márcio Ribeiro, and Gunter Saake. 2018. A Context-Aware Recommender System for Extended Software Product Line Configurations. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*. ACM, 1–8.

[38] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software product line engineering: foundations, principles and techniques*. Springer, Berlin Heidelberg.

[39] Carlos Ruiz, Hector A Duran-Limon, and Nikos Parlavantzas. 2016. Towards a software product line-based approach to adapt IaaS cloud configurations. In *International Conference on Utility and Cloud Computing*. ACM, 398–403.

[40] Karsten Saller, Malte Lochau, and Ingo Reimund. 2013. Context-aware DSPLs: model-based runtime adaptation for resource-constrained systems. In *Proceedings of the International Software Product Line Conference (SPLC)*. ACM, 106–113.

[41] Amir Molzam Sharifloo, Andreas Metzger, Clément Quinton, Luciano Baresi, and Klaus Pohl. 2016. Learning and evolution in dynamic software product lines. IEEE, 158–164.

[42] Kai Shi. 2017. Combining Evolutionary Algorithms with Constraint Solving for Configuration Optimization. IEEE, 665–669.

[43] Ian Sommerville and Pete Sawyer. 1997. Viewpoints: principles, problems and a practical approach to requirements engineering. *Annals of software engineering* 3, 1 (1997), 101–130.

[44] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. 2009. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research* 10 (June 2009), 623–656.

[45] Lei Tan, Yuqing Lin, Huilin Ye, and Guoheng Zhang. 2013. Improving product configuration in software product line engineering. In *Australasian Computer Science Conference*. Australian Computer Society, Inc., 125–133.

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

[47] C Wohlin, P Runeson, M Host, MC Ohlsson, B Regnell, and A Wesslen. 2000. Experimentation in software engineering: an introduction.

[48] Yi Xiang, Yuren Zhou, Zibin Zheng, and Miqing Li. 2018. Configuring Software Product Lines by Combining Many-Objective Optimization and SAT Solvers. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 26, 4 (2018), 0–14.