Association Rule Learning Based Approach to Automatic Generation of Feature Model Configurations

Olfa Ferchichi[1][a], Raoudha Beltaifa[2,4][b] and Lamia Labed[3][c]

[1]*Laboratoire Riadhi Ecole Nationale des Sciences de l'Informatiques, Tunisia*
[2]*Sesame University, Tunisia*
[3]*Carthage University, Tunisia*
[4]*Tunis University, Tunisia*

Keywords: Software Product Lines, Variability, Feature Models, Configuration, Advanced Apriori Algorithms, Association Rule Learning.

Abstract: The evolution the Software Product Line processes requires targeted support to address emerging customer functionals and non functionals requirements, evolving technology platforms, and new business strategies. Enhancing the features of core assets is a particularly promising avenue in Software Product Line evolution.The manual configuration of SPLs is already highly complex and error-prone. The key challenge of using feature models is to derive a product configuration that satisfies all business and customer requirements. However, proposing a unsupervised learning-based solution to facilitate this evolution is a growing challenge. To address this challenge, in this paper we use association rules learning to support business during product configuration in SPL. Based on extended feature models, advanced apriori algorithm automatically finds an optimal product configuration that maximizes the customer satisfaction. Our proposal is applied on a practical case involving the feature model of a Mobile Phone Product Line.

1 INTRODUCTION

Software Product Line (SPL), a widely recognized approach in software engineering (Clements, 2001) (Ferchichi et al., 2020) (Pohl et al., 2005) involves generating a range of interconnected products by blending reusable core elements with custom assets tailored to each specific product. Demonstrated as an efficient strategy for leveraging architecture-level reuse, software product lines have established their effectiveness in the field. The SPL engineering framework consists of two main processes: domain engineering and application engineering. The domain engineering process begins with domain analysis to identify both common and variable features. These features serve as the foundation for designing and implementing the domain. Through domain activities, software assets known as core assets are created. Core assets are reusable components utilized in developing Product Line (PL) products during the application engineering process. These assets can encompass various elements such as feature models, architectures, components, or any other reusable outcomes of domain activities.

One of the key aspects of product lines is variability. Variability (Metzger and Pohl, 2014) (Bashroush et al., 2017) (El-Sharkawy et al., 2019) is defined as the ability of a software system or artifact to be efficiently extended, changed, customized, or configured (White et al., 2014) for use in a particular context.

In this paper, we propose an approach for automatically deriving PL configrations based on association rules learning.

The remainder of this paper is divided into six sections. in section 2 we present the fundamental concepts of SPL. In Section 3, we present the problem description. In Section 4, we present related work. Section 5 describes the proposed advanced apriori algorithms approach for configuration SPL. The experiment and the interpretation of the result are presented in section 6. We give a conclusion in Section 7.

[a] https://orcid.org/0000-0003-2520-7588
[b] https://orcid.org/0000-0003-4096-5010
[c] https://orcid.org/0000-0001-7842-0185

Ferchichi, O., Beltaifa, R. and Labed, L.
Association Rule Learning Based Approach to Automatic Generation of Feature Model Configurations.
DOI: 10.5220/0012742000003687
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2024)*, pages 262-272
ISBN: 978-989-758-696-5; ISSN: 2184-4895
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

2 BACKGROUND

2.1 Feature Model

Variability is expressed in a software variability model, which such as Feature Model (FM). A FM (Lee et al., 2002) is a tree or a directed acyclic graph of features. A FM is organized hierarchically. A mandatory feature has to be selected if its parent feature is mandatory or if its parent feature is optional and has been selected. Mandatory features define commonalities. Optional, alternative, and 'or' features define variability in feature models. As a result, a feature model is a compact representation of all mandatory and optional features of a software product line. Each valid combination of features represents a potential product line application.

In Figure 1, we present a Feature-Oriented Domain Analysis (FODA) model associated with the 'Mobile Phone' SPL. In this model, we identify the following elements: the root feature 'Mobile Phone,' optional features such as 'stores' and 'Bluetooth,' mandatory features like 'keyboard' and 'camera'. The group_or relationship between 'Connectivity' and its children 'Mobile-Network', 'Bluetooth' and 'NFC'. The alternative relationship with cardinality $< 1..1 >$ is represented between the feature 'Sensors' and its children 'Indicative' and 'Capactive'. The interval noted [1..2] which determine the possible number of instances of a feature 'Sim'. Extensions of FMs can be grouped into three categories: 1) basic feature models (offering mandatory, alternative and 'or' features, as well as 'requires' and 'excludes' cross-tree constraints), 2) cardinality-based feature models (Sreekumar, 2023) and 3) extended feature models (adding arbitrary feature attributes; e.g., to express variation in non_functionals requirements such as quality (Ferchichi et al., 2021).

2.2 Non_Functional Requirements

Non_Functional requirements (NFR) are classified as quantitative when they involve measurable information (e.g., cost) and qualitative when the information lacks measurability (e.g., customer preference level) as discussed by (Gérard et al., 2007). FMs with Cardinalities address the challenges of SPL by modeling the variability in terms of choices in Features. The variability in the product family is represented by Feature cardinality, a cardinality group of features.

The concepts of configuration and configuration processes were initially used in the fields of artificial intelligence (Kumar, 2017) and problem-solving in operational research, primarily to enable the configuration of physical products. (Faltings and Freuder, 1998) introduced a special issue of 'Intelligent Systems and their Applications' focused on configuration processes. The two authors define this configuration process as a means of customizing parts of products to meet specific consumer needs. They particularly emphasize three criteria that a configuration must meet: (i) it must be correct so that the company can deliver the product; (ii) it must be produced quickly to avoid losing the customer to the competition; (iii) it must be optimal to convince the consumer. They also note that these criteria strongly favor the automation of the configuration process, and the progress of e-commerce tends to amplify this trend. In SPL,the configuration process is a major activity in application engineering, involving the selection of desired features within the PL before the product is realized. A FM represents all possible product configurations in an SPL defined in Figure 1 . As an example, the mobile phone feature model can generate up to 1232 different product variants (configurations). A sample product configuration for the mobile phone product line is illustrated in Figure 2 with FeatureIDE (Kaur and Kumar, 2014).

In SPLs, a significant hurdle is the task of toggling features within a feature model to craft new software product configurations (Machado et al., 2014). As the number of features increases in the model, so does the array of potential product options. This process resembles an optimal feature selection challenge within an Extended Feature Model (EFM) when configuring products within an SPL. However, without automated assistance, optimizing this selection becomes cumbersome. It involves addressing various objectives concurrently, such as aligning with user preferences and maximizing feature selection. Identifying the 'optimal' products within these vast and constrained parameter spaces exceeds human intuition. Thus, automated techniques for feature selection are essential to streamline configuration efforts.

3 PROBLEM DESCRIPTION

The primary concern with SPL is determining how to configure an optimized feature set that meets the customer's requirements. In Figure 1, features correspond to the functional requirements of the Mobile Phone SPL. However, features may also be linked to non-functional requirements. (Ferchichi et al., 2021), have highlighted the importance of considering non-functional requirements. For example, in the context of the Mobile Phone SPL, it's possible to identify non-functional requirements associated with each feature, such as quality. This implies that every prod-

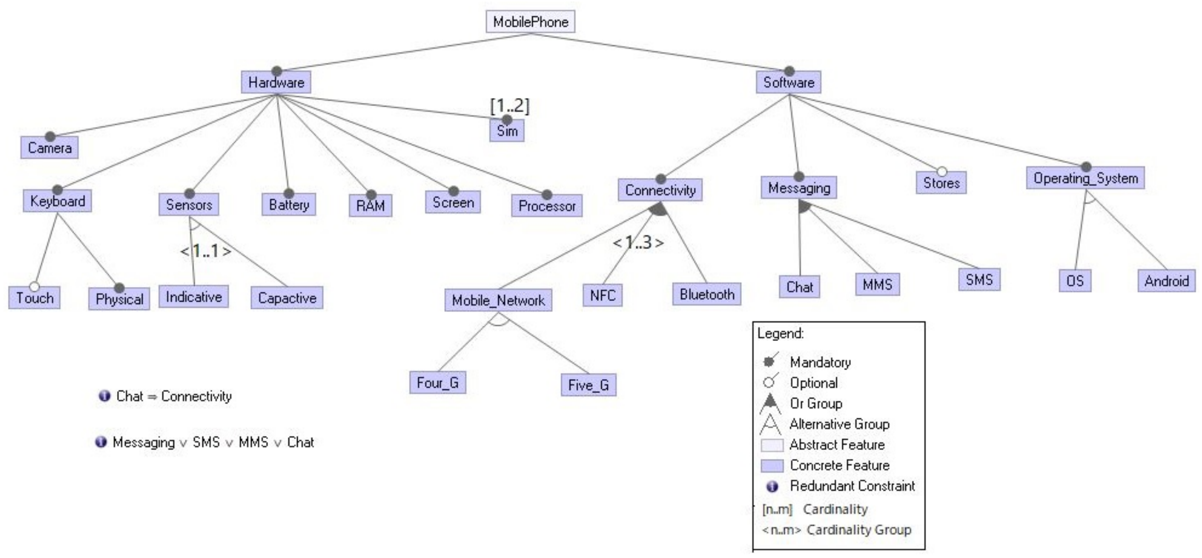


Figure 1: A Feature Model of the Mobile Phone SPL.

uct differs not only in terms of its functional features but also due to its non-functional requirements.

Therefore,in our previous work (Ferchichi et al., 2021), we proposed an extension of feature models with non-functional requirements in the quality aspect.

Figure 3 illustrates the non-functional requirements to extend the FM of Figure 1. In Figure 3, all features have the quality attribute (see table 1). As a motivating example, given a Mobile Phone PL that includes a variety of varying features, what is the product that best meets the customer requirements limited by a given quality? The challenge is that with hundreds or thousands of features, it is hard to analyze all different product configurations to find an optimal configuration.

When it comes to manually configuring extensive features of interdependent SPLs, it might become impossible, especially if numerous features and dependencies between features are involved. Additionally, the configuration process must be reiterated whenever the configuration or implementation of an SPL changes.

Ideally, a user should only have to configure a SPL that encompasses the entire application scenario. The user should not need to be concerned with the implementation details of underlying SPLs.

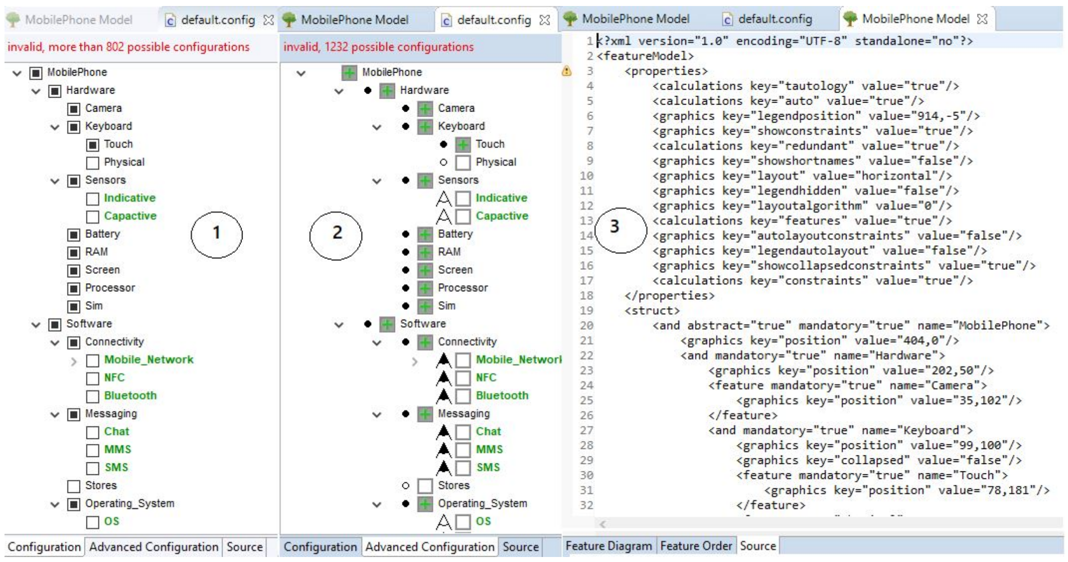
In this papier, we will discuss association rules, a fundamental concept in data mining that allows us to identify relationships among elements in a dataset. We will explain the basic principles of association rule mining, the various measures used to evaluate association rules, as well as the different algorithms used to generate these rules.

Table 1: Non functionals requirements.

	Quality attribute
Hardware	Refers to the physical components of a mobile phone, such as the processor, memory, display, battery, cameras, speakers, ports
Software	Refers to the programs, applications, and operating systems that run on a mobile phone, enabling various functions and features beyond the hardware components.
Security	Refers to feature designed to enhance the security of a system, device, application, or component
Behavioral	Refers to a feature or attribute of a system, organism, or entity that pertains to its behavior or the way it acts or operates.
Structural	Refers to a feature or attribute of an object, system, or entity that pertains to its physical or organizational structure.

Therefore, the main goal of the papier is to propose an automatic product configuration approach (see Figure 4) based on unsupervised learning.

Based on the association rule learning, our approach derives all the valid configurations of a PL, which is modeled by an extended feature model. Then the user can fix some fucntional and non-fucntional requirements and automatically the most satisfying configuration of a product is extracted.



```
invalid, more than 802 possible configurations
MobilePhone
  Hardware
    Camera
    Keyboard
      Touch
      Physical
    Sensors
      Indicative
      Capactive
    Battery
    RAM
    Screen
    Processor
    Sim
  Software
    Connectivity
      Mobile_Network
      NFC
      Bluetooth
    Messaging
      Chat
      MMS
      SMS
    Stores
    Operating_System
      OS
```

```
invalid, 1232 possible configurations
MobilePhone
  Hardware
    Camera
    Keyboard
      Touch
      Physical
    Sensors
      Indicative
      Capactive
    Battery
    RAM
    Screen
    Processor
    Sim
  Software
    Connectivity
      Mobile_Networl
      NFC
      Bluetooth
    Messaging
      Chat
      MMS
      SMS
    Stores
    Operating_System
      OS
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<featureModel>
    <properties>
        <calculations key="tautology" value="true"/>
        <calculations key="auto" value="true"/>
        <graphics key="legendposition" value="914,-5"/>
        <graphics key="showconstraints" value="true"/>
        <calculations key="redundant" value="true"/>
        <graphics key="showshortnames" value="false"/>
        <graphics key="layout" value="horizontal"/>
        <graphics key="legendhidden" value="false"/>
        <graphics key="layoutalgorithm" value="0"/>
        <calculations key="features" value="true"/>
        <graphics key="autolayoutconstraints" value="false"/>
        <graphics key="legendautolayout" value="false"/>
        <graphics key="showcollapsedconstraints" value="true"/>
        <calculations key="constraints" value="true"/>
    </properties>
    <struct>
        <and abstract="true" mandatory="true" name="MobilePhone">
            <graphics key="position" value="404,0"/>
            <and mandatory="true" name="Hardware">
                <graphics key="position" value="202,50"/>
                <feature mandatory="true" name="Camera">
                    <graphics key="position" value="35,102"/>
                </feature>
                <and mandatory="true" name="Keyboard">
                    <graphics key="position" value="99,100"/>
                    <graphics key="collapsed" value="false"/>
                    <feature mandatory="true" name="Touch">
                        <graphics key="position" value="78,181"/>
                    </feature>
```

Figure 2: Product Line Configuration 'Mobile Phone'.

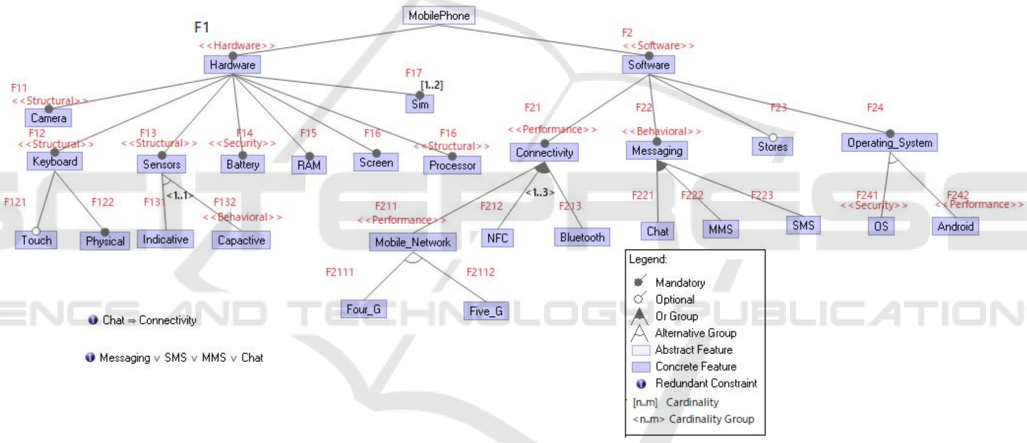


Figure 3: Extended FM by non-functional requirements.

4 RELATED WORK

Many past works were devoted to automatic configuration of SPL In this section, we present approaches related to our proposal.

(Batory, 2004) outlines AHEAD, a method for configuring Software Product Lines (SPLs) employing step-wise refinement, where configurations are iteratively refined. Our approach shares similarities, as it also involves selecting additional features across multiple steps to achieve a desired configuration (Hubaux et al., 2012)

introduced a formalism for establishing the workflow necessary to configure a feature model in several steps. The MUSCLES approach also emphasizes configuring a model through multiple steps. Nevertheless, (Hubaux et al., 2009)'s study does not explore feature model drifts or the automated derivation of a configuration path from an initial to a final configuration. Additionally, MUSCLES includes support for optimizations.

Various techniques for configuring and validating feature models in a single step have been proposed ((Benavides et al., 2013) (Heradio et al., 2022) (Benavides et al., 2005) (Ochoa et al., 2019))

These approaches leverage Constraint Satisfaction Problems (CSPs) and propositional logic to generate feature model configurations in a single stage and ensure their validity. Such techniques are valuable in addressing the high complexity associated with determining a valid feature selection for a feature model that satisfies a set of intricate constraints.

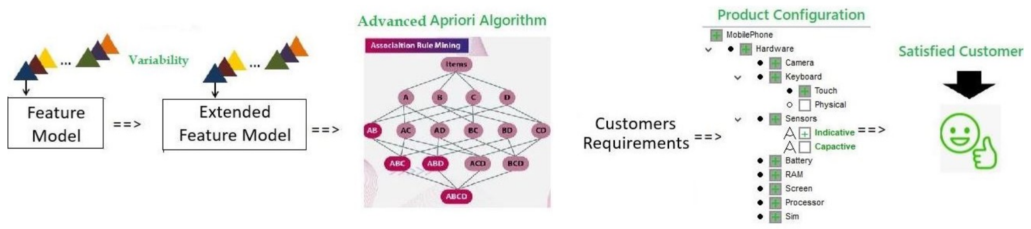


Figure 4: Overview of the Approach of Automatic generation of Product Configurations.

(Elsner et al., 2010) have examined variability over time and the challenges associated with understanding the relationships between variability points. MUSCLES concentrates on automating three essential tasks identified by Elsner et al. for managing variability over time. Specifically, MUSCLES offers capabilities for automating and optimizing tasks termed by Elsner et al. as: (1) proactive planning, (2) tracking, and (3) analysis. While Elsner et al. focus on the general identification of issues in managing variability over time, MUSCLES provides a framework for automating the specific tasks outlined by Elsner et al. as necessary in this context.

(Machado et al., 2014), present the SPLConfig, to support business during product configuration in SPL. Based on feature models, SPLConfig automatically finds an optimal product configuration that maximizes the customer satisfaction.

In their studies, (Berger et al., 2013) and (Mendonca et al., 2009) utilized propositional logic to automate the validation of FM, depict staged feature configurations, and offer assistance for manual feature selection. Nevertheless, their methods offer only semi-automatic support for product configuration, relying solely on Feature Properties (FPs). Furthermore, these methodologies rely on exact exponential-time algorithms such as Satisfiability (SAT) or binary decision diagrams, which are unsuitable for SPL applications due to their extensive computational requirements and cumbersome nature.

Our proposal is defined by an approach for automatically deriving PL configurations based on the associating rule learning. From the generated configurations the user can choose the quality attribute to be considered in the configuration of his need. In fact, based the quality attributes defined in the extended feature model, the extracted configuration, from all the derived configurations, may satisfy the user.

5 AUTOMATIC CONFIGURATIONS OF SPL

In Figure 4, we present an approach of deriving automatically all the valid configurations of a PL defined by an extended feature model, as a first step. We note that the extended FM is a FM enriched with quality steretypes. Having all the configurations enables the automatic detection of dead features and falsely optional ones. It's important to note that a feature is considered dead if it doesn't appear in any configuration of the SPL. A configuration is considered falsely optional if it appears in all configurations even though it's optional. With all the configurations of an SPL, if these situations occur, the evolution process automatically initiates the removal of the dead feature and changes the optionality (from optional to mandatory) of the falsely optional feature.

Then, having the valid configurations implies removing automatically all configurations having dead and optional features. In the next step, the customer inserts a set of functional and nonfunctional features to define his requirements, and automatically he gets the better configuration satisfying his requirements.

6 SPL CONFIGURATIONS BASED ON ASSOCIATION RULE MINING

Association rule mining is one of the most important application fields of the data mining tasks. Association rule is used to find out the dependency of among multiple domains based on the given degree of support and confidence.

In this section, we present the unsupervised Learning approach, and then we present our motivations to introduce this approach in improving EFM configuration.

Association Rule Mining (ARM) is a data mining (Saxena and Rajpoot, 2021) technique that aims to discover interesting relationships or associations among variables in large datasets. These associations are often expressed in the form of rules that describe the co-occurrence patterns of items or attributes within transactions.

ARM is unsupervised learning approach and is a category of machine learning that uses labeled datasets to train algorithms to predict outcomes and recognize patterns. In our approach, we used the advanced apriori algorithms (Suneetha and Krishnamoorti, 2010) which is based on an association rule learning.

6.1 Advanced Apriori Algorithms

An advanced apriori algorithm is used for frequent item set mining and association rule learning over databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. In classical Apriori algorithm, when candidate itemsets are generated, the algorithm needs to test their occurrence frequencies. The manipulation with redundancy will result in high frequency in querying, so tremendous amount of resources will be expended in time or in space. Therefore advanced algorithm was proposed for mining the association rules in generating frequent k-item sets.

Instead of judging whether these candidates are frequent item sets after generating new candidates, this new algorithm finds frequent item sets directly and removes the subset that is not frequent, based on the classical Apriori algorithm.

6.1.1 Key Metrics for Apriori Algorithm

The potential number of associations can be extensive, particularly with a large number of items. The challenge lies in determining which associations are most meaningful. When employing the Apriori algorithm, three essential metrics (Alsanad and Altuwaijri, 2022) are used : Support, Confidence and Lift. They are which are applied on rules.

- **Support:** The support of a rule or a set of attributes (items) indicates the percentage of records that satisfy the rule. (see formula 1)

$$Supp(X \Rightarrow Y) = \frac{|X\&Y|}{|BD|} \tag{1}$$

  We denote:

  – $|X|$: as the number of records containing attributes X in the database (BD)

  – $|BD|$ as the total number of records.

- **Confidence:** The confidence of a rule measures the validity of the rule: the percentage of examples that verify the conclusion among those that verify the premise. (see formula 2)

$$\begin{aligned} Conf(X \Rightarrow Y) &= |X\&Y|\,/\,|X| \\ Conf(X \Rightarrow Y) &= Supp(X\&Y)/Supp(X) \end{aligned} \tag{2}$$

- **Lift:** is a measure that tells us whether the probability of an item Y increases or decreases given item X. (see formula 3)

$$Lift(X \Rightarrow Y) = \frac{Conf(X \Rightarrow Y)}{Supp(Y)} \tag{3}$$

In this paper we describe the improvement of classical Apriori algorithm in the following two aspects: a) Reducing the passes of DB scan. b) Reducing the unnecessary features generation.

6.1.2 Advanced Apriori Algorithm

The advanced apriori algorithm we applied in our approach is defined in the following:

- **Algorithm**

```
Input : transactional database D and minimum support threshold min_sup.
Output: L, frequent item-sets in D.
Method:
1) $L_1$= frequent item-set of length 1,
2) Generate power set of $L_1$ and named as SPL initialize with item-set_count=0, it will global for entire algorithm.
3) For each transaction t in database Do.
a) For each item I in t Do,
Compare I with $L_1$
If (not match) then delete item from transaction t.
End Do.
b) Generate power set of t and named as items feaure
c) Compare item-sets of SPL with items feaure
d) If (item-set match) increase the item-set_count by 1 of SPL.
4) End Do.
Pruning phase:
5) For each item-set I in SPL Do,
6) If item-set_count of I is less than to min_sup threshold the delete I
7) End Do,
8) Remaining item-set in SPL will be frequent item-sets which holds min_sup threshold.
```

6.1.3 Description of the Algorithm Steps

The adavanced Apriori algo is described by the following steps

- (step 1): Initializing candidate sets items (cardinality k = 1)
- (step 2): Filtering: Retain only frequent sets L, i.e., with cardinality k whose support is ≥ supmin (fixed)
- (step 3): Joining: Combine all frequent sets of cardinality k to calculate sets of cardinality k1 -
  - Search for frequent items F1 :item-sets1
  - Among F1, search for frequent item pairs F2: item-sets2
  - From F2, search for frequent item triplets F3: item-sets3
- (steps 4) Testing: If the set of sets of cardinality k+1 is empty, stop; otherwise, k = k+1, return to step 2."

6.2 Configuration of SPL Based on the Advanced Apriori Algorithm

In this section, we present the application of the Advaced apriori algorithm for automatic generation of SPL configurations. The database D is an Extented Feature Model (EFM) of a SPL. An item is a feature defined in the EFM. A rule is defined by a relationship between two features. The support of a rule is defined according to the type of the relationship beween two features(items).

Each relationship bewteen two features indicates if they can be included in the same configuration or not, which has an influence on the support value of associated rule.

- Mandatory
  When a sub-feature has a mandatory relationship with its parent, the configuration cannot exhibit the child feature unless it also exhibits the parent feature. The support of this relationship is equal to 100%. So, every mandatory feature must be included in all configurations of the line (see Figure 5).

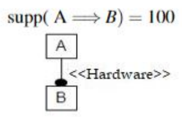


Figure 5: Support of the mandatory relationships.

- Optional
  In an optional relationship, the feature can either be chosen or not, so the probability of being chosen is 1/2 and the probability of not being chosen is also 1/2, thus the supmin=50.
- Alternative Group , cardinality group of features $< 1..1 >$ and Constraint X-or

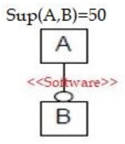


Figure 6: Support of the optional relationships.

With in the Basic and Extended Feature Models, alternative Group, cardinality group of features $< 1..1 >$ and Constraint X-OR have the same semantics. In this relation, the minsupp is equal to 1/3, and we can only choose a single feature, which implies that we can extract only one rule.

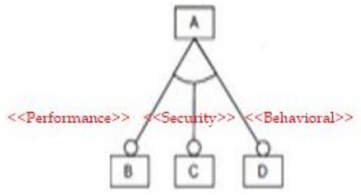


Figure 7: Support of the alternative relationships.

- Or Group , cardinality group of features $< n..m >$ and Constraint Or
  Within the Basic and Extended Feature Models, Or Group , cardinality group of features $< n..m >$ and Constraint have the same semantics. The set of sub-features C, D, and B are linked to the parent feature A through a multiple-choice relationship if one or more sub-features can be included in a product presenting the parent feature. In this example, we will apply in detail all the steps of the advanced Apriori algorithm .

In this case, the supmin is the support of a feature equal to 1/3 .We follow the next three steps of the algorithm to compute the frequent features.

- **Searching for Frequent Items F1 Item-Set 1**
  In this section, we will calculate the support of each feature which belongs to item-set1. Support (B)= 1/3, support (C)=1/3 support (D)= 1/3 , Thus, the supmin is 1/3, and we apply the formula support ≥ supmin so F1={B,C,D}
- **Among F1, Searching for Frequent Items F2: Item-Set 2**
  The itemsets are BC, BD, CD, support(BC)= 2/3, support(BD)= 2/3 ,support(CD)= 2/3, so sup-

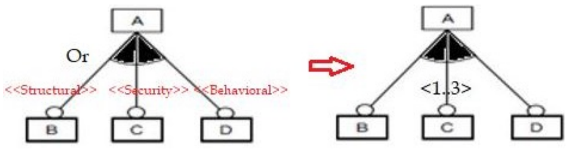


Figure 8: Representation of the Or Group and cardinality group of features $< n..m >$ relationships.

port(BD),support(DC) and support (BC) =2/3 $\geq$ 1/3 . F2= {BC, BD, CD}.

- **From F2, Search for Frequent Item Triplets Items F3: Item-Set 3**

  F3= {BCD}, support(BCD)=1 $\geq$ 1/3.

The Figure 9 illustrates the process of extracting frequent itemsets. This involves traversing a lattice and calculating the supports associated with each combination. The number of configurations quickly becomes very high, and each configuration would require scanning the database. It is essential to consider the minsupp parameter.

6.2.1 The Selected Rules

"Considering all the retained association rules in the following table based on support calculations. In the table, we have extracted all frequent rules across the different relations of the 'Group' or group.

Table 2: Extraction of Association Rules.

N°	Rules	Confidence
1	$A \Longrightarrow B$	100%
2	$A \Longrightarrow C$	100%
3	$A \Longrightarrow D$	100%
4	$A \Longrightarrow BC$	100%
5	$A \Longrightarrow BD$	100%
6	$A \Longrightarrow CD$	100%
4	$A \Longrightarrow BCD$	100%

The table represents the number of associated configurations for these relationships.

Table 3: Numbers of configurations.

Features	C1	C2	C3	C4	C5	C6	C7
B	1	0	0	1	0	1	1
C	0	1	0	1	1	0	1
D	0	0	1	0	1	1	1

6.2.2 Representation of Lift

The enhanced algorithm is outlined in the following steps:

Input: Data: a transaction database
Min_sup: the minimum support count threshold

- Initially, each item is considered a member of the set of feature item-set C1. The algorithm scans all transactions to count the occurrences of each item.
- The set of frequent item-sets, L1, is determined by comparing the feature count with the minimum support count, containing candidate 1-itemsets satisfying the minimum support.
- To generate the set of frequent item-sets 2, L2, the algorithm generates a feature set of item-sets 2. Then, the transactions in Data are scanned, and the support count of each feature item set in C2 is accumulated, repeating step 2.
- C2 is determined from L2.
- Generate C3 features from L2 and scan C2 for the count of each feature, then repeat step 2.
- At the end of the pass, determine which feature item sets are actually large, and those become the seed for the next pass.
- This process continues until no new large item sets are found (see Figure 10).

7 APPROACH EXPERIMENT

In this section, we present an application of the approach proposed for the Mobile Phone extended feature model (as depicted in Figure 3). We implemented the approach using Python. Following this, we demonstrate the application of the advanced Apriori algorithm.

The extended feature model created for the SPL mobile phone using FeatureIDE is stored in an XML format. We edit the feature model both graphically and textually, simultaneously identifying features. The feature model is stored in an XML file, which we then imported into Python. As illustrated in Figure 11 below.

Subsequently, we tested the algorithm with 6 transactions, selecting the first three features with the FeatureIDs F, F1, F2. We then applied the algorithm to extract frequent rules. Through these rules, we aim to retrieve the possible configurations' versions.

The Figure 13 depicts a simulation of the example of EFM with the mandatory relationship between the three features: MobilePhone (F), Hardware (F1), and Software (F2). Since the relationship between these features is mandatory, we have set the support value and minsup to 1.

So, itemset1 consists of these three features: itemset1 = MobilePhone (F), Hardware (F1), Software (F2)with cardinality = 1. Subsequently, from itemset1, the combination of three features yields itemset2 with cardinality = 1. The association rules are generated with a confidence of 100%, as shown in Figure 12.

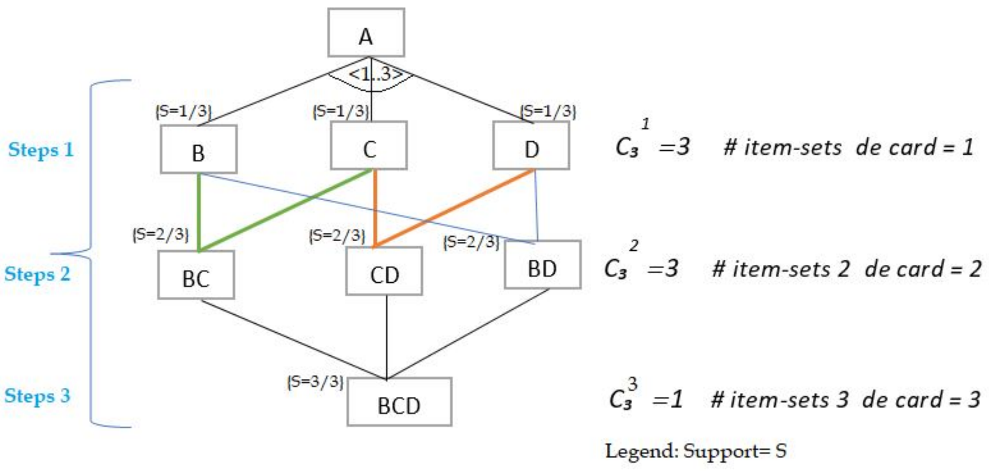
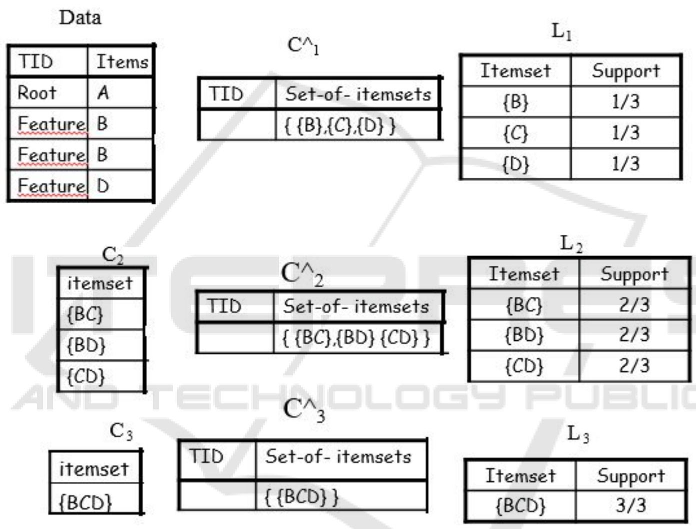

A
<1..3>
(S=1/3)
(S=1/3)
(S=1/3)
Steps 1
B
C
D
$C_3^1 = 3$ # item-sets de card = 1
(S=2/3)
(S=2/3)
(S=2/3)
Steps 2
BC
CD
BD
$C_3^2 = 3$ # item-sets 2 de card = 2
(S=3/3)
Steps 3
BCD
$C_3^3 = 1$ # item-sets 3 de card = 3
Legend: Support= S

Figure 9: Frequent itemsets extraction.



Data

TID	Items
Root	A
Feature	B
Feature	B
Feature	D

C^1

TID	Set-of- itemsets
	{ {B},{C},{D} }

L_1

Itemset	Support
{B}	1/3
{C}	1/3
{D}	1/3

C_2

itemset
{BC}
{BD}
{CD}

C^2

TID	Set-of- itemsets
	{ {BC},{BD} {CD} }

L_2

Itemset	Support
{BC}	2/3
{BD}	2/3
{CD}	2/3

C_3

itemset
{BCD}

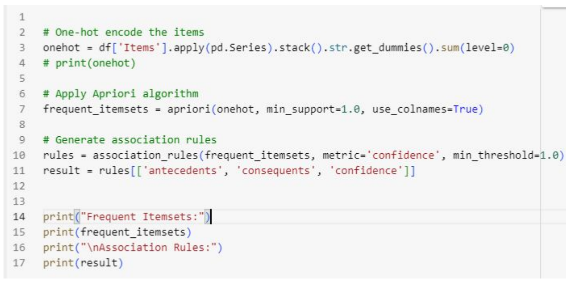
C^3

TID	Set-of- itemsets
	{ {BCD} }

L_3

Itemset	Support
{BCD}	3/3

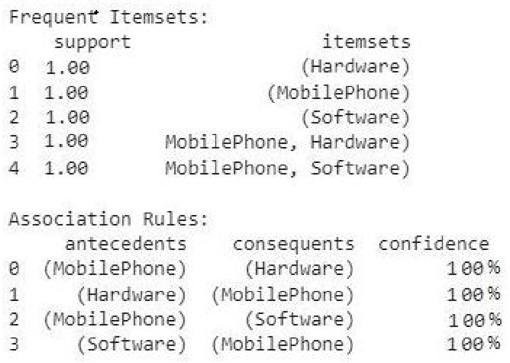
Figure 10: Example of Generation of feature item set and frequent item-set.



```python

# One-hot encode the items
onehot = df['Items'].apply(pd.Series).stack().str.get_dummies().sum(level=0)
# print(onehot)

# Apply Apriori algorithm
frequent_itemsets = apriori(onehot, min_support=1.0, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric='confidence', min_threshold=1.0)
result = rules[['antecedents', 'consequents', 'confidence']]


print("Frequent Itemsets:")
print(frequent_itemsets)
print("\nAssociation Rules:")
print(result)
```

Figure 11: Example of Generation of feature item set and frequent item-set.

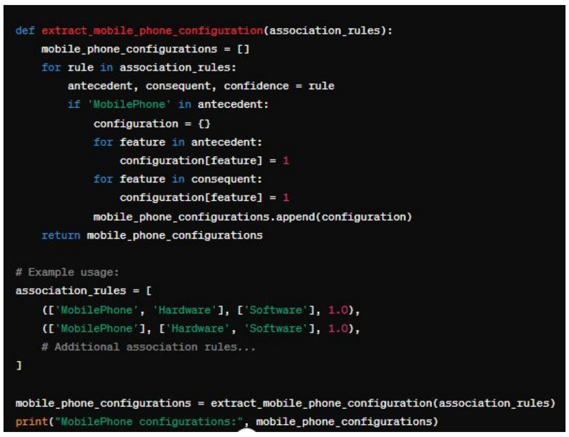
To extract the configuration of the SPL Mobile Phone, we can use the association rules generated by the advanced Apriori algorithm. Each rule represents a possible configuration of features. We can filter out the rules that include the Mobile Phone feature and



```
Frequent Itemsets:
   support                  itemsets
0   1.00                  (Hardware)
1   1.00               (MobilePhone)
2   1.00                  (Software)
3   1.00     MobilePhone, Hardware)
4   1.00     MobilePhone, Software)

Association Rules:
     antecedents     consequents   confidence
0  (MobilePhone)      (Hardware)        100%
1     (Hardware)   (MobilePhone)        100%
2  (MobilePhone)      (Software)        100%
3     (Software)   (MobilePhone)        100%
```

Figure 12: Result of implementing the example of Generation of feature item set and frequent item-set.

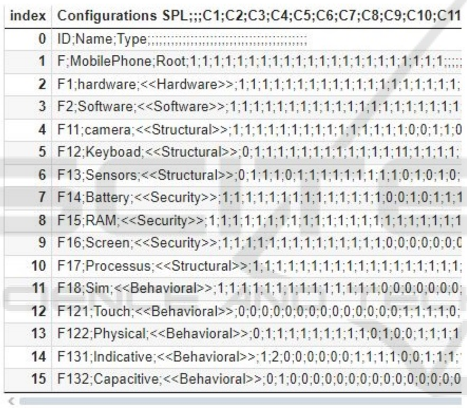
extract the corresponding configuration. The Figure 13 represents an excerpt of code from SPLs configuration



```
def extract_mobile_phone_configuration(association_rules):
    mobile_phone_configurations = []
    for rule in association_rules:
        antecedent, consequent, confidence = rule
        if 'MobilePhone' in antecedent:
            configuration = {}
            for feature in antecedent:
                configuration[feature] = 1
            for feature in consequent:
                configuration[feature] = 1
            mobile_phone_configurations.append(configuration)
    return mobile_phone_configurations

# Example usage:
association_rules = [
    (['MobilePhone', 'Hardware'], ['Software'], 1.0),
    (['MobilePhone'], ['Hardware', 'Software'], 1.0),
    # Additional association rules...
]

mobile_phone_configurations = extract_mobile_phone_configuration(association_rules)
print("MobilePhone configurations:", mobile_phone_configurations)
```

Figure 13: Code snippet assumes that the configurations.

The Figure 14 represents the SPL configuration using the advanced Apriori algorithm. If a feature is present in a configuration, its value is set to 1; otherwise, it is set to zero.



index	Configurations SPL;;;C1;C2;C3;C4;C5;C6;C7;C8;C9;C10;C11
0	ID;Name;Type;;;
1	F;MobilePhone;Root;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;;;;;
2	F1;hardware;<<Hardware>>;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;
3	F2;Software;<<Software>>;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1
4	F11;camera;<<Structural>>;1;1;1;1;1;1;1;1;1;1;1;1;1;1;0;0;1;1;0
5	F12;Keyboad;<<Structural>>;0;1;1;1;1;1;1;1;1;1;1;1;11;1;1;1;1;
6	F13;Sensors;<<Structural>>;0;1;1;1;0;1;1;1;1;1;1;1;1;0;1;0;1;0;
7	F14;Battery;<<Security>>;1;1;1;1;1;1;1;1;1;1;1;1;0;0;1;0;1;1;1
8	F15;RAM;<<Security>>;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1
9	F16;Screen;<<Security>>;1;1;1;1;1;1;1;1;1;1;1;1;1;0;0;0;0;0;0;0
10	F17;Processus;<<Structural>>;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;
11	F18;Sim;<<Behavioral>>;1;1;1;1;1;1;1;1;1;1;1;1;1;0;0;0;0;0;0;
12	F121;Touch;<<Behavioral>>;0;0;0;0;0;0;0;0;0;0;0;0;0;1;1;1;1;0;
13	F122;Physical;<<Behavioral>>;0;1;1;1;1;1;1;1;1;0;1;0;0;1;1;1;1
14	F131;Indicative;<<Behavioral>>;1;2;0;0;0;0;0;0;1;1;1;1;0;0;1;1;1;
15	F132;Capacitive;<<Behavioral>>;0;1;0;0;0;0;0;0;0;0;0;0;0;0;0;0

Figure 14: Extraction of the SPL Mobile Phone configuration.

8 CONCLUSIONS

Feature model configurations in Software Product (SPL) Lines still remain problematic and specifically for Extended Feature models. This paper proposes the use of an unsupervised learning approach for configuration generation. Hence, we use an advanced apriori algorithm for frequent item set minig and association rule. These latters will be able to let feature model generation in order to derive specific applications (SPL products). We have exeprimented all the proposed approach on an extended mobile phone feature model. This latter is enriched by non functional requirements, so complicating the configuration. The obtained results are promising and show that learning is really an efficient way for product derivations in the context of Software Product Line. As future work, we will experiment our approach on more SPL cases and try to use other learning methods and compare their results in the context of configuration generations.

REFERENCES

Alsanad, A. and Altuwaijri, S. (2022). Advanced persistent threat attack detection using clustering algorithms. *International Journal of Advanced Computer Science and Applications*, 13(9).

Bashroush, R., Garba, M., Rabiser, R., Groher, I., and Botterweck, G. (2017). Case tool support for variability management in software product lines. *ACM Computing Surveys (CSUR)*, 50(1):1–45.

Batory, D. (2004). A tutorial on feature oriented programming and the ahead tool suite (ats). *Revised: Sept*, 8:2004.

Benavides, D., Felfernig, A., Galindo, J. A., and Reinfrank, F. (2013). Automated analysis in feature modelling and product configuration. In *Safe and Secure Software Reuse: 13th International Conference on Software Reuse, ICSR 2013, Pisa, June 18-20. Proceedings 13*, pages 160–175. Springer.

Benavides, D., Segura, S., Trinidad, P., and Ruiz-Cortés, A. (2005). Using java csp solvers in the automated analyses of feature models. In *International Summer School on Generative and Transformational Techniques in Software Engineering*, pages 399–408. Springer.

Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K., and Wąsowski, A. (2013). A survey of variability modeling in industrial practice. In *Proceedings of the 7th International Workshop on Variability Modelling of Software-intensive Systems*, pages 1–8.

Clements, P., N. L. (2001). Software product lines practices and patterns, addison-wesley professional.

El-Sharkawy, S., Yamagishi-Eichler, N., and Schmid, K. (2019). Metrics for analyzing variability and its implementation in software product lines: A systematic literature review. *Information and Software Technology*, 106:1–30.

Elsner, C., Botterweck, G., Lohmann, D., and Schröder-Preikschat, W. (2010). Variability in time-product line variability and evolution revisited. *VaMoS*, 10:131–137.

Faltings, B. and Freuder, E. C. (1998). Guest editors' introduction: Configuration. *IEEE Intelligent Systems*, 13(04):32–33.

Ferchichi, O., Beltaifa, R., and Jilani, L. L. (2020). An ontological rule-based approach for software product lines evolution. In *2020 International Multi-Conference on:“Organization of Knowledge and Advanced Technologies”(OCTA)*, pages 1–9. IEEE.

Ferchichi, O., Beltaifa, R., Jilani, L. L., and Mazo, R. (2021). Towards a smart feature model evolution. *ICSEA 2021*, page 159.

Gérard, S., Espinoza, H., Terrier, F., and Selic, B. (2007). 6 modeling languages for real-time and embedded systems: Requirements and standards-based solutions.

In *Dagstuhl Workshop on Model-Based Engineering of Embedded Real-Time Systems*, pages 129–154. Springer.

Heradio, R., Fernandez-Amoros, D., Galindo, J. A., Benavides, D., and Batory, D. (2022). Uniform and scalable sampling of highly configurable systems. *Empirical Software Engineering*, 27(2):44.

Hubaux, A., Classen, A., and Heymans, P. (2009). Formal modelling of feature configuration workflows. In *Proceedings of the 13th International Software Product Lines Conference (SPLC'09), San Francisco, CA, USA*, pages 221–230. SEI, Carnegie Mellon University.

Hubaux, A. et al. (2012). *Feature-based Configuration: Collaborative, Dependable, and Controlled.* PhD thesis, Citeseer.

Kaur, M. and Kumar, P. (2014). Mobile media spl creation by feature ide using foda. *Global Journal of Computer Science and Technology*, 14(3).

Kumar, S. L. (2017). State of the art-intense review on artificial intelligence systems application in process planning and manufacturing. *Engineering Applications of Artificial Intelligence*, 65:294–329.

Lee, K., Kang, K. C., and Lee, J. (2002). Concepts and guidelines of feature modeling for product line software engineering. In *International Conference on Software Reuse*, pages 62–77. Springer.

Machado, L., Pereira, J., Garcia, L., and Figueiredo, E. (2014). Splconfig: Product configuration in software product line. In *Brazilian Congress on Software (CBSoft), Tools Session*, pages 1–8. Citeseer.

Mendonca, M., Wąsowski, A., and Czarnecki, K. (2009). Sat-based analysis of feature models is easy. In *Proceedings of the 13th International Software Product Line Conference*, pages 231–240.

Metzger, A. and Pohl, K. (2014). Software product line engineering and variability management: achievements and challenges. *Future of software engineering proceedings*, pages 70–84.

Ochoa, L., González-Rojas, O., Cardozo, N., González, A., Chavarriaga, J., Casallas, R., and Díaz, J. F. (2019). Constraint programming heuristics for configuring optimal products in multi product lines. *Information Sciences*, 474:33–47.

Pohl, K., Böckle, G., and Van Der Linden, F. (2005). *Software product line engineering: foundations, principles, and techniques*, volume 1. Springer.

Saxena, A. and Rajpoot, V. (2021). A comparative analysis of association rule mining algorithms. In *IOP Conference Series: Materials Science and Engineering*, volume 1099, page 012032. IOP Publishing.

Sreekumar, A. (2023). Guided requirements engineering using feature oriented software modeling.

Suneetha, K. and Krishnamoorti, R. (2010). Advanced version of a priori algorithm. In *2010 First International Conference on Integrated Intelligent Computing*, pages 238–245. IEEE.

White, J., Galindo, J. A., Saxena, T., Dougherty, B., Benavides, D., and Schmidt, D. C. (2014). Evolving feature model configurations in software product lines. *Journal of Systems and Software*, 87:119–136.