Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2023.041627

ARTICLE

Binary Oriented Feature Selection for Valid Product Derivation in Software Product Line

Muhammad Fezan Afzal[1], Imran Khan[1], Javed Rashid[1,2,3], Mubbashar Saddique[4,*] and Heba G. Mohamed[5]

[1]Department of CS&SE, International Islamic University, Islamabad, 44000, Pakistan

[2]Information Technology Services, University of Okara, Okara, 56300, Pakistan

[3]Departmet of Computer Science, MLC Lab, Okara, 56300, Pakistan

[4]Department of Computer Science & Engineering, University of Engineering & Technology Lahore, Narowal Campus, Narowal, 51601, Pakistan

[5]Department of Electrical Engineering, College of Engineering, Princess Nourah bint Abdulrahman University, P.O. Box 84428, Riyadh, 11671, Saudi Arabia

*Corresponding Author: Mubbashar Saddique. Email: dr.mubbashar@uet.edu.pk

Received: 29 April 2023 Accepted: 04 July 2023 Published: 08 October 2023

ABSTRACT

Software Product Line (SPL) is a group of software-intensive systems that share common and variable resources for developing a particular system. The feature model is a tree-type structure used to manage SPL's common and variable features with their different relations and problem of Crosstree Constraints (CTC). CTC problems exist in groups of common and variable features among the sub-tree of feature models more diverse in Internet of Things (IoT) devices because different Internet devices and protocols are communicated. Therefore, managing the CTC problem to achieve valid product configuration in IoT-based SPL is more complex, time-consuming, and hard. However, the CTC problem needs to be considered in previously proposed approaches such as Commonality Variability Modeling of Features (COVAMOF) and Genarch + tool; therefore, invalid products are generated. This research has proposed a novel approach Binary Oriented Feature Selection Crosstree Constraints (BOFS-CTC), to find all possible valid products by selecting the features according to cardinality constraints and cross-tree constraint problems in the feature model of SPL. BOFS-CTC removes the invalid products at the early stage of feature selection for the product configuration. Furthermore, this research developed the BOFS-CTC algorithm and applied it to, IoT-based feature models. The findings of this research are that no relationship constraints and CTC violations occur and drive the valid feature product configurations for the application development by removing the invalid product configurations. The accuracy of BOFS-CTC is measured by the integration sampling technique, where different valid product configurations are compared with the product configurations derived by BOFS-CTC and found 100% correct. Using BOFS-CTC eliminates the testing cost and development effort of invalid SPL products.

KEYWORDS

Software product line; feature model; internet of things; crosstree constraints; variability management

1 Introduction

A group of products with a common set of features to serve particular market segments is known as a Software Product Line (SPL). The development of the software family is SPL's primary objective. Due to the reusability of core assets' common and variable features, the SPL software family facilitates the development of a wide range of software systems. All necessary features that indicate the family of software's specification and scope are the core asset of SPL. SPL is giving the best improvement in the programming industry due to the quick advancement of programming by the reusability of elements from center resources [1]. SPL is used by industries, such as mobile phones, Internet of Things (IoT) applications, and automobiles for the development of a family of software to target specific market segments and claimed that it provides a promising path for better, faster, and cheaper development of a wide range of software systems. From the perspective of individual end-users or market requirements, each product that aims to develop from SPL differs from the others. Reusable parts of SPL are normal and variable elements utilized to foster the group of items. Due to the reusability of all SPL-derived products, common features are simple to manage; however, variability features are chosen based on the end user's needs and create product differentiation and variety [2]. Therefore, variable features must manage their relationship to other features during selection and rejection from actual product development because they are not included in every product. A feature's selection or rejection during product configurations may result in relationship constraints and an invalid product. As a result, SPL's variability management is the primary obstacle encountered during the software's development. Variable features must be handled and managed inefficiently to enable the high reusability of SPL features, as SPL supports the high reusability of features [3].

Fig. 1 illustrates how a feature model, a tree-like structure, is used in the literature to manage SPL's varied, common properties and crosstree constraints problem. An SPL's configuration rules and the relationships between its variation points can be recorded using the feature model. A feature model concisely represents the entire SPL, complete with constraints and relationships between features [1,4]. For a functioning product configuration, parent-child relationships must be observed when selecting features from the feature model. Features with predefined relationships in the feature model are (i) additional features, in which a single feature from a group of features is chosen; (ii) optional features, which can be chosen or not, (iii) OR set, where at least one feature from each group must be selected and (iv) Crosstree Constrains, in which the relationships among features are included or excluded of features in different parent sub-tree. It is necessary to develop all features in advance without creating a running application at the domain level to ensure they can be used in real-world product configurations [3,5].

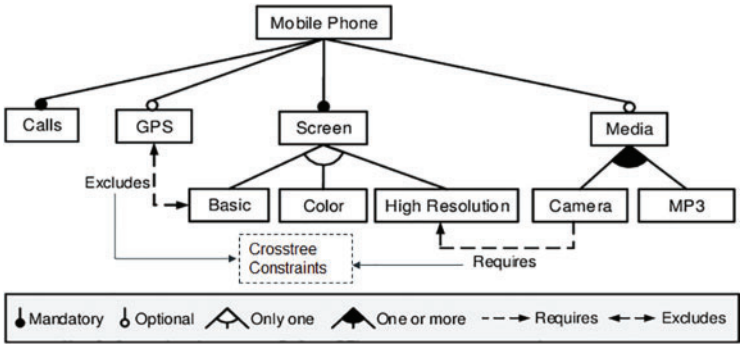

Figure 1: Mobile phone feature model

Organizations put in the time, money, and effort necessary for the product configuration based on the feature model before constructing the features. The initial costs of SPL and single product development are depicted in Fig. 2, indicating that SPL organizations invest in initial development costs without benefiting from the market [3]. The break-even point of SPL shown in Fig. 2 depends on the size of SPL, i.e., the total number of product configurations. The total valid number of products is a major parameter for the advanced cost estimation of SPL. However, calculating the total number of valid products is challenging due to the feature model's predefined relationships and crosstree constraints. Therefore, multiple methods and approaches exist, such as determining how many products are included in the feature model. Binary Pattern for Nested Cardinality Constraints (BPNCC) cardinality Constraints (dealing of Features (approach is applied to the Internet of Things (IoT) and Software Product Line of Things (SPLOT) are discussed in the literature. However, these approaches only consider the basic and nested cardinality constraints such as "OR," "AND," "Alternate," and "OR group" relationships to calculate the total number of products. However, there are still possibilities of invalid product derivation due to the crosstree constraints in the sub-tree of the feature model. This problem leads to wrong cost estimation of SPL due to invalid products [6].

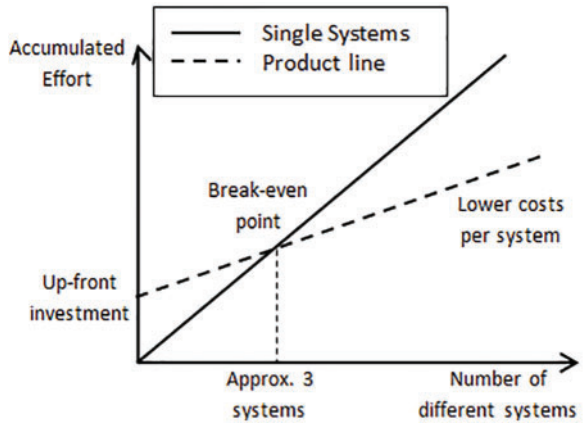

Figure 2: Cost estimation of SPL and single product

The first problem is crosstree constraints in the IoT-based feature model, invalid feature combinations become part of SPL, leading to extra effort and cost in developing SPL. As shown in Fig. 1, the crosstree constraints "Global Positioning System (GPS)" and "Basic" exclude each other; therefore, any product that contains GPS and Basic will be invalid. Moreover, the crosstree constraint "Camera" includes the "High Resolution"; if the camera is selected, the high resolution must be part of the product. Fig. 1 shows "mobile phone" SPL where ten products are invalid due to the crosstree constraints problem. It is important to remove the invalid products from the total number of products before developing SPL. However, existing approaches find the total number of products but do not consider the crosstree constraints that lead to both valid and invalid products. Due to invalid products, the development cost and effort increase. "Hence, invalid feature combinations are generated due to constraints problem, and relationships between varied features resultantly make this process complex and consume extra effort during integration testing of SPL."

This paper proposed a novel Binary Oriented Feature Selection Crosstree Constraint (BOFS-CTC) approach that calculates the valid feature product combinations by considering IoT devices' basic and nested cardinality and crosstree constraints. BOFS-CTC is applicable for all kinds of small and large feature models with low to high complexity of constraints. The contribution of this paper is to

mitigate the invalid feature combinations for product derivation at an early stage of SPL development. Furthermore, BOFS-CTC has applied different complexity feature models to obtain the total valid digit of products and found 100% accuracy. However, the previous approaches need to consider the crosstree constraints problem to get valid products. In this paper, different approaches are compared with the proposed BOFS-CTC algorithm, and it is found that BOFS-CTC is a more appropriate and applicable approach for an accurate features' combination of the feature model. In resultant, by using BOFS-CTC, the total cost and effort of SPL product development are minimized. Furthermore, BOFS-CTC is the independent approach of any specific tool as we have proposed its algorithm.

2 Related Work

This section discusses previous contributions related to the total number of products and product configurations and for features selection of the SPL feature model. Due to the intricate interactions between features in the SPL feature model, managing variability is difficult [7]. In addition, estimating the precise total number of features and every potential combining feature in a large-scale feature model is time-consuming and prone to error tasks [8]. Furthermore, constraint violations frequently happen for SPL product configurations with several objectives, so it is challenging to identify the best approach.

Cavalcanti et al. approached for SPL feature model variants traceability provided by the meta-model level, the link between features and the constraints are defined using Unified Modeling Language (UML) notations. The meta-model is based on SPL's key assets and is arranged according to UML models. Due to the necessity of common features being a component of every product, requirement traceability in each product with variants must be linked. By keeping track of the variation points, the meta-model variant traceability technique makes it possible to track product variation and improve testability across the board in a single SPL scope [9].

AMPLET Traceability Framework (ATF) of SPL development process artifacts was proposed by Anquetila et al. and is model-driven and based on AMPLET. The ATF traceability management system improves the development of features, their import and export, their search for product variations, and their visualization of linkages between artifacts. Implementations of the matrix model, such as the domain engineering of SPL and the traceability link information stored in the database repository, can be accessed by the query to find the variation points [10].

For modeling the needs of SPL, Shaker et al. [11] introduced Feature-Oriented Formal Language (FORMAL). If the end user adds additional needs, they can be modeled using formal modeling. To promote reusability, it allows the modularity of features to eliminate feature dependencies. Decompose the features into tiny discrete features to increase reusability among SPL products. The objectives of FORMAL modeling include precision, associative and commutative composition, associative and commutative modularity, simplicity of development, modeling differences, and feature modularity [11].

Cechticky et al. [8] suggested an Extensible Markup Language (XML) based modeling strategy for the SPL feature model. All major feature model drawbacks can be found using an XML-based modeling technique. The feature model is mapped using an XML schema, turned into an XML file, and translated to an XML Schema Definition (XSD) by defining the needs and constraints of the end user throughout application development. A primary information of feature relationships, such as alternative, obligatory, optional, and OR group, must be predefined at the stage of domain engineering to translate all constraints and relationships of the feature model in the XML schema. Also, at the

application engineering level, feature constraints and end-user requirements must be specified at the XSD level for proper feature selection [8].

Hartman et al. [7] suggested handling feature model contextual variability in diverse application development contexts. Textual variation distinguishes each of SPL's products in light of the end customer's needs. Due to the number of systems that interface with one another across numerous product lines, contextual variability rises. This study suggested modeling contextual variability across multiple product lines. Merging the various feature models in domain engineering, such as product lines for tablets and mobile phones, shares some similarities. The opportunity to reuse features across many product lines increases, resulting in faster time to market and lower development costs [7]. Relationships between features in the particular environment must be modeled using contextual variability at the requirement engineering stage of product lines. Due to complicated interactions (include or exclude) between features, several features affect the other features during the development of the program.

Ali et al. [12] addressed SPL's contextual variability in their work and suggested a unified framework that begins with the application development goals before mapping those goals to feature models at the beginning of modeling. The feature model classifies each aim as the terminal feature, and their relationships are specified. This framework's problem frame manages the cross-tree restrictions between features to reduce complexity as goals are enhanced during requirement elicitation and feature model complexity rises [12]. The limitations of variation points in each SPL product make it difficult to manage variance. Hence, carefully managing these restrictions in the variability feature model makes the right product configurations possible. The authors' method for automatically creating variability models for product development defines build-time faults and heuristic feature extraction limitations from the code. Three goals were set for this study's evaluation of four open-source SPL systems: (1) correctness of the suggested technique, (2) recovery of the original constraint variability feature model, and (3) classification of constraints. The suggested method effectively manages variability utilizing build-time and extracts feature model constraints for the proper product derivation [13].

Selecting practicable and significant characteristics is challenging and time-consuming for stakeholders in big feature models with complicated relationships. The inability of stakeholders to examine the feature model and pinpoint the key features for a given proposal. Hence, to make it simple for stakeholders to choose key features for particular product development, authors have proposed the goal-based design of the SPL domain. The suggested method transforms the feature model into a goal-oriented one and identifies the functionally complete characteristics that are most useful across various applications. As a result, stakeholders can choose features from the goal-oriented feature model based on application needs. This procedure will aid in the better understanding of stakeholders and the better application development of developers [14].

Finding suitable configurations from the fundamental assets that make up SPL is difficult because of the high dimension of the data and the constraints. The author offered a practical way of creating an SPL model to analyze features and produce a reliable product configuration. The basic asset of SPL is defined using the schema in the suggested approach, which is based on XML. Products are verified using an alloy analyzer once the feature properties from the core asset defined in the schema are extracted. Moreover, decision models are generated for each configuration using Extensible Style Sheet Language Transformation (XSLT) [15].

The complexity of SPL's diversity and heterogeneity makes extracting the proper configuration knowledge challenging. Authors have put up the idea of Domain Knowledge Modeling Language (DKML) to specify the knowledge of SPL configuration. Additionally, the GenArch+ tool is used in

this work to enhance the specification of configuration information for the SPL feature model and to facilitate the composition and production of DKMLs. DKML and GenArch+ working together suggested that SPL product derivations will perform better [16].

Cost-estimating models like Structured Intuitive Model for Product Line Economics (SIMPLE) and Constructive Product Line Investment Model (COPLIMO) need information from the feature diagram to determine the entire estimated cost of SPL. In addition to the overall number of potential goods, these models require the total number of variables and shared attributes. To determine, authors have suggested the method [17] based on Non-Fungible token (NFT) to account for all potential products, which employs the same notations as VFD+ and NFT. For the management of variability in feature models, the framework for the Feature Model Analyzer FAMA has been put forth in the literature [5]. FAMA assists in identifying the valid feature model product by utilizing XML that complies with all restrictions. It also computes the total number of all feature model products that are theoretically possible. Another method requiring only one call to execute has been proposed to calculate the entire number of possible products [18]. This algorithm, which improves on NFT and VFD+ notations, is much more effective in computation and runtime efficiency. This technique manages the restrictions between features like alternative, optional, and OR by using the cardinality of leaf features.

Table 1 describes the advantages and disadvantages of existing approaches in literature.

Table 1: Comparison of existing approaches in literature

Source	Year	Approach	Advantages	Disadvantages
Cavalcanti et al. [9]	2011	UML base feature model design	Transform the feature model into UML diagram to manage the common and variable features.	UML diagrams map the common and variable features but do not support the product configurations.
Anquetil et al. [10]	2010	AMPLET traceability framework	Stored the variation points into database and trace the variability in each product of SPL.	ATF is not supporting if there is any update or change in the existing feature model.
Shaker et al. [11]	2012	FORMAL	Modeling include precision, associative and commutative composition, associative and commutative modularity.	Convert the feature model into separate modules such as OR group and alternate group. But unable to convert the feature model that have crosstree constraints.
Cechticky et al. [8]	2004	XML and XSD based variability modeling	Mapping of feature model into variability and commonality and their relationship.	Contextual variability is not considered in this study.
Ali et al. [12]	2009	SPL's contextual variability	Cardinality of features in SPL.	Calculate the cardinality of each group of features in feature model. It do not calculate the total number of products.

(Continued)

Table 1 (continued)

Source	Year	Approach	Advantages	Disadvantages
Nadi et al. [13]	2014	Build-time faults and heuristic feature extraction	Application based product development by extracting faults of relationships.	Detect the faults only cardinality relationships of feature model but do not cover the crosstree constraints.
Noorian et al. [14]	2014	Transforms the feature model into a goal-oriented	Development of single product from the feature model	Only single product derivation is considered but not whole SPL.
Lee [15]	2015	XML and XSLT based variability modeling	Mapping of features and their relationship in XML and XSLT.	Do not cover crosstree constraints.
Cirilo et al. [16]	2013	DKML	GenArch+ tool is used in this work to enhance the specification of configuration information for the SPL feature model.	Update the feature model with new features at run time without crosstree constraints.
Fernandez-Amoros et al. [17]	2009	SIMPLE	Cost estimation of applications development of SPL.	Cost estimation of valid and invalid products of SPL.
Abbas et al. [6]	2017	BPNCC	Product configurations of SPL feature model and find the total number of products.	Binary oriented product configurations without crosstree constraint.

Through a literature review, it is concluded that this problem can be solved through combinatorial testing. This type of testing selects a subset of products that covers all possible interactions of features [6].

3 Material and Methods

All of the abovementioned approaches ignore the cross-tree constraints problem while using feature models that produce some invalid products. Thus, if they consider these constraints, they can reduce invalid configurations. Furthermore, they should have explored how we can automatically test the feasibility of products for their cross-tree constraints problems such as include and exclude. Our proposed algorithm overcomes these limitations and improves the correctness of feature selection. It helps to automatically memorize all the constraints through our new algorithm while using the feature model. Then, check these constraints among all products to get valid products. This approach reduces the development cost, effort, and time before SPL product development.

3.1 Complexity of Crosstree Constraints

The complexity of the feature model depends on the crosstree constraints of the feature model. CTCs include and exclude relationships among features and groups of the feature models. By increasing the CTCs in the feature model, more inclusive and exclude operations are performed that affect the other feature combinations of SPL. Developing complex systems that provide consumers

with various functions takes much work. The Challenge lies in providing many options for various application contexts with high versatility while restricting the customization of systems to achieve maintainability and growth management. The Feature model is an important contribution to dealing with invalid feature combinations by capturing and visualizing the similarities and dependencies between features and the components that provide the features. Feature models have been widely used in technical systems and as an element of implementing a line of software products for more than ten years. Table 2 shows the comparison of existing approaches. Typically, the feature model depicts a tree structure with various nodes known as features [19].

Table 2: Existing approaches comparison for managing variability with CTC

Approaches	FM tree relationship	CTC	Total number of products	Mapping of feature model
Web interface to construct syntactically and semantically Feature model Miguel Horcas et al. 2020 [1]	Yes	No	No	Yes
Extensible model driven engineering approach Shatnawi et al. 2020 [5]	Yes	No	Yes	Yes
Multi-Objective Optimization-Binary Pattern for Nested Cardinality Constraints Abbas et al. 2018 [20]	Yes	No	No	No
Binary Pattern for Nested Cardinality Constraints Abbas et al. 2017 [6]	Yes	No	Yes	Yes

Our proposed framework consists of two phases. In the first phase, we identify the valid and invalid features from the feature model according to the complexity of crosstree constraint problems. In the second phase, we drive the product configurations of SPL based on valid and invalid features.

3.2 Factors of Invalid Features

Valid and invalid features are based on the complexity of crosstree constraints. Valid features have low crosstree constraints, and invalid features have high crosstree constraints. Invalid features increase the probability of invalid product configurations. Table 3 shows the product configurations of the "Mobile Phone" feature model in Fig. 1. Table 3 consists of valid and invalid product configurations due to not considering the crosstree constraints. In Fig. 1, "GPS" and "Basic" features exclude each other, i.e., only one can be part of the product configuration. Therefore, Table 2 shows the invalid product configurations that consist of both "GPS" and "Basic," such as product numbers 3, 9, 15, and 21. Furthermore, the "Camera" requires "High Resolution", i.e., if any product configuration adds the camera in the final product derivation, then there must be a screen "High Resolution". All the products in Table 2 are invalid where the camera is one, and the high resolution is 0, such as 14, 15, 17, 20, 21, 23, and 24 are invalid. Therefore, we propose a framework that distinguishes the valid and invalid features of the feature model.

Table 3: Mobile phone SPL product configurations without considering crosstree constraints

Mobile phone products	Accuracy	Call	GPS	Basic	Color	High resolution	Camera	MP3
1	Valid	1	1	0	0	1	0	0
2	Valid	1	1	0	1	0	0	0
3	Invalid	1	1	1	0	0	0	0
4	Valid	1	0	0	0	1	0	0
5	Valid	1	0	0	1	0	0	0
6	Valid	1	0	1	0	0	0	0
7	Valid	1	1	0	0	1	0	1
8	Valid	1	1	0	1	0	0	1
9	Invalid	1	1	1	0	0	0	1
10	Valid	1	0	0	0	1	0	1
11	Valid	1	0	0	1	0	0	1
12	Valid	1	0	1	0	0	0	1
13	Valid	1	1	0	0	1	1	0
14	Invalid	1	1	0	1	0	1	0
15	Invalid	1	1	1	0	0	1	0
16	Valid	1	0	0	0	1	1	0
17	Invalid	1	0	0	1	0	1	0
18	Invalid	1	0	1	0	0	1	0
19	Valid	1	1	0	0	1	1	1
20	Invalid	1	1	0	1	0	1	1
21	Invalid	1	1	1	0	0	1	1
22	Valid	1	0	0	0	1	1	1
23	Invalid	1	0	0	1	0	1	1
24	Invalid	1	0	1	0	0	1	1

Violations of the given below factors lead to invalid product configurations:

- Or group relationships
- Alternative relationship
- Include crosstree constraints
- Exclude crosstree constraints
- One-to-One (optional to optional)
- One-to-many (optional to optional)
- One-to-One (optional to alternate)
- One-to-many (optional to alternate)
- One-to-One (optional to optional)
- One-to-many (alternate to alternate)
- One-to-one (alternate to alternate)

3.3 Types of Crosstree Constraints

There are two types of crosstree constraints in the feature model:

- Include (Require)
- Exclude (Reject)

The complexity of the feature model is based on the number of features in SPL and the crosstree constraint relationships. Increasing the features in the feature model gradually increases the crosstree constraint problems [21]. Therefore, this research focuses on all types of feature models, such as,

- Small feature model with fewer crosstree constraints
- Small feature model with maximum crosstree constraints
- Large feature model with fewer crosstree constraints
- Large feature model with maximum crosstree constraints

Furthermore, these crosstree constraints are categorized into One-to-One and One-to-Many. One-to-One crosstree constraint is simple due to the relationship between only two features. However, the One-to-Many crosstree constraint is complex due to the relationship of one feature with more than one feature that increases the dependency. These One-to-One and One-to-Many crosstree constraints further imply optional and alternative feature model groups, categorized as optional to optional and optional to an alternative.

4 Binary Oriented Feature Selections (BOFS)

The BOFS-CTC is a novel approach built on the binary combinations of features for crosstree (sub-tree), leaf, and parent node restrictions. The BOFS-CTC is a linear method for counting all feature model products without violating crosstree and cardinality restrictions. Additionally, this technique counts all products in a large feature model, with n backtrace nested constraints having zero violation of the constraints. Since terminal features (leaf nodes) are usable and obvious to end users, they are necessary for product derivation. Functional features known as terminal features are used to create SPL goods because they do not have any further child features. At the terminal, the product's benefits and real functionality are visible. All connections between parents of terminal features are represented by non-terminal features [6]. As a result, consider the connections between the constraints on the sub-tree and the terminal characteristics of each group (alternative, optional, OR).

4.1 BOFS-CTC Framework

The framework suggested a fresh and efficient method to count all SPL products, as shown in Fig. 3. OG is the number of optional features in one group, and OF is the number of optional features in any group. The required, optional, alternative, and OR groups make up the SPL feature model. All products must always have the required characteristics. However, varying features set the goods apart in the wide range of features. The six stages that make up this BOFS-CTC strategy:

- In the fourth stage, formulas corresponding to various variable groups use a backtrace tree structure to determine the products.
- The fifth step, which considers crosstree constraints of features, creates binary combinations of each group and its subgroups.
- The third stage entails dividing the crosstree constraints in Fig. 3 into the groups listed below:
- Optional to Optional.
  - One-to-One

- One-to-Two
- One-to-Three or more

- Alternate to Alternate.
  - One-to-Many
- Optional to Alternate and vice versa.
  - One-to-Many
- In the fourth stage, equations corresponding to various variable groups use a backtrace tree structure to determine the products.
- The fifth step, which considers crosstree constraints of features, creates binary combinations of each group and its subgroups.
- The final sexist stage is to count all potential products in the feature model.

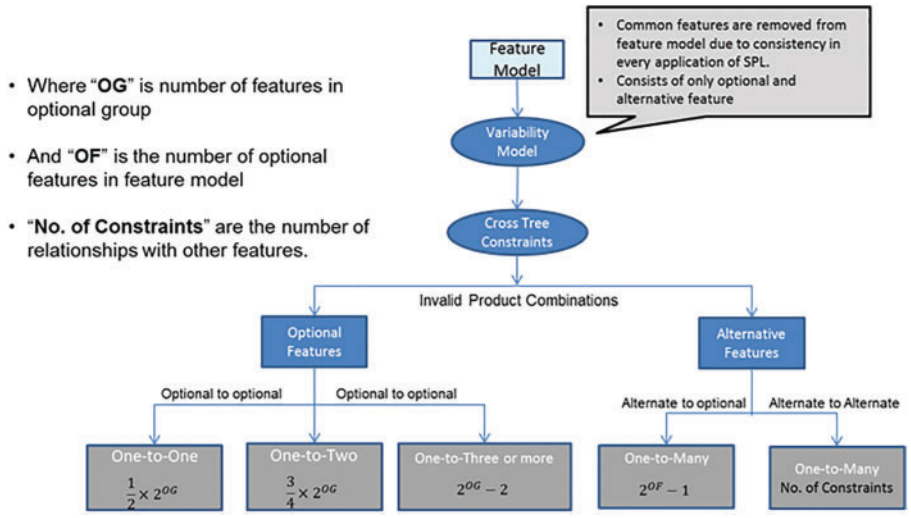

Figure 3: BOFS-CTC framework

4.2 BOFS-CTC Product Derivation

In the case of "one optional feature has the CTC with the single feature, " to find the invalid products from the feature model, we have derived the mathematical Eq. (1) "Accuracy Function" as given below:

$$Number\ of\ invalid\ products = \frac{1}{2} \times 2^{OG} \tag{1}$$

Here, OG shows the number of features in the OR group with constraints. Therefore, valid products from the OR group can be derived from Eq. (2).

$$Total\ valid\ products = \#P = 2^{n} - \left(\frac{1}{2} \times 2^{OG}\right) \tag{2}$$

where n is the total optional features that have CTC.

In the case of "one optional feature has CTC with two features of OR group", to find the invalid products from the feature model, we have derived the mathematical Eq. (3).

$$Number\ of\ invalid\ products = \frac{3}{4} \times 2^{OG} \tag{3}$$

Therefore, valid products from the OR group can be derived from the Eq. (4).

$$Total\ valid\ products = \#P = 2^n - \left(\frac{3}{4} \times 2^{OG}\right) \tag{4}$$

In the case of "one optional feature has CTC with three or more features of OR group", to find the invalid products from the feature model, we have derived the mathematical Eq. (5).

$$Number\ of\ invalid\ products = 2^{OG} - 2 \tag{5}$$

Therefore, valid products from the OR group can be derived from Eq. (6).

$$Total\ valid\ products = \#P = 2^n - (2^{OG} - 2) \tag{6}$$

In the case of "Alternate to optional (One-to-many)", to find the invalid products from the feature model, we have derived the mathematical Eq. (7), and for all valid products, we have derived Eq. (8).

$$Number\ of\ Invalid\ Products = (2^{OF} - 1) \tag{7}$$

$$Total\ valid\ products = 2^{OF} \times A - (2^{OF} - 1) \tag{8}$$

where OF is the number of optional features, A is the number of alternate features. Eq. (7) calculate the invalid products of CTC between alternate and optional OR group. Eq. (8) evaluates the total number of valid products. In the case of "alternate to alternate (one-to-many)", to find the invalid products from the feature model, we have derived the mathematical Eq. (9).

Invalid products = #constraints are applied on the alternate group of features as only one feature is selected among n number of features.

$$Total\ valid\ products = n \times n - invalid\ products \tag{9}$$

4.3 BOFS-CTC Algorithm

In this paper, the BOFS-CTC algorithm is developed to automatically generate product feature combinations in binary form, whereby characteristics selected are denoted by one and those not chosen by 0. BOFS-CTC algorithm consists of six modules and one main module that calls the other six modules, as given below.

The first module of BOFS-CTC structured a tree known as the feature model, where root, parent and chilled nodes with their name have been defined. This module requires the data set of features, and their cardinality relationships such as mandatory, optional, alternate, and OR group.

Algorithm for Valid Features

```
import random
from anytree import Node, RenderTree, render, AsciiStyle
from anytree.exporter import DotExporter
# Creating tree structure
A = Node("Mobile") # root
B = Node("Mandatory", parent = A)
C = Node("Optional", parent = A)
D = Node("c", parent = B)
```

```
E = Node("Screen", parent = B)
F = Node("GPS", parent = C)
G = Node("Media", parent = C)
H = Node("Basic", parent = E)
I = Node("Color", parent = E)
J = Node("High Resolution", parent = E)
K = Node("Camera", parent = G)
L = Node("MP3", parent = G)
```

In the second module, a list is generated of features that an SPL domain contains according to their specific groups and relationships. As mentioned, the features name of the mobile feature model is given below.

```
# Defining lists of features
    Screen = ["Basic","Color", "High Resolution"]
    Media = ["Camera", "MP3"]
    Mandatory = ["Calls", "Screen"]
    Optional = ["GPS", "Media"]
```

Mandatory features are always part of the product; however, constraints can also exist in leaf nodes of mandatory features. Therefore, the third module deals with the mandatory features where an alternate relationship exists. In the given bellow module, only one feature can be part of the product configuration from the three mandatory alternate features (Basic et al. Resolution).

```
# Define function to display Mandatory Features
    def display_mandatory_features(Mandatory, Screen, select):
        print("Mandatory Features for Product: ", Mandatory [0])
        print("Mandatory Features for Product: ", Mandatory [1])
        print("Select Screen Type: ", Screen[select])
```

Forth module deals with optional features that may or may not be part of product configuration. Therefore, it has only two options (1) select, i.e., 1, and (2) not selected, i.e., 0. The given bellow module is applied on optional group media of mobile feature model where parent node media consists of further two leaf nodes MP3 and camera.

```
# Define function to display Optional Features
    def display_optional_features (Optional, Media, select1):
        print("Optional Features for product:", Optional [0])
        print("Optional Features for product:", Optional [1])
        if select1 == 0:
            print("Selected Optional Feature: ", Optional [select1])
        elif select1 == 1:
            print("Selected Optional Feature: ")
            print("Media Types: ", Media [select1])
```

```
print("Media Types: ", Media [0])
```

In the fifth and last module, input the crosstree constraints, including and excluding features; if any configurations violate the crosstree constraints, that configuration is excluded from the total number of products. This process generates final valid feature combinations for the whole SPL domain. Therefore, get all valid features a combination without any cardinality relationship violation and crosstree constraints.

```
# Define function to display total features and selected features count
    def display_count_plot(O_count, T_count, T_M_count, T_S_count):
        S_M_count = 2
        selected = S_M_count + O_count
        print("Total Features:", T_count)
        print("Selected Features:", selected)
        import matplotlib.pyplot as plt
        left = [1,2,3,4]
        height = [T_M_count, T_S_count, T_count, selected]
        tick_label = ['Mandatory', 'Optional', 'Total Features', 'Selected Features']
        plt.bar(left, height, tick_label = tick_label, width = 0.8, color=['blue', 'red'])
        plt.xlabel('Labels')
        plt.ylabel('Count')
        plt.title('Features Modeling')
        plt.show()
# Define main function to call all functions
    def main_function():
        print(RenderTree(A, style = AsciiStyle()))
        display_mandatory_features(Mandatory, Screen, random.randint(0, 2))
        display_optional_features(Optional, Media, random.randint(0, 1))
        display_count_plot(random.randint(1, 2), 7, 4, 3)
# Call main function
    main_function()
```

Previously proposed algorithms have been applied to the mobile phone feature model in Fig. 1 and get 24 product configurations where some invalid configurations were also generated due to crosstree constraints, as shown in Table 3. Therefore, BOFS-CTC is applied to the same feature model with cardinality and crosstree constraints and has 14 product configurations. From Table 3, BOFS-CTC removed ten invalid product configurations, as shown in Table 4. To verify the valid product configurations in Table 3, use the relationships below. GPS has no relationship (exclude) with Basic such as "GPS¬Basic," where GPS is selected, i.e., GPS = 1, then Basic should not be selected, i.e., Basic = 0. GPS can be selected where the screen must be color or high resolution. The other CTC of the camera requires a high-resolution screen; if camera = 1, then the high resolution must be 1. These CTC are satisfied. Therefore, all 14 products are valid in Table 4.

Table 4: Mobile phone feature model valid product configurations

Mobile phone products	Call	GPS	Basic	Color	High resolution	Camera	MP3
1	1	1	0	0	1	0	0
2	1	1	0	1	0	0	0
3	1	0	0	0	1	0	0
4	1	0	0	1	0	0	0
5	1	0	1	0	0	0	0
6	1	1	0	0	1	0	1
7	1	1	0	1	0	0	1
8	1	0	0	0	1	0	1
9	1	0	0	1	0	0	1
10	1	0	1	0	0	0	1
11	1	1	0	0	1	1	0
12	1	0	0	0	1	1	0
13	1	1	0	0	1	1	1
14	1	0	0	0	1	1	1

4.4 Experimental Work

A comparative study is performed of BOFS-CTC with previously proposed approaches in the literature, such as COVAMOF, GenArch+, Common Variability Language (CVL), and BPNCC, as shown in Table 4. A comparative study is based on major parameters defining the proposed approaches' working and accuracy. These proposed approaches calculate and generate the total number of SPL products. Table 5 indicates that BOFS-CTC is more appropriate and covers all the major parameters used to generate all product configurations. The previously proposed approaches do not consider the crosstree constraints during the product configurations; however, BOFS-CTC generates binary combinations with the single-level, nested, and crosstree constraints. Therefore, BOFS-CTC is the best approach to calculate and generate the binary combinations of SPL product configurations.

Table 5: BOFS-CTC comparison with other proposed approaches based on feature model level

Approaches	CTC	Binary combinations	Nested constraints	Single level constraints
COVAMOF	No	Yes	No	Yes
GenArch+	No	No	No	Yes
CVL	No	Yes	No	Yes
BPNCC	No	Yes	Yes	Yes
BOFS-CTC	Yes	Yes	Yes	Yes

BOFS-CTC is applied to small and large feature models with different relationships and limitations. Table 6 shows the results of a total number of valid products by considering all the feature

model's basic relationships and crosstree constraints. Results show that the crosstree constraints significantly affect the total number of valid products. If the crosstree constraints are not considered, the total number of products is higher than the given products due to invalid product combinations. Therefore, BOFS-CTC is more effective and accurate for all feature models, such as small, large, simple, and complex (nested cardinality constraints, crosstree constraints).

Table 6: BOFS-CTC Applied on small and large feature models

Feature models	No. of features	Mandatory	Optional	XOR	OR	Grouped	CTC	# Valid products
Web content delivery	15	1	4	3	1	9	6	23
Delay block semantics specification	23	8	7	1	0	7	20	41
Epic slice machine	32	7	4	0	6	20	9	275352
Sale computers Specification	38	0	2	10	1	35	23	12088
Route finder feature model	51	10	1	7	11	39	6	9997020
Smart home	78	38	27	1	4	14	10	14480162

5 Conclusions and Future Work

SPL is a successful strategy for resource reuse. The commonalities and variable characteristics of SPL are managed using a feature model. For an organization to implement SPL, cost estimates for the entire SPL and individual applications are crucial knowledge. The budget anticipated to be needed for creating all SPL products is identified by cost estimation. Organizations must compute the budget before adaptation to ascertain whether or not a specific product line is within budget. The total number of items is the main factor that cost-estimating models consider. Developing applications using all feature combinations also allows for calculating each application's functional and non-functional characteristics and selecting and selecting features. The cardinality of each group and crosstree constraints are required by the BOFS-CTC method. In order to determine the overall number of products and solutions of all combinations, it finally combines all groups. We have demonstrated the complete correctness of BOFS-CTC without violating the constraint (cardinality and crosstree) in the complicated constrained feature model. The conclusions are validated because separately applied BOFS-CTC for total numbers and total solutions of feature combinations are equivalent.

Future work will focus on choosing the best optimization features for each application by using binary patterns and considering crosstree constraints. We will optimize speed and quality while optimizing the minimal parameters, such as cost and memory utilization, following end-user needs. Additionally, we will improve every product's functional and non-functional aspects.

Acknowledgement: The authors would like to thank Dr Asad Abbas (Assistant Professor), Department of Computer Science & Information Technology, University of Central Punjab, Lahore, Pakistan for providing us wonderful technical support in the research. The authors acknowledge the MLC Research Lab, Okara for their important and fruitful comments to enhance the quality of the current article.

Funding Statement: Princess Nourah bint Abdulrahman University Researchers Supporting Project Number (PNURSP2023TR140), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: M. F. Afzal, I. Khan, J. Rashid; data collection: M. F. Afzal, M. Saddique; analysis and interpretation of results: H. G. Mohamed, J. Rashid. I. Khan; draft manuscript preparation: M. F. Afazl, J. Rashid, M. Saddique, H. G. Mohamed. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Data (Feature Model SPLOT) will be provided on request.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] T. Bordis, T. Runge, A. Knüppel, T. Thüm and I. Schaefer, "Variational correctness-by-construction," in *Proc. of the 14th Int. Working Conf. on Variability Modelling of Software-Intensive Systems*, Magdeburg, Germany, pp. 1–9, 2020.

[2] R. Lindohf, J. Krüger, E. Herzog and T. Berger, "Software product-line evaluation in the large," *Empirical Software Engineering*, vol. 26, no. 2, pp. 1–41, 2021.

[3] V. M. Le, T. N. T. Tran and A. Felfernig, "Consistency-based integration of multi-stakeholder recommender systems with feature model configuration," in *Proc. of the 26th ACM Int. Systems and Software Product Line Conf.*, Graz, Austria, pp. 178–182, 2022.

[4] A. Abbas, I. F. Siddiqui and S. U. J. Lee, "Contextual variability management of IoT application with xml-based feature modeling," *Journal of Theoretical and Applied Information Technology*, vol. 95, no. 6, pp. 1300–1308, 2017.

[5] D. Benavides, S. Segura, P. Trinidad and A. R. Cortés, "FAMA: Tooling a framework for the automated analysis of feature models," *VaMoS*, vol. 1, pp. 1–6, 2007.

[6] A. Abbas, I. F. Siddiqui, S. U. J. Lee and A. K. Bashir, "Binary pattern for nested cardinality constraints for software product line of IoT-based feature models," *IEEE Access*, vol. 5, pp. 3971–3980, 2017.

[7] H. Hartmann and T. Trew, "Using feature diagrams with context variability to model multiple product lines for software supply chains," in *Proc. of the 2008 12th Int. Software Product Line Conf.*, Washington DC, USA, pp. 12–21, 2008.

[8] V. Cechticky, A. Pasetti, O. Rohlik and W. Schaufelberger, "Xml-based feature modelling," in *Proc. of the 8th Int. Conf. Software Reuse (ICSR)*, Madrid, Spain, pp. 101–114, 2004.

[9] Y. C. Cavalcanti, I. do Carmo Machado, P. A. da Mota, S. Neto, L. L. Lobato *et al.,* "Towards metamodel support for variability and traceability in software product lines," in *Proc. of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, New York, USA, pp. 49–57, 2011.

[10] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J. C. Royer *et al.,* "A model-driven traceability framework for software product lines," *Software & System Modeling*, vol. 9, no. 4, pp. 427–451, 2010.

[11] P. Shaker, J. M. Atlee and S. Wang, "A feature-oriented requirements modeling language," in *Proc. of the 2012 20th IEEE Int. Requirements Engineering Conf. (RE)*, Chicago, USA, pp. 151–160, 2012.

[12] R. Ali, Y. Yu, R. Chitchyan, A. Nhlabatsi and P. Giorgini, "Towards a unified frame-work for contextual variability in requirements," in *Proc. of the 2009 Third Int. Workshop on Software Product Management*, Atlanta, USA, pp. 31–34, 2009.

[13] S. Nadi, T. Berger, C. Kästner and K. Czarnecki, "Mining configuration constraints: Static analyses and empirical results," in *Proc. of the 36th Int. Conf. on Software Engineering*, Hyderabad, India, pp. 140–151, 2014.

[14] M. Noorian, E. Bagheri and W. Du, "From intentions to decisions: Understanding stakeholders' objectives in software product line configuration," in *Proc. of the 26th Int. Conf. on Software Engineering*, Washington DC, USA, pp. 671–677, 2014.

[15] S. U. J. Lee, "An effective methodology with automated product configuration for software product line development," *Mathematical Problems in Engineering*, vol. 2015, no. 5, pp. 435316, 2015.

[16] E. Cirilo, U. Kulesza, A. Garcia, D. Cowan, P. Alencar *et al.,* "Configurable software product lines-supporting heterogeneous configuration knowledge," in *Proc. of the 13th Int. Conf. on Software Reuse*, Pisa, Italy, pp. 176–191, 2013.

[17] D. Fernandez-Amoros, R. H. Gil and J. C. Somolinos, "Inferring information from feature diagrams to product line economic models," in *Proc. of the 13th Int. Software Product Line Conf.*, California, USA, pp. 41–50, 2009.

[18] D. Fernandez-Amoros, R. Heradio, J. A. Cerrada and C. Cerrada, "A scalable approach to exact model and commonality counting for extended feature models," *IEEE Transactions on Software Engineering*, vol. 40, no. 9, pp. 895–910, 2014.

[19] C. Sundermann, M. Nieke, P. M. Bittner, T. Heß, T. Thüm *et al.,* "Applications of SAT solvers on feature models," in *Proc. of the 15th Int. Working Conf. on Variability Modelling of Software-Intensive Systems*, Krems, Austria, pp. 1–10, 2021.

[20] A. Abbas, I. F. Siddiqui, S. U. J. Lee, A. K. Bashir, W. Ejaz *et al.,* "Multi-objective optimum solutions for IoT-based feature models of software product line," *IEEE Access*, vol. 6, pp. 12228–12239, 2018.

[21] M. Bhushan, J. A. G. Duarte, P. Samant, A. Kumar and A. Negi, "Classifying and resolving software product line redundancies using an ontological first-order logic rule based method," *Expert Systems with Applications*, vol. 168, no. 6, pp. 114167, 2021.