A Novel Valid Features Combination Approach for Product Derivation in Software Product Line

Ph.D Thesis

By

Muhammad Fezan Afzal

23-FBAS/PHDSE/F16

Supervisor

Dr. Imran Khan

Assistant Professor, Department of Computer Science, IIUI.

Co-Supervisor

Dr. Asad Abbas

Assistant Professor, University of Central Punjab.

Department of Software Engineering, Faculty of Computing

International Islamic University, Islamabad

January 2024

PhD
OOS 15
AFN

Software documentation
Software product Line
Valued Features

A dissertation submitted to the
Faculty of Computing and Information Technology,
International Islamic University, Islamabad
as a partial fulfillment of the requirements
for the award of the degree of
Doctor of Philosophy in Software Engineering.

INTERNATIONAL ISLAMIC UNIVERSITY ISLAMABAD
FACULTY OF COMPUTING & INFORMATION TECHNOLOGY
DEPARTMENT OF SOFTWARE ENGINEERING

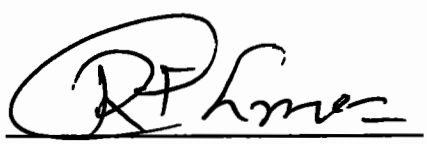
Date: 19-01-2024

Final Approval

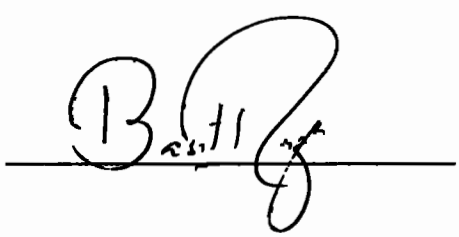
It is certified that we have read this thesis, entitled ": A Novel Valid Features Combination Approach for Product Derivation in Software Product Line "submitted by **Muhammad Fezan Afzal Registration No. 23-FBAS/PHDSE/F16**. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by the International Islamic University Islamabad for the award of the degree of PhD in Software Engineering.

Committee

External Examiner:
Dr. Arif Ur Rehman,
Professor
Bahria University, Islamabad

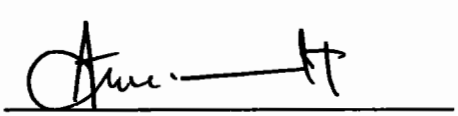


External Examiner:
Dr. Basit Raza,
Associate Professor
COMSATS University, Islamabad

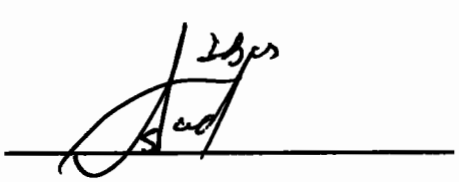


Internal Examiner:
Dr. Syed Muhammad Saqlain,
Assistant Professor,
Department of Computer Science,
FoC, IIUI

Supervisor:
Dr. Imran Khan ,
Assistant Professor,
Department of Computer Science,
FoC, IIUI



Co-Supervisor:
Dr. Asad Abbas,
Assistant Professor
University of Central Punjab, Lahore



Declaration

I hereby declare that this thesis, neither as a whole nor any part thereof has been copied out from any source. It is further declared that no portion of the work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Muhammad Fezan Afzal

Dedication

This thesis is dedicated to my family, especially to my father Rana Muhammad Afzal Khan, mother Sidra Beghum, wife Saira Fezan, sisters Aroosa Afzal, Arooj Afzal, Amreena Afzal and brothers Mohsin Jabran, Ahsan Arslan.

Muhammad Fezan Afzal

Acknowledgments

This thesis and all my efforts are fruitful only due to ALLAH Almighty, the Most Merciful and Beneficent, Who gave me strength to complete this task to the best of my abilities and knowledge.

I would like to thank my supervisor *Prof. Dr. Imran Khan* and Co-Supervisor *Prof. Dr. Asad Abbas*, who gave all their knowledge, guidance and support to boost my confidence and learning. I would also like to thank my wife who has supported me patiently and firmly during completion of my task.

I would also like to acknowledge my brothers, friends, students and colleagues especially *Prof. Dr. Shahbaz Ahmed Khan Ghayyor, Prof. Dr. Imran Khan, Prof. Dr. Asad Abbas, Prof. Dr. Salma Imtiaz chairperson Department of Software Engineering, Prof. Dr. Mudasar Ghafoor, Prof. Dr. Hafiz Abid Masood, Prof. Imran Saeed, Dr. Muhammad Shabir Kallu, Prof Dr. Nasir Ali, Prof. Dr. Muhammad Azmat (NUST), Dr. Rana Javaid Rashid, Mr. Rana Muhammad Ashfaq, Mr. Hafiz Saleh, Mr. Muhammad Arshad, Mr. Numan Arshad, Mr. Dawood Shabeer, Mr. Usman Haider Kallu* and *Higher Education Commission (HEC), Pakistan.* All of them encouraged and provided logistic and technical help during this research.

I would like to admit that I owe all my achievements to my truly, sincere and most loving parents and friends who mean the most to me, and whose prayers have always been a source of determination for me.

Abstract

Software Product Line (SPL) is a group of software-intensive systems that share common and variable resources for developing a particular system. SPL epitomizes the notion of a planned reuse. It encourages the development of a series of software applications by reusing basic functionality to the maximum extent. All the products part of SPL are technically called "features". Features can be represented through a compact graphical format called a feature diagram. This visual representation is further termed a feature model. The feature model is a tree-type structure used to manage SPL's common and variable features with their different relations and problems of Crosstree Constraints (CTC). Common features exist in every SPL product, while variable features are part of the product according to application requirements, constraints, and relationships. Therefore, invalid feature combinations can be generated due to constraints, and relationships between varied features resultantly make this process complex and consume extra effort while developing applications in SPL. This happens due to the need for better algorithms working when implementing cross-tree constraints.

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. Numerous methods are available to cope with the complexity and extra effort when selecting features for specific product derivation. The selected subset of products has both valid and invalid configurations. That is why complexity and effort are increased during the development of SPL. Therefore, managing the CTC problem to achieve valid product configuration in IoT-based SPL is more complex, time-consuming, and hard. In literature, multiple algorithms are proposed for selecting features from the feature model for product derivation. However, proposed algorithms only consider the cardinality constraints of feature models such as OR group and alternative. However, the CTC problems are not considered in previously proposed approaches such as Commonality Variability Modeling of Features (COVAMOF) and Genarch+ tool; therefore, invalid products are generated due to the violation of feature selection constraints.

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 and applied the BOFS-CTC algorithm 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. In its first step, an intensive literature review is conducted to understand the working and weak areas of existing feature models. Secondly, based on literature findings, understand the limitations of existing algorithms used to calculate the valid total number of products of the SPL feature model with primary cardinality constraints. In the last phase, the proposed algorithm will be developed to calculate the valid total number of products by considering the cross-tree constraints of the feature model.

Furthermore, we will validate our results using the "simple random sampling" technique, where random products (combination of features) will be chosen from different small and large feature models. Validation will be based on comparing manually generated combinations and system-generated results. For the development of SPL, organizations require advanced investment in domain engineering. 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 to be reasonable correctness. Using BOFS-CTC reduces the testing cost of SPL as invalid products are removed from the total number of products. Eliminates the testing cost and development effort of invalid SPL products.

Contents

1 Introduction 1

1.1 Objectives and Scope 5

2 Literature Review 7

2.1 Valid Products of SPL 7

2.2 Testing Efforts of SPL Products 8

2.3 Research Gap 18

2.4 Problem Statement 19

2.5 Research Questions 19

3 Binary Oriented Feature Selection Crosstree Constraints (BOFS-CTC) 20

3.1 Material and Methods 20

3.2 Complexity of Crosstree Constraints 21

3.3 Factors of Invalid Features 22

3.4 Types of Crosstree Constraints 24

3.5 Binary Oriented Feature Selections (BOFS) 24

3.5.1 BOFS-CTC Framework 25

3.6 BOFS-CTC Product Derivation 26

3.6.1 BOFS-CTC Algorithm 28

3.6.2 BOFS-CTC Comparison 31

4 Binary Oriented Feature Selection-Crosstree Constraints Validation 33

4.1 Feature Model Optional Selection 34

4.2 Alternate Feature Model Optional Selection 41

4.3 Increase Alternate Feature Model Optional Selection 45

5 Results and Analysis **52**
5.1 Impact of Managing Crosstree Constraints 53
5.2 Applications of CTC 55
5.3 BOFS-CTC Feature Models 58

6 Conclusions and Future Directions **77**
6.1 Potential Impact of Research 78

List of Figures

1.1 Mobile Phone Feature Model [19] 3
1.2 Cost estimation of SPL and single product 4

3.1 BOFS-CTC framework 26

4.1 CTC of One optional to two optional features 34
4.2 Crosstree constraints with one optional to two optional 34
4.3 Valid and Invalid Products indicates table 4.1 35
4.4 Four Optional Features with One-to-Two CTC 36
4.5 Valid and Invalid Product configurations indicates table 4.2 37
4.6 Five Optional Features with One-to-Two CTC 37
4.7 Valid, Invalid Configurations from table 4.3 39
4.8 Six Optional Features with One-to-Two CTC, Valid, Invalid Configurations 39
4.9 Six Optional Features Valid and Invalid Configurations from table 4.4 41
4.10 Two Alternate Features with One-to-Two CTC Feature Model 42
4.11 Valid and Invalid Configurations from table 4.5 44
4.12 Two Alternate Features with One-to-Two CTC, Four optional Feature Model 44
4.13 Valid and Invalid Configurations from table 4.6 45
4.14 Three Alternate Features with One-to-Three CTC Optional 45
4.15 Valid and Invalid Configurations from table 4.7 47
4.16 Feature model of n alternate and n optional features 48
4.17 Valid and Invalid product configurations from fig 4.18 49
4.18 Four optional features with one-to-three optional CTC 49
4.19 Four Alternate and Four optional features with one-to-three optional CTC 49
4.20 Valid and Invalid product configurations from fig 4.19 50
4.21 Four Alternate and Four optional features with one-to-three optional CTC 50

4.22 Valid and Invalid product configurations from fig 4.21 50
4.23 Three Alternate and six optional features with one-to-three optional CTC 51
4.24 Valid and Invalid product configurations from fig 4.23 51

5.1 Mobile Phone Feature Model 57
5.2 Algorithm for require and exclude 57
5.3 Eight feature models with CTC and without CTC 60
5.4 CTC based Feature Models with 20, 31, 32 63
5.5 Complex and large feature models with and without CTC 64
5.6 Four feature models dataset 65
5.7 Feature model of fifty feature with CTC and basic relationships 67
5.8 Small feature model with large CTC 68
5.9 Large feature model with less CTC (1) 69
5.10 Large feature model with less CTC (2) 70
5.11 Large feature model with less CTC (3) 71
5.12 Large feature model with less CTC (4) 72
5.13 Large feature model with less CTC (5) 73
5.14 Large feature model with mediam size of CTC 74
5.15 Mediam size feature model with large CTC 75

List of Tables

2.1 Comparison of existing approaches 11
2.2 Analysis of litrature with respect invalid feature combination 12
2.3 Valid and invalid feature combinations due to crosstree consraints 13
2.4 Problems identified due to invalid feature combinantions 14
2.5 Method/Approach followed in the wake of invalid feature combinations 16
2.6 Mapping SPL issues with SDLC phases 17

3.1 Existing approaches comparison for managing variability with CTC 21
3.2 Mobile phone SPL product configurations without considering crosstree constraints 23
3.3 Mobile phone feature model valid product configurations 30
3.4 BOFS-CTC Comparison with other proposed approaches based on feature model level 31
3.5 BOFS-CTC Applied on small and large feature models 32

4.1 Valid and invalid product configurations of Figure 4.2 35
4.2 Valid and Invalid Product configurations from Figure 4.4. 36
4.3 Valid and Invalid Product configurations from Figure 4.6. 38
4.4 Invalid Product with six optional features one-to-one CTC from fig 4.8 40
4.5 Combination of two alternate and thre optional from fig. 4.10 42
4.6 Two Alternate Features with One-to-Two CTC, Four optional from fig 4.12 43
4.7 Three Alternate Features with One-to-Two CTC, Four optional from fig 4.14 46

5.1 Valid Product Configurations by using BOFS-CTC Algorithm 59
5.2 BOFS-CTC algorithm results of 20, 31 and 32 features of feature model 66
5.3 BOFS-CTC Valid configuration of 50 to 60 features 76

Chapter 1

Introduction

Software applications have become central to today's age and daily life. Irrespective of age. sex, profession, or geographical boundaries, all of us depend upon software applications. This dependency may be direct or indirect in any sense. Many professionals, from doctors to engineers, parents to students, buyers to suppliers, and manufacturers to service providers, depend entirely on software [1, 2]. However, it is disappointing that even though heavy amounts are spent on software applications to make them accurate, efficient, secure, and reliable, vulnerabilities and defects remain. This is not a one-time investment; instead, it is a continuous investment that will be exceeded yearly to fix the flaws and overcome software vulnerabilities [3, 4].

According to the National Institute of Standards and Technology (NIST) 2002 research, the annual software error cost is approximately $59.5 billion. This figure is estimated just for the United States. This situation worsens daily as software gets more complex, has numerous features, and goes online. On the other hand, IT companies are reducing their budgets each year to overcome their research, development, and maintenance costs [5, 6].

Due to the increased expenses, organizations are trying to overcome their issues related to the need for more resources. IT companies are trying to improve productivity, enhance quality, decrease cost, decrease labor needs, decrease the time required to market the product and reduce the time necessary to cater to the market. Domain engineering or product line development is critical in systematic software reuse. It is the whole process of reusing domain knowledge to produce new software systems [7, 8].

Software Product Line (SPL) is a paradigm for developing and managing internal software sys-

tems from a common set of resources using a specific production process. It is a technique used to create software products with similar characteristics that share the exact nature of code, experience, and developer documentation [9]. Specifically, SPL is a group of software-intensive systems with common manageable characteristics. These functions are combined to meet specific market requirements. SPL improves the reuse of existing resources, i.e., common and variable features, and reduces development time, cost, effort, and time to market [9, 10].

SPL is a group of related products; features are the characteristics of a program, and the feature model manages these features. The feature model is used to manage SPL's common and variable features [11]. Therefore, the feature model has become one of the most used in the SPL community to develop the software family. A feature is an option to include specific functions in the system configuration. Features can be presented in a compact graphical format called a feature diagram. This visual tree-type structure is also called a feature model [12]. The feature model manages generic and variable SPL functions with various relationships and constraints. During application development, products are developed by selecting features from the SPL domain [13, 14]. Fig. 1.1 shows the feature model of mobile phones.

SPL consists of two processes: 1) Domain and 2) Application engineering. Domain engineering consists of all possible common and variable features of specific SPL. Furthermore, the feature model is a tree-type structure that shows the domain of SPL with all common and variable features [[15, 16]. Common features are part of every SPL product; however, variable features are selected according to user requirements and pre-defined relationships between features. Therefore, the selection of variable features differentiates the products of SPL [17].

The feature model is composed of pre-defined relationships between features, as shown below [8, 18]:

- Required. These are the standard features that must be part of every SPL product. In Fig. 1.1, a call is the common feature always selected in every product.
- Optional. These features may or may not be part of the SPL product. The selection of these features is according to end-user requirements. In Fig. 1.1. "GPS" is an optional feature.
- Alternative. The group features from where one and only one feature can be selected for the product development of SPL. If there are more than two features in an alternative group, then only one feature can be selected for the product derivation of SPL. In Fig 1.1, there are three features, "basic, color, and high resolution," on the screen; we can select only one feature at a time in the development of the product.

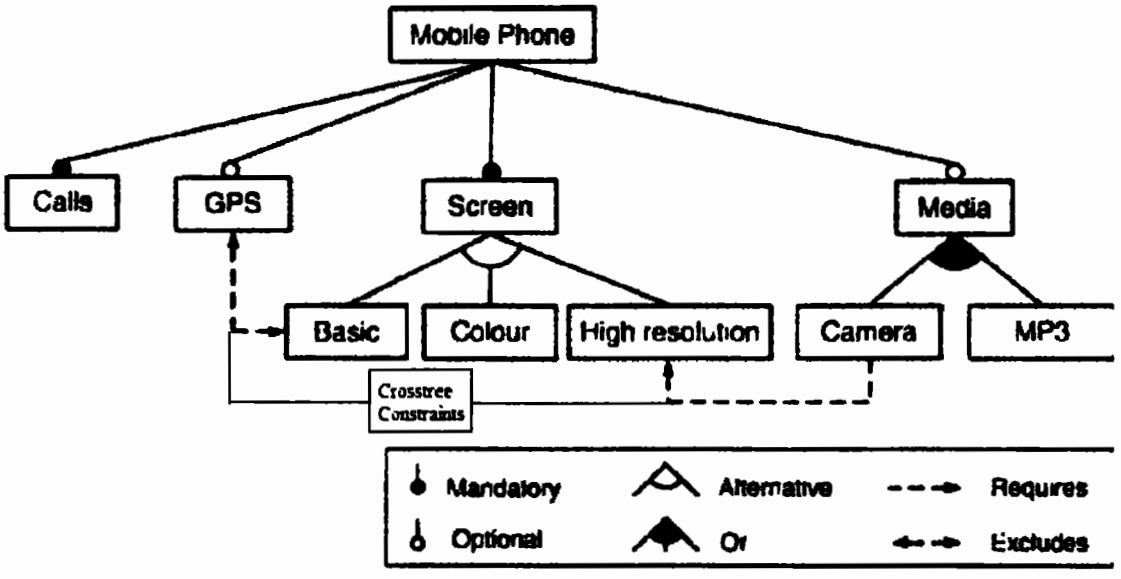


Figure 1.1: Mobile Phone Feature Model [19]

- Or group. A collection of child features is associated with its parent, and more than one feature can be selected for SPL product development. In Fig 1.1. the camera MP3 has the Or group relationship, where one or both can be set in the Product development of SPL.

The pre-defined relationships of feature models, such as optional, alternative, and Or group, are defined by every feature model. However, another relationship or constraint, known as the crosstree constraint, is also part of the feature model. Crosstree constraints are the relationships between sub-trees of the feature model. There are two types of crosstree constraints: 1) include features and 2) exclude features [19], as shown in Fig 1.1. "GPS" has the exclude crosstree constraint with the "Basic screen," and "camera" includes the "high-resolution screen."

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. 1.2, indicating that SPL organizations invest in initial development costs without benefiting from the market [20, 21]. The break-even point of SPL shown in Fig. 1.2 depends on the size of SPL, i.e., the total number of product configurations. The total valid number of products is a significant parameter for the advanced cost estimation of SPL [23, 24]. 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

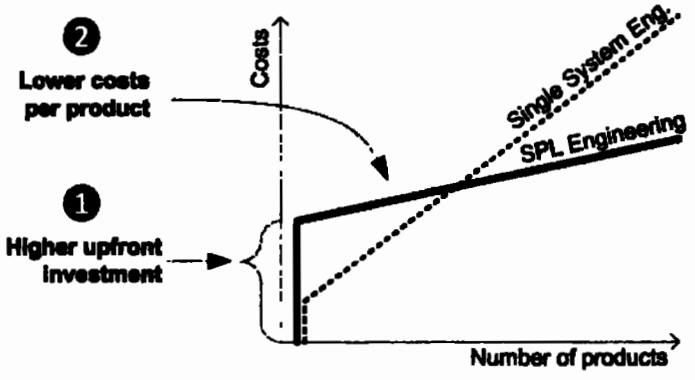


Figure 1.2: Cost estimation of SPL and single product

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 primary 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 [25, 26].

There are multiple research problems in SPL such as aspect oriented domain for the multithreading of software and crosscutting concerns. However, these problems are undertaken at the time of development of SPL applications and need to solve at the time of product development. It is more important to solve the problem of crosstree constraints at domain level of SPL. Therefore, this research solve the crosstree problems in feature model of software product line. Cross-tree constraints provide feature models with maximum expressive power as they enable the capture of any interdependency between features through arbitrary propositional logic formulas. However, the presence of these constraints adds complexity to the process of reasoning about feature models. whether using SAT solvers or compiling the model into a binary decision diagram for efficient analyses. While certain efforts have attempted to streamline constraints by eliminating them, these approaches typically focus on simple constraints like "requires" and "excludes," or they necessitate the introduction of an extra set of features, thereby raising the overall complexity of the resulting feature model [27]. Conventional methods for analyzing feature models rely on addressing algorithmic challenges like boolean satisfiability, satisfiability modulo theories, or integer linear programming. While these existing approaches effectively handle small and medium-sized problem instances, challenges arise when dealing with the scalability of large-sized feature models.

Quantum computers offer the potential for super-polynomial speedups in solving specific algorithmic problems, presenting an opportunity to overcome the scaling issues observed in the analysis of larger feature models [28].

The first problem is crosstree constraints in the IoT-based feature model, which cause invalid feature combinations to become part of SPL, leading to extra effort and cost in developing SPL. As shown in Fig.1.1, 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. 1.1 shows "mobile phone" SPL where ten products are invalid due to the crosstree constraints problem. It is essential 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 research proposed a novel binary-oriented feature selection crosstree constraint (BOFS-CTC) approach that calculates the valid feature product combinations by considering IoT devices' primary and nested cardinality and crosstree constraints. BOFS-CTC applies for small and large feature models with low to high complexity 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 digits of products and found high accuracy. However, the previous approaches need to consider the crosstree constraints problem to get valid products. In this thesis, different methods 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. As a result, using BOFS-CTC minimizes the total cost and effort of SPL product development. Furthermore, BOFS-CTC is the independent approach of any specific tool as we have proposed its algorithm.

1.1 Objectives and Scope

The aims of our research are:

- To highlight the strength and weaknesses of existing models presenting their work towards

Invalid Feature Combinations, we analyze the methodology and approach researchers used in their work.

- To find the invalid product combinations of SPL.
- To derive the valid product configurations of SPL with the selection of features
- To identify the valid feature combinations for each product derivation.
- To find the cost for each SPL using valid feature combinations and a total number of valid products.

We will apply the "simple random sampling" technique to validate our proposed solution. The simple random sampling technique is based on randomly selecting products from the population. In a simple random sampling technique, each product has an equal chance of occurrence from the population. Therefore, our validation of results will be based on the random selection of manually generated products from small and large feature models and will be compared with system-generated results. We have also driven all invalid products and reached out to them with the list of all products (combination of features); the absence of invalid products validated the correctness of our proposed solution.

This thesis is structured as Chapter 2 is Literature Review, Chapter 3 is Binary Oriented Feature Selection Crosstree Constraints (BOFS-CTC), Chapter 4 is Binary Oriented Feature Selection-Crosstree Constraint Validation, Chapter 5 is Result and Analysis and Chapter 6 is Conclusions and Future Directions.

Chapter 2

Literature Review

2.1 Valid Products of SPL

The development of a software product line consists of two life cycles: domain engineering and application engineering. In domain engineering, all possible common and variable features under the SPL domain exist. In application engineering, the common and variable features are selected from the domain engineering to develop the product according to the end-user requirements [27, 28]. To design multiple combinations, we can use the Binary Pattern for Nested Cardinality Constraints (BPNCC) Method, which calculates the exact number of combinations of products. BPNCC finds all possible products automatically by using a top-to-bottom approach. However, invalid products are generated in the total number of products due to the crosstree constraint between features. Authors have ignored the exclude and include constraints between features [2]. It is only possible to test some of the products individually as a lot of effort, cost, and time are required because a large SPL has thousands of products. 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 [25].

Different approaches are used for the development of products. When we drive other products by combining multiple product line features, many things need to be corrected. Therefore, integration testing of components is essential for detecting faults and failures in products. Integration testing tests the interfaces of different SPL features and detects faults due to incompatibility among multiple parts of an SPL [29]. Due to the increasing number of products, testing them individually and detecting defects takes a lot of work. Thus, only a subset of products that covers all possible

interactions of features is selected for testing. In many types of research, different methods are used for feature integration testing of SPL products and to reduce the number of products. All of them aim to reduce the effort and cost of integration testing and increase the fault detection ratio. To minimize testing efforts, a subset of products are selected using different approaches such as independent pathway tests, combination tests, priority-based tests, mutation-based tests, and model-based tests [30, 31].

In all these approaches, feature models or product lines are used to select a subset of products. Many researchers use these approaches differently, with some benefits and limitations. An integration testing method has been presented to improve testing by dealing separately with optional features and alternative features of an SPL. This paper recognizes that by growing the number of items, alternative features have a negative effect. They offer a new approach, the simple form, to black-box testing. This approach converts the model of the function into a graph of feature inclusion and then associates cases of use with each feature [32, 33]. Then, select the base paths on this graph using an algorithm that tests feasibility using the Depth First order to find the longest path. Finally, if they are linearly independent, add these path algorithms. In this algorithm, group all possible routes by alternative characteristics and use the base route algorithm for each group. However, this method has some limitations, such as the cost of calculation being higher than the cost of all characteristics and efforts to create a characteristic model dependency structure. These efforts are reduced by providing a new algorithm called the "Full Feature Algorithm." In this method, there is no need to design any graphs, such as the Feature Integration Graph (FIG). This algorithm aims to select a subset of products covering all functions. We must greedily add functions to the variable until we have a product that includes the most unused functions. Then, update the set of user functions, and once all functions are enabled, you're done [34].

2.2 Testing Efforts of SPL Products

Reducing testing effort by pruning irrelevant features based upon multiple test cases. Outside features are those whose absence or presence has no change in the system's behavior. Thus, they only test the combinations of relevant features and reduce the testing effort. But at the same time, it has some limitations, such as we already need to know about all test cases and different usage scenarios. Furthermore, in other studies, multiple approaches focus on increasing the fault detection rate using higher strength suites [35].

Researchers also use the prioritization method to detect faults and decrease the testing effort effi-

ciently. A research work describes a process of effective product-line testing using similarity-based prioritization. In this method, incrementally select the products that are diverse in features to increase the feature interaction coverage. Prioritize the products on a similar basis and then test all of them. The results show that it potentially increases the fault detection ratio but needs to decrease the testing effort efficiently. Moreover, introduces a PINE method, which is applied before generating integration test case scenarios. PINE consists of four significant steps: in the first step, feature model analysis is done by partitioning the FM into some independent sub-trees. Then, prioritize each feature with the score given by the domain engineer based on customer needs, time to market, relationship between elements, etc. At the same time, the score is also obtained through "event probability," these two factors are multiplied through an algorithm to get the score for each feature. In the third step, prune the feature model but cut less important branches by prioritizing the features on three preference levels: high, medium, and low. These ranges are defined through Boehm's method. In the last step, another algorithm is presented for selecting configurations in which every feature is covered at least once in a design. To choose selected configuration features, they are analyzed based on their relationship and find configurations that cover the maximum no of features. Thus, these two algorithms reduce the number of products for testing, and the effort of integration testing decreases [36].

In some research, testing is done through the mutation process, like they perform testing on feature model mutation by using the fault-based approach. They define some mutation operators that help find faults and check whether all products are valid. Therefore, it detects glitches related to several characteristics and relationships. The main measures of these operators are

- The wrong cardinality of a single feature.
- Faulty elements of a grouped relation.
- Existence of a set relation.
- Wrong cardinality of a fixed relation.
- Wrong constraint.

Moreover, this method increases the probability of finding faults, so confidence is developed that the products in the feature model match their requirements. However, the problem of growing complexity is due to the large number of invalid feature combinations and crosstree constraints. It also ignores some feature model constraints [37]. In addition, some studies are also improving mutation testing based on function-oriented programming, while the problem of increasing complexity still

needs to be solved. In their work, they receive appropriate mutation operators that validate their approach across four software product lines and further discuss the challenges of mutation testing. The result shows that it improves the defect detection method. Still, its limitations are that it is relatively expensive due to the large number of products or options and does not reduce the testing effort [38].

As we step forward, for generating Integration testing scenarios (ITS), we studied different methods in the literature, such as creating test scenarios through activity diagrams, which are the most promising approaches. In our research work, we are focusing on valid feature combinations for developing SPL. Errors are caused due to invalid feature combinations concerning cross-tree constraints.

In the Software Product Line, core assets are reused to develop a family of products, which helps reduce development costs, time, and effort. Still, much effort, time, and cost are required to test this family of products as it exposes errors or compatibility issues caused by integrating different components or features. Hence, trying all the products individually is challenging [12]. The literature solves this problem through combinatorial testing in which, instead of all of these products, a subset of products for testing that covers all possible interactions of features is individually selected [39]. Different methods are defined for choosing this type of product subset, which are helpful but still have some limitations, as mentioned in Table 2.1. In research work, use a model-based testing method to test all possible interactions of components. They create placeholders and integration scenarios that cover all interactions for integration testing. Hence, it is helpful to uncover inter-component failures through these placeholder integration scenarios [40].

Before proceeding further, it is imperative to present evidence from the literature that the problem statement stated in this research exists. The following sections give a detailed overview of invalid feature combinations in the literature.

In continuation to the probe to present literature on invalid feature combinations, table 2.2 provides a detailed overview of the ten research publications from renowned journals to give the reader an idea about the existence of invalid feature combinations.

Table 2.3 shows the combination of the features in binary form by applying the BPNCC approach [25]. After shedding light on invalid feature combinations, table 2.4 gives an idea about the problems highlighted due to the invalid feature combinations. These findings are gleaned after an intensive literature review of the same research papers ACM, IEEE, and Springer presented.

In previous sections, table 2.1 and Table 2.2 have already given enough information on the exis-

Table 2.1: Comparison of existing approaches

References	Approaches	FM Tree Relationship	CrossTree Constraints	Total .No. of products	Mapping of FM
J. Miguel Horcas, et. al. Software Product Line Conference. 2020. Canada [41]	Web interface to construct syntactically and semantically Feature model	Yes	No	No	Yes
H. Shatnawi et. al. ACM Software Engineering 2020. USA [42]	Extensible model driven engineering approach	Yes	No	Yes	Yes
Abbas, A et. al. IEEE Access, 2018 [25]	Multi-Objective Optimization-Binary Pattern for Nested Cardinality Constraints	Yes	No	No	No
Abbas, A. et. al. IEEE Access, 2017 [26]	Binary Pattern for Nested Cardinality Constraints	Yes	No	Yes	Yes

Table 2.2: Analysis of litrature with respect invalid feature combination

Source Document	Finding from the Literature
Framework for Refactoring Software Product Line Architecture [15]	• Inconsistencies arise • Create invalid combinations of features. • Misconfigured product features and the evolution of SPL.
Environment Modeling-Based Requirements Engineering for Software Intensive Systems [16]	• Generates a set of functions according to variability constraints, • FM accepts as output to minimize the invalid joins to a multi-step configuration.
PACOGEN [17]	• Combinations of functions are common problems • Generate paired tests • Determine the number of N rows
Feature-Oriented Software Product Lines [18]	• Mismatched mutability given by a specific application. • Code integration of teamwork. • Optional feature expresses errors.
Pairwise Testing of SPL [43]	• Different qualities and similarities are combined • Combining functions make test configurations. • High variability combination results in unmanaged test
Test Generation Using Minimum Invalid Tuples [44]	• Certain settings may be prohibited. • Settings are null and are not protected. • U-tuple is true if it can be used in a valid f-test.
Code Generation to Support Static and Dynamic Composition [45]	• Validation process configuration provided against the constraints. • Functions that are not present in the model. • Incomplete mandatory functions and invalid function.
Genetic Algorithm for optimized feature selection [19]	• Computationally expensive and time-consuming. • Developers need the ability to easily create and evaluate the selection.
Detection of Feature Interactions Feature-Aware [46]	• Functional interactions unforeseen • Combination of functions are source of failure.
Automated Diagnosis of Configuration Errors [47]	• Require several steps configure broad functions • Making difficult to prevent conflicts and error • New techniques are needed to debug invalid configurations • Erroneous configurations

Table 2.3: Valid and invalid feature combinations due to crosstree consraints

#Products	Nature of Products	GPS	Basic	Color	High Resolution	Camera	MP3
1	Invalid Products GPS exclude	1	1	0	0	0	0
2		1	1	0	0	0	1
3		1	1	0	0	1	0
4		1	1	0	0	1	1
5	Invalid Products Basic Camera Include High	0	0	1	0	1	0
6		0	1	0	0	1	0
7		1	0	1	0	1	0
8		0	0	1	0	1	1
9		0	1	0	0	1	1
10		1	0	1	0	1	1
11	Valid Products	0	0	0	1	0	0
12		0	0	1	0	0	0
13		0	1	0	0	0	0
14		1	0	0	1	0	0
15		0	0	0	1	1	1
16		1	0	0	1	1	1
17		1	0	0	1	1	0
18		1	0	1	0	0	0
19		0	0	0	1	0	1
20		0	0	1	0	0	1
21		0	1	0	0	0	1
22		1	0	0	1	0	1
23		1	0	1	0	0	1
24		0	0	0	1	1	0

Table 2.4: Problems identified due to invalid feature combinantions

Source Document	Overview of the Problems
Framework for Refactoring SPL Architecture [48]	• Suspicious product does not perform correct functions. • it is necessary to evaluate the models to demonstrate the architecture and functionality of the SPL.
Environment Modeling-Based RE for Software Intensive Systems [49]	• Specified by a collection of functional and non-functional requirements • Coherent set of individual requirements • Functions are invalid expressions.
Automatic Generation of Pairwise Test Configurations [50]	• Minimal instrumental support for the setup of combination of test functions. • Tool returns the minimum number of configurations.
Feature-Oriented SPL [51]	• Unacceptable combination of functions and interaction, • Challenge is to identify the missing behavior while in the problem of additional functions, • Coordination of code in a way that does not affect mutability.
Integration Testing for SPL [43]	• Various methods and tools reveal scalability issues • inefficiencies outside of a range of product variants.
Combinatorial Test Generation for SPL [44]	• Challenge in this effort is to deal with the large number of constraints between different functions.
Code Generation for Static and Dynamic Composition of SPL [45]	• Static and dynamic methods disadvantages of versatility in composition, efficiency, and consumption of resources. • Force the programmer to develop between static and dynamic composition.
Genetic Algorithm for Optimized Feature Selection [19]	• Optimizing the SPL function with limited resources is a very limited problem. Optimizing • Exact problem-solving algorithms do not scale well.
Detection of Feature Interactions [46]	• different combinations of functions are possible, • Not possible to detect the interactions of functions by creating all possible combinations.
Automated Diagnosis of SPL Configuration Errors [47]	• Difficult to debug conflicts and bugs in large function models.

tence of invalid feature combinations and the problems that arise due to the issue. Table 2.5 details proposed solutions and approaches to counter the effects of invalid feature combinations.

In these approaches, they use feature models or product lines to select a subset of products. Many researchers use these approaches differently, with some benefits and limitations. The FIG method has been presented to improve testing by dealing separately with optional features and alternative features of an SPL. This paper recognizes that by growing the number of items, alternative features have a negative effect. They offer a new approach, the simple FIG form, to black-box testing. This approach converts the model of the function into a graph of feature inclusion and then associates cases of use with each feature. Then, select the base paths on this graph using an algorithm that tests feasibility using the Depth First order to find the longest path. Finally, if they are linearly independent, add these path algorithms. In this algorithm, group all possible routes by alternative characteristics and use the base route algorithm for each group. However, this method has some limitations, such as calculating FIG, which is higher than the cost of all characteristics' efforts to create a characteristic model (FIG) dependency structure. These efforts are reduced by providing a new algorithm called the "Full Feature Algorithm." In this method, there is no need to design any graphs such as FIG. This algorithm aims to select a subset of products covering all functions. We must greedily add functions to the variable until we have a product that includes the most unused functions [8].

Finally, Table 2.6 presents the mapping of various analysis methods onto the Software Development Life Cycle (SDLC). This section gives an overview of the literature on how multiple researchers analyzed the problems arising from invalid feature combinations and, as per their findings, at which phase of SDLC this issue may be dealt with.

Table 2.5: Method/Approach followed in the wake of invalid feature combinations

Source Document	Methods	Approaches Used
Intensive Systems [49]	Goals of autonomous objects.	Reflecting autonomous subjects.
PACOGEN [50]	Greedy algorithms, and meta-heuristics.	Permitted interactions between pairs of functions.
Feature-Oriented SPL [51]	Functional Model to Prevent Interaction.	Abstracted for function interaction.
Pairwise Testing for SPL [43]	Captures features linked by required, optional, and alternative.	Combinatorial Approach.
Combinatorial Test Generation [44]	Minimum Invalid Tuples (MITS).	Notion of Minimum Iinvalid tuples (MIT).
Code Generation and Composition of SPL [45]	Dynamic composition in Object Teams.	Static and Dynamic Composition.
Genetic algorithm for SPL [19]	GAFES	Genetic Algorithm
Detection of Feature Interactions [46]	Intentional Testing of features .	SPL VERIFIER
Automated Diagnosis of SPL Configurations [47]	CURE Diagnostics	Debugging configuration and constraint solver

Table 2.6: Mapping SPL issues with SDLC phases

Source Document	Analysis Method	Specifications Strategy	SDLC Phase			
			Req. Gathering	Design	Development	Testing
Variational Correctness -by- Construction [41]	theorem proving	feature-based specification				Yes
Feature-Oriented Contract Composition [42]	analysis method independent	feature-based specification			Yes	
Modularization of Refinement Steps for Agile Formal Methods [52]	model checking, theorem proving	feature-based specification	Yes		Yes	
Strategies for Product-Line Verification: Case Studies and Experiments [53]	model checking	composition-based implemen-tation				Yes
Symbolic Model Checking of Product-Line Requirements Using SAT-Based Methods [54]	model checking	family-wide specification, feature-based specification		Yes		Yes
Probabilistic Model Checking for Energy Analysis in Software Product Lines [55]	model checking	feature-based specification		Yes		Yes

Towards Formal Safety Analysis in Feature-Oriented Product Line Development [31	Model checking	feature-based specification		Yes	Development	Tesing
Potential Synergies of Theorem Proving and Model Checking for Software Product Lines [32	Model checking, theorem proving	feature-based specification		Yes		Yes
SPLat: Lightweight Dynamic Analysis for Reducing Combinatorics in Testing Configurable Systems [33	Testing	feature-based specification		Yes		
Compositional Verification of Software Product Lines [34]	Model checking	feature-based specification	Yes			Yes

2.3 Research Gap

BPNCC finds all possible products automatically using a top-to-bottom approach to all valid and invalid combinations [25]. MOO-BPNCC [26] consists of three independent paths (first, second, and third). Furthermore, heuristics on these paths found that the first path could be more feasible due to space and execution time complexity. The second path reduces the space complexity; however, time complexity increases due to the increasing group of features. BPNCC and MOO-BPNCC are the latest techniques that cannot find invalid, valid, and partially invalid/valid feature combinations due to cross-tree constraints. The proposed solution will be able to identify the issues mentioned above in the context of cross-tree constraints.

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. 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.

2.4 Problem Statement

All of the approaches mentioned above ignore the cross-tree constraints while using feature models that produce some invalid products. Thus, if they consider these constraints, they can reduce invalid configurations. Furthermore, they did not exploit how we can automatically test the feasibility of products for their cross-tree controls, such as include and exclude. Due to crosstree constraints in the feature model, invalid feature combinations become part of SPL, leading to extra effort and cost in developing SPL. As shown in Fig. 1.1, the crosstree constraints "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. 1.1 shows "mobile phone" SPL where ten products are invalid due to the crosstree constraints. Table 2.3 1 shows the selection of features, and 0 indicates the feature is not part of the product. It is essential 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, and relationships between varied features resultantly make this process complex and consume extra effort during integration testing of SPL." Furthermore, table 2.6 shows a comparative study of different research studies that map the issues of SPL in the software development life cycle.

2.5 Research Questions

- Mitigating the invalid feature combinations:

  How do we minimize the number of invalid feature combinations generated during integrating software product line testing while dealing with cross-tree constraints?

- Formal method:

  What would be the method for generating the finite prioritize feature set to test invalid feature combinations during integrating testing in the software product line?

Chapter 3

Binary Oriented Feature Selection Crosstree Constraints (BOFS-CTC)

We proposed an 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 applies for small and large feature models with low to high complexity 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 thesis, different methods 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. As a result, using BOFS-CTC minimizes the total cost and effort of SPL product development. Furthermore, BOFS-CTC is the independent approach of any specific tool as we have proposed its algorithm.

3.1 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.

Table 3.1: 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] J. Miguel Horcas, et. al. 2020. [41]	Yes	No	No	Yes
Extensible model driven engineering approach H. Shatnawi et. al. 2020. [32]	Yes	No	Yes	Yes
Multi-Objective Optimization-Binary Pattern for Nested Cardinality Constraints Abbas, A et. al. 2018 [26]	Yes	No	No	No
Binary Pattern for Nested Cardinality Constraints Abbas, A. et. al. 2017 [25]	Yes	No	Yes	Yes

Our proposed algorithm overcomes these limitations and improves the correctness of feature selection. It helps to memorize all the constraints automatically 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. Table 3.1 describes the practices that consider the CTC variability of the feature model. However, research needs to be done to manage the CTC variability.

3.2 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 excluded operations that affect the other feature combinations of SPL are performed. Developing complex systems that provide consumers with various functions takes much work [58, 59]. The Challenge lies in providing many options for different application contexts with high versatility while restricting the customization of systems to achieve maintainability and growth management. The Feature model is essential to dealing with invalid feature combinations by capturing and visualizing the similarities and dependencies between features and the components that provide the features [60, 61]. 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 [62, 63].

3.3 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 3.2 shows the product configurations of the "Mobile Phone" feature model in Fig. 1.1. which consists of valid and invalid product configurations due to not considering the crosstree constraints. "GPS" and "Basic" features exclude each other, i.e., only one can be part of the product configuration. Therefore, in Table 3.2, the invalid product configurations consist of "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.

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)

Table 3.2: 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

- One-to-one (alternate to alternate)

3.4 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 [64]. 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.

3.5 Binary Oriented Feature Selections (BOFS)

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. The BOFS-CTC is a novel approach built on the binary combinations of features for cross-tree (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 an extensive 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 actual functionality are visible. All connections between parents of terminal features are represented by non-terminal features [65, 66, 67]. As a result, consider the relationships between the constraints on the sub-tree and the terminal characteristics of each group (alternative, optional, OR).

3.5.1 BOFS-CTC Framework

The framework suggested a fresh and efficient method to count all SPL products, as shown in Fig. 3.1. 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 necessary characteristics. However, varying features set the goods apart in the wide range of features. Six stages make up this BOFS-CTC strategy.

- In the first stage, formulas corresponding to various variable groups use a backtrace tree structure to determine the products.
- The second 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. 3.1 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

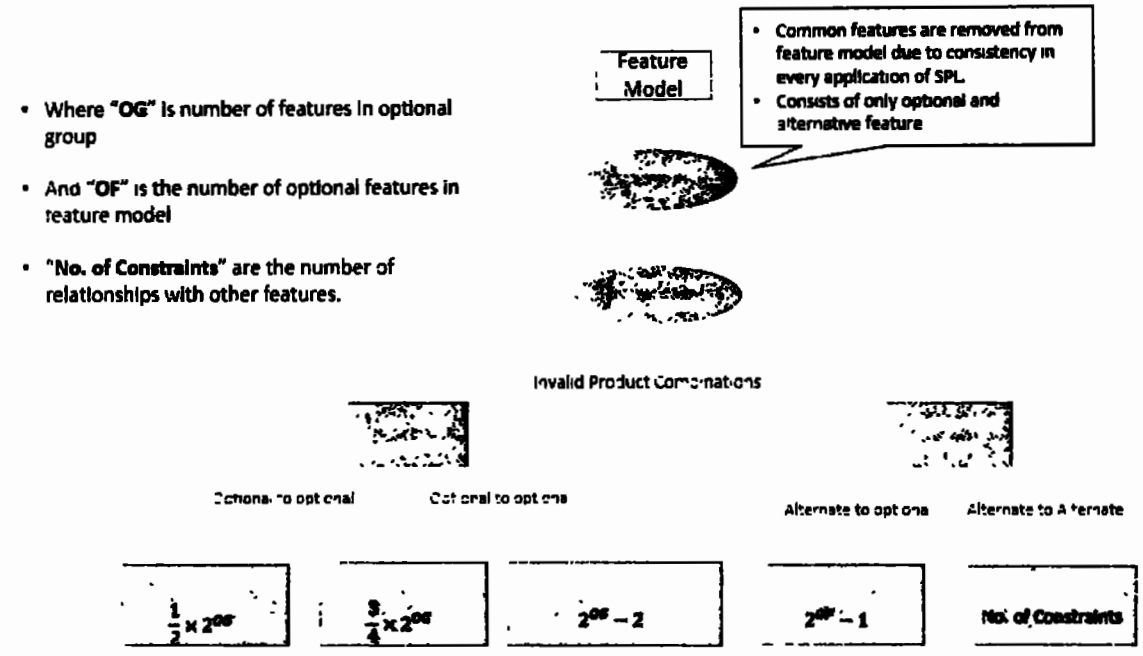


Figure 3.1: BOFS-CTC framework

- 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 final sexist stage is to count all potential products in the feature model.

3.6 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 equation 3.1 "Accuracy Function" as given below":

$$Number\ of\ invalid\ products = \frac{1}{2} \times 2^{OG} \tag{3.1}$$

Here, OG shows the number of features in the OR group with constraints. Therefore, valid products from the OR group can be derived from equation 3.2.

$$Total\ valid\ products = 2^n - (\frac{1}{2} \times 2^{OG}) \tag{3.2}$$

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 equation 3.3.

$$Number\ of\ invalid\ products = \frac{3}{4} \times 2^{OG} \tag{3.3}$$

Therefore, valid products from the OR group can be derived from the equation 3.4.

$$Total\ valid\ products = 2^n - (\frac{3}{4} \times 2^{OG}) \tag{3.4}$$

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 equation 3.5.

$$Total\ valid\ products = 2^{OG} - 2 \tag{3.5}$$

Therefore, valid products from the OR group can be derived from equation 3.6.

$$Total\ valid\ products = 2^n - (2^{OG} - 2) \tag{3.6}$$

In the case of "Alternate to optional (One-to-many)," to find the invalid products from the feature model, we have derived the mathematical equation 3.7 and for all valid products, we have derived equation 3.8.

$$Number\ invalid\ products = 2^{OF} - 1 \tag{3.7}$$

$$Total\ valid\ products = 2^{OF} \times A - (2^{OF} - 1) \tag{3.8}$$

Where OF is the number of optional features, A is the number of alternate features. Equation 3.7 calculates the invalid products of CTC between the alternate and optional OR groups. Equation 3.8

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 equation 3.9. Invalid products = #constraints are applied on the alternate group of features as only one feature is selected among n number of features.

$$Total\ valid\ products = n \times n - invalid\ products \tag{3.9}$$

3.6.1 BOFS-CTC Algorithm

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. Algorithm 1. requires the data set of features and their cardinality relationships, such as mandatory, optional, alternate, and OR group.

```
Valid Features Extraction
    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)
```

Algorithm 1: Feature model data and constraints input

Algorithm 2. consists of five modules, where the first and second modules generate a list of features that an SPL domain contains according to their specific groups and relationships. As mentioned,

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

Algorithm 2: Algorithm for entering features with relationships

Table 3.3: Mobile phone feature model valid product configurations

M.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

the name of the features of the mobile feature model is given below.

In the third module, 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).

The fourth 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 two leaf nodes MP3 and camera.

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.

Previously proposed algorithms have been applied to the mobile phone feature model in Fig. 1.1 and get 24 product configurations where some invalid configurations were also generated due to

Table 3.4: BOFS-CTC Comparison with other proposed approaches based on feature model level

Approaches	CTC	Binary Combination	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

crosstree constraints, as shown in Table 3.1. Therefore, BOFS-CTC is applied to the same feature model with cardinality and crosstree constraints and has 14 product configurations. From Table 3.3, BOFS-CTC removed ten invalid product configurations, as shown in Table 3.3. Use the relationships below to verify the valid product configurations in Table 3.3. GPS has no relationship (exclude) with Basic, such as "GPS¬Basic," where GPS is selected, i.e., GPS=1, then Basic should not be chosen, i.e., Basic=0. GPS can be set 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 CTCs are satisfied. Therefore, all 14 products are valid in Table 3.3.

3.6.2 BOFS-CTC Comparison

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 3.4. A comparative study is based on significant parameters defining the working and accuracy of the proposed approaches. These proposed approaches calculate and generate the total number of SPL products. BOFS-CTC is more appropriate and covers all the significant parameters that 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 calculating and generating the binary combinations of SPL product configurations.

BOFS-CTC is applied to small and large feature models with different relationships and limitations. Table 3.5 shows the results of a total number of valid products by considering all the feature model's primary 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 prod-

Table 3.5: BOFS-CTC Applied on small and large feature models

Feature Model	No. of Features	Manda-tory	Optio-nal	XOR	OR	Grouged	CTC	# Valid Product
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

uct 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).

Chapter 4

Binary Oriented Feature Selection-Crosstree Constraints Validation

This thesis proposes the BOFS-CTC approach and is used to find the valid total number of products in SPL. BOFS-CTC is a novel approach that evaluates the crosstree constraints in the feature model to avoid invalid product configurations and find the valid products of SPL. Based on the feature model structure, we have categorized six cases of the feature model for the application of BOFS-CTC according to the crosstree constraints. These cases are based on the different relationships of the features of one parent node to the features of another parent node. Furthermore, these relationships are between variable features, such as alternate and optional features. The first step is to find the invalid product configurations that do not follow the crosstree constraints, and the second step is to find the total number of products that include all the valid and invalid product configurations. The third step is removing the invalid products calculated in the first step from the total number of products. Only valid products that do not violate the crosstree and relationship constraints are obtained through this process.

SPL companies that develop products from a specific domain by reusing the features and expanding the family of products only spend the exact cost and effort on invalid products using our proposed solution.

BOFS is valid for the 'n' number of features in a feature model where different features have crosstree constraints with individual or group of features. Therefore, we have categorized the crosstree constraints of features in the feature model as given below.

4.1 Feature Model Optional Selection

Fig. 4.1 shows feature model that only consists of crosstree constraints among optional features, where only optional features exist.

- $One - to - One\ (O_{f1} \rightharpoonup O_{f2})$
- $One - to - Two\ (O_{f1} \rightharpoonup (O_{f2}\ \&\&\ O_{f3}))$

Case I: Feature model that exists zero alternate features and "n" number of optional features. Crosstree Constraint: one optional to two optional features.

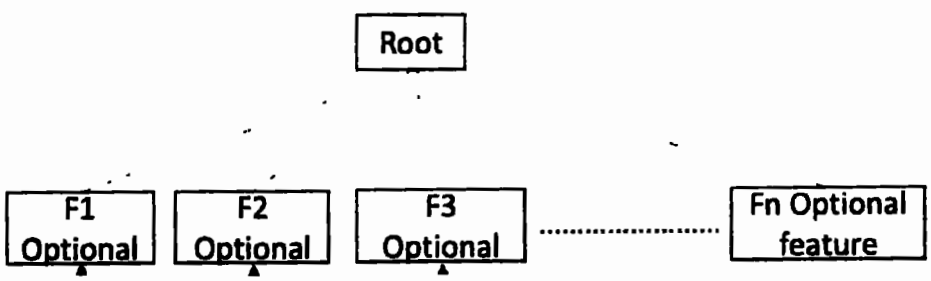


Figure 4.1: CTC of One optional to two optional features

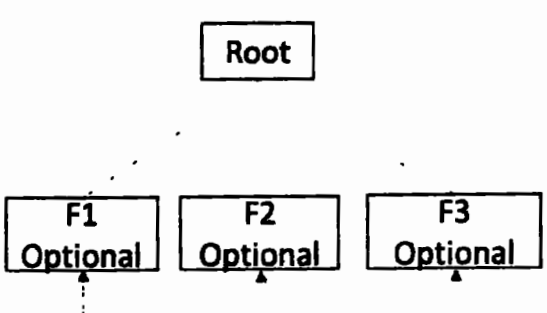


Figure 4.2: Crosstree constraints with one optional to two optional

Fig. 4.2 shows only three features that have crosstree constraints $(O_{f1} \rightharpoonup (O_{f2} \parallel O_{f3}))$ Equation 4.1 is used to calculate the invalid products given bellow.

$$Invalid\ products = 2^{n-2} + 2^{n-3} \tag{4.1}$$

$$Invalid\ products = 2^{3-2} + 2^{3-3}$$

Table 4.1: Valid and invalid product configurations of Figure 4.2

# Products	OF1	OF2	OF3	Comment
1	0	0	0	Valid
2	1	0	0	Valid
3	0	1	0	Valid
4	1	1	0	Invalid: $(O_{f1} \leftarrow O_{f2})$
5	0	0	1	Valid
6	1	0	1	Invalid: $(O_{f1} \leftarrow O_{f3})$
7	0	1	1	Valid
8	1	1	1	Invalid: $(O_{f1} \leftarrow (O_{f2} \, OR \, O_{f3}))$

$$Invalid\,products = 2^1 + 2^0$$

$$Invalid\,products = 2 + 1 = 3$$

In table 4.2. eight total products and three products (4, 6, and 8) are invalid due to defined crosstree constraints that have been calculated using Equation 1.

From Figure 4.2. where optional features are a minimum of three, i.e. n=3. Calculate the invalid products by using equation 4.1. Fig 4.3 shows the total number of products, valid and invalid products of the feature model of Fig 4.2

Figure 4.3: Valid and Invalid Products indicates table 4.1

Fig 4.4 shows the feature model with the same crosstree mentioned above constraints. However, the number of optional features is increased by one, i.e., 4.

From the figure 4.4, where optional feature are minimum 4 i.e. n=4. Calculate the invalid products by using equation 4.1

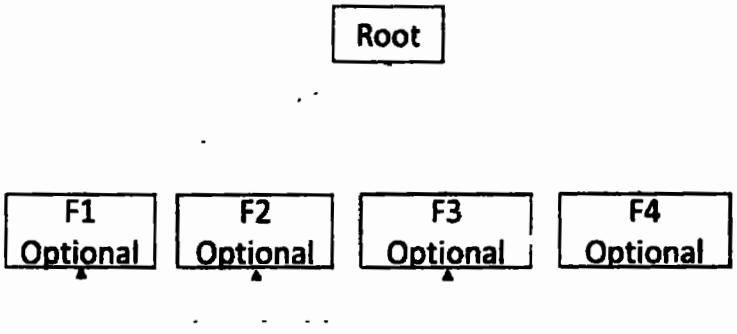


Figure 4.4: Four Optional Features with One-to-Two CTC

Table 4.2: Valid and Invalid Product configurations from Figure 4.4.

# Products	OF1	OF2	OF3	OF4	Comment
1	0	0	0	0	Valid
2	1	0	0	0	Valid
3	0	1	0	0	Valid
4	1	1	0	0	Invalid: $(O_{f1} \leftarrow O_{f2})$
5	0	0	1	0	Valid
6	1	0	1	0	Invalid: $(O_{f1} \leftarrow O_{f3})$
7	0	1	1	0	Valid
8	1	1	1	0	Invalid: $(O_{f1} \leftarrow (O_{f2} \, OR \, O_{f3}))$
9	0	0	0	1	Valid
10	1	0	0	1	Valid
11	0	1	0	1	Valid
12	1	1	0	1	Invalid: $(O_{f1} \leftarrow O_{f2})$
13	0	0	1	1	Valid
14	1	0	1	1	Invalid: $(O_{f1} \leftarrow O_{f3})$
15	0	1	1	1	Valid
16	1	1	1	1	Invalid: $(O_{f1} \leftarrow (O_{f2} \, OR \, O_{f3}))$

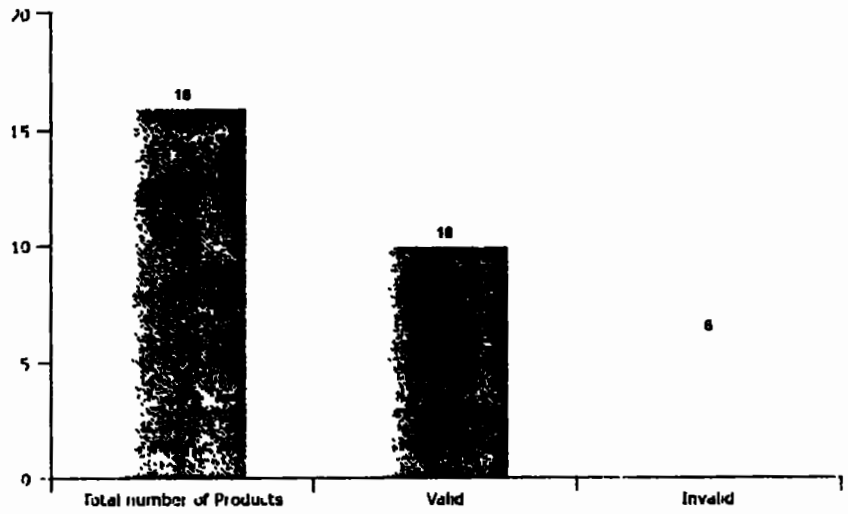


Figure 4.5: Valid and Invalid Product configurations indicates table 4.2

$$Invalid\ products = 2^{4-2} + 2^{4-3}$$

$$Invalid\ products = 2^2 + 2^1$$

$$Invalid\ products = 4 + 2 = 6$$

Table 4.2 shows sixteen total products, and six products (4, 6, 8, 12, 14, and 16) are invalid due to defined crosstree constraints. Fig. 4.5 shows the total number of products, valid and invalid products from the table 4.2

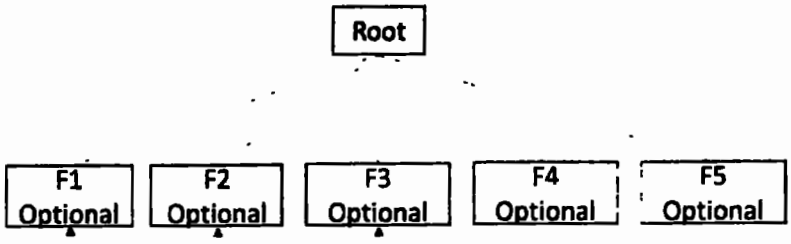


Figure 4.6: Five Optional Features with One-to-Two CTC

Fig 4.6 shows the five optional features; the constraints remain the same, i.e., one-to-two optional features. In Table 4.3, thirty-two total number of products and twelve products (4, 6, 8, 12, 14, 16, 20, 22, 24, 28, 30, and 32) are invalid due to defined crosstree constraints. From the figure 4.6, where optional feature are minimum 5 i.e. n=5. Calculate the invalid products by using equation 4.1

Table 4.3: Valid and Invalid Product configurations from Figure 4.6.

# Products	OF1	OF2	OF3	OF4	OF5	Comment
1	0	0	0	0	0	Valid
2	1	0	0	0	0	Valid
3	0	1	0	0	0	Valid
4	1	1	0	0	0	Invalid: $(O_{f1} \leftarrow O_{f2})$
5	0	0	1	0	0	Valid
6	1	0	1	0	0	Invalid: $(O_{f1} \leftarrow O_{f3})$
7	0	1	1	0	0	Valid
8	1	1	1	0	0	Invalid: $(O_{f1} \leftarrow (O_{f2} \, OR \, O_{f3}))$
9	0	0	0	1	0	Valid
10	1	0	0	1	0	Valid
11	0	1	0	1	0	Valid
12	1	1	0	1	0	Invalid: $(O_{f1} \leftarrow O_{f2})$
13	0	0	1	1	0	Valid
14	1	0	1	1	0	Invalid: $(O_{f1} \leftarrow O_{f3})$
15	0	1	1	1	0	Valid
16	1	1	1	1	0	Invalid: $(O_{f1} \leftarrow (O_{f2} \, OR \, O_{f3}))$
17	0	0	0	0	1	Valid
18	1	0	0	0	1	Valid
19	0	1	0	0	1	Valid
20	1	1	0	0	1	Invalid: $(O_{f1} \leftarrow O_{f2})$
21	0	0	1	0	1	Valid
22	1	0	1	0	1	Invalid: $(O_{f1} \leftarrow O_{f3})$
23	0	1	1	0	1	Valid
24	1	1	1	0	1	Invalid: $(O_{f1} \leftarrow (O_{f2} \, OR \, O_{f3}))$
25	0	0	0	1	1	Valid
26	1	0	0	1	1	Valid
27	0	1	0	1	1	Valid
28	1	1	0	1	1	Invalid: $(O_{f1} \leftarrow O_{f2})$
29	0	0	1	1	1	Valid
30	1	0	1	1	1	Invalid: $(O_{f1} \leftarrow O_{f3})$
31	0	1	1	1	1	Valid
32	1	1	1	1	1	Invalid: $(O_{f1} \leftarrow (O_{f2} \, OR \, O_{f3}))$

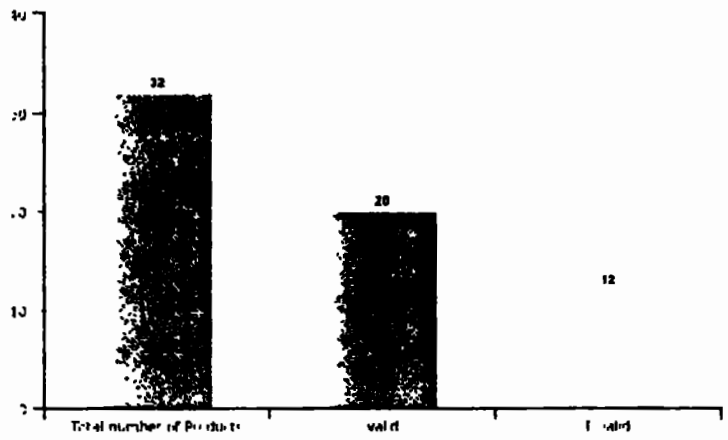


Figure 4.7: Valid, Invalid Configurations from table 4.3

$$Invalid\ products = 2^3 + 2^2$$

$$Invalid\ products = 8 + 4 = 12$$

Fig 4.7 shows the total number of products, valid and invalid products of feature model combinations of table 4.3

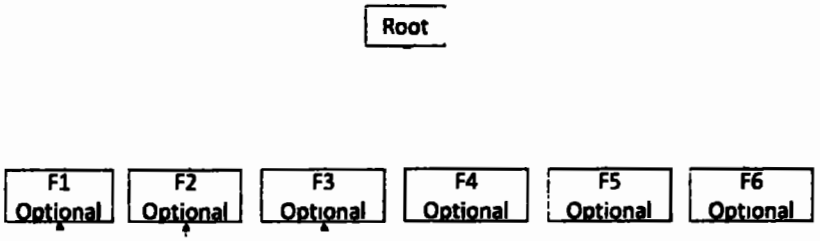


Figure 4.8: Six Optional Features with One-to-Two CTC, Valid, Invalid Configurations

From Fig 4.8, there are sixty-four possible total number of products and 24 products (4, 6, 8, 12, 14, 16, 20, 22, 24, 28, 30, 32, 36, 38, 40, 44, 46, 48, 52, 54, 56, 60, 62 and 64) are invalid due to defined crosstree constraints as shown in table 4.4.

Figure 4.8 shows optional features with a minimum of 6, i.e., n=6. Calculate the invalid products by using equation 4.1 Fig 4.9 shows the total number of products, valid and invalid products of the feature model of Fig 4.8

$$Invalid\ products = 2^{6-2} + 2^{6-3}$$

$$Invalid\ products = 2^4 + 2^3$$

Table 4.4: Invalid Product with six optional features one-to-one CTC from fig 4.8

# products	OF1	OF2	OF3	OF4	OF5	OF6	Comment
1	1	1	0	0	0	0	Invalid: $(O_{f1} \leftarrow O_{f2})$
2	1	0	1	0	0	0	Invalid: $(O_{f1} \leftarrow O_{f3})$
3	1	1	1	0	0	0	Invalid: $(O_{f1} \leftarrow (O_{f2} OR O_{f3}))$
4	1	1	0	1	0	0	Invalid: $(O_{f1} \leftarrow O_{f2})$
5	1	0	1	1	0	0	Invalid: $(O_{f1} \leftarrow O_{f3})$
6	1	1	1	1	0	0	Invalid: $(O_{f1} \leftarrow (O_{f2} OR O_{f3}))$
7	1	1	0	0	1	0	Invalid: $(O_{f1} \leftarrow O_{f2})$
8	1	0	1	0	1	0	Invalid: $(O_{f1} \leftarrow O_{f3})$
9	1	1	1	0	1	0	Invalid: $(O_{f1} \leftarrow (O_{f2} OR O_{f3}))$
10	1	1	0	1	1	0	Invalid: $(O_{f1} \leftarrow O_{f2})$
11	1	0	1	1	1	0	Invalid: $(O_{f1} \leftarrow O_{f3})$
12	1	1	1	1	1	0	Invalid: $(O_{f1} \leftarrow (O_{f2} OR O_{f3}))$
13	1	1	0	0	0	1	Invalid: $(O_{f1} \leftarrow O_{f2})$
14	1	0	1	0	0	1	Invalid: $(O_{f1} \leftarrow O_{f3})$
15	1	1	1	0	0	1	Invalid: $(O_{f1} \leftarrow (O_{f2} OR O_{f3}))$
16	1	1	0	1	0	1	Invalid: $(O_{f1} \leftarrow O_{f2})$
17	1	0	1	1	0	1	Invalid: $(O_{f1} \leftarrow O_{f3})$
18	1	1	1	1	0	1	Invalid: $(O_{f1} \leftarrow (O_{f2} OR O_{f3}))$
19	1	1	0	0	1	1	Invalid: $(O_{f1} \leftarrow O_{f2})$
20	1	0	1	0	1	1	Invalid: $(O_{f1} \leftarrow O_{f3})$
21	1	1	1	0	1	1	Invalid: $(O_{f1} \leftarrow (O_{f2} OR O_{f3}))$
22	1	1	0	1	1	1	Invalid: $(O_{f1} \leftarrow O_{f2})$
23	1	0	1	1	1	1	$(O_{f1} \leftarrow O_{f3})$
24	1	1	1	1	1	1	Invalid: $(O_{f1} \leftarrow (O_{f2} OR O_{f3}))$

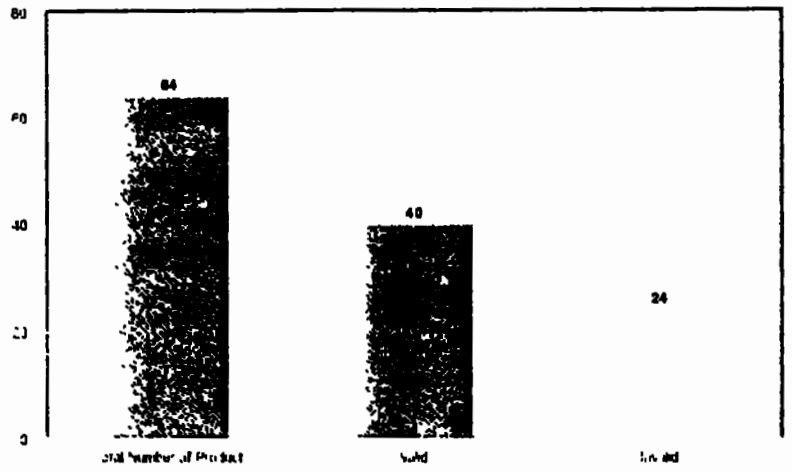


Figure 4.9: Six Optional Features Valid and Invalid Configurations from table 4.4

$$Invalid\ products = 16 + 8 = 12$$

- $One - to - Three\ (O_{f1} \leftarrow (O_{f2}\ \&\&\ O_{f3}\ \&\&\ O_{f4}))$

4.2 Alternate Feature Model Optional Selection

Crosstree constraint: One optional to two optional features where two mandatory alternate features and 'n' optional features exist, as shown in Fig 4.10.

- $A_{f1} \parallel A_{f2}\ \&\&\ One - to - One\ (O_{f3} \leftarrow O_{f4})$
- $A_{f1} \parallel A_{f2}\ \&\&\ One - to - Two\ (O_{f3} \leftarrow (O_{f4}\ \&\&\ O_{f5}))$

Where A_{f1} and A_{f2} shows the alternate features, where only one feature can be part of product configuration.

Table 4.5 shows sixteen total number of products, and six products (7, 8, 11, 12, 15, and 16) are invalid due to defined crosstree constraints. Fig 4.11 shows the total number of products, valid and invalid products of feature model configurations of table 4.5.

$$Invalid\ Combinantions = Alt \times (2^{n-2} + 2^{n-3}) \tag{4.2}$$

Equation 4.2 is used to identify the invalid product configurations for the feature model that consists of a group of two alternate features and three optional features. The crosstree

Table 4.5: Combination of two alternate and thre optional from fig. 4.10

# Products	AF1	AF2	OF3	OF4	OF5	Comment
1	0	1	0	0	0	Valid
2	1	0	0	0	0	Valid
3	0	1	1	0	0	Valid
4	1	0	1	0	0	Valid
5	0	1	0	1	0	Valid
6	1	0	0	1	0	Valid
7	0	1	1	1	0	Invalid
8	1	0	1	1	0	Invalid
9	0	1	0	0	1	Valid
10	1	0	0	0	1	Valid
11	0	1	1	0	1	Invalid
12	1	0	1	0	1	Invalid
13	0	1	0	1	1	Valid
14	1	0	0	1	1	Valid
15	0	1	1	1	1	Invalid
16	1	0	1	1	1	Invalid

constraint among these three optional features is One-to-two. "n" is the total number of optional features, and "Alt" is the total number of alternate features. The above figure shows the value of n=3 and alt=2 in equation 4.2.

$$Invalid\ Combinantions = 2 \times (2^{3-2} + 2^{3-3})$$

$$Invalid\ Combinantions = 2 \times (2^{1} + 2^{0})$$

$$Invalid\ Combinantions = 2 \times (3) = 6$$

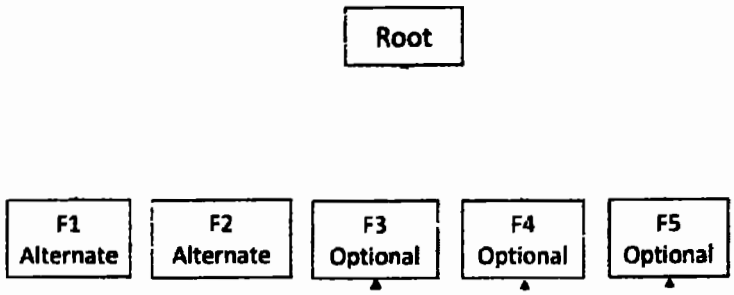


Figure 4.10: Two Alternate Features with One-to-Two CTC Feature Model

Table 4.6: Two Alternate Features with One-to-Two CTC, Four optional from fig 4.12

#Products	AF1	AF2	OF3	OF4	OF5	OF6	Comment
1	0	1	0	0	0	0	Valid
2	1	0	0	0	0	0	Valid
3	0	1	1	0	0	0	Valid
4	1	0	1	0	0	0	Valid
5	0	1	0	1	0	0	Valid
6	1	0	0	1	0	0	Valid
7	0	1	1	1	0	0	Invalid
8	1	0	1	1	0	0	Invalid
9	0	1	0	0	1	0	Valid
10	1	0	0	0	1	0	Valid
11	0	1	1	0	1	0	Invalid
12	1	0	1	0	1	0	Invalid
13	0	1	0	1	1	0	Valid
14	1	0	0	1	1	0	Valid
15	0	1	1	1	1	0	Invalid
16	1	0	1	1	1	0	Invalid
17	0	1	0	0	0	1	Valid
18	1	0	0	0	0	1	Valid
19	0	1	1	0	0	1	Valid
20	1	0	1	0	0	1	Valid
21	0	1	0	1	0	1	Valid
22	1	0	0	1	0	1	Valid
23	0	1	1	1	0	1	Invalid
24	1	0	1	1	0	1	Invalid
25	0	1	0	0	1	1	Valid
26	1	0	0	0	1	1	Valid
27	0	1	1	0	1	1	Invalid
28	1	0	1	0	1	1	Invalid
29	0	1	0	1	1	1	Valid
30	1	0	0	1	1	1	Valid
31	0	1	1	1	1	1	Invalid
32	1	0	1	1	1	1	Invalid

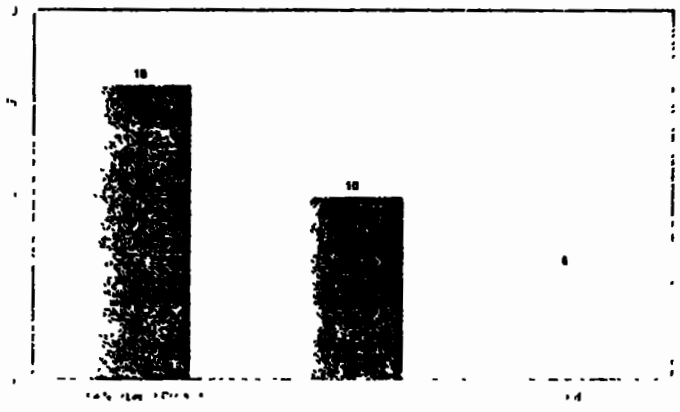


Figure 4.11: Valid and Invalid Configurations from table 4.5

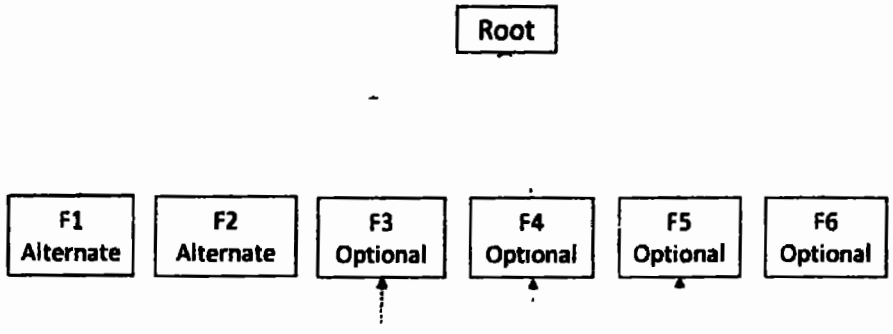


Figure 4.12: Two Alternate Features with One-to-Two CTC, Four optional Feature Model

Table 4.6 shows 32 total number of products, and 12 products (7, 8, 11, 12, 15, 16, 23, 24, 27, 28, 31, and 32) are invalid due to defined crosstree constraints. Fig 4.13 shows the total number of products, valid and invalid products of feature model configurations of table 4.6

In figure,4.12 the value of n=4 and alt=2 put in equation 4.2.

$$Invalid\ Combinantions = 2 \times (2^{4-2} + 2^{4-3})$$

$$Invalid\ Combinantions = 2 \times (2^{2} + 2^{1})$$

$$Invalid\ Combinantions = 2 \times (6) = 12$$

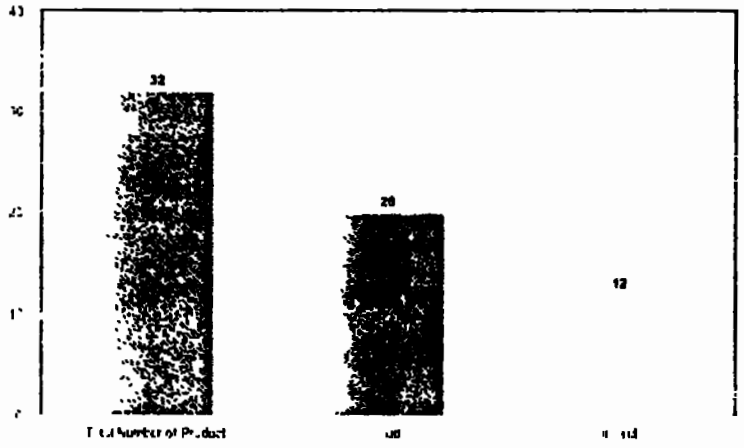


Figure 4.13: Valid and Invalid Configurations from table 4.6

4.3 Increase Alternate Feature Model Optional Selection

Crosstree constraint: One optional to 'n' optional features where 'n' mandatory alternate and 'n' optional features exist.

- $A_{f1} \parallel A_{f2} \parallel A_{f3}$ && $One - to - Three\ (O_{f1} \leftarrow (O_{f5}$ && $O_{f6}))$

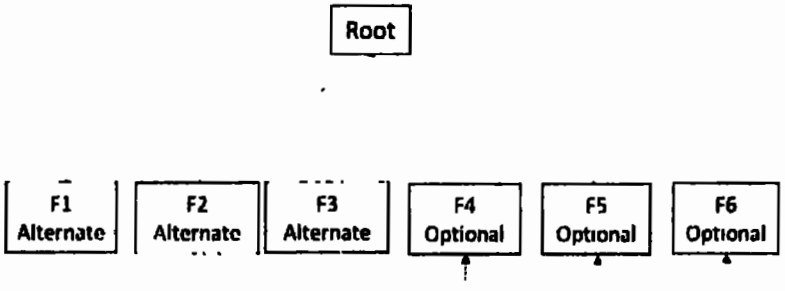


Figure 4.14: Three Alternate Features with One-to-Three CTC Optional

$$Invalid\ Combinantions = 3 \times (2^{3-2} + 2^{3-3})$$

$$Invalid\ Combinantions = 3 \times (2^{1} + 2^{0})$$

$$Invalid\ Combinantions = 3 \times (3) = 9$$

From the feature model Fig 4.14, there are nine invalid product configurations in Table 4.7. However, the total number of products is 24, and the valid products are 13.

Table 4.7: Three Alternate Features with One-to-Two CTC, Four optional from fig 4.14

# Products	AF1	AF2	AF3	OF4	OF5	OF6	Comment
1	0	0	1	0	0	0	Valid
2	0	1	0	0	0	0	Valid
3	1	0	1	0	0	0	Valid
4	0	1	0	1	0	0	Valid
5	0	0	1	1	0	0	Valid
6	1	1	0	1	0	0	Valid
7	0	0	1	0	1	0	Valid
8	0	1	0	0	1	0	Valid
9	1	0	1	0	1	0	Valid
10	0	1	0	1	1	0	Invalid
11	0	0	1	1	1	0	Invalid
12	1	1	0	1	1	0	Invalid
13	0	0	1	0	0	1	Valid
14	0	1	0	0	0	1	Valid
15	1	0	1	0	0	1	Valid
16	0	1	0	1	0	1	Invalid
17	0	0	1	1	0	1	Invalid
18	1	1	0	1	0	1	Invalid
19	0	0	1	0	1	1	Valid
20	0	1	0	0	1	1	Valid
21	1	0	1	0	1	1	Valid
22	0	1	0	1	1	1	Invalid
23	0	0	1	1	1	1	Invalid
24	1	1	0	1	1	1	Invalid

Fig 4.15 shows the total number of products valid and invalid product configurations of the feature model from Table 4.7.

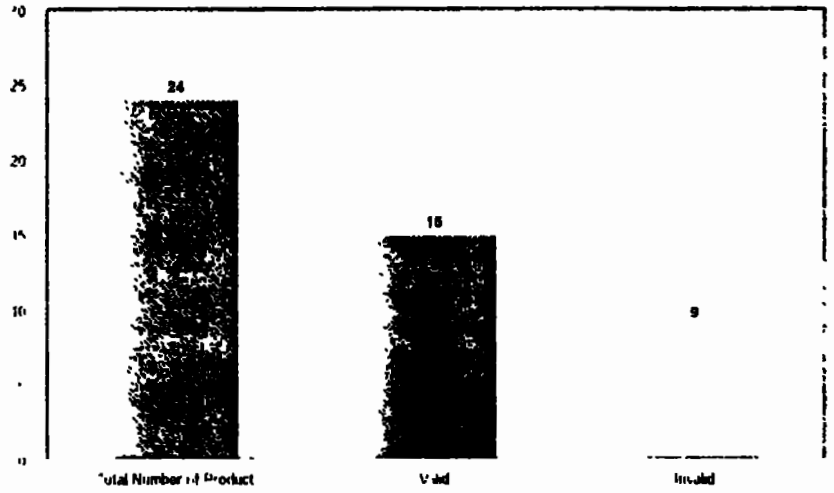


Figure 4.15: Valid and Invalid Configurations from table 4.7

Case III: Any number of Alternate and N number of optional

$$Invalid\,Combinantions = k \times Alt \times 2^m \tag{4.3}$$

Where "alt" is total number of alternate features. K is constant whose value is 7 i.e. k=7 and m= 0,1,2,3,4,5,6,...

Minimum optional feature should be 4 where value of m=0. Where optional features will be 5 then the value of m=1 as so on.

Equation 4.3 is only valid where any alternate and n number of optional features are used. Still, the constraint is between one optional and three optional features, as mentioned in Fig 4.16. Put the values in equation 4.3.

$$Invalid\,Combinantions = 7 \times 2 \times 2^0 = 14$$

As mention in table, there are 14 invalid product configurations that are the same as derived from the equation 4.3.

- $A_{f1} \parallel A_{f2} \parallel A_{f3}\ \&\&\ One-to-One\,(O_{f4} \leftarrow (O_{f5}\ \&\&\ O_{f6}\ \&\&\ O_{f7}))$

From figure 4.18, put the values in equation 4.3.

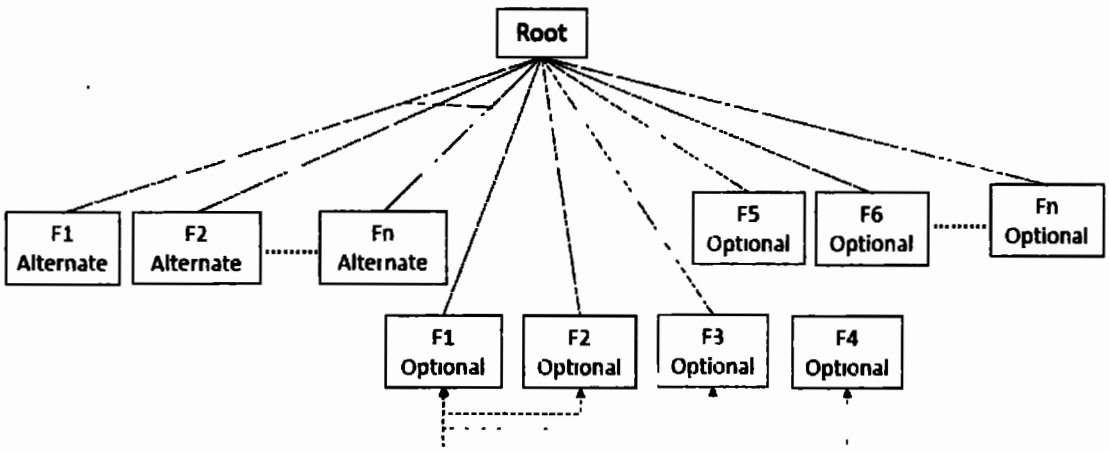


Figure 4.16: Feature model of n alternate and n optional features

$$Invalid\ Combinantions = 7 \times 4 \times 2^0 = 28$$

As mention in table, there are 28 invalid product configurations that are the same as derived from the equation 4.3.

From figure 4.20, put the values in equation 4.2.

$$Invalid\ Combinantions = 7 \times 2 \times 2^1 = 28$$

As mention in Fig 4.17, there are 28 invalid product configurations that are the same as derived from the equation 4.3.

From above figure, put the values in equation 4.3.

$$Invalid\ Combinantions = 7 \times 3 \times 2^2 = 84$$

As mention in Fig.4.24 , there are 84 invalid product configurations that are the same as derived from the equation 4.3.

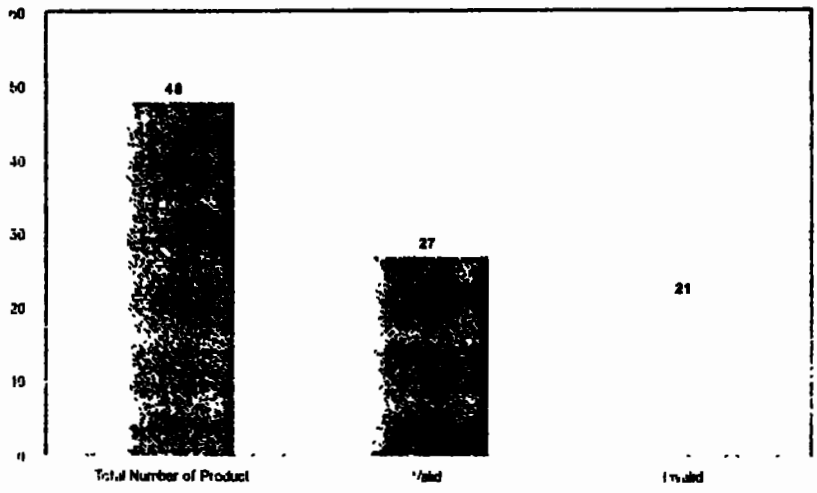


Figure 4.17: Valid and Invalid product configurations from fig 4.18

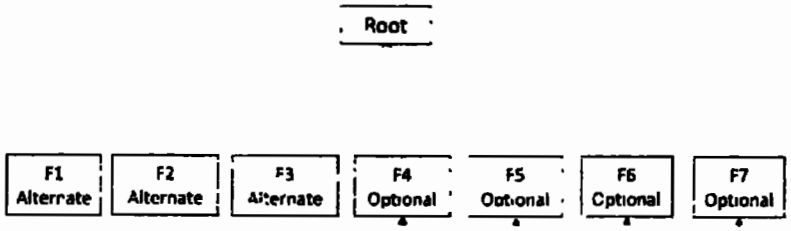


Figure 4.18: Four optional features with one-to-three optional CTC

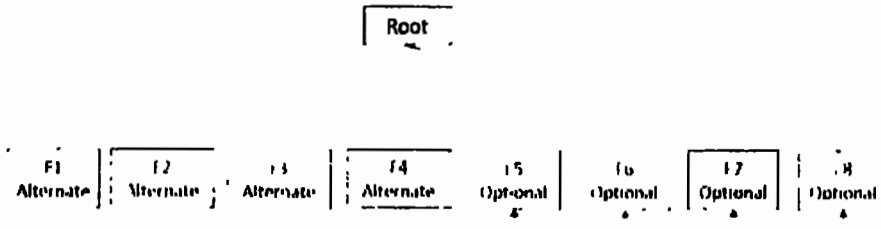


Figure 4.19: Four Alternate and Four optional features with one-to-three optional CTC

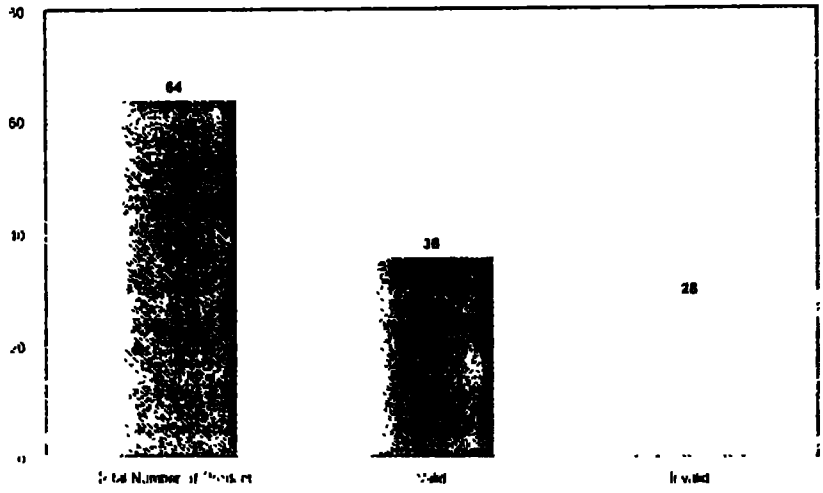


Figure 4.20: Valid and Invalid product configurations from fig 4.19

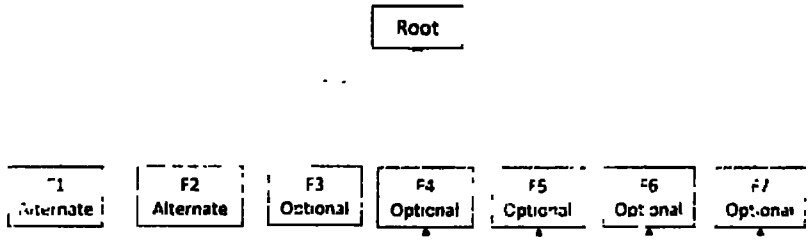


Figure 4.21: Four Alternate and Four optional features with one-to-three optional CTC

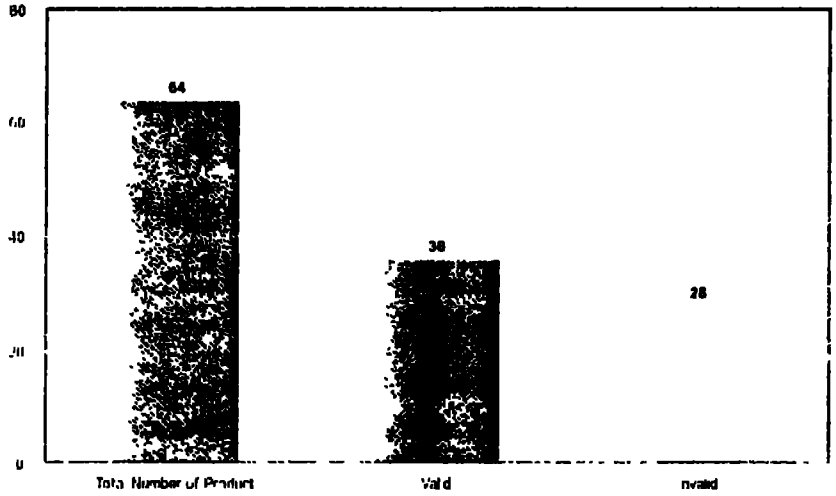


Figure 4.22: Valid and Invalid product configurations from fig 4.21

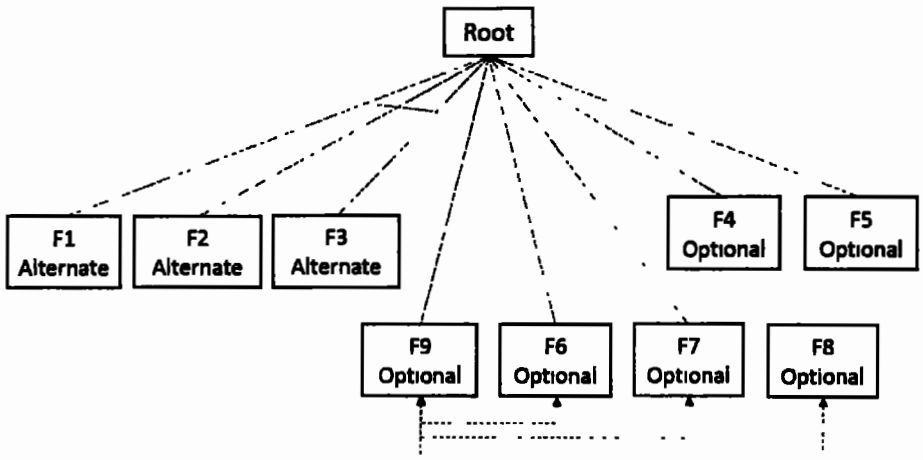


Figure 4.23: Three Alternate and six optional features with one-to-three optional CTC

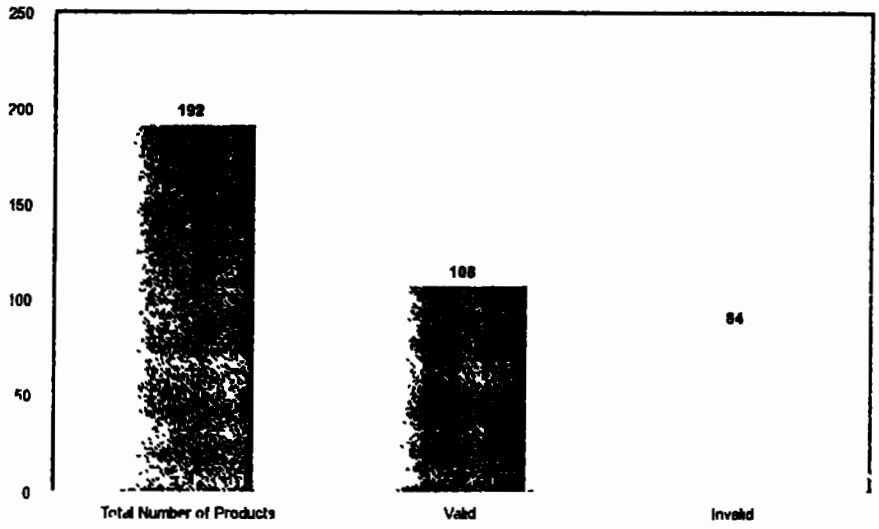


Figure 4.24: Valid and Invalid product configurations from fig 4.23

Chapter 5

Results and Analysis

Crosstree constraints are essential to feature modeling because they allow for analyzing and extracting insightful knowledge from feature models. A software system's configurations and interactions between features are represented using feature models, which help stakeholders comprehend and analyze the system's variability.

Crosstree constraints are used in feature modeling to specify dependencies or interactions between several features or feature groups. These restrictions outline the requirements that must be met for the system to be configured validly. Several important conclusions can be drawn from the outcomes and the crosstree constraints.

Crosstree constraints, in the first place, aid in discovering feature model discrepancies or conflicts. Conflicts arise when conflicting restrictions prohibit certain feature combinations. Such disputes can be found by analyzing the crosstree constraints, which enables stakeholders to fix them and guarantee the feature model is internally consistent.

Second, crosstree constraints make comprehending how feature interactions affect things easier. Crosstree constraints capture the interactions that frequently occur between features in a software system. By analyzing the constraints, stakeholders can learn more about how the existence or absence of particular features impacts the availability or restrictions of other features. Making decisions concerning feature dependencies and their effects on system behavior as a whole is made easier with this information.

Thirdly, crosstree restrictions allow for identifying permissible feature pairings and precise product configurations. Stakeholders can determine whether or not feature combinations are permitted by

looking at the limitations. This information is useful when creating legitimate product setups or assisting consumers during the configuration process.

Crosstree constraints also make evaluating whether the feature model is comprehensive easier. By examining the limitations, Stakeholders can identify gaps or incomplete links between characteristics. This makes it possible for the feature model to fully depict the system's variability by capturing all necessary relationships and restrictions.

In conclusion, crosstree restrictions in feature model findings and analysis are crucial for spotting inconsistencies, comprehending how features interact, figuring out legitimate feature combinations, and evaluating the completeness of the feature model. These limitations offer insightful information that aids in decision-making and supports stakeholders in managing software unpredictability.

5.1 Impact of Managing Crosstree Constraints

Depending on the complexity of the feature model and the particular constraints involved, addressing crosstree constraints in the feature model setup might have different effects. While crosstree constraints help ensure that feature configurations are valid and consistent, it isn't easy to pinpoint the precise cost and labor savings. However, the following are some potential advantages that may be connected to successfully handling cross-tree constraints:

1. Reduced Configuration Errors:

   Crosstree restrictions enforce dependencies and exclusions between features to prevent improper configurations. By effectively controlling these limits, users are less likely to choose feature combinations that are incompatible or contradictory, which lowers configuration problems and the need for further debugging or troubleshooting.

2. Improved Efficiency in Decision-Making:

   Users are guided during configuration by transparent and well-managed crosstree limitations. They facilitate decision-making and save time and effort by assisting users in understanding the relationships and constraints between features, expediting the decision-making process.

3. Enhanced Reusability and Scalability: Crosstree constraints allow for the modularization and reuse of feature models by accurately describing the connections and interactions between

features. Well-managed constraints reduce the work needed for feature model maintenance and evolution by explicitly defining dependencies and restrictions, making adding or modifying future features more straightforward.

4. Minimized Rework and Validation Efforts:

   With appropriate crosstree constraints, choosing between conflicting or insufficient feature configurations may be more accessible, which could result in validation or rework problems. Effectively controlling the restrictions decreases the likelihood of such difficulties, saving time and money that would have been needed to fix configuration-related issues.

5. Requirement Analysis:

   Crosstree constraints help to make the interdependencies and connections between features more understandable. This clarity can speed up the requirement analysis process, allowing stakeholders to make better decisions and lowering the time and effort required for revisions and iterations brought on by misunderstandings or misinterpretations.

6. Testing and Validation:

   Properly maintained crosstree constraints offer a more reliable and consistent feature model. This, in turn, makes it easier to test and validate the software system to ensure it functions as intended in various feature configurations. The testing effort can be more concentrated and effective if there are fewer contradictions and disputes to be resolved.

7. Maintenance and Evolution:

   A well-managed feature model with precise crosstree restrictions facilitates the maintenance and development of the software system. The effect analysis and adaptation process might go more smoothly when new features are added, or old ones are changed. Long-term cost reductions are achieved by reducing the work necessary for maintaining the integrity of the feature model and upgrading it.

It is important to remember that handling crosstree constraints can reduce errors, increase efficiency, promote reuse, and save rework. Still, it does take time and knowledge to define, validate, and manage these constraints effectively. The amount of work necessary may vary depending on the complexity of the feature model and the number of constraints. The cost and labor associated with this procedure can be reduced using appropriate tool assistance and a systematic approach to handling crosstree restrictions.

5.2 Applications of CTC

It is essential to ensure that all potential connections and interactions between features are accurately represented while building a feature model. Crosstree constraints can be used to infer the model's completeness in this situation. By analyzing these limitations, stakeholders can find any missing or imperfect links between features.

Consider a feature model for an e-commerce application as an illustration. The feature model could have functions like "Product Search," "User Registration," and "Shopping Cart." The "User Registration" feature might only be accessible if the "Shopping Cart" feature is chosen, according to the crosstree limitations. If there is no such restriction in the feature model, it may be possible for users to register without a cart, leading to an inconsistent system.

Stakeholders can locate these holes in the feature model by analyzing the crosstree constraints. They can use it to decide whether more constraints or dependencies are required to portray the system's unpredictability accurately. By filling in these gaps, stakeholders can avoid any inconsistencies or conflicts in the software system brought on by inadequate feature modeling.

Crosstree constraints also help preserve the feature model's consistency over time. A software system may add or modify new features as it develops. Participants in the evolution process can confirm that adjustments follow the dependencies and constraints already in place by looking at the crosstree constraints. This guarantees that the feature model is accurate and current and indicates the system's behavior.

In conclusion, analyzing crosstree constraints in the context of feature model outputs is essential for determining the model's level of completeness. It aids in locating any missing dependencies or links, enabling stakeholders to improve the feature model and guarantee that it appropriately captures the system's heterogeneity. Stakeholders can maintain a consistent and trustworthy feature model as the scheme develops by considering these restrictions throughout the software development lifecycle.

Feature A is needed for Feature B.

- B implies A

Feature C and Feature D are mutually exclusive:

- C excludes D
- D excludes C

Feature E and Feature F cannot be selected together:

- E excludes F
- F excludes E

Feature G can only be selected if both Feature B and Feature C are selected:

- G requires B
- G requires C

Feature H can only be selected if Feature E is selected:

- H requires E

Feature I can only be selected if Feature D is not selected:

- I requires not D

Feature J can only be selected if Feature G is selected and Feature H is not selected:

- J requires G
- J requires not H

These crosstree restrictions specify the dependencies and exclusions between the features. Constraint 1 states, for instance, that if Feature B is chosen, Feature A must also be selected. Constraint 4 ensures the appropriate link between these features by stating that Feature G can only be chosen if Feature B and Feature C are also preferred.

Stakeholders can reason about the legitimate feature model configurations by analyzing these crosstree constraints. They support the user in choosing the best combinations of features based on the indicated dependencies and aid in identifying conflicts, such as the mutual exclusivity of Features C and D. Additionally, by eliminating incorrect configurations and ensuring the feature model's internal consistency and completeness, these restrictions raise the software system's overall quality.

The many components of a mobile phone system are shown in connection to one another in Fig 5.1. Operating system, display, camera, storage, connectivity, battery, and other functionalities like water resistance, facial recognition, and fingerprint scanner are all included in the features.

The top-level feature, in this case, the cell phone itself, is represented by the feature model's root. The feature model is then divided into many feature categories and those categories' corresponding

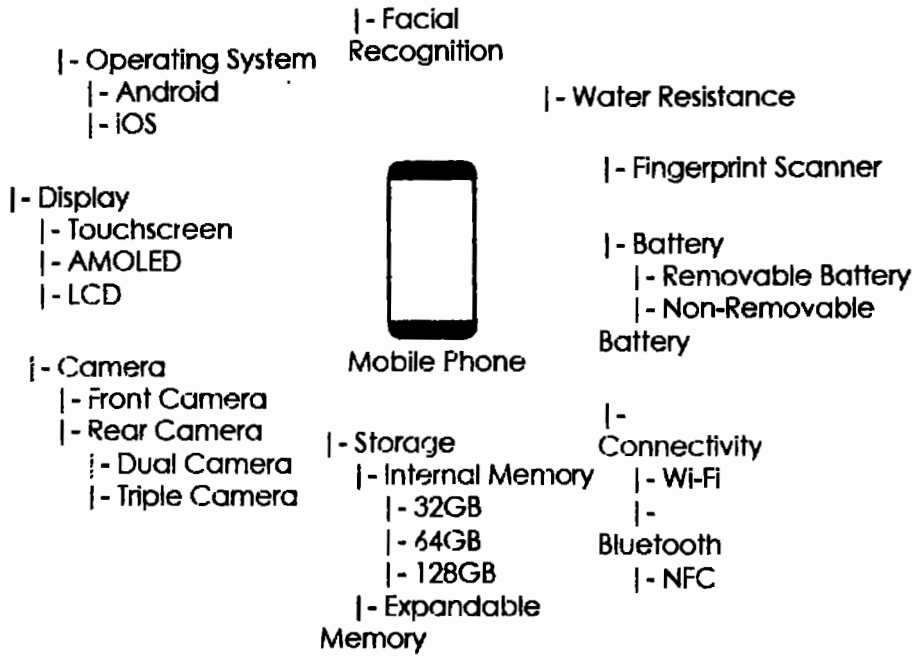


Figure 5.1: Mobile Phone Feature Model

... features.

There are two sub-features, for instance, under the "Operating System" feature: "Android" and "iOS." Similar sub-features exist under the "Display" feature, including "Touchscreen," "AMOLED," and "LCD." by using the algorithm mentioned in Fig 5.2.

The "Camera" feature has the subfeatures "Front Camera" and "Rear Camera," as well as other subfeatures like "Dual Camera" and "Triple Camera" under "Rear Camera."

```
function manageCrosstreeConstraints(featureModel, configuration):
  for each constraint in featureModel.crosstreeConstraints:
    if constraint.type == "Requires":
      if constraint.dependentFeature not in configuration:
        return false
    else if constraint.type == "Excludes":
      if constraint.dependentFeature in configuration:
        return false
  return true
```

Figure 5.2: Algorithm for require and exclude

The feature model also provides options for battery life ("Removable Battery" and "Non-Removable Battery"), connectivity ("Wi-Fi," "Bluetooth," and "NFC"), and storage ("Internal Memory" and "Expandable Memory").

Last but not least, the feature model offers extras like "Water Resistance," "Facial Recognition," and "Fingerprint Scanner."

The managed Crosstree Constraints method in Figure 5.2 pseudo-code accepts a feature model and a configuration as inputs. Each crosstree constraint in the feature model is iterated through. It determines whether the dependent feature described in the constraint is present or absent in the configuration depending on the type of constraint (needs or excludes). The function returns false, indicating an invalid configuration if the constraint is broken. The function returns true, indicating a valid setup if all restrictions are met. The feature model and configuration must be expressed using the proper data structures in the programming language of choice for this pseudo-code to work. Additionally, it presumes that the feature model consists of a list of crosstree constraints, where each constraint has properties like type (needs or excludes) and the dependent feature.

With the feature model and the current setup, you would call the manage Crosstree Constraints method to implement this algorithm. The format is valid if the function returns true according to crosstree restrictions. If it gives a false result, one or more constraints have been broken, and the configuration needs to be changed.

The stakeholders can comprehend and make sense of the variety of the mobile phone system thanks to this feature model's organized representation of the numerous features and their interactions.

5.3 BOFS-CTC Feature Models

The BOFS-CTC algorithm has applied to the SPLOT feature models. SPLOT is a library of real-time feature models where every type of feature model exists with all defined variability relationships and constraints. There are small and large feature models with complex relationships. This section consists of small and large feature models and product configurations in binary-oriented combinations of features.

Fig 5.3 consists of four small feature models: 1) System, 2) Computer selection, 3) AA sample, and 4) Match Engine. In these feature models, some have both crosstree constraints and feature variability relationships. However, some feature models consist of only feature variability relationships.

Table 5.1: Valid Product Configurations by using BOFS-CTC Algorithm

Feature Model: 10 Feature						
Feature Model	Mandatory	Optional	XOR group	OR group	CTC	Valid Configuration
BDS	0	9	0	0	1	112
Eshop	3	2	1	1	4	9
Mobile Phone	2	2	1	1	2	14
Car	3	6	0	0	0	16
Match Engine	3	4	1	1	0	16
System	4	1	1	1	2	8
Computer Selection	8	1	0	0	0	2
Test	1	2	1	1	1	32
MyFM	1	2	2	0	2	19
Biciclete	3	0	3	0	1	6

- "System" feature model consists of two crosstree constraints: "feature 7" constrained with "feature 10" and "feature 8 with "feature 10."
- "Computer Selection" feature model consists of 9 features without any crosstree constraints.
- "AA" feature model consists of 9 features without any crosstree constraints.
- "Match Engine" feature model consists of 9 features (3 mandatory and four optional features) without any crosstree constraints.

Table 5.1 shows the valid product configurations of feature models that consist of 10 features. The table clearly shows that several variable features generate more configurations. The feature model "BDS" consists of 9 optional features, with valid product configurations 112. On the other hand, "Computer Selection" has eight mandatory features and one optional feature, and it has only two valid product configurations.

Fig 5.4 consists of three feature models with 20, 31, and 32 features with more crosstree constraints compared to Fig 5.3. The feature models are "online shopping software," "historiaclinca," and "Urna." These feature models consist of various variabilities, such as alternate, optional, alternate optional, mandatory alternate, and mandatory, optional groups. In Fig 5.4, the crosstree constraints are:

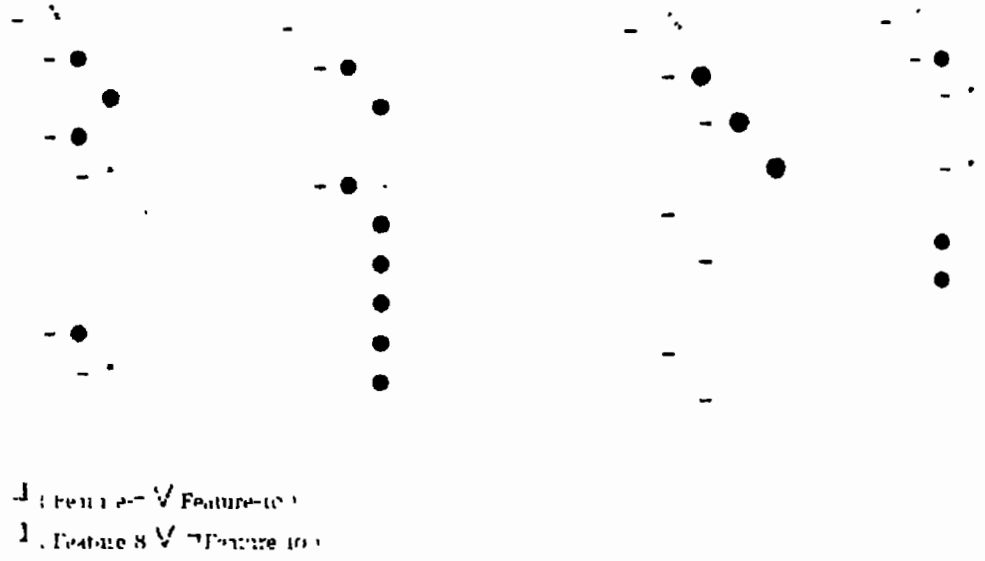


Figure 5.3: Eight feature models with CTC and without CTC

- "Online Shopping Software" feature model consists of seven crosstree constraints.
- "Historiaclinca" feature model consists of 31 features without any crosstree constraints.
- "Urna" feature model consists of 32 features with complex crosstree constraints such as the second CTC among eight features, i.e., eight features are in include and exclude relationship.

Fig 5.5 also consists of more complex feature models with many crosstree constraints. BOFS-CTC takes these feature models' input and generates the total possible feature combinations and number of products, as shown in Table 5.2. In Fig 5.5, the crosstree constraints are:

- "Building" feature model comprises 20 features with six crosstree constraints.
- "Cloud Storage Software" feature model consists of 21 features without any crosstree constraints.
- "Reference Management Software" feature model consists of 36 features without any crosstree constraints.

Fig 5.6 consists of four "GreenHouse," "Reference Management System," "TAM Reservas," and "Family of Bike Computers" feature models with and without crosstree constraints. The "Reference Management System" feature model has four crosstree constraints.

Fig 5.7 shows the single large feature model that consists of 54 features, and only two crosstree

constraints exist. The two crosstree constraints are "Set Profile and From Memory" and "Show Event and Access" among the four features.

Fig 5.8 shows the feature model with many crosstree constraints, i.e., every feature has the crosstree constraint with other features. The crosstree constraints are:

- Feature 0 OR Feature 9 OR Feature lan OR Feature 7 OR Destination.
- Feature 0 OR Feature 35 OR Feature 24 OR time date OR True OR walk OR mode.
- Feature false OR Triprequest OR InterModelRoute.

Fig 5.9 shows the "video player" large feature model with only seven crosstree constraints, i.e., a prominent feature model with fewer constraints. These crosstree constraints are:

- Envios excludes Factura Fisica.
- Notification excludes Listas.
- Gestion excludes Publication products.
- Notification excludes Factura Online.
- Notification excludes Por Correo.
- Notification excludes Notification page.
- Notification excludes Certificado Compra.

Fig 5.10 shows the "System Bandara SiBRAM" feature model existing of 44 features with three crosstree constraints. These crosstree constraints are:

- SMS Gateway excludes Basic Phone.
- SMS exclude Basic Phone.
- App Notifier excludes smart phone.

Fig 5.11 shows the "Carte SD" feature model that consists of more than fifty features where only five include condition crosstree constraints are exist. These crosstree constraints are:

- DMA requires USB.
- 32bits reuires Cortex M3.
- Red requires 33V.

- 32bits requires Cortex M4
- 64bits requires Cortex A8

Fig 5.12 shows the "e-commerce" feature model exists more than 35 features and five exclude condition crosstree constraints. These crosstree constraints are:

- Phone negate notification.
- Tablet negate notification.
- High negate Bank transfer.
- Account negate Reward program.
- Account negate Receipt history.

Fig 5.13 shows the "Tiendas-pos" feature model that exists more than 40 features with six include condition crosstree constraints. These crosstree constraints are:

- Caja includes Dispositivos.
- Tasa includes Comun.
- Peso includes Balanzas.
- Usuarios includes Maneja-turnos.
- Dispositivos includes Terminales.
- Clientes includes Pedidos

Fig 5.14 shows the "Facturacion-Serv-Publicos" complex feature model, which comprises more than 50 features. This feature model exists to exclude condition crosstree constraints. These crosstree constraints are among every subtree of the feature model.

Fig 5.15 shows the large feature model with 25 exclude condition crosstree constraints. These crosstree constraints are among every feature of a subtree with other subtrees. Table 5.3 shows the valid product derivations, and results show that many valid products, such as more than a billion, also exist.

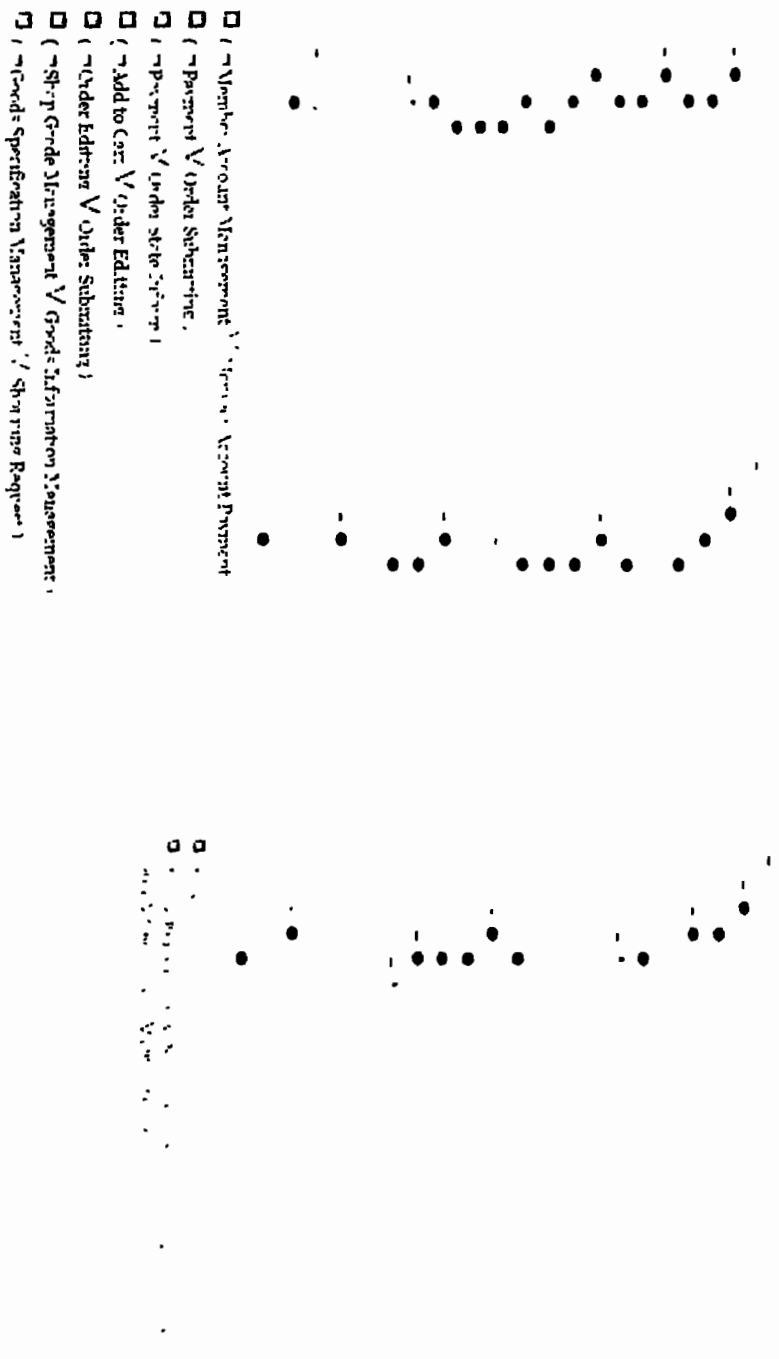


Figure 5.4: CTC based Feature Models with 20, 31, 32

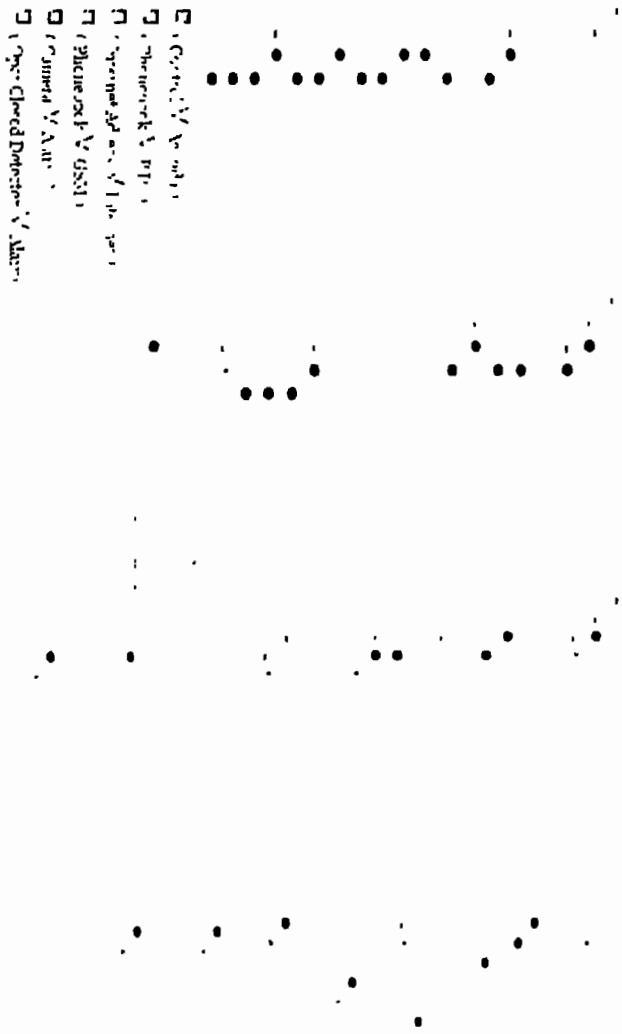


Figure 5.5: Complex and large feature models with and without CTC

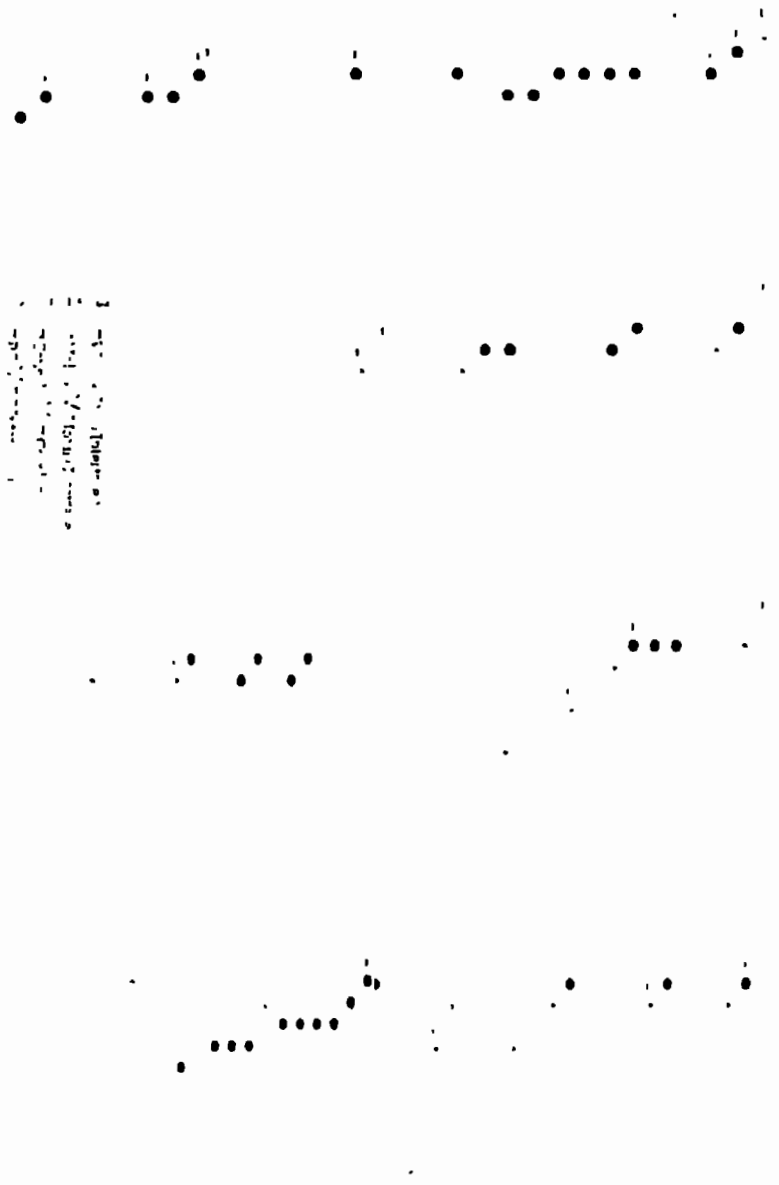


Figure 5.6: Four feature models dataset

Table 5.2: BOFS-CTC algorithm results of 20, 31 and 32 features of feature model

Feature Model: 20,31,32 Feature						
Feature Model	Mandatory	Optional	XOR group	OR group	CTC	Valid Configuration
Online Shopping Software	15	1	0	0	7	11
Historiaclinica	13	6	0	0	0	48
Urna	11	2	2	0	2	36
Cloud Storage Software	11	6	0	1	0	102
Software Product line Service	14	5	0	0	6	18
Automotive System	8	3	7	1	9	1344
Reference Management System	6	10	1	3	7	87480
Green House	15	15	0	0	4	3712
TAM Reservas	8	4	6	0	2	1296
Family of Bike Computer	13	4	6	0	5	240

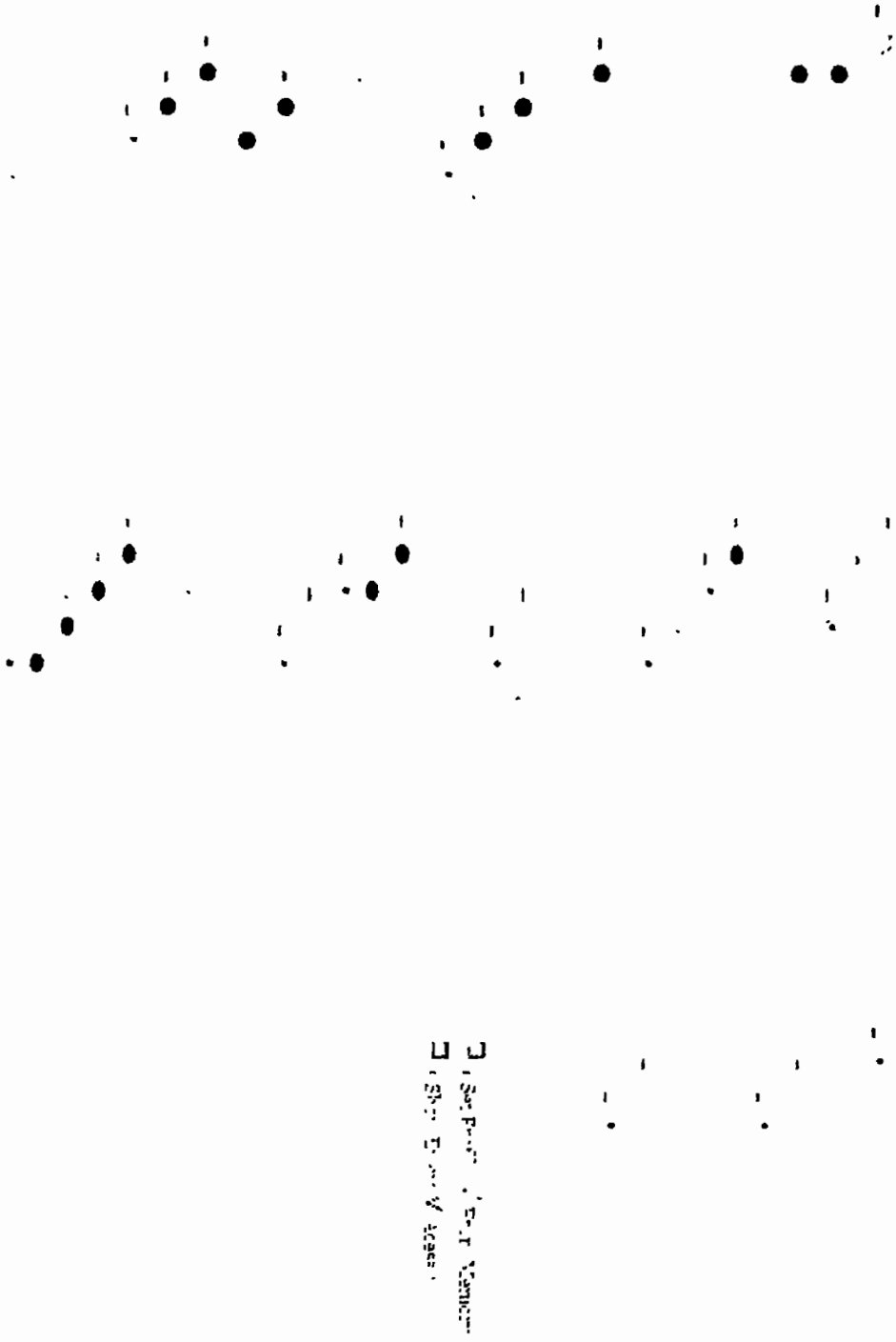


Figure 5.7: Feature model of fifty feature with CTC and basic relationships

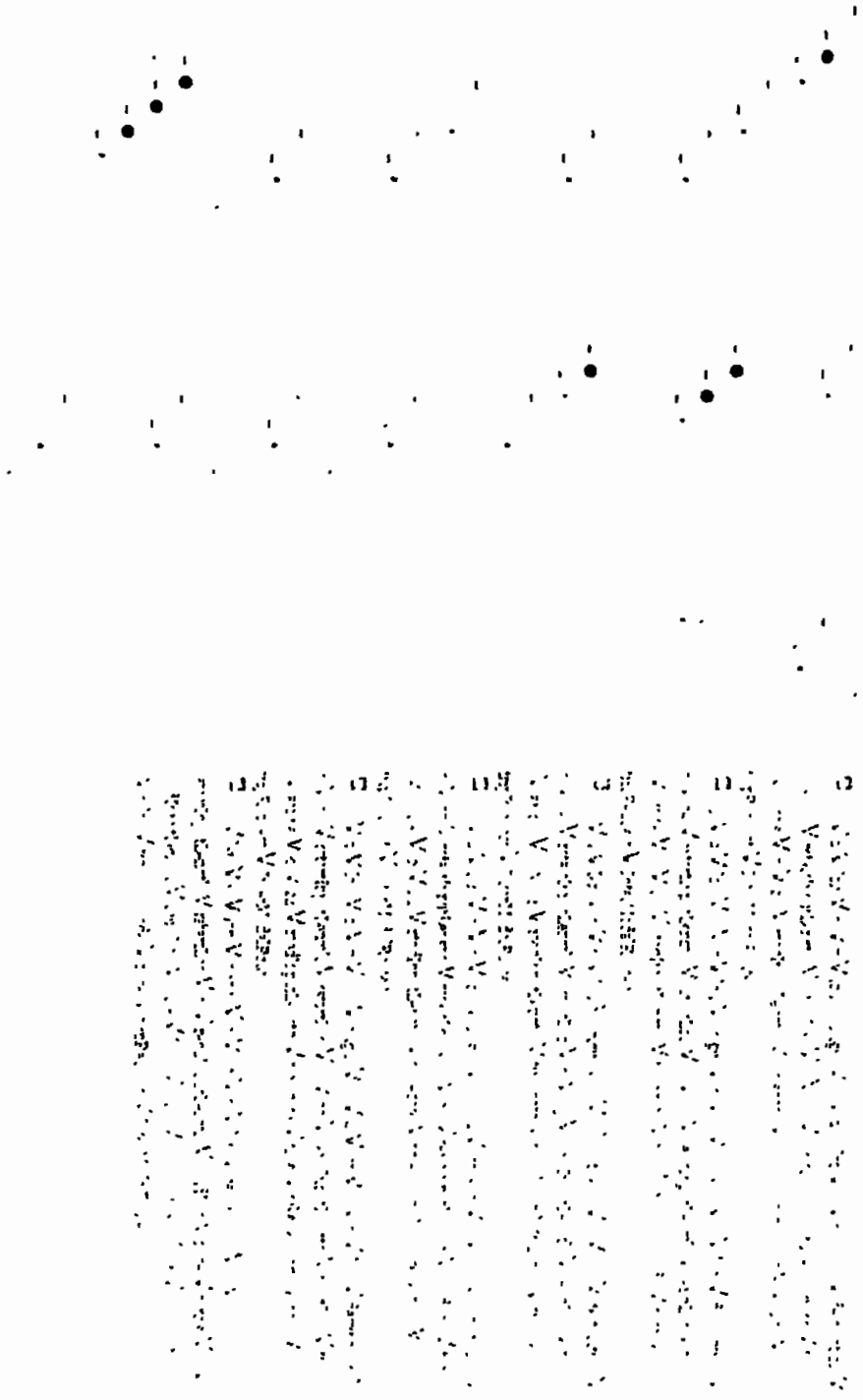


Figure 5.8: Small feature model with large CTC

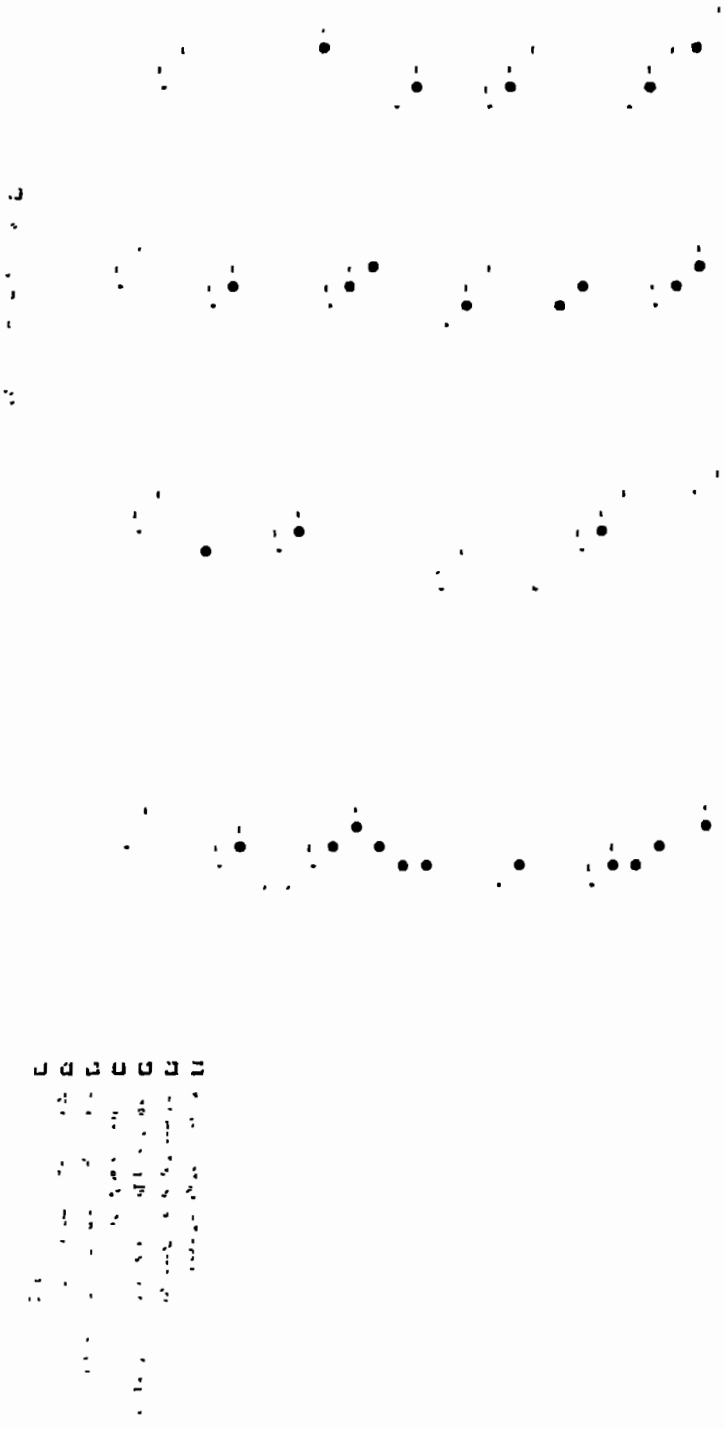


Figure 5.9: Large feature model with less CTC (1)

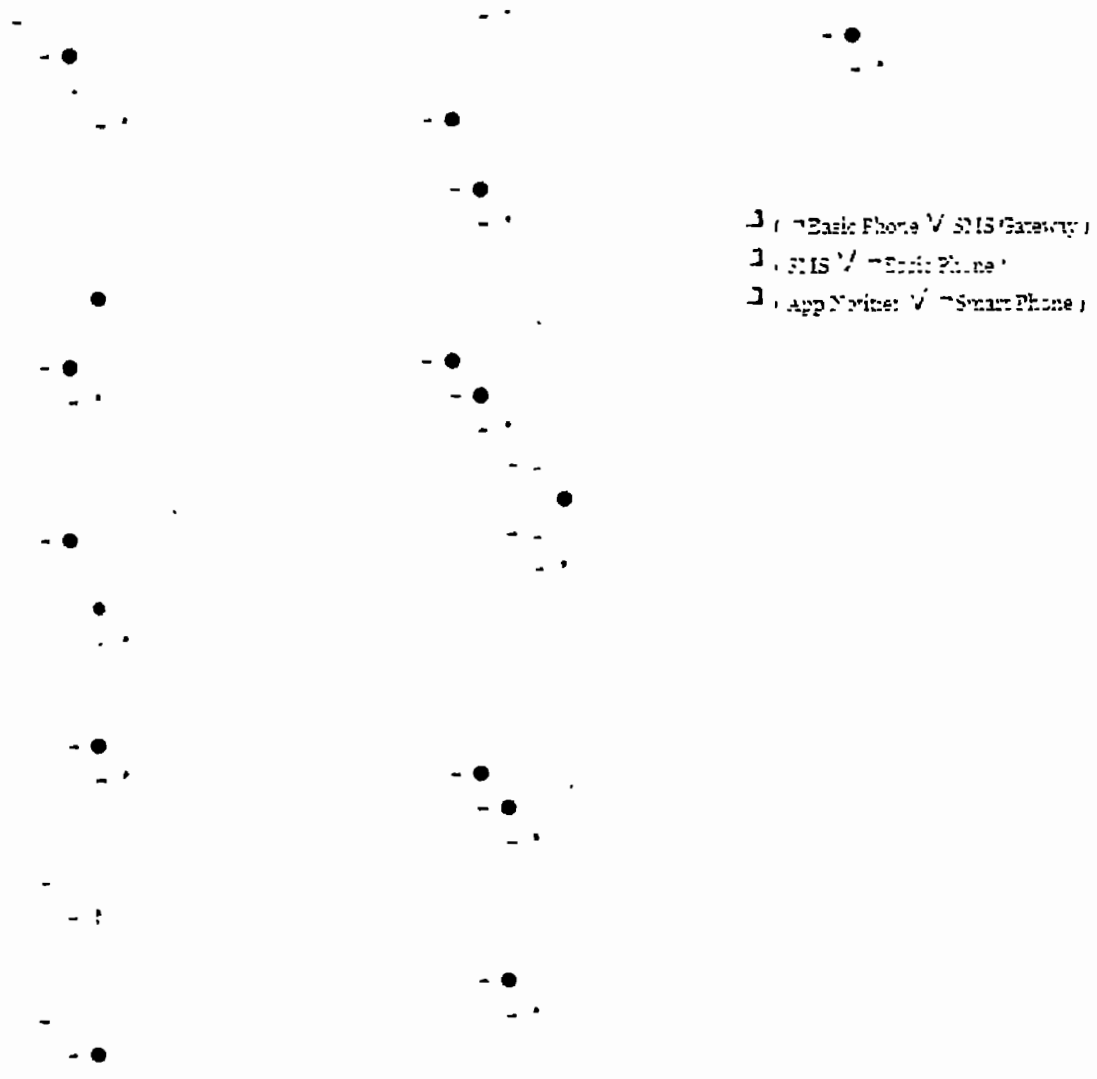


Figure 5.10: Large feature model with less CTC (2)

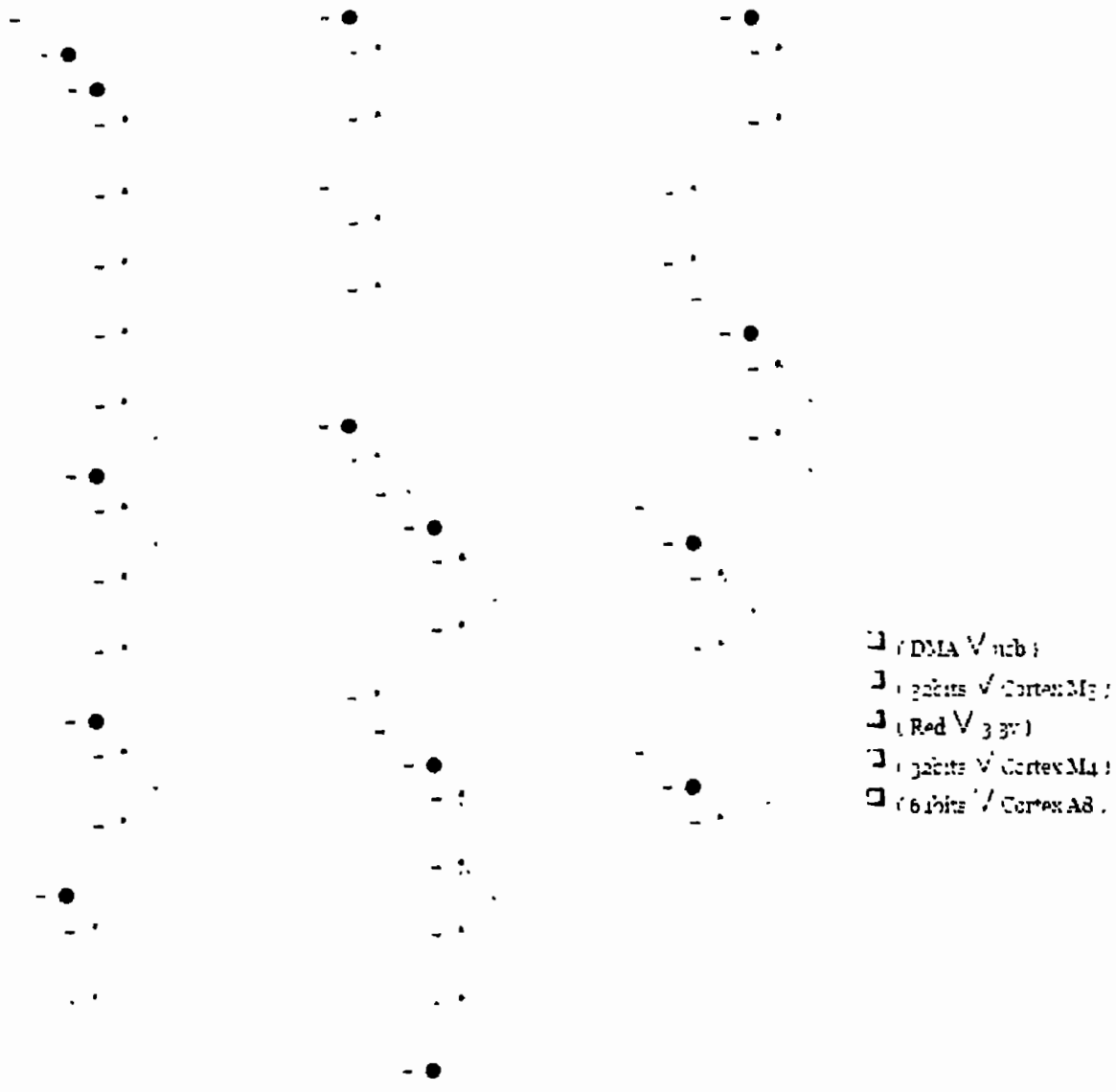


Figure 5.11: Large feature model with less CTC (3)

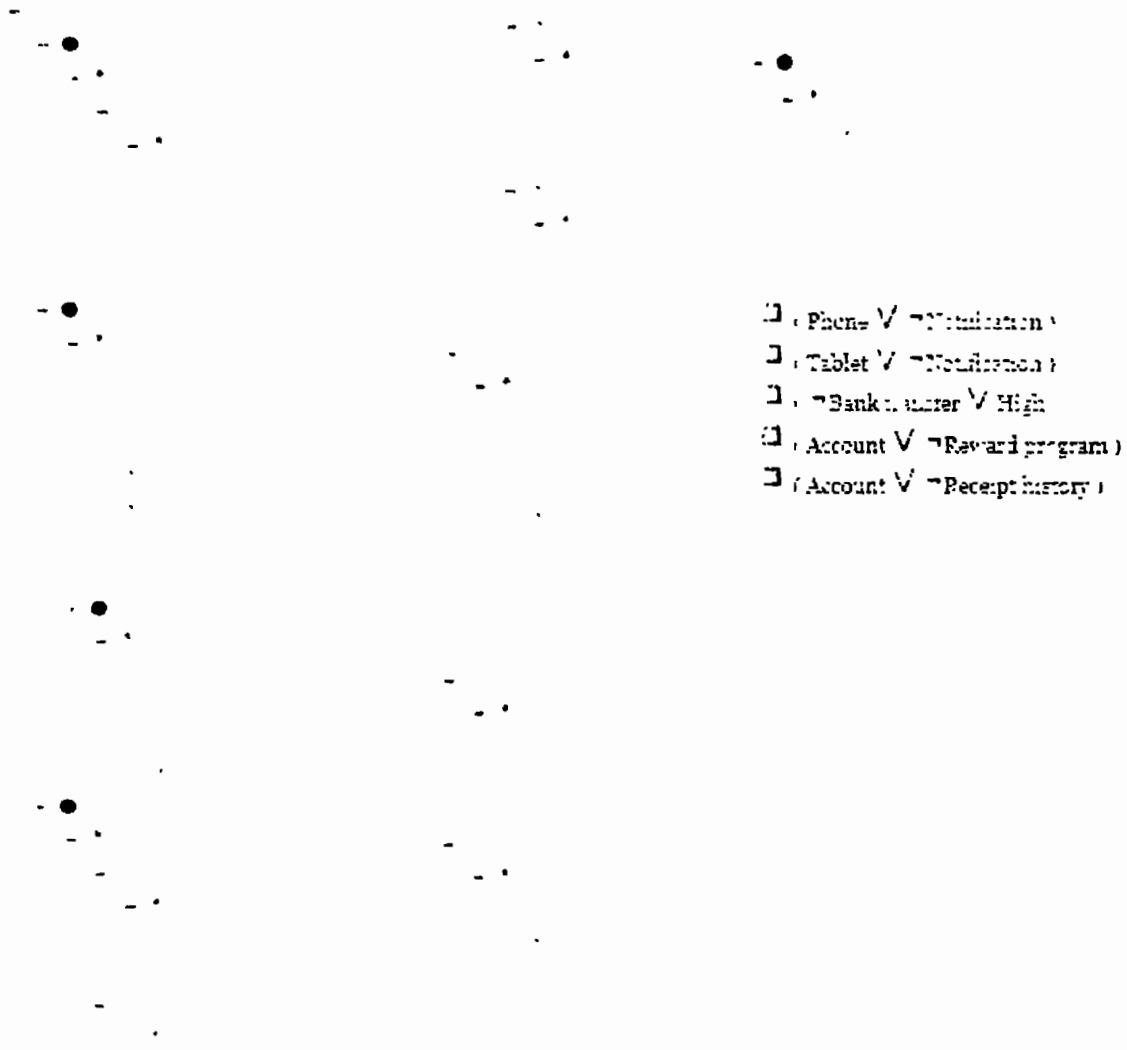


Figure 5.12: Large feature model with less CTC (4)

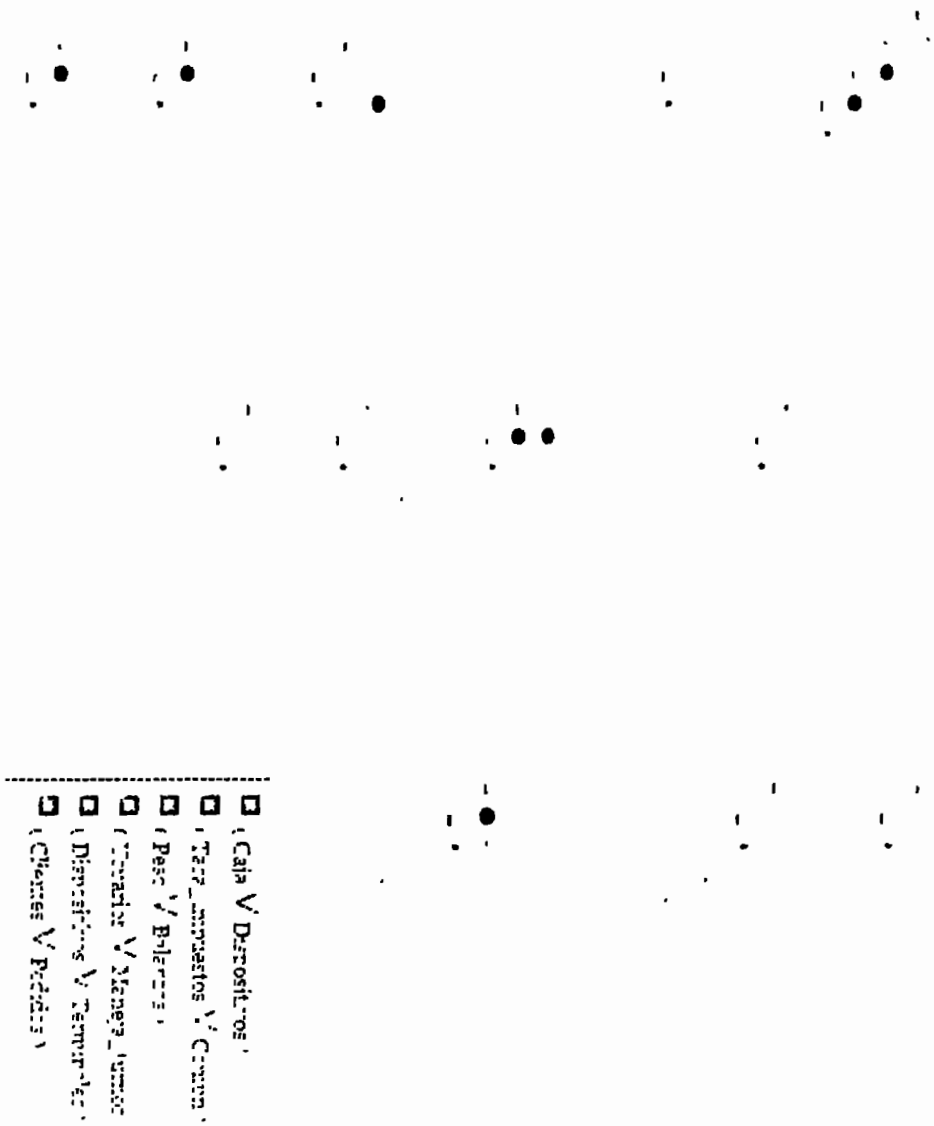


Figure 5.13: Large feature model with less CTC (5)

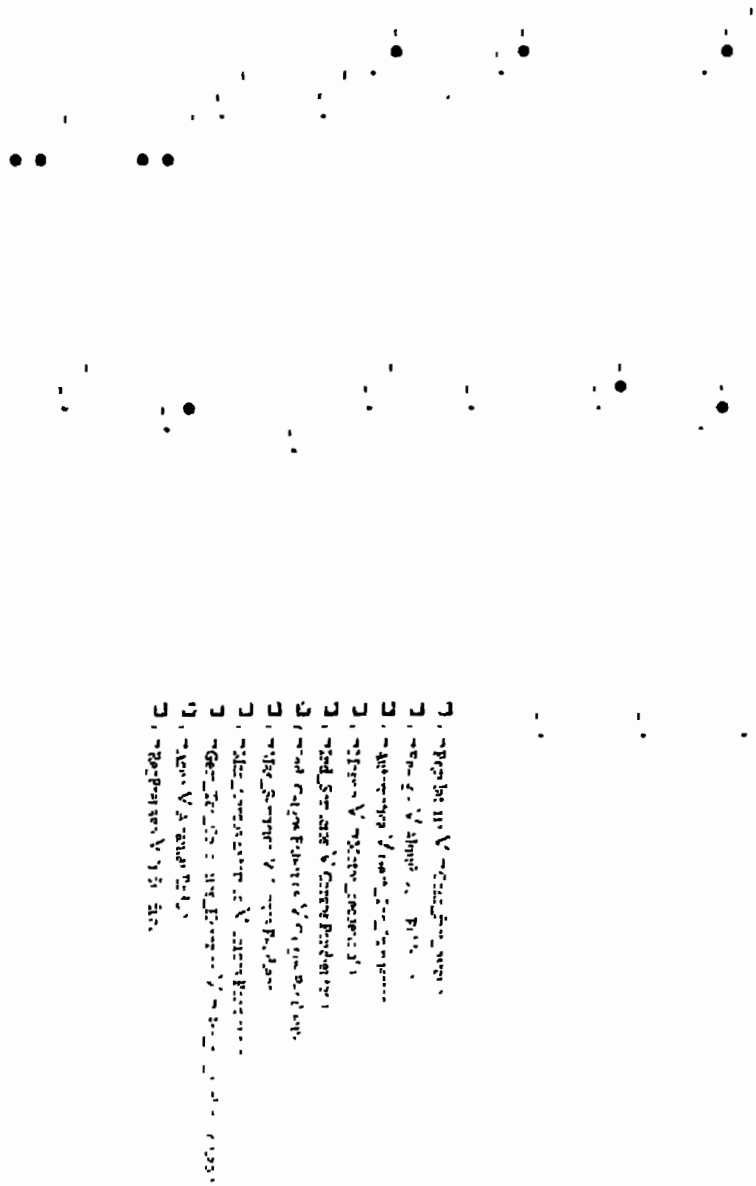


Figure 5.14: Large feature model with mediam size of CTC

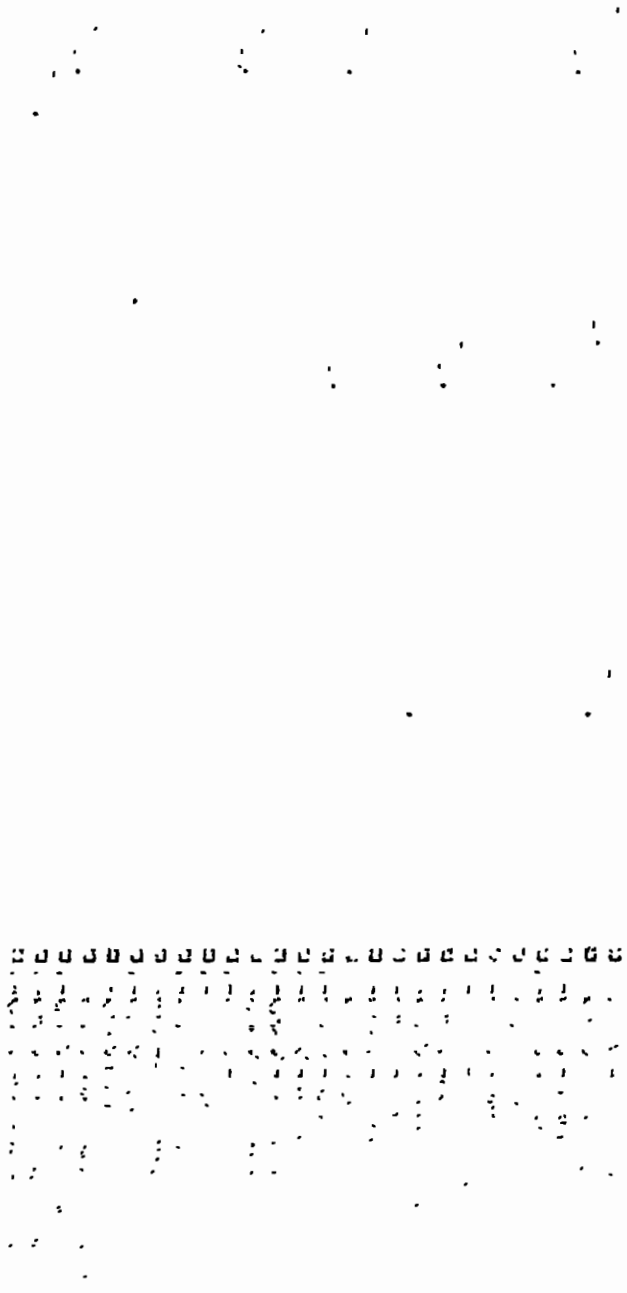


Figure 5.15: Mediam size feature model with large CTC

Table 5.3: BOFS-CTC Valid configuration of 50 to 60 features

Feature Model: 51 to 60 Feature							
Feature Model	Mandatory	Optional	XOR group	OR group	CTC	Invalid Products	Valid Products
Mobile Visit Guide	16	8	6	5	2	40105	1636800
Rout Finder	10	1	7	11	6	240123	9997020
Video Player	13	16	0	9	1	200945	482549760
Tienda Virtual	14	10	2	8	7	647855	168972288
Feature Model SibRAM	16	7	5	7	3	326980	387760128
SD Model	13	7	22	11	5	47	128
Ecommere	5	7	1	11	5	More than 2 million	More than 1 billion
Facturacion	10	6	5	10	11	More than 2 million	137582550
Market Place Nootbook	0	7	5	7	20	More than 4 million	More than1 billion
Tiend POS	8	14	4	6	6	More than 1 million	More than 1 billion

Chapter 6

Conclusions and Future Directions

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, finding the total valid number of products to calculate advance SPL cost is essential. The total valid number of products is a crucial parameter that needs to be determined early in SPL domain development. Invalid product configurations become the cause of wrong cost estimation as well as invalid application development in the domain of SPL. Invalid product configurations are due to the violation of relationships and crosstree constraints between features of the feature model. Therefore, finding the total number of valid product configurations that do not violate the essential connections and crosstree constraints of the feature model is necessary. In this work, we have proposed a BOFS-CTC framework and algorithm for valid product configurations that do not violate the feature model's essential relationships and crosstree constraints. BOFS-CTC is a sequential approach based on mathematical equations to apply to feature models according to the relationships and crosstree constraints. By comparing the features of the feature model according to their relationship and crosstree constraints, BOFS-CTC eliminates the invalid product configurations and generates the valid product configurations. BOFS-CTC is applied early in SPL's domain engineering to find the advanced initial development cost of complete products. The BOFS-CTC framework is used to find the total number of valid products by applying the mathematical equations that calculate the invalid products and then remove these products from the total number of products. As a result, we find the total valid products.

Furthermore, we have developed the BOFS-CTC algorithm based on the framework of mathematical equations. BOFS-CTC algorithm finds the total valid product configurations automatically.

The dataset of the BOFS-CTC framework is a complete feature model structure, i.e., variable features with their relationships (optional, alternate, optional group, alternate group) and crosstree constraints. The algorithm computes the total valid product configurations by categorizing each variable feature (optional, alternate) and then maps the crosstree constraints between these features predefined in the dataset.

We have applied the BOFS-CTC algorithm on various large and small feature models, from low to high complexity feature constraints. Our results show no crosstree constraints and relationship violations and found valid product configurations. We verified our results by comparing the valid products of BOFS-CTC with sampling techniques of different product configurations.

From the experimental results of the BOFS-CTC algorithm, we found a significant difference between the total number of products and valid product configuration. This indicates that the development cost of SPL also decreases due to eliminating the invalid product configurations from the domain of SPL. Resultantly, it also reduces resource utilization and enhances the integration testing of features in the feature model. Integration testing, i.e., the compatibility of features in application development from the total number of valid products, is more accessible due to eliminating invalid product configurations.

6.1 Potential Impact of Research

Reduced Rework and Debugging Activities: Errors in feature configurations can be found early in the process by recognizing and controlling crosstree restrictions. As a result, configuration issues that would have needed to be fixed later in the development cycle can now be set with a significant reduction in rework and debugging efforts. The time, resources, and expenses necessary to resolve errors are reduced when they are fixed as soon as possible.

Prevention of Downstream Issues: Violations of crosstree constraints can result in invalid setups that can have a cascading effect on downstream procedures. The likelihood of further issues like compatibility issues, data inconsistencies, or functional failures is significantly decreased by identifying and fixing these problems early on. This preventative strategy contributes to the overall software system's stability and integrity.

Reduced Rework and Debugging Activities: By identifying and managing crosstree constraints, feature configuration errors can be discovered early. As a result, configuration issues that required fixing later in the development cycle may now be resolved with a significantly lower investment

in rework and debugging. When faults are corrected as quickly as feasible, the time, money, and resources needed to rectify them decreases.

Prevention of Downstream Problems: Crosstree constraint violations can lead to invalid setups that might negatively impact downstream procedures. The risk of further problems like compatibility problems, data inconsistencies, or functional failures is significantly reduced by spotting and correcting these errors as soon as they arise. This preventative measure enhances the overall stability and integrity of the software system.

Streamlined Development Process: Early error detection sanctioned the development process to run more smoothly. The development team can maintain a constant development pace without substantial disruptions or delays by fixing configuration issues early. This enhances the development process's general effectiveness and productivity.

Facilitates Agile and Iterative Development: Agile and iterative development approaches are compatible with effective cross-tree constraint management. Early error detection and correction make quick iterations and course adjustments possible, guaranteeing that the software system develops based on tested and trustworthy settings. The development process is more agile, accelerating market time and improving responsiveness to shifting client needs.

In conclusion, the early error detection attained by recognizing and managing crosstree constraints saves rework, averts downstream problems, boosts testing effectiveness, raises customer happiness, streamlines the development process, and is by agile techniques. It guarantees a development lifecycle that is more effective and efficient, with fewer risks and higher overall software quality.

References

[1] S. Dogan, N. A. Dogan, I. Celik, "Teachers' skills to integrate technology in education: Two path models explaining instructional and application software use". Education and Information Technologies. Vol. 26 no. 13 pp. 11-32, Jan 2021.

[2] C. Watson, N. Cooper, D. N. Palacio, K. Moran, D. Poshyvanyk, "A systematic literature review on the use of deep learning in software engineering research". ACM Transactions on Software Engineering and Methodology (TOSEM). Vol. 31 no. 2, pp. 1-58 March 2022.

[3] D. Cock, A. Ramdas, D. Schwyn, M. Giardino, A. Turowski, Z. He, N. Hossle, D. Korolija, M. Licciardello, K. Martsenko, R. Achermann, "Enzian: an open, general, CPU/FPGA platform for systems software research," InProceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 434-451, Feb 2022.

[4] M. A. Akbar, K. Smolander, S. Mahmood, A. Alsanad, "Toward successful DevSecOps in software development organizations: A decision-making framework", Information and Software Technology. no. 147 pp. 68-94, July 2022.

[5] D. D. Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, M. Wimmer, "Low-code development and model-driven engineering: Two sides of the same coin?", Software and Systems Modeling. Vol. no. 4pp. 37-46, Apr 2022.

[6] M. Lochau, and J. Kamischke, "Parameterized preorder relations for model-based testing of software product lines", In International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, Springer, Berlin, Heidelberg, pp. 223-237, 2012.

[7] P. Dhore, L. Wadhwa, P. Shinde. D. Chaudhri, P. Vyas, "Brief Review On Different Manual Software Testing Approaches and Procedure", Journal of Pharmaceutical Negative Results, pp. 455-464, Jan 2023.

[8] M. A. Hadi, F. H. Fard, "Evaluating pre-trained models for user feedback analysis in software engineering: A study on classification of app-reviews", Empirical Software Engineering. Vol. 4, pp. 88, July 2023.

[9] A. A. Pratama, A. B. Mutiara, "Software quality analysis for halodoc application using iso 25010: 2011", Int. J. Adv. Comput. Sci. Appl, Vol. 12, no. 8, 383-392, 2021.

[10] E. R. Sepasi, K. N. Balouchi, J. Mercier, and R. E. Lopez-Herrejon, "An Empirical Eye-Tracking Study of Feature Model Comprehension", arXiv preprint arXiv:2203.05068, 2022.

[11] E. R. Sepasi, K. N. Balouchi, J. Mercier, and R. E. Lopez-Herrejon, "An Empirical Eye-Tracking Study of Feature Model Comprehension", arXiv preprint arXiv:2203.05068, 2022.

[12] F. Bertolotti, W. Cazzola, L. Favalli, "Features, believe it or not! a design pattern for first-class citizen features on stock jvm", In Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume A (pp. 32-42, 2022.

[13] D. Romano, K. Feichtinger, D. Beuche, U. Ryssel, and R. Rabiser, "Bridging the gap between academia and industry: transforming the universal variability language to pure:: variants and back", In Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume B, pp. 123-131, 2022.

[14] I. Ayala, M. Amor, L. Fuentes. and A. V. Papadopoulos, "Self-adapting Industrial Augmented Reality Applications with Proactive Dynamic Software Product Lines", In 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, pp. 01-08, 2021.

[15] A. Valdezate, R. Capilla, J. Crespo, and R. Barber, "Ruva: A runtime software variability algorithm", IEEE Access, 10, pp. 52525-52536, 2022.

[16] G. Kahraman, and L. Cleophas, "A tool for modeling and analysis of relationships among feature model views", In Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume B, pp. 103-109, 2022.

[17] 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, 168, 114167, 2021.

[18] V. M. Le, A. Felfernig, M. Uta, T. N. T. Tran, and C. V. Silva, "WipeOutR: automated

redundancy detection for feature models", In Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume A (pp. 164-169, 2022.

[19] M. Kowal, S. Ananieva, and T. Thüm,"Explaining anomalies in feature models", ACM SIGPLAN Notices, 5vol. 2, no. 3, 132-143, 2016.

[20] J. Guo, J. White, G. Wang, J. Li, and Y. Wang, "A genetic algorithm for optimized feature selection with resource constraints in software product lines", Journal of Systems and Software, 84(12), pp. 2208-2221, 2011.

[21] T. Thum, C. Kastner, S. Erdweg, and N. Siegmund, "Abstract features in feature modeling". In 15th International Software Product Line Conference, IEEE, pp. 191-200, 2011.

[22] H. Holdschick, "Challenges in the evolution of model-based software product lines in the automotive domain". In Proceedings of the 4th International Workshop on Feature-Oriented Software Development, pp. 70-73, 2012.

[24] L. Ochoa, O. González-Rojas, T. Thüm, "Using decision rules for solving conflicts in extended feature models", In Proceedings of the ACM SIGPLAN International Conference on Software Language Engineering (pp. 149-160, 2015.

[24] S. Urli, A. Bergel, M. Blay-Fornarino, P. Collet, and S. Mosser, "A visual support for decomposing complex feature models", In IEEE 3rd Working Conference on Software Visualization (VISSOFT),IEEE, pp. 76-85, 2015.

[25] 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.

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

[27] J. M. Horcas, J. Ballesteros, M. Pinto, and L. Fuentes, "Elimination of constraints for parallel analysis of feature models", In Proceedings of the 27th ACM International Systems and Software Product Line Conference-Volume A, August 2023, pp. 99-110.

[28] D. Eichhorn, T. Pett, T. Osborne, and I. Schaefer. "Quantum Computing for Feature Model Analysis: Potentials and Challenges", In Proceedings of the 27th ACM International Systems and Software Product Line Conference-Volume A. August 2023. pp. 1-7.

[29] A. Wasowski, and T. Berger, "Feature Modeling. In Domain-Specific Languages: Effective Modeling, Automation, and Reuse", Cham: Springer International Publishing, pp. 437-457, 2023.

[30] J. M. Horcas, M. Pinto, and L. Fuentes, "Extensible and modular abstract syntax for feature modeling based on language constructs", In Proceedings of the 24th ACM Conference on Systems and Software Product Line: Vol. A, pp. 1-7, 2020.

[31] C. Bezerra, R. Lima, and P. Silva, "Dymmer 2.0: A tool for dynamic modeling and evaluation of feature model,. In Proceedings of the V Brazilian Symposium on Software Engineering, pp. 121-126. 2021.

[32] F. Damiani, D. Faitelson, C. Gladisch, and S. Tyszberowicz, "A novel model-based testing approach for software product lines", Software and Systems Modeling, vol. 16, no. 4, pp 1223-1251, 2017.

[33] H. Lackner, "Model-Based Product Line Testing: Sampling Configurations for Optimal Fault Detection", In International SDL Forum Springer, Cham pp. 238-251, 2014.

[34] H. Foidl, and M. Felderer, "Integrating software quality models into risk-based testing" Software quality journal, vol. 26, pp 809-847, 2018.

[35] S. Reis, A. Metzger, and K. Pohl, "Integration testing in software product line engineering: a model-based technique", In International Conference on Fundamental Approaches to Software Engineering pp. 321-335, Berlin, Heidelberg, March, 2007.

[36] F. Ensan, E. Bagheri, and D. Gašević, "Evolutionary search-based test generation for software product line feature models", In International Conference on Advanced Information Systems Engineering , Springer, Berlin, Heidelberg pp. 613-628, 2012.

[37] A. Schürr, S. Oster, and F. Markert, "Model-driven software product line testing: An integrated approach", In International Conference on Current Trends in Theory and Practice of Computer Science Springer, Berlin, Heidelberg, pp. 112-131, 2010.

[38] M. Lochau, S. Oster, U. Goltz, and A. Schürr, "Model-based pairwise testing for feature interaction coverage in software product line engineering", Software Quality Journal, vol. 20, no. 4, pp 567-604, 2012.

[39] B. P. Lamancha, M. P. Usaola, and M. P. Velthius, "A model based testing approach for model-driven development and software product lines", In International Conference on

Evaluation of Novel Approaches to Software Engineering, Springer, Berlin, Heidelberg pp. 193-208, 2010.

[40] P. Reales, M. Polo, and D. Caivano, " Model based testing in software product lines", In International Conference on Enterprise Information Systems, Springer, Berlin, Heidelberg, pp. 270-283, 2011.

[41] J. M. Ferreira, S. R. Vergilio, and M. Quinaia, "Software product line testing based on feature model mutation", International Journal of Software Engineering and Knowledge Engineering, 27(05), pp 817-839, 2017.

[42] M. Lochau, D. Reuling, J. Bürdek, T. Kehrer, S. Lity, A. Schürr, and U. Kelter, "Model-Based Round-Trip Engineering and Testing of Evolving Software Product Lines". In Managed Software Evolution, Springer, Cham, pp. 141-173, 2019.

[43] T. Bordis, T. Runge, A. Knüppel, T. Thüm, and I. Schaefer, "Variational correctness-by-construction", In Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems, pp. 1-9, 2020.

[44] T. Thüm, A. Knüppel, S. Krüger, S. Bolle, and I. Schaefer, "Feature-oriented contract composition", Journal of Systems and Software, 152, pp. 83-107, 2019.

[45] D. Marijan, A. Gotlieb, S. Sen, and A. Hervieu, "Practical pairwise testing for software product lines", In Proceedings of the 17th international software product line conference, pp. 227-235, 2013.

[46] L. Yu, F. Duan, Y. Lei, R. N. Kacker, and D. R. Kuhn, " Combinatorial test generation for software product lines using minimum invalid tuples", In IEEE 15th International Symposium on High-Assurance Systems Engineering, IEEE, pp. 65-72, 2014.

[47] M. Rosenmuller, N. Siegmund, G. Saake, and S. Apel, "Code generation to support static and dynamic composition of software product lines", In Proceedings of the 7th international conference on Generative programming and component engineering, pp. 3-12, 2008.

[48] S. Apel, H. Speidel, P. Wendler, A. Von Rhein, and D. Beyer, "Detection of feature interactions using feature-aware verification", In 26th IEEE/ACM International Conference on Automated Software Engineering, IEEE, pp. 372-375, 2011.

[49] J. White, D. C. Schmidt, D. Benavides, P. Trinidad, and A. Ruiz–Cortés, "Automated diag-

nosis of product-line configuration errors in feature models", In 12th International Software Product Line Conference, IEEE, pp. 225-234, 2008.

[50] M. Tanhaei, J. Habibi, and S. H. Mirian-Hosseinabadi, "A feature model based framework for refactoring software product line architecture", Journal of Computer Science and Technology, vol. 31, no. 5, pp 951-986, 2016.

[51] M. T. Fulop, M. Guban, A. Guban, M. Avornicului, "Application research of soft computing based on machine learning production scheduling". Processes, Vol. 10, no. 3, 520, 2022.

[52] A. Hervieu, B. Baudry, and A. Gotlieb, " Pacogen: Automatic generation of pairwise test configurations from feature models", In IEEE 22nd International Symposium on Software Reliability Engineering , IEEE , pp. 120-129, 2011.

[53] S. Apel, D. Batory, C. Kästner, and G. Saake, "Software Product Lines. In Feature-Oriented Software Product Lines", Springer, Berlin, Heidelberg, pp. 3-15, 2013.

[54] F. Benduhn, T. Thum, I. Schaefer, and G. Saake, "Modularization of refinement steps for agile formal methods", In International Conference on Formal Engineering Methods, Springer, Cham, (pp. 19-35, 2017.

[55] S. Apel, A. Von Rhein, P. Wendler, A. Größlinger, and D. Beyer, "Strategies for product-line verification: case studies and experiments", In 35th International Conference on Software Engineering (ICSE), IEEE, pp. 482-491, 2013.

[56] S. Ben-David, B. Sterin, J. M. Atlee, and S. Beidu, "Symbolic model checking of product-line requirements using sat-based methods", In IEEE/ACM 37th IEEE International Conference on Software Engineering, IEEE, Vol. 1, pp. 189-199, 2015.

[57] C. Dubslaff, S. Klüppelholz, and C. Baier, "Probabilistic model checking for energy analysis in software product lines". In Proceedings of the 13th international conference on Modularity, pp. 169-180, 2014.

[58] S. Bessling, and M. Huhn, "Towards formal safety analysis in feature-oriented product line development", In International Symposium on Foundations of Health Informatics Engineering and Systems, Springer, Berlin, Heidelberg, pp. 217-235, 2013.

[59] T. Thüm, J. Meinicke, F. Benduhn, M. Hentschel, A. Von Rhein, and G. Saake, "Potential synergies of theorem proving and model checking for software product lines", In Proceedings of the 18th International Software Product Line Conference, Vol. 1, pp. 177-186, 2014.

[60] C. H. P. Kim, D. Marinov, S. Khurshid, D. Batory, S. Souto, P. Barros, and M. d'Amorim, "SPLat: Lightweight dynamic analysis for reducing combinatorics in testing configurable systems", In Proceedings of the 9th Joint Meeting on Foundations of Software Engineering, pp. 257-267, 2013.

[61] J. V. Millo, S. Ramesh, S. N. Krishna, and G. K. Narwane, " Compositional verification of software product lines", In International Conference on Integrated Formal Methods, Springer, Berlin, Heidelberg, pp. 109-123, 2013.

[62] T. Thüm, A. Knüppel, S. Krüger, S. Bolle, and I. Schaefer, "Feature-oriented contract composition. Journal of Systems and Software", Vol. 152, pp. 83-107, 2019.

[63] M. Lochau, D. Reuling, J. Bürdek, T. Kehrer, S. Lity, A. Schürr, and U.Kelter, "Model-Based Round-Trip Engineering and Testing of Evolving Software Product Lines", In Managed Software Evolution, Springer, Cham, pp. 141-173, 2019.

[64] T. Bordis, T. Runge, A. Knüppel, T. Thüm, and I. Schaefer, " Variational correctness-by-construction". In Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems, pp. 1-9, 2020.

[65] Z. Jin, "Environment modeling-based requirements engineering for software intensive systems", Morgan Kaufmann, 2018.

[66] X. Li, W. E. Wong, R. Gao, L. Hu, and S.Hosono, "Genetic algorithm-based test generation for software product line with the integration of fault localization techniques", Empirical Software Engineering, 23(1), pp. 1-51, 2018.

[67] E. Bagheri, T. Di Noia, A. Ragone, and D. Gasevic, "Configuring software product line feature models based on stakeholders' soft and hard requirements", In International Conference on Software Product Lines, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 16-31, 2010.

[68] J. Van Gurp, J. Bosch, and M. Svahnberg, "On the notion of variability in software product lines", In Proceedings Working IEEE/IFIP Conference on Software Architecture, IEEE, pp. 45-54, 2001.

[69] K. Czarnecki, M. Antkiewicz, C. J. Kim, S. Lau, K. Pietroszek, "Model-driven software product lines",. In Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pp. 126-127, Oct 2005.