A GRAPH BASED FEATURE SELECTION APPROACH FOR CONFIGURING SOFTWARE PRODUCT LINES

BY

MOHAMMAD AHSAN JAVED

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SOFTWARE ENGINEERING

MAY 2016

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **MOHAMMAD AHSAN JAVED** under the direction his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN SOFTWARE ENGINEERING.**

Dr. Sajjad Mahmood
(Advisor)

Dr. Adel Fadhl Noor Ahmed
Department Chairman

Dr. Mohammad Rabah Alshayeb
(Member)

Dr. Salam A. Zummo
Dean of Graduate Studies

Dr. Mahmood Khan Niazi
(Member)

6/11/16

Date

To my beloved father, without whom none of my success would be possible.

ACKNOWLEDGMENTS

First and foremost, I praise Allah the Almighty, without whom it would have never been possible for me to complete this thesis in such an esteemed institution. Secondly, I send my salutations to Prophet Mohammad (P.B.U.H.). I then thank my father, mother and siblings for being the constant source of support, prayers and inspiration throughout my life and the journey towards the completion of this thesis.

I express my deep thanks to my Thesis Advisor, Dr. Sajjad Mahmood, for believing in me and taking me under his mentorship. It was his brilliance and relentless help that kept me going and focused on the right track, to achieve this milestone. It seems highly fit to thank Dr. Mohammad Rabah AlShayeb and Dr. Mahmood Khan Niazi, for teaching me the art of research and honing my academics by allowing me to have some pearls from their knowledgeable minds.

I genuinely thank my friends, who were there for me through thick and thin, especially, Mohsin Javed, Abdullah Siddiqui, Osamah Al Dhafer, Haris Mumtaz and Samran Naveed. In the end, I thank all those who offered me their help and prayers, at occasions where I was low and assisted me in climbing out of those pit holes.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

PL	:	Product Line
SPL	:	Software Product Line
FDG	:	Feature Dependency Graph
FM	:	Feature Model
CLF	:	Concrete Level Feature
GA	:	Genetic Algorithm
GGA	:	Grouping Genetic Algorithm
FODA	:	Feature Oriented Domain Analysis
AS	:	Automotive System
LCA	:	Life Cycle Assessment

ABSTRACT

Full Name : Mohammad Ahsan Javed

Thesis Title : A Graph Based Feature Selection Approach for Configuring Software Product Lines

Major Field : Software Engineering

Date of Degree : May, 2016

A Software Product line (SPL) is configuration-centric with a focus on developing a collection of related software products, all of which share some core functionality, and differ in some specific features. SPL uses a feature model to specify the commonalities and variabilities in terms of software features, to identify and develop reusable software assets. Selection of features is a key process to derive new SPL configurations that aim to either minimize or maximize a business objective, subject to a set of constraints. To date, feature selection techniques have focused on finding an optimal solution to objective functions such as cost and resource constraints. However, the existing approaches have not considered the structural relationships and configuration dependencies encoded in a feature model, together with the business objectives, leaving open the question of how best to optimize SPL feature selection in the presence of feature interdependencies. In this research thesis, we have developed a feature selection technique that consolidates interdependent features into related clusters and uses a Genetic algorithm (GA) to find a near optimal solution for feature selection with respect to clustering error, product priority and product integrity. This thesis provides a solution to SPL feature selection problem such that it helps a developer to analyze interdependencies and select suitable features for SPL configurations. This thesis also applies the approach on two case studies to evaluate the workings of the developed approach.

ملخص الرسالة

الاسم الكامل : محمد احسن جاويد

عنوان الرسالة : منهج اختيار الخاصية المعتمد على الرسم لتكوين خط انتاج البرمجيات

التخصص : هندسة البرمجيات

تاريخ الدرجة العلمية : مايو, 2016

خط انتاج البرمجيات هو التكوين المركزي بالاعتماد على تطوير مجموعة من المنتجات البرمجية المرتبطة ببعضها، وكلها تشترك في بعض الوظائف الأساسية، وتختلف في بعض السمات المحددة. خط انتاج البرمجيات يستخدم نموذج يعتمد على الخاصية لتحديد القواسم المشتركة والمتغيرات من حيث خصائص البرمجيات، لتحديد وتطوير برمجيات يمكن إعادة استخدامها. الاختيار من بين هذه الخصائص هي عملية أساسية لإنتاج تكوينات جديدة لخط انتاج البرمجيات التي تهدف إما إلى تقليل أو زيادة الهدف التجاري، والذي عادة يخضع لمجموعة من القيود. حتى الآن، تقنيات اختيار الخاصية ركزت على إيجاد الحل الأمثل لدالة الهدف مثل القيود في التكلفة والموارد. مع ذلك، فان الطرق الحالية المستخدمة لا تعتمد العلاقات الهيكلية والاعتمادات التكوينية المشفرة في النموذج المعتمد على الخاصية، جنبا إلى جنب مع أهداف المؤسسة، وترك الباب مفتوحا أمام مسألة كيفية ايجاد أفضل السبل لتحسين اختيار الخصائص لخط انتاج البرمجيات في ظل وجود خاصية الترابط. في هذه الأطروحة البحثية، قمنا بتطوير تقنية اختيار الخصائص التي تعزز الخصائص المترابطة في مجموعات ذات صلة، وباستخدام الخوارزمية الجينية لإيجاد أقرب حل للحل الأمثل لاختيار الخاصية مع الاخذ بالاعتبار الخطأ الوارد من توزيع المجموعات، أولوية المنتج وسلامة المنتج. وتقدم هذه الأطروحة حل لمسالة لاختيار الخاصية لخط انتاج البرمجيات والتي تساعد المطور في تحليل الترابط واختيار الخاصية المناسبة لتكوينات خط انتاج البرمجيات. هذه الاطروحة أيضا تم تطبيقها على اثنين من دراسات الحالة لتقييم عمل المنهج المقترح.

CHAPTER 1

INTRODUCTION

Over the last decade, the extensive use of software has placed new challenges for the software industry in terms of expectations to enhance development productivity, quality and reduce associated costs [1]. These expectations have influenced the software industry to recreate the idea of reuse [2, 3]. Software product line is an approach with an aim to move the software engineers away from developing each system from scratch. It focuses on developing a set of software systems in a domain that shares more commonalities than uniqueness [3] while developing software systems by releasing product variants. Due to these advantages, SPL development has been utilized in a variety of software applications such as mobile phones, elevator control systems, and the list keeps growing. In this research thesis, we adopt Clements and Northrop's definition of software product line [3]: *"software product line is a software engineering approach for creating configurable software applications that can be adapted to a variety of requirement sets"*.

The success of SPL project requires a substantial initial investment for the development of core assets. This development process of the core assets needs to maximize the coverage of the domain in a product line within budget and a given time frame. SPL core asset development consists of two main activities: (a) Domain engineering and (b) Application engineering [4]. Domain engineering process defines the commonality and variability of the product line while application engineering process deals with the

development of the applications of the product line by reusing domain artifacts for a specific set of requirements.

Feature modeling [5-8] is the de-facto standard to represent core assets of a SPL. A feature model expresses the commonality and variability in a product line in terms of features [9]. In a feature model, features are units of capability that are delivered to customers, product configuration and configuration management, parameterization for reusable components and product management for targeting specific market segments [7, 10, 11]. A feature model shows a set of features in a hierarchical arrangement that describes successive refinements of the variability in a product line [9, 12]. The common and variable features in a feature model are organized using structural relationships like aggregation and generalization. Furthermore, the organization of the features also use configuration relations which are defined in terms of different type of dependencies [6, 13].

Previous research indicates that deriving SPL configurations is a time consuming and expensive activity [11, 14-16]. A major challenge while deriving valid SPL configurations using feature model, is determining, for a given set of constraints, how an optimized feature selection can be found [11]. The process of feature selection needs to consider the structural and configuration dependencies encoded in a feature model when grouping related features into clusters. The consolidation of related features into clusters, considering the dependencies among them, is an essential aspect but it is not sufficient on its own. The feature selection process also needs to consider how feature assignments to products affect different desired objectives required of a product line like value and integrity of product line [2]. Hence, it is implied that the feature selection process in SPL

should not be opportunistic; it should be carefully planned while considering feature dependencies and keeping a balance among the different objectives of a SPL.

1.1 Research Objectives

In this research work, the feature selection problem is represented as a multi-objective optimization problem; with the desired system attributes formulated as objective functions. We present a feature selection process that helps SPL developers to consolidate interdependent features into clusters and use a genetic search algorithm to search for solutions with minimum clustering error, maximum product integrity and maximum priority of the product.

The overarching objective of this thesis is to present a software feature selection process that provides guidelines for software product line developers to consolidate related requirements into clusters and then optimally select the consolidated feature sets based on the user preferences. The objectives of this research can be formally stated as the following:

Objective**:** Rectify the problem of multi-objective feature selection in software product lines by providing optimal (or near optimal) solutions while considering the impact of dependency relationships among features. This objective is divided into following sub-objectives:

Sub-Objective 1: Develop dependency analysis technique for SPL features and model inter-dependency between SPL features.

Sub-Objective 2: Adapt signed graph clustering for consolidating SPL features.

Sub-Objective 3: Apply Genetic Algorithm to optimize feature selection in SPL.

Sub-Objective 4: To present application of our approach to two case studies.

Addressing above objectives will assist software product line development organizations in better understanding, planning and managing software feature selection decisions in software product line development projects.

1.2 Thesis Outline

The remainder of the thesis is organized as follows: Chapter 2 gives the background on the topics that must be highlighted before going deep into the workings of the developed approach. Chapter 3 defines the research methodology and framework and explains the approach step by step. Chapter 4 applies the approach to two case studies separately and depicts the results achieved. Chapter 5 discusses the results and compares the solutions. Chapter 6 concludes the thesis, states the outcomes of this thesis, points out the threats to validity and mentions the future directions to this research.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

Before moving on to the research specifics, it is important to highlight some fundamental background information on the topics that will help in understanding the problem fully, leading to the complete understanding of the approach this research dictates.

This chapter is comprised of four sections. Section 2.1 gives a background on software product lines. Section 2.2 describes feature modeling and the different type of dependencies that can exist among the features. Section 2.3 discusses the different approaches and algorithms that attempt to solve this feature optimization problem. Another section, Section 2.4, identifies the open problems and formulates the problem that we have solved in this research. Section 2.5 introduces the Grouping Genetic Algorithms (GGA). This chapter then concludes with Section 2.6, where the different approaches of GGA are mentioned alongside their applications.

2.1 Software Product Line Overview

Software pervades every sector. It has become the bottom line for many organizations, even those who never envisioned themselves in the software business are heavily involved with software [3]. Businesses now expect improved efficiency and productivity from software to help achieve their business goals like *low-cost production, high quality, and quick time to market, etc.* In such circumstances, a reuse strategy makes more sense. Although there are several reuse strategies SOA [17], CBS [18] that are being practiced by the industry, they have a little economic effect [3]. Traditional reuse techniques' focus

is small-grained, opportunistic, and technology-driven rather than meeting business goals. There is a need for strategic reuse to achieve business benefits [3].

This is where the innovative and growing concept in software engineering, Software Product Lines (SPL), is introduced. A *software product line* is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way. It is a new application of a proven concept in several engineering fields. Many businesses have put the product-line concept to their advantage achieving their business goals in less time. Few examples include the *Celsiustech's Ship System 2000, Cummins Inc.'s Diesel Control Systems, National Reconnaissance Office/ Raytheon's Control Channel Toolkit, Market Maker GMBH's Merger, Nokia Mobile Phones.* These companies gained a lot of benefits like major cost cuttings, a drastic decrease in the time to market etc. The success of SPL in software engineering suggests the future of reuse lies in SPL.

There are two approaches to software product line, a proactive and a reactive one. We will be using the former approach since it is more consistent with the SPL principle of proactive reuse [3]. Moreover, since we are catering the situation where SPL has to be newly introduced in a place where there is no prior use of SPL; proactive evolution was the right approach to adopt.

2.2 Feature Modeling and Dependency Analysis

Kang et al. [5] first introduced the feature models as part of Feature-Oriented Domain Analysis (FODA) for SPLs. Features are units of capability that are delivered to customers, product configuration and configuration management, parameterization for

reusable components and product management for targeting specific market segments [7, 10, 11]. A feature model is a hierarchically arranged set of features showing the relationships and dependencies between a set of features for a SPL.

Typically, a feature model specifies structural relationships, configuration dependencies and operational dependencies between features of a SPL [6, 13]. We discuss the main types of feature dependencies as follows:

- "Required Dependency" – It exists between two features if a feature that is selected in a product requires the presence of another feature in the same product.
- "Excluded Dependency" – It is present among two features if a feature is selected in a product where the other feature cannot be selected.
- "Usage dependency" – It's there among the features if a feature depends on another feature in order to correctly function or be implemented.

A sample feature model can be seen in Figure 1, followed by the description of several dependencies that exist between the features of the depicted feature model.

For instance, in Figure 1, there exists a "required" dependency between maps and GPS feature; whenever a maps feature is selected the GPS will also be selected. The dependency is due to the fact that in real-world setting the maps are a useless feature without the GPS, not vice versa. Similarly, a "required" dependency between iPhone6 and fingerPrint and retinaScan where, whenever an iPhone6 feature is selected, the fingerPrint and retinaScan feature must also be present in the product. The dependency was based on the fact that iPhone6, being the latest and flagship mobile for Apple Inc., it

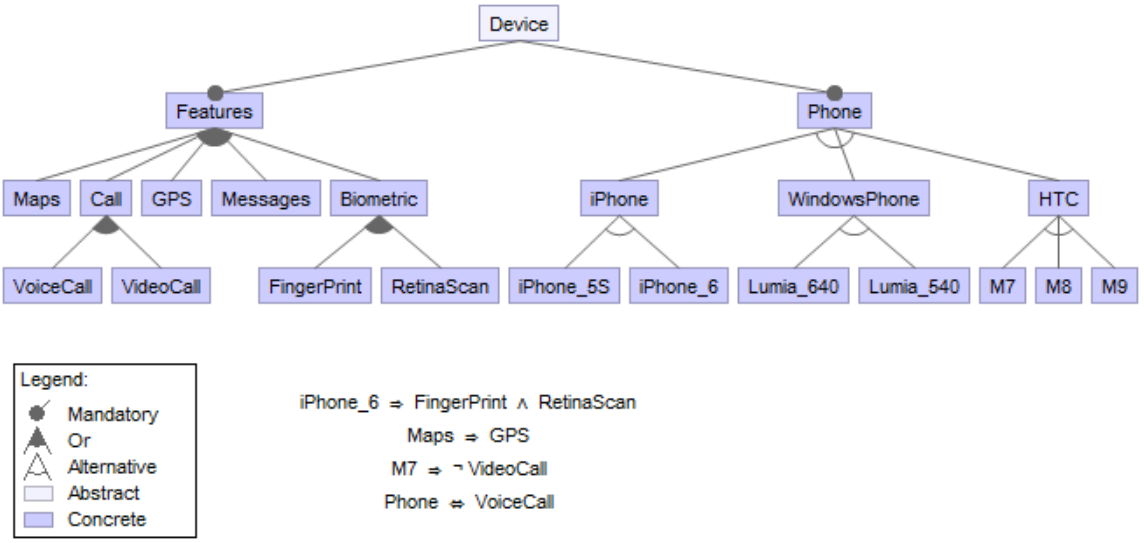

Figure 1 - Feature model for Smartphone

contains both the fingerprint and retinaScan features by default. Next, a "required" dependency between phones and voiceCall feature, which is quite logical when it is put to real-life environment, where a mobile phone is of no use if it unable make a call, hence the phone feature will have the voiceCall feature by default. In addition to these three "required" dependencies, there exists an "excluded/threat" dependency which resides between m7 and videoCall feature, which states that an m7 feature can never have a videoCall feature since in real-life settings m7 is not able to support the video calling feature.

2.3 Feature Selection Approaches

It can be seen in the previous researches that finding an optimal feature selection is a NP-hard problem [11, 19]. Researchers indicate similarity between feature selection with resource constraints and configuration optimization problems as addressed by rest of the automated feature selection approaches in the literature who have not consider potential constraints [20]. A significant number of researchers have applied techniques like BDD [21], CSP [22], and SAT solvers [23] to solve the product line feature selection problem,

but they haven't considered resource constraints. Furthermore, the time complexities for these techniques is shown to be exponential [11, 19]. Similarly, there have been efforts by researchers in developing polynomial-time approximation algorithms to select highly optimal feature sets [19].

Furthermore, there are many researchers in recent times that are trying to solve this feature optimization problem along the consideration of resource constraints and user requirements. In a certain research, Henard et al. [1] introduce an algorithm called SATIBEA, where the authors address the problem by combining constraint solving with multi-objective search-based optimization. The research evaluates the algorithm over five large real world SPLs considering some quality indicators and diversity measures. Furthermore, this research demonstrates the significance of using constraint solving with search-based approaches. To use the two techniques together they consider two key aspects called diversity promotion and searching via smart operators. By using this they show how SATIBEA outperforms the 'Sate-of-the-art' Indicator based evolutionary algorithm (IBEA) [24].

In another research, Guo et al. [11] tries to solve this problem by using a GA-based AI approach which they name as GAFES. Here they generate a set of related feature while simultaneously considering the resource constraints. They first generate encodings of products randomly to initialize the modified GA and then use their functions to eradicate the invalid configurations using their approach.

Wang and Pang [25] attempts to solve this problem. In order to solve, they preprocess the feature model by modifying the feature model such that the features that have some user

constraints are always selected. Then they convert this model into a graph using defined rules, which follows the optimization step where they apply the Ant colony optimization algorithm. However, their technique only caters to the *required* and *excluded* dependencies and ignores the other types of dependencies mentioned in section 2.2, which may result in products that don't conform to the feature dependencies that exist among the features.

There are several researchers [26, 27] that augment the feature model with some attribute and the use it to solve the multi-objective optimization problem. The attributes act as the resource constraints that the researchers try to cater while optimizing the feature selection process. Both the researchers add the same attributes to the feature but, with different value ranges assigned to each attribute. Sayyad et al. [26] defined optimization objectives which guide their search for optimized feature selection. Due to the optimization objectives selected, the research results in a viewpoint that is only useful for researchers but not the business end users which negates the ultimate purpose of using the product-line engineering. On the other hand, Lian [27] explicitly mentions the feature attributes as NFRs and then solves the problem using the two-dimensional fitness function to also integrate the user preferences. Their algorithm IVEA does the selection in one step which tends to increase the time of the algorithm takes. Moreover, they tested it on two feature models which make the results hard to be generalized upon a variety of feature models available.

In a similar research Lizhang [28] shows two evolutionary algorithm templates to solve the problem of feature selection. First, to simplify the treatment for different feature constraints they encode them into a uniform format, defining them as rules using

Chomsky grammar. They then use their solutionRevies algorithm upon IBEA technique, Sayyad et al. [29], to select features using optimization objectives. However, in this paper, they assume that NFRs considered are quantified and classified.

Li et al. [4] attempts to solve this problem from a new perspective using an approach called 0-1 programming. They transform the feature model constraint into inequality constraint to make them compatible to be solved by linear programming. Although they claim to solve this problem of feature selection in less polynomial time, they assume there is only one kind of resource to be consumed, which is not the case in real life situations.

The literature surveyed is evaluated in the light of the following criterion: (a) Consideration of features interdependencies/Cross-tree constraints (b) Stated Objectives (c) Experimental or Theoretical base/Validation (V), which helps in identification of the problems that remain open in the domain of software product line engineering.

Based on the discussion in Section 2.3, we formalized the problem for our research, by addressing open problems that were posed by the literature surveyed.

2.4 Addressing the open problems

Despite the interest in software feature selection problem, there are some key research questions with regards to the selection of suitable features that remain open. The approach for feature selection that we have developed in this research addresses the open problems raised in the literature. For example, Cho et al. [13] advocate the need to analyze and consider feature dependencies before designing and developing product line assets.

The proposed approach does consider different types of dependencies encoded in a feature model, as mentioned in Section 2.2. Similarly, Karimpour and Ruhe [2] argue the need for developing techniques that find a solution for a trade-off between alternative features for products while balancing overall value and product integrity.

The feature selection approach developed in this research work aims to rectify this problem by providing optimal (or near optimal) solutions for feature selection while considering the impact of dependency relationships, product priority and the overall integrity of the product. The objectives are selected keeping the business perspective in mind. Moreover, to best of our knowledge, the related studies have used their techniques on automatically generated feature models only; so we validate our research by applying it to real life case studies to confirm the beneficial aspect of our approach, as a proof-to-concept.

2.5 Grouping Genetic Algorithm (GGA)

Ensuring that the SPL configurations fulfill the business objectives while considering the feature dependencies makes the problem more complex. As there are multiple objectives to be optimized, this yields for some tradeoffs still ensuring best possible results. This makes the problem an optimization problem.

A popular and promising approach to solve such optimization problem is the use of genetic algorithm and it has been used by many researchers as used by Karimpour and Guo, [2, 11].

The techniques and approaches that use GA, mentioned in Section 2.3, use the normal binary chromosomal representations to solve the feature selection problem. So in order to

avoid the invalid solutions to be created they have to assign huge or moderate penalties on individuals; which causes the GA to converge before finding some of the solutions.

If they incorporate a high penalty during evaluation and the domain is one in which production of an individual violating the constraint is likely, the genetic algorithm might spend most of its time evaluating illegal individuals. Further, it can happen that when a legal individual is found, it drives the others out and the population converges on it without finding better individuals, since the likely paths to other legal individuals require the production of illegal individuals as intermediate chromosomes/structures (two illegal parents might produce best of the children), and the penalties for violating the constraint make it unlikely that such intermediate chromosomes/structures will reproduce. If one imposes moderate penalties, the system may evolve individuals that violate the constraint but are rated better than those that do not because the rest of the evaluation function can be satisfied better by accepting the moderate constraint penalty than by avoiding it. If one builds a "decoder" into the evaluation procedure that intelligently avoids building an illegal individual from the chromosome, the result is frequently computation-intensive to run. Further, not all constraints can be easily implemented in this way.

As seen, normal binary encodings in a GA are unnatural for many problems as they don't fully accommodate the problem specific information [30]. Hence, our GA's encoding scheme is inspired by the encoding scheme suggested by Falkenaeur [31] and Michalewicz [32] , we have modified the scheme just to ease the coding of the genetic algorithm. No changes have been made to the gist of the encoding scheme.

Falkenauer [31] proposed so-called Grouping Genetic Algorithm (GGA) to deal with a variety of grouping (partitioning) problems; his efforts aimed at designing appropriate chromosomal representation to capture the structure of the problem. Many researchers [33-38] have applied his chromosomal representations or a modified version of his chromosomal representations to represent and then solve the different type of partitioning problems like the bin-packing, bin balancing or graph-coloring problems.

2.6 GGA Approaches and applications

This section highlights some of the work that has been done using the grouping genetic algorithm. The section also discusses some of the applications of the GGA.

Quiroz et al [33] presents a new grouping genetic algorithm called GGA-CGT to solve the bin packing problem. The algorithm includes heuristic strategies that promote the transmission of the best genes of the chromosomes and that allow for exploration of the search space. In this research GGA-CGT controls the selection of individuals, to create a balance between the selective pressure and population diversity, avoiding the premature convergence of the algorithm and obtaining better solutions in a small number of generations.

In another research[36], proposes a grouping genetic algorithm for clustering along with the following stages: application of various numbers of clusters in a data set in order to find the suitable number of clusters, optimization of the algorithm by means of effective crossover and mutation operators, quality enhancement by implementation of the local search method.

Zulawinski [39] shows an application of a modified version of the GGA, which proposed by [31]. This paper shows the effectiveness of this approach on various Bin Balancing problems.

E. C. Brown and R. T. Sumichrast [35] proposes a new solution to solve the machine-part cell formation (MPCF) problem. MPCF is a problem that addresses the issues surrounding the creation of part families based on component processing requirements, and the identification of machine groups based on their ability to process specific part families. This methodology is based on a grouping genetic algorithm and employs a specialized replacement heuristic within the crossover operator.

Another application of GGA is shown by Rhydian Lewis and Ben Paechter [34]. They apply GGA to solve University Course Timetabling-Problems (UCTPs) which involves the allocation of resources (such as rooms and timeslots) to all the events of a university; satisfying a set of hard-constraints and, as much as possible, some soft constraints.

CHAPTER 3

RESEARCH METHODOLOGY AND FRAMEWORK

The evolution of SPL is particularly a challenging job. The functionalities that a SPL is composed of are in the form of features. These features naturally have dependencies among them. When evolving SPL these feature dependencies must be taken into consideration for smooth and efficient evolution. SPL development needs a feature selection process where the user requirements and the features are analyzed together to provide a portfolio of sets of related features that must be implemented together.

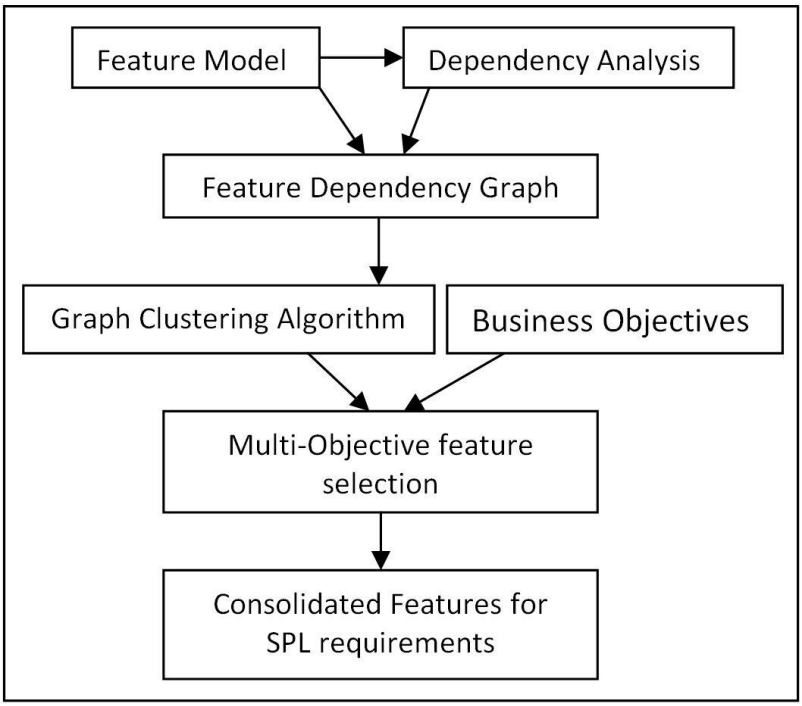

Figure 2 - Research Methodology and Framework

We believe that this consolidation of related features into sets must address three essential concerns: understanding the needs of stakeholder via SPL specifications, analyzing dependencies among features; and then identifying a group of suitable sets (containing related features) that best matches SPL specifications and user preferences. The multi-objective feature selection process shown in Figure 2 composed of following four phases: (a) SPL specification, (b) Features dependency analysis, (c) Cluster analysis and (d) Feature optimization. The first phase, SPL specification, uses a feature model to specify all the features present in the SPL. In the second phase, feature dependency analysis is carried out which examines the relationships between the features. The third phase applies the graph clustering algorithm to organize interdependent features into clusters and create consolidated features. Lastly, in the feature optimization phase, an evolutionary algorithm is used to optimize feature selection with respect to the objective functions. As a result, a set of possible solutions that fit user preferences is produced.

3.1 Design Science Research Guidelines

Our approach, a graph-based feature selection technique, adheres to the research guidelines mentioned in the *design science research framework* [40]. The framework sets forth seven design science research guidelines which guide in producing a research carrying viable contributions. The guidelines and a summary of what each guideline indicates are presented in Table 1.

Table 1 - Summary of Design Science Research Guidelines

	Design science research guideline	Guideline indications
1	"Design as an artifact"	The artifacts produced by a research must be in the form of a model or a method.
2	"Problem relevance"	The technology-based solutions, produced by the research, must be relevant to the problems.
3	"Design evaluation"	Case studies, experiments or other evaluation methods must be used to gauge and exhibit the quality of the design artifact.
4	"Research contribution"	Clear contributions in the scope of the design artifact, must be made by the research.
5	"Research rigor"	Rigorous methods must be used by the research model while evaluating and developing the design artifact.
6	"Design as search process"	For the production of an effective design artifact, desired ends must be reached via available means while adhering to the problem domain's laws.
7	"Communication of research"	The audience for the research should be both, technology and management oriented.

We discuss our approach with reference to the guidelines in Table 2 as follows:

3.1.1 Design as an artifact

The major artifact in this research is the multi-objective feature selection process that is led by features dependency analysis, consolidated features (a feature that contains several related features) and the objective functions (user preferences). The feature selection process uses the feature model to elicit SPL requirements while the feature dependency analysis is performed to analyze the relationships between SPL features. The graph

clustering is used to consolidate related features which lead to the main artifact of our research.

3.1.2 Problem Relevance

The problem we solve in our research has high relevance to software engineering field. Firstly, SPL success critically depends upon the selection of appropriate features [26, 41, 42]. Secondly, the consolidation of related features should be performed based on the feature dependencies, as discussed in Chapter 2, and the SPL requirements. The discussion in literature review section of Chapter 2 highlights the need for a technique that can select features based on the feature dependency/requirements analysis [18], and can present several solution sets of consolidated features that match SPL specifications and the user preferences. The feature selection approach developed in this research aims to fulfill the aforementioned needs, eventually contributing to solving an important problem in the field of software engineering.

3.1.3 Design Evaluation

The feature selection approach is evaluated using two real life case studies. This allows the research audience to gauge the efficiency of our approach compared to other related approaches. The case studies presented are discussed in relation to our feature selection process in detail, in chapter 4. The observations on the findings of the case studies are discussed in detail. Moreover, we have incorporated experts' qualitative feedback in our approach to confirm the viability of our approach.

3.1.4 Research Contribution

The principal research contributions of our research are the following: (a) Development of a feature dependency analysis technique, (b) Introduction to the concept of FDG (feature dependency graph), (c) Adaptation to signed graph clustering for combining related features, (d) Applying a Genetic Algorithm to optimize feature selection in SPL. These research contributions are then be evaluated as mentioned in the previous part, *"Design Evaluation"*.

3.1.5 Research Rigor

The feature selection process that is presented in our research uses a feature model to acquire the SPL specifications; it then applies feature dependency analysis to analyze the dependencies among the features while forming a feature dependency graph; followed by the graph clustering step where related feature are consolidated accordingly; leading to the final step where the multi-objective feature selection is done by using genetic algorithm. The SPL to-be needs are elicited using feature modeling [5] while the consolidation of related features is done using a local optimization signed graph clustering algorithm [43, 44] which has a long history and follows a sound mathematical model. It can therefore be seen, that our research work is drawn from a clearly defined and tested base of literature and techniques.

3.1.6 Design as a Search Process

The design of our research is based on an iterative search that can effectively balance the SPL requirements, the feature dependencies and the user preferences. Moreover, our process ensures that the laws that are commonly accepted and practiced in software

engineering are satisfied; this can be seen in the previous item that our work is heavily drawn from already accepted processes and approaches.

3.1.7 Communication of Research

The audience targeted by our research is the SPL analysts who are well aware of the feature modeling; clustering approaches, particularly graph based, and feature selection processes and algorithms. However, since the business requirements play a central role in our approach, there is strong motivation for the managerial audience to adopt our approach.

3.2 Feature Modeling

The first step is to form a suitable feature model that has sufficient amount of features and dependencies which are also close to a real world setting. These features act as the design requirements for the SPL. A similar feature model for smartphone device product line can be seen in Figure 1, earlier in chapter 2.

3.3 Feature Dependency Analysis

To proceed with the research it is important to analyze the dependencies that exist among the features and come up with a technique that allows us to convert the feature model into a graph upon which the graph clustering can be carried out.

These dependencies are induced in the feature model when it is being created. The dependencies reflect upon the relationships features have among themselves; based on the structural and relational constraints.

The dependencies must be fully analyzed and translated into a structure that can easily depict the relations among the various features of a product line. The semantic

dependency relationship among the concrete level features (CLFs), leaves of the feature model, can be seen in Figure 3.

The child features that were connected to their parent via AND connector were given the edge weight ‘1’ between them as these features must exist together. Similarly, those connected via ALT or NOT connectors were assigned the edge weight value of ‘-1’ among them, as only one of the child can be selected for a certain product or both can’t/must not exist in the same product. Likewise, edge weight between the features that were connected via OR connector were assigned the value ‘0.5’.

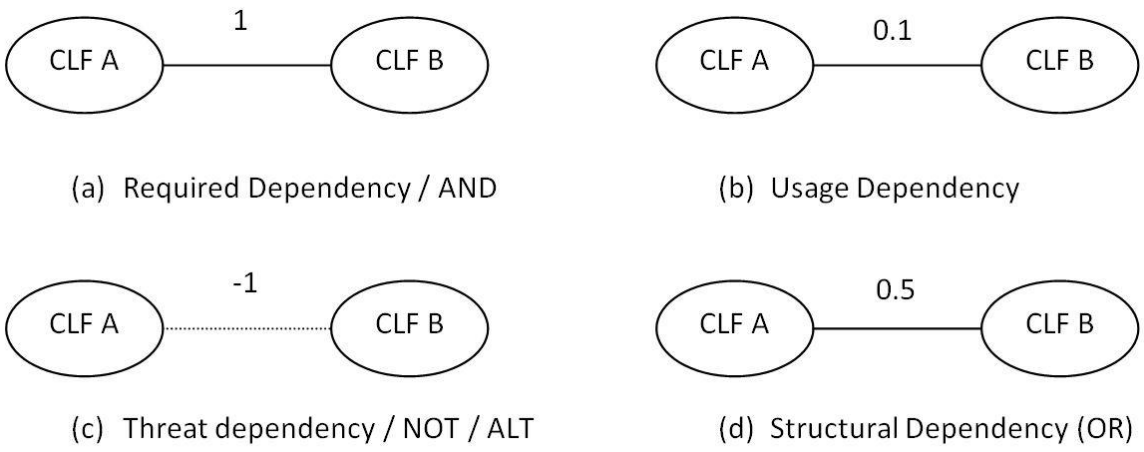

Figure 3 - Semantic Dependency Relationships among the CLFs

The weights proposed for the required, usage, structural OR and threat dependencies are derived from the literature surveyed to compile this study. A similar method to the one used by Khan in [18]. The inherent limitation of using weights lies in the subjective nature of their values. However, the sensitivity analysis presented previously indicates that our approach is robust and is able to produces stable solutions even if the weight values are varied.

3.4 Feature Dependency Graph (FDG)

We construct an undirected signed graph G(N, E), called the feature dependency graph, to model the semantic dependencies between CLFs. The set of nodes N of the feature dependency graph consists of all CLFs of the SPL, with positive or negative edges connecting pairs of CLFs with required, usage or threat dependencies.

Every edge E is assigned a positive or negative weight between -1 and 1 to specify the nature and the strength of the interdependence between its end nodes. The signed graph clustering algorithm subsequently used in our approach tries to merge nodes (CLFs) with positive edges in the same cluster (group), preferring edges with higher positive weights, while separating nodes with negative edges, preferring edges with more negative weights. Therefore, in order to differentiate between the three dependences, we have to choose three weight values w1, w2 and w3 such that w1, w2 > 0 (as the usage and required are positive in nature) while w3 < 0 (as the threat dependency is a negative relationship). We assign weights of 1, 0.1 and -1 to the required, usage and threat dependencies, respectively. The choice of the weights is based on the case studies experiences mentioned in [18].

We also introduced another dependency for the structural OR and assigned a weight, w4, of 0.5 to the edge among the nodes that have a structural OR among themselves. The notion behind the value of w4 is that, if there exist two nodes with structural OR among themselves, they have an equal probability of being selected for a certain cluster; either a node (CLF) will be selected or either it won’t be selected.

Based on the dependencies discussed in Chapter 2, a feature dependency graph can be made. A sample FDG can be seen in Figure 4. This FDG contains 10 nodes, which represent 10 concrete level features. Features that don't have any relation with other feature can exist too; like *'CLF 3'* shown in Figure 4. These features can be placed in any cluster. Such features are one of the causes why we receive several different solutions having the same clustering error.

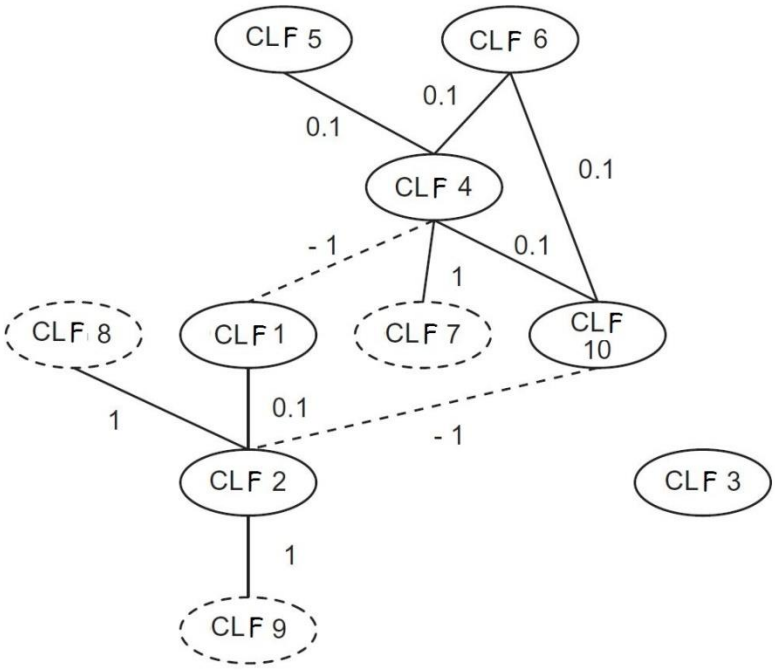

Figure 4 - A sample Feature dependency graph

3.5 Graph Clustering

Once the feature model is converted into the graph, we run graph clustering algorithm, to come up with the several solutions including some near-optimal solutions each of which contains clusters of features (consolidated features) that are interrelated and can efficiently be developed together when developing a software product line.

A custom graph clustering algorithm was developed to cluster elements into groups. Our graph clustering algorithm works like a relocation algorithm that is used to partition signed graphs. Signed graphs are the graphs that have positive or negative weights over the edges that connect the vertices. As seen in Figure 4, the edges in the graph contain both negative and positive weights so; our algorithm seems to be the appropriate graph clustering algorithm to apply. This algorithm optimizes a certain partition by including as many positive edges as possible with the cluster and negative edges between the clusters.

3.5.1 Cluster Formation

In this step, the nodes of the feature dependency graph are clustered based on their interdependencies. We use the local optimization signed graph clustering algorithm [43] that partitions nodes of a signed graph in such a way that pairs of nodes joined by positive edges are grouped in the same cluster, whereas pairs of nodes joined by negative edges are separated into different clusters. Our objective is to cluster the concrete-level features (CLFs) with required and usage relationships together while separating concrete-level features with threat dependencies. The cluster formation helps to combine those CLFs in a cluster that work together to achieve a functionality of the SPL. On the other hand, any two concrete-level features with a threat dependency will be separated into different clusters.

However, it is important to note that it is not always possible to cluster a signed graph. In fact, a signed graph is clusterable if and only if it contains no cycle with exactly one negative edge [45]. If the feature dependency graph is not clusterable, the local optimization algorithm finds a partition that minimizes the clustering error. The negative error *neg* of a partition is the sum of weights of all negative edges that lie inside clusters.

While the positive error *pos* can be defined as the sum of weights of positive edges joining different clusters. The clustering error, Er is defined [43] as:

$$\mathrm{Er} = \mathrm{pos} + |\mathrm{neg}| \qquad (1)$$

where |.| stands for the absolute value. Given Eq. (1) we can outline the local optimization clustering algorithm as shown in Figure 5:

(1) Start with a random initial clustering *Cls*.
(2) For $i{:=}1 \ldots, n$; repeat steps 3 and 4.
(3) Obtain a new clustering Cls' by exchanging nodes between two clusters.
(4) If $Er(Cls) > Er(Cls')$ then select Cls' as the local optimal.

Figure 5 - Cluster formation procedure for graph clustering

Here Er(Cls) denotes the error of the clustering Cls and n denotes the number of iterations performed by the algorithm before stopping. The output of the Procedure is a local optimal clustering that is not necessarily a global optimal. However, for large n (typically 1000), this local optimal provides a good approximation to the global optimal [43]. Procedure 1 is the most widely used signed graph clustering algorithm as the problem of finding a global optimal clustering is NP-hard [46].

3.6 Business Objectives

Once the set of solutions or configurations to SPL are provided, there can be several different near optimal solutions based on the objectives that must be reached out of the set of SPL solutions. The objective functions can be created based on the structural needs of the solutions or they can be elicited from the stakeholders, like a business analyst, who

wants a certain trait to be depicted at all times from their products within a product line or as in our case, software product line.

For our approach, overall there are three objectives which are considered while defining near-optimal solutions to a software product line. One was the clustering error that resulted after applying our graph clustering algorithm over the FDG; the second was the overall product priority and last the product integrity. Each of the objectives is defined in the following sub-sections.

3.6.1 Clustering Error

The clustering error is generated by the graph clustering algorithm, which depicts how much of a tradeoff between the features' dependencies have been made in order to generate that particular error.

$$Er = pos + |neg| \qquad (1)$$

Where, *neg* of a partition is the sum of weights of all negative edges that lie inside clusters and *pos* can be defined as the sum of weights of positive edges joining different clusters.

3.6.2 Overall product priority

One of the major requirements of businesses or software system planner is to produce products that are not only built in a certain fashion but they also amount to a required value it provides to the company. As a single product contains several features, we start off by assigning a priority value based on the importance of each concrete level feature (CLF). A common approach to find the value or priority of a feature is to ask the stakeholders to vote for features. Hence, the assignment was made based on a small

survey where we asked the respondents, experts and developers, to rate the features based on the importance of the feature in the product. We then took the average value of the respondents answer and assigned that value as the priority for a particular feature.

As we have set of clustered/consolidated features in a single product configuration/solution, the priority of a cluster may depend on the selection state of other features such that selection of a feature can increase or decrease the value/priority of the whole cluster. Therefore, we use average of averages method to calculate the overall product priority or value. So, the priority of each feature in a group is added and divided by the number of features in a group/cluster, hence the average priority within each cluster increases or decreases based on the priority of each individual. This is repeated for every group. The average of these averages is then the overall product priority of the resulting product.

$$PP = (\sum_{i=0}^{m} (\sum_{j=0}^{n} CPj)/n)/m \quad (2)$$

Where PP is the overall product priority, m is the number of clusters in the product, n is the number of CLFs in that particular cluster and CP_j is the priority of a single CLF, 'j'.

3.6.3 Overall product integrity

As talked about earlier, one of the main goals for businesses to use SPL as a reuse technique is to achieve a range of products in efficient and effective manner that also comply to the business requirements which SPL tends to fulfill. One such goal that is required by businesses or stakeholders is the overall integrity of the resulting product [2, 47].

According to C. Takahiro [47], the product integrity can be defined as “the degree to which the features of a product are perceived as cohesive”. From end-user perspective the higher the product’s integrity is, the higher the synergy among the features of the product, rather than a product just having a collection of features each doing a solitary task.

To formulate integrity, we use NCP (Nearest common Predecessor). NCP measures the semantic distance among tow CLFs. To calculate NCP for two CLFs, we first have to label the levels, root being the zero level, while the parent of the deepest leaf will have the highest level, as the number is incremented at each tree level. NCP for two CLFs is their first common abstract reached when moving from leafs towards the root. That level number will be the NCP of those two features. Figure 6 shows how the integrity between two features is measured by using NCP.

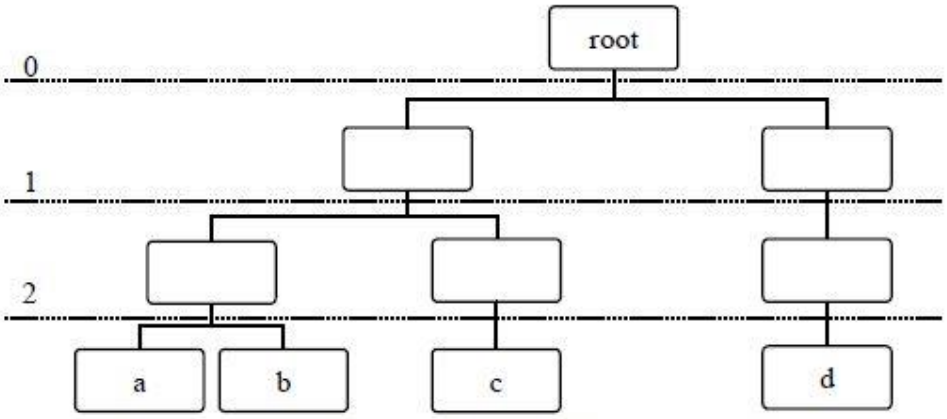

Figure 6 - Sample Calculations for NCP. Ex: NCP(a,b) = 2, NCP(c,d) = 0

3.7 Multi-objective feature selection using Genetic Algorithm

Ensuring that the SPL configurations fulfill the business objectives while considering the feature dependencies makes the problem more complex. As there are multiple objectives

to be optimized, this yields for some tradeoffs still ensuring best possible results. This makes the problem an optimization problem.

A popular and promising approach to solve such optimization problem is the use of genetic algorithm and it has been used by many researchers as in [2, 11].

3.7.1 Grouping Genetic Algorithm (GGA)

Falkenauer [31] proposed so-called Grouping Genetic Algorithm (GGA) to deal with a variety of grouping (partitioning) problems; his efforts aimed at designing appropriate chromosomal representation to capture the structure of the problem. Many researchers [33-38] have applied his chromosomal representations or a modified version of his chromosomal representations to represent and then solve the different type of partitioning problems like the bin-packing, bin balancing or graph coloring problems.

3.7.2 Encoding Scheme

Normal binary encodings are unnatural for many problems as they don't fully accommodate the problem specific information[30]. Hence, our encoding scheme is inspired by the encoding scheme suggested by Falkenaeur [31] and Michalewicz [32] , we have modified the scheme just to ease the coding of the genetic algorithm. No change has been made to the gist of the encoding scheme. This encoding scheme is used to enhance the performance of the genetic algorithm by using problem specific genetic operators.

The crossover and mutation genetic operators remain the same and work in the same fashion as suggested in [31, 32]. This encoding scheme is specially designed to solve the grouping problems, like bin-packing and bin-balancing problems. It scheme had to be

modified because in the bin-balancing and the bin-packing problems either the bin/group size is fixed or the number of objects/features inside a bin/group are fixed. In our approach both the amount of groups and the number of features within a group are kept variable.

The following scheme has been used to represent the chromosome:

(11 22 31 43 53 63 71 81 92 : 1 2 3)

- The part of the chromosome to the left of the colon is the feature part, while the other is called the group part.
- The 1st digit of each number in the feature part is the feature number or ID. The second digit is the group they are associated with.
- The group part represents the total clusters that all the features in this solution reside in.
- So, in the above example, there are 9 features associated with 3 different groups.

The encoding scheme to encode a SPL configuration into a chromosome is depicted in Figure 6.

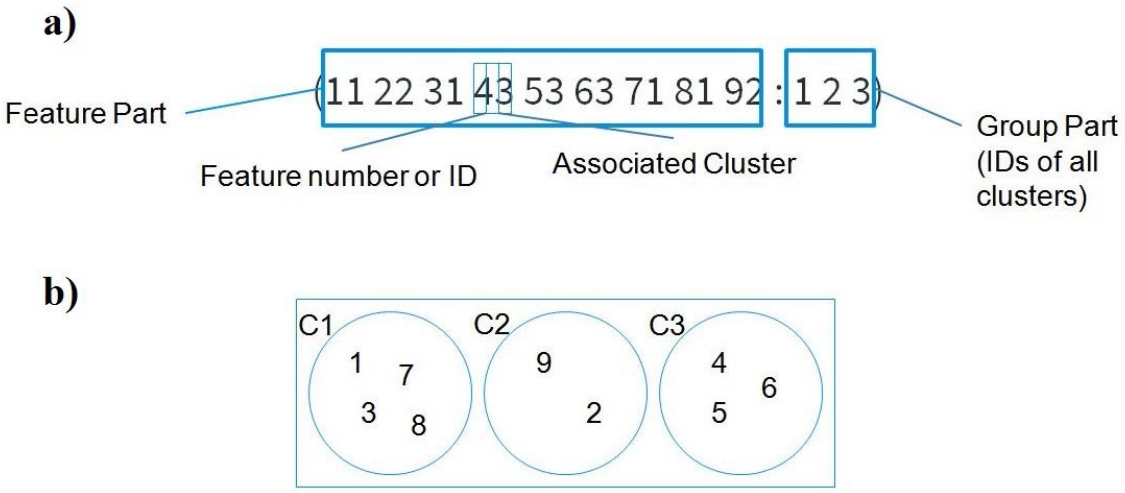

Figure 7 - a) Description of Chromosome b) Solution representation of the chromosome

3.7.3 Selection

Once the population is generated, the next step is to select some individuals from the given population, upon which the crossover and mutation genetic operators can be applied.

The individuals will be selected based on a fitness value which is calculated as follows:

$$\text{Fitness} = 1/PP + 1/PI + Er \quad (3)$$

Where, PP is the overall product priority, PI is the overall product integrity and Er is the clustering error for that product/SPL configuration. The lower the value of fitness the fittest an individual is.

To select the individuals we have used the tournament selection approach [48]. It is a method to select an individual from a given population of several individuals. Several

tournaments are carried out among some individuals (chosen at random) in a given population. The one with the best fitness among the two is then selected for crossover.

The tournament selection method has the following steps:

- Select X individuals from a given population. (X = tournament size)
- Select the individual with best fitness value, having some probability p.
- Then select the next best-fit individual, with probability p*(1-p)
- Then select the 2nd next best fit individual with probability p*((1-p)^2)
- Do this K times and then perform crossover upon each pair of individuals to generate the next population.

3.7.4 Crossover

One of the genetic operators in a GA is a crossover. When crossover between two individual of a given population occurs, a new individual is generated. This individual inherits the traits of its parent. When performing crossover, the hope is that by combining two individuals an even 'fitter' offspring will be created, while inheriting the traits of its parents.

The words chromosome and individual will be used interchangeably. The crossover is explained using the following example:

Individual 1: (11 22 31 43 53 63 71 81 92 : 1 2 3)

Individual 2: (12 23 33 45 51 64 72 82 96 : 1 2 3 4 5 6)

Two crossing sites are then selected in each of the individuals

Individual 1: (11 22 31 43 53 63 71 81 92 : 1 | 2 3|)

Individual 2: (12 23 33 45 51 64 72 82 96 : 1 2 |3 4| 5 6)

The idea is to inject the contents bounded between the two crossings sites of the first parent are inserted at the first crossing site of the second parent and vice versa. This creates two children, who are then mutated or sent directly to the new population, depending on the mutation probability.

This form of crossover results in duplicate elements being grouped into the different clusters. To solve this problem the old groups of the child with duplicate elements are deleted if the new group also has those same elements. This, in turn, leaves some elements not being assigned to any of the group. To cater this problem a repair function is developed that randomly assigns the abandoned elements to any group or creates a new group and assigns the abandoned features to it.

The process of crossover can be seen in Figure 8. The two chromosomes are denoted with different colors in order to identify the groups and features of each individual chromosome.

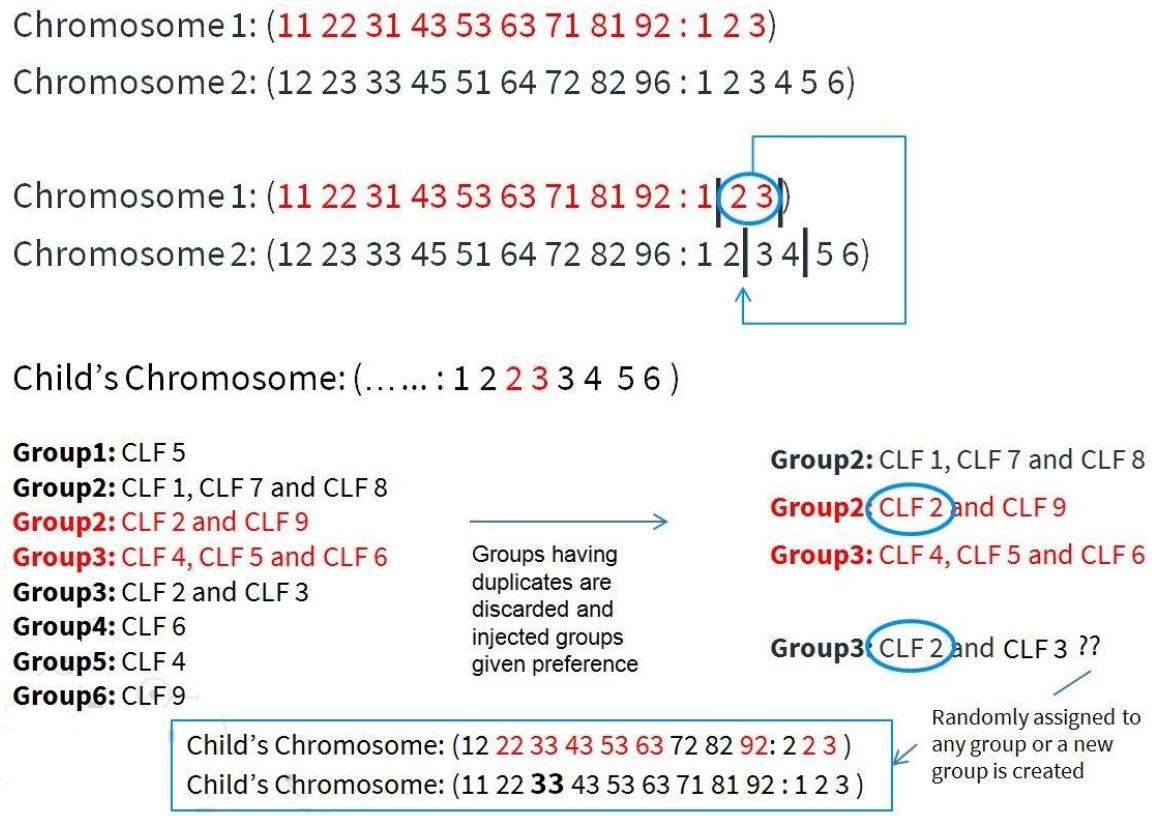

Figure 8 - GA Crossover process

3.7.5 Mutation

Once the crossover produces a child, mutation is performed based on the mutation probability. Mutation is performed to bring a little bit of randomness in the created individual, to ensure that this particular individual is not among the initial population.

The small change we perform is swapping of a feature from one cluster to another. This process of mutation of a chromosome can be seen in Figure 9.

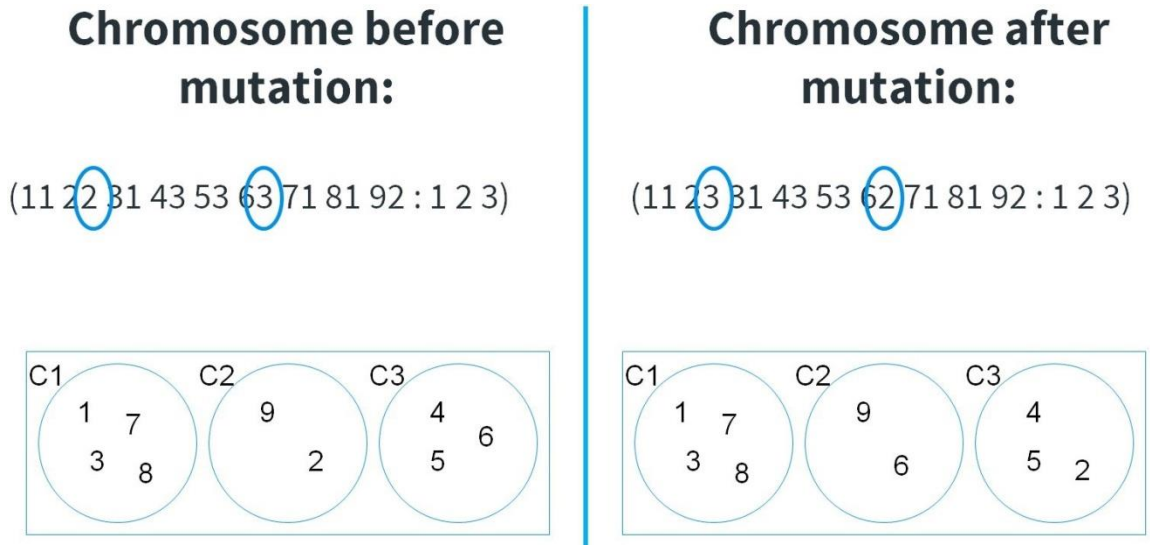

Figure 9 - Mutating a chromosome by swapping a feature among clusters

3.7.6 Termination of GA

There are two ways to terminate the GA either a certain number of generations (1000) are reached or a certain fitness value individual is created and no further improvement to the fitness value occur, for a certain amount of generations.

3.8 Consolidated features for SPL requirements

Once the GA terminates, The GA produces a single best solution having the best fitness value among the last generation that was generated. This solution is the near optimal solution that has evolved from a random initial population. The solution suggests that elements in each cluster must be developed together in order to benefit the most in terms of the business objectives.

CHAPTER 4

CASE STUDIES AND RESULTS

This chapter comprises of two case studies where we apply our approach and discuss the outcomes of each case study separately. The first case study is a real life example of an automotive system (AS), which is used to easily understand the concepts discussed in our approach. The second case study is based on a well known tool used by the architects and structural engineers, life cycle assessment tool (LCA). These case studies show the applicability of our approach in different environments, which will help the system analysts and SPL managers to easily adopt to this approach.

4.1 Case Study 1 – Automotive System (AS)

This case study includes some commonly utilized features of an automotive system. As automobiles are widespread and knowledge about them is a very common, it will make our approach easy to relate and understand.

4.1.1 AS - Feature Modeling

For the purpose of the case study a suitable feature model for an Automotive System was formulated based on the SPL requirements of an automotive system, which contained 34 features and 8 cross-tree constraints. These constraints were the basis upon which the relationships between the features were defined. The feature model can be seen Figure 9; it depicts the core and optional features having a variety of relations among the features.

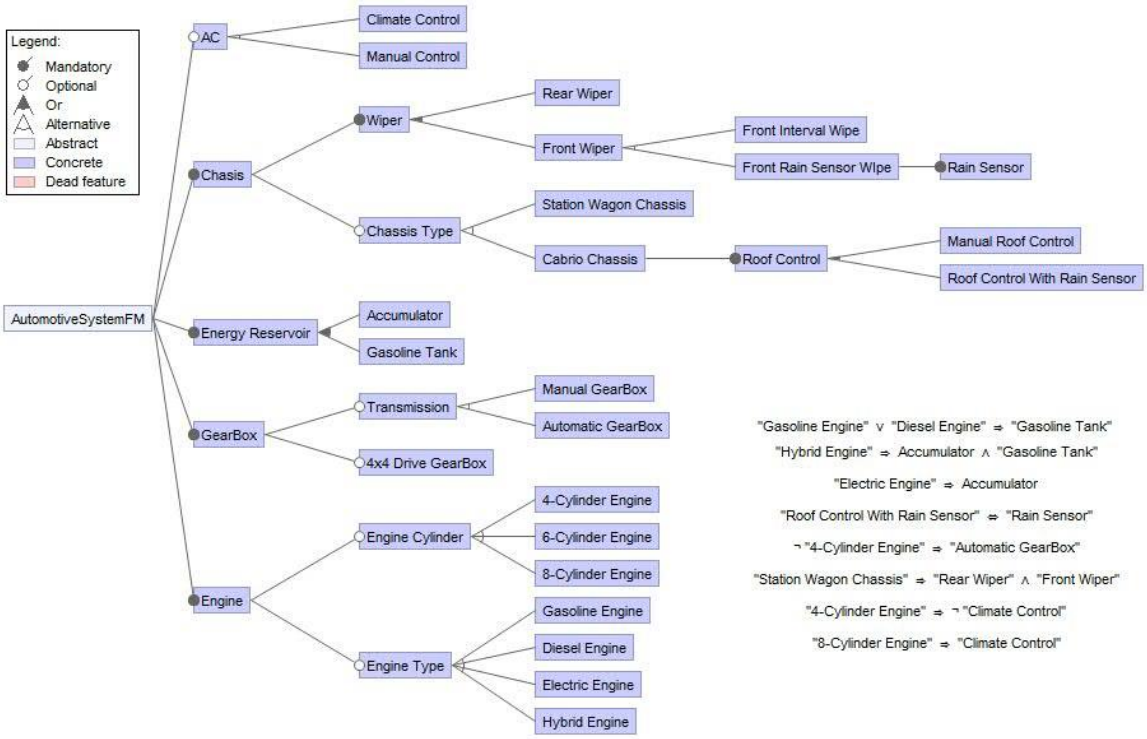

Figure 10 - Feature Model for an Automotive system

4.1.2 AS - Feature Dependency Analysis

The feature dependency analysis was performed over the feature model depicted in Figure 10. The analysis considered the feature dependencies that existed among the concrete level features, as shown in Figure 3. The dependencies that existed among the CLFs of the automotive system can be seen in Table 2.

Table 2 - Feature Dependency analysis for automotive system

Concrete Level Features		Required Dependency / AND	Threat Dependency / Not / ALT	Structural Dependency OR	Usage Dependency
CLF 1	4-Cylinder Engine		CLF 2, CLF 3, CLF 9, CLF 19		
CLF 2	6-Cylinder Engine		CLF 1, CLF 3		
CLF 3	8-Cylinder Engine	CLF 19	CLF 1, CLF 2		
CLF 4	Gasoline Engine	CLF 12	CLF 5, CLF 6, CLF 7		
CLF 5	Diesel Engine	CLF 12	CLF 4, CLF 6, CLF 7		
CLF 6	Electric Engine	CLF 11	CLF 4, CLF 5, CLF 7		
CLF 7	Hybrid Engine	CLF 11, CLF 12	CLF 4, CLF 5, CLF 6		
CLF 8	Manual GearBox	CLF 10	CLF 9		
CLF 9	Automatic GearBox	CLF 10	CLF 1, CLF 8		
CLF 10	4x4 Drive GearBox	CLF 8, CLF 9			
CLF 11	Accumulator	CLF 6, CLF 7			
CLF 12	Gasoline Tank	CLF 4, CLF 5, CLF 7			
CLF 13	Rear Wiper	CLF 16		CLF 14, CLF 15	
CLF 14	Front Internal Wipe		CLF 15	CLF 13	
CLF 15	Rain Sensor		CLF 14	CLF 13	CLF 18
CLF 16	Station Wagon Chassis	CLF 13	CLF 17, CLF 18		
CLF 17	Manual Roof Control		CLF 16	CLF 18	
CLF 18	Roof Control with Rain Sensor		CLF 16	CLF 17	CLF 15
CLF 19	Climate Control	CLF 3	CLF 1, CLF 20		
CLF 20	Manual Control		CLF 19		

4.1.3 AS - Feature Dependency Graph (FDG)

The undirected feature dependency graph was created by developing a graph clustering algorithm defined in Chapter 3 of the thesis. The undirected FDG can be seen in Figure 11. The nodes of the graph represent the CLFs in the feature model, which was created for the automotive system.

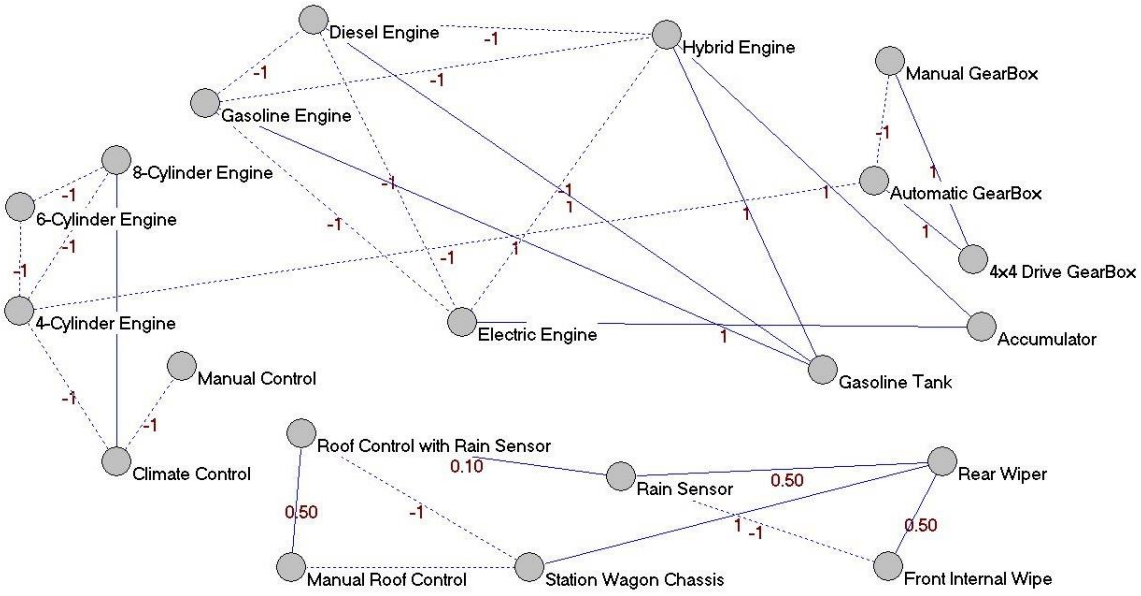

Figure 11 - FDG for the Automotive FM using the feature dependencies

4.1.4 AS - Graph Clustering

The graph clustering algorithm was run on the feature dependency graph; the following steps were followed to get the clustering results depicted in Table 3:

1. Provide the number of clusters
2. Run graph clustering algorithm 1000 times with 95% confidence level
3. Note the clustering error
4. Decrement the number of clusters
5. Repeat step 1-4 (starting from 'no. of clusters = number of CLFs' to 'no. of clusters = 1')

4.1.5 AS - Clustering Results

The clusters are formed based on how related the features are (i.e. feature dependencies). Clustering error closer to zero suggests that during clusters formation better compromises between the placement of features (based on the dependencies) into clusters, were made than the results produced with higher clustering error.

Table 3 - Results after applying graph clustering on the FDG

Number of Clusters	Clusters	Clustering Error
20	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5), (CLF 6), (CLF 7), (CLF 8), (CLF 9), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 18), (CLF 19), (CLF 20)	1.06
19	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5), (CLF 6), (CLF 7), (CLF 8), (CLF 9), (CLF 10), (CLF 11), (CLF 12), (CLF 13, CLF 16), (CLF 14), (CLF 15), (CLF 17), (CLF 18), (CLF 19), (CLF 20)	0.96
18	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5), (CLF 6), (CLF 7), (CLF 8, CLF 10), (CLF 9), (CLF 11), (CLF 12), (CLF 13, CLF 16), (CLF 14), (CLF 15), (CLF 17), (CLF 18), (CLF 19), (CLF 20)	0.86
17	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 12), (CLF 6), (CLF 7, CLF 11), (CLF 8), (CLF 9), (CLF 10), (CLF 13, CLF 16), (CLF 14), (CLF 15), (CLF 17), (CLF 18), (CLF 19), (CLF 20)	0.76
16	(CLF 1), (CLF 2), (CLF 3), (CLF 4, CLF 12), (CLF 5), (CLF 6, CLF 11), (CLF 7), (CLF 8), (CLF 9, CLF 10), (CLF 13, CLF 16), (CLF 14), (CLF 15), (CLF 17), (CLF 18), (CLF 19), (CLF 20)	0.66
15	(CLF 1), (CLF 2), (CLF 3, CLF 19), (CLF 4), (CLF 5), (CLF 6), (CLF 7, CLF 11, CLF 12), (CLF 8, CLF 10), (CLF 9), (CLF 13, CLF 16), (CLF 14), (CLF 15), (CLF 17), (CLF 18), (CLF 20)	0.56
14	(CLF 1), (CLF 2), (CLF 3, CLF 19), (CLF 4, CLF 12), (CLF 5), (CLF 6), (CLF 7, CLF 11), (CLF 8, CLF 10), (CLF 9), (CLF 13, CLF 16), (CLF 14), (CLF 15), (CLF 17, CLF 18), (CLF 20)	0.51
13	(CLF 1), (CLF 2), (CLF 3, CLF 19), (CLF 4), (CLF 5), (CLF 6), (CLF 7, CLF 11, CLF 12), (CLF 8, CLF 10), (CLF 9), (CLF 13, CLF 15, CLF 16), (CLF 14), (CLF 17, CLF 18), (CLF 20)	0.46
12	**(CLF 1), (CLF 2), (CLF 3, CLF 19), (CLF 4, CLF 12), (CLF 5), (CLF 6, CLF 11), (CLF 7), (CLF 8), (CLF 9, CLF 10), (CLF 13, CLF 14, CLF 16), (CLF 15, CLF 17, CLF 18), (CLF 20)**	**0.45**
11	(CLF 1), (CLF 2), (CLF 3, CLF 19), (CLF 4, CLF 12), (CLF 5), (CLF 6, CLF 11), (CLF 7, CLF 20), (CLF 8, CLF 10), (CLF 9), (CLF 13, CLF 14, CLF 16), (CLF 15, CLF 17, CLF 18)	0.45
10	(CLF 1, CLF 6), (CLF 2), (CLF 3, CLF 19), (CLF 4, CLF 13, CLF 14, CLF 16), (CLF 5, CLF 12), (CLF 7, CLF 11), (CLF 8, CLF 10), (CLF 9), (CLF 15, CLF 17, CLF 18), (CLF 20)	0.45
9	(CLF 1, CLF 5, CLF 12), (CLF 2), (CLF 3, CLF 19), (CLF 4), (CLF 6, CLF 8, CLF 11), (CLF 7), (CLF 9, CLF 10, CLF 20), (CLF 13, CLF 14, CLF 16), (CLF 15, CLF 17, CLF 18)	0.45
8	(CLF 1, CLF 8, CLF 15, CLF 17, CLF 18), (CLF 2), (CLF 3, CLF 7, CLF 19), (CLF 4, CLF 12), (CLF 5, CLF 20), (CLF 6, CLF 11), (CLF 9, CLF 10), (CLF 13, CLF 14, CLF 16)	0.45
7	(CLF 1, CLF 5), (CLF 2, CLF 13, CLF 14, CLF 16), (CLF 3, CLF 19), (CLF 4, CLF 9, CLF 12, CLF 15, CLF 17, CLF 18), (CLF 6, CLF 8, CLF 10, CLF 11), (CLF 7), (CLF 20)	0.45
6	(CLF 1, CLF 15, CLF 17, CLF 18, CLF 20), (CLF 2, CLF 6, CLF 13, CLF 14, CLF 16), (CLF 3, CLF 5, CLF 19), (CLF 4, CLF 8), (CLF 7, CLF 11, CLF 12), (CLF 9, CLF 10)	0.45
5	(CLF 1, CLF 7, CLF 8, CLF 10, CLF 12, CLF 13, CLF 14, CLF 16), (CLF 2, CLF 6, CLF 11), (CLF 3, CLF 15, CLF 17, CLF 18, CLF 19), (CLF 4, CLF 9), (CLF 5, CLF 20)	0.45
4	(CLF 1, CLF 7, CLF 15, CLF 17, CLF 18), (CLF 2, CLF 5, CLF 12, CLF 20), (CLF 3, CLF 4, CLF 8, CLF 10, CLF 13, CLF 14, CLF 16, CLF 19), (CLF 6, CLF 9, CLF 11)	0.45
3	(CLF 1, CLF 5, CLF 7, CLF 8, CLF 12), (CLF 2, CLF 6, CLF 9, CLF 10, CLF 11, CLF 15, CLF 17, CLF 18, CLF 20), (CLF 3, CLF 4, CLF 13, CLF 14, CLF 16, CLF 19)	2.25
2	(CLF 1, CLF 2, CLF 4, CLF 5, CLF 8, CLF 10, CLF 12, CLF 15, CLF 17, CLF 18, CLF 20), (CLF 3, CLF 6, CLF 7, CLF 9, CLF 11, CLF 13, CLF 14, CLF 16, CLF 19)	5.95
1	(CLF 1, CLF 2, CLF 3, CLF 4, CLF 5, CLF 6, CLF 7, CLF 8, CLF 9, CLF 10, CLF 11, CLF 12, CLF 13, CLF 14, CLF 15, CLF 16 , CLF 17 , CLF 18, CLF 19, CLF 20)	30.4

Hence, the optimum clustering error is 0.45 in our experiments. The reason it remained constant for several different numbers of clusters is that there might always be a compromise made between placements of features into clusters that if the error decreases by removal of a feature from a cluster, it tends to increase due to the addition of that feature into another cluster.

In case, if the error is same for several different solutions, the near optimal solution is the one that gives minimal error we select the solution with a the most number of clusters, as it gives more choices to distribute the product among the development teams. Therefore, the results in Table 3 suggest having the most number of clusters with minimum clustering error, 12 clusters (consolidated features) of related features.

Therefore, the algorithm clustered the FDG into 12 clusters where the cluster number can be seen beside each CLF, within the parenthesis in the graph in Figure 12.

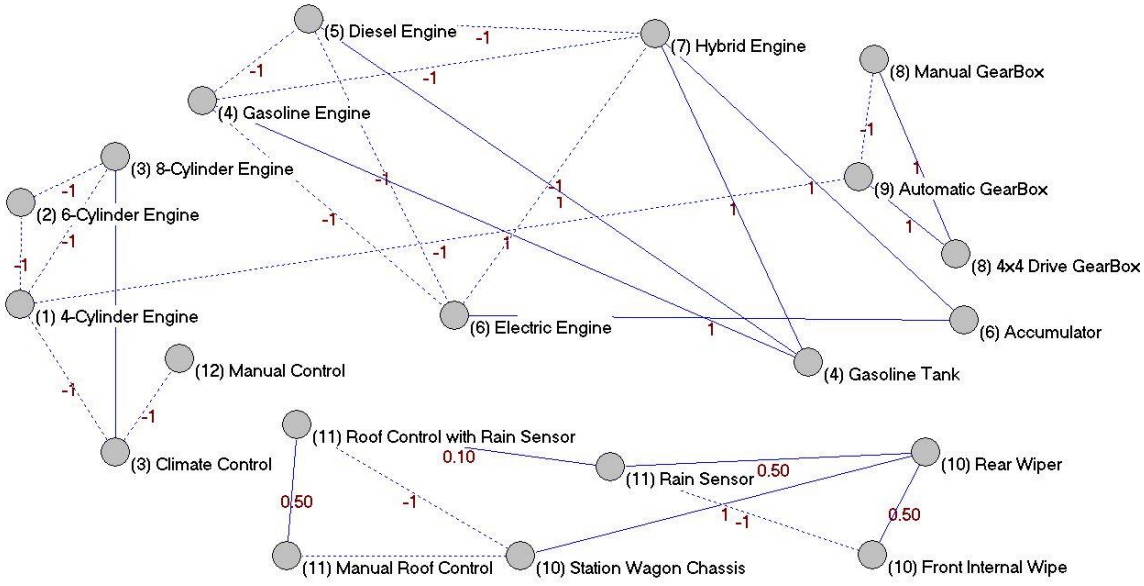

Figure 12 - A clustered FDG after running the Graph Clustering algorithm

4.1.6 Multi-objective feature selection using Genetic Algorithm

GA algorithm was used to generate a near-optimal solution. The GA started off by initializing a population of 50 random solutions. All the solutions were evaluated based on the fitness value and then the 20 best among the population were selected to be the part of a new population. These 20 were then selected for crossover, using tournament selection. Based on the *crossover probability, 0.5*, the crossover was performed between each pair of solutions. The children produced by these crossovers were then mutated in accordance with the *mutation probability, 0.1*. These children were then added to the new created population. The steps were then repeated until a *certain fitness value* was achieved. The one with the best fitness value was suggested as the near-optimal solution.

To calculate the fitness value equation (3) is used. To compute the overall fitness value we need the computations of the PP, PI and Err or the clustering error is generated when we apply our graph clustering algorithm, PP is calculated by using equation (2) while PI is calculated using a technique called Nearest Common Predecessor, NCP. Calculation of PP and PI are discussed individually and then the resulting configuration is shown.

4.1.7 AS – Calculating Product Priority, PP

As mentioned in Section 3.6.2, the assignment of priority was made based on a small survey where we asked the respondents, experts and developers, to rate the features based on the importance of the feature in the product. We then took the average value of the respondents' answers and assigned that value as the priority for a particular feature. Next, we inverted the priority for each feature as the higher the priority numbering the lower the importance and vice versa (i.e. the feature having priority 1 is of most importance).

This value was assigned to each individual feature as its priority and then used in the GA's objective function for calculating the product priority as discussed in Section 3.6.2..

The priorities for each concrete level feature in the 'Automotive System (AS)' feature model can be seen in the Table 4.

Table 4 - Priorities of CLFs in AS feature model

Concrete Level Features		Priority	Priority Inverted
CLF 1	4-Cylinder Engine	1	1
CLF 2	6-Cylinder Engine	1	1
CLF 3	8-Cylinder Engine	1	1
CLF 4	Gasoline Engine	1	1
CLF 5	Diesel Engine	1	1
CLF 6	Electric Engine	1	1
CLF 7	Hybrid Engine	1	1
CLF 8	Manual GearBox	1	1
CLF 9	Automatic GearBox	1	1
CLF 10	4x4 Drive GearBox	4	0.25
CLF 11	Accumulator	1	1
CLF 12	Gasoline Tank	1	1
CLF 13	Rear Wiper	4	0.25
CLF 14	Front Internal Wipe	3	0.333333333
CLF 15	Rain Sensor	5	0.2
CLF 16	Station Wagon Chassis	4	0.25
CLF 17	Manual Roof Control	5	0.2
CLF 18	Roof Control with Rain Sensor	5	0.2
CLF 19	Climate Control	2	0.5
CLF 20	Manual Control	2	0.5

4.1.8 AS – Calculating Product Integrity, PI

As mentioned in the previous chapter we used Nearest Common Predecessor (NCP) to calculate the integrity of the product. The NCP was calculated for every concrete level feature with all other concrete level features. To ease up the calculations of NCP we converted the Automotive System's feature model to a similar sample that was seen in

the previous chapter in section 3.6.3 (i.e. Figure 6). The modified feature model is shown in Figure 13.

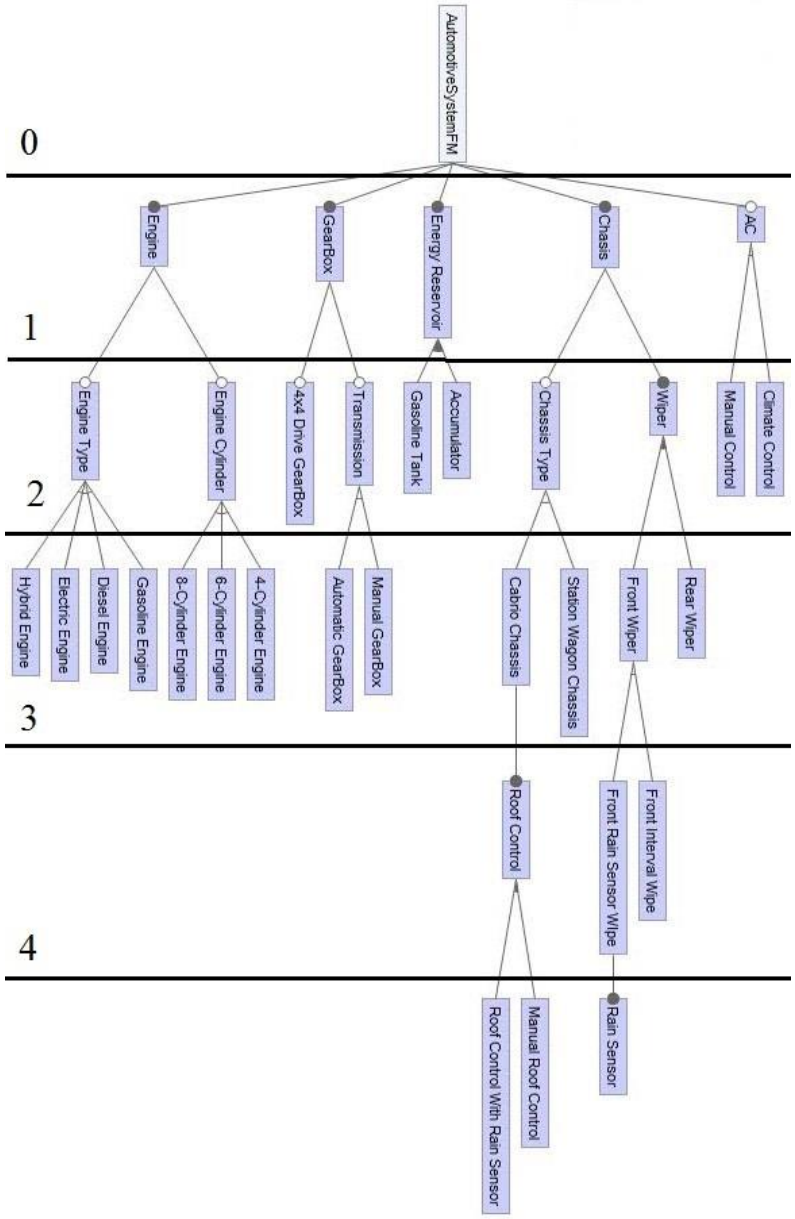

Figure 13 - Modified feature model to calculate NCP for AS

Based on the Figure 13, the NCP calculations were carried out on AS feature model. The NCP value was calculated among each pair of concrete level features and then in the

objective function for product integrity in the GA, the product integrity for each solution was calculated. The pair-wise NCP calculations can be seen in table 5.

Table 5 - NCP calculations for all CLFs in AS feature model

		4-Cylinder Engine	6-Cylinder Engine	8-Cylinder Engine	Gasoline Engine	Diesel Engine	Electric Engine	Hybrid Engine	Manual GearBox	Automatic GearBox	4x4 Drive GearBox	Accumulator	Gasoline Tank	Rear Wiper	Front Internal Wipe	Rain Sensor	Station Wagon Chassis	Manual Roof Control	Roof Control with Rain Sen	Climate Control	Manual Control
		CLF 1	CLF 2	CLF 3	CLF 4	CLF 5	CLF 6	CLF 7	CLF 8	CLF 9	CLF 10	CLF 11	CLF 12	CLF 13	CLF 14	CLF 15	CLF 16	CLF 17	CLF 18	CLF 19	CLF 20
4-Cylinder Engine	CLF 1	**0**																			
6-Cylinder Engine	CLF 2	2	**0**																		
8-Cylinder Engine	CLF 3	2	2	**0**																	
Gasoline Engine	CLF 4	1	1	1	**0**																
Diesel Engine	CLF 5	1	1	1	2	**0**															
Electric Engine	CLF 6	1	1	1	2	2	**0**														
Hybrid Engine	CLF 7	1	1	1	2	2	2	**0**													
Manual GearBox	CLF 8	0	0	0	0	0	0	0	**0**												
Automatic GearBox	CLF 9	0	0	0	0	0	0	0	2	**0**											
4x4 Drive GearBox	CLF 10	0	0	0	0	0	0	0	1	1	**0**										
Accumulator	CLF 11	0	0	0	0	0	0	0	0	0	0	**0**									
Gasoline Tank	CLF 12	0	0	0	0	0	0	0	0	0	0	1	**0**								
Rear Wiper	CLF 13	0	0	0	0	0	0	0	0	0	0	0	0	**0**							
Front Internal Wipe	CLF 14	0	0	0	0	0	0	0	0	0	0	0	0	2	**0**						
Rain Sensor	CLF 15	0	0	0	0	0	0	0	0	0	0	0	0	2	3	**0**					
Station Wagon Chass	CLF 16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**				
Manual Roof Control	CLF 17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	**0**			
Roof Control with Ra	CLF 18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	4	**0**		
Climate Control	CLF 19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	
Manual Control	CLF 20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**0**

The results for the clustering errors, corresponding product priority and integrity values can be seen in Table 6.

Table 6 - Clustering Error, Priority and Integrity values for each resulting SPL solutions of AS

Number of Clusters	Clusters	Clustering Error	Priority	Integrity
20	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5), (CLF 6), (CLF 7), (CLF 8), (CLF 9), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 18), (CLF 19), (CLF 20)	1.06	0.684	0
19	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5), (CLF 6), (CLF 7), (CLF 8), (CLF 9), (CLF 10), (CLF 11), (CLF 12), (CLF 13, CLF 16), (CLF 14), (CLF 15), (CLF 17), (CLF 18), (CLF 19), (CLF 20)	0.96	0.707	0
18	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5), (CLF 6), (CLF 7), (CLF 8, CLF 10), (CLF 9), (CLF 11), (CLF 12), (CLF 13, CLF 16), (CLF 14), (CLF 15), (CLF 17), (CLF 18), (CLF 19), (CLF 20)	0.86	0.711	1
17	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 12), (CLF 6), (CLF 7, CLF 11), (CLF 8), (CLF 9), (CLF 10), (CLF 13, CLF 16), (CLF 14), (CLF 15), (CLF 17), (CLF 18), (CLF 19), (CLF 20)	0.76	0.79	0
16	(CLF 1), (CLF 2), (CLF 3), (CLF 4, CLF 12), (CLF 5), (CLF 6, CLF 11), (CLF 7), (CLF 8), (CLF 9, CLF 10), (CLF 13, CLF 16), (CLF 14), (CLF 15), (CLF 17), (CLF 18), (CLF 19), (CLF 20)	0.66	0.8	1
15	(CLF 1), (CLF 2), (CLF 3, CLF 19), (CLF 4), (CLF 5), (CLF 6), (CLF 7, CLF 11, CLF 12), (CLF 8, CLF 10), (CLF 9), (CLF 13, CLF 16), (CLF 14), (CLF 15), (CLF 17), (CLF 18), (CLF 20)	0.56	0.845	2
14	(CLF 1), (CLF 2), (CLF 3, CLF 19), (CLF 4, CLF 12), (CLF 5), (CLF 6), (CLF 7, CLF 11), (CLF 8, CLF 10), (CLF 9), (CLF 13, CLF 16), (CLF 14), (CLF 15), (CLF 17, CLF 18), (CLF 20)	0.51	0.704	5
13	(CLF 1), (CLF 2), (CLF 3, CLF 19), (CLF 4), (CLF 5), (CLF 6), (CLF 7, CLF 11, CLF 12), (CLF 8, CLF 10), (CLF 9), (CLF 13, CLF 15, CLF 16), (CLF 14), (CLF 17, CLF 18), (CLF 20)	0.46	0.741	8
12	(CLF 1), (CLF 2), (CLF 3, CLF 19), (CLF 4, CLF 12), (CLF 5), (CLF 6, CLF 11), (CLF 7), (CLF 8), (CLF 9, CLF 10), (CLF 13, CLF 14, CLF 16), (CLF 15, CLF 17, CLF 18), (CLF 20)	0.45	0.77	7
11	(CLF 1), (CLF 2), (CLF 3, CLF 19), (CLF 4, CLF 12), (CLF 5), (CLF 6, CLF 11), (CLF 7, CLF 20), (CLF 8, CLF 10), (CLF 9), (CLF 13, CLF 14, CLF 16), (CLF 15, CLF 17, CLF 18)	0.45	0.782	7
10	(CLF 1, CLF 6), (CLF 2), (CLF 3, CLF 19), (CLF 4, CLF 13, CLF 14, CLF 16), (CLF 5, CLF 12), (CLF 7, CLF 11), (CLF 8, CLF 10), (CLF 9), (CLF 15, CLF 17, CLF 18), (CLF 20)	0.45	0.753	8
9	(CLF 1, CLF 5, CLF 12), (CLF 2), (CLF 3, CLF 19), (CLF 4), (CLF 6, CLF 8, CLF 11), (CLF 7), (CLF 9, CLF 10, CLF 20), (CLF 13, CLF 14, CLF 16), (CLF 15, CLF 17, CLF 18)	0.45	0.757	8
8	(CLF 1, CLF 8, CLF 15, CLF 17, CLF 18), (CLF 2), (CLF 3, CLF 7, CLF 19), (CLF 4, CLF 12), (CLF 5, CLF 20), (CLF 6, CLF 11), (CLF 9, CLF 10), (CLF 13, CLF 14, CLF 16)	0.45	0.75	8
7	(CLF 1, CLF 5), (CLF 2, CLF 13, CLF 14, CLF 16), (CLF 3, CLF 19), (CLF 4, CLF 9, CLF 12, CLF 15, CLF 17, CLF 18), (CLF 6, CLF 8, CLF 10, CLF 11), (CLF 7), (CLF 20)	0.45	0.731	8
6	**(CLF 1, CLF 15, CLF 17, CLF 18, CLF 20), (CLF 2, CLF 6, CLF 13, CLF 14, CLF 16), (CLF 3, CLF 5, CLF 19), (CLF 4, CLF 8), (CLF 7, CLF 11, CLF 12), (CLF 9, CLF 10)**	**0.45**	**0.74**	**10**
5	(CLF 1, CLF 7, CLF 8, CLF 10, CLF 12, CLF 13, CLF 14, CLF 16), (CLF 2, CLF 6, CLF 11), (CLF 3, CLF 15, CLF 17, CLF 18, CLF 19), (CLF 4, CLF 9), (CLF 5, CLF 20)	0.45	0.76	9
4	(CLF 1, CLF 7, CLF 15, CLF 17, CLF 18), (CLF 2, CLF 5, CLF 12, CLF 20), (CLF 3, CLF 4, CLF 8, CLF 10, CLF 13, CLF 14, CLF 16, CLF 19), (CLF 6, CLF 9, CLF 11)	0.45	0.74	10
3	(CLF 1, CLF 5, CLF 7, CLF 8, CLF 12), (CLF 2, CLF 6, CLF 9, CLF 10, CLF 11, CLF 15, CLF 17, CLF 18, CLF 20), (CLF 3, CLF 4, CLF 13, CLF 14, CLF 16, CLF 19)	2.25	0.716	12
2	(CLF 1, CLF 2, CLF 4, CLF 5, CLF 8, CLF 10, CLF 12, CLF 15, CLF 17, CLF 18, CLF 20), (CLF 3, CLF 6, CLF 7, CLF 9, CLF 11, CLF 13, CLF 14, CLF 16, CLF 19)	5.95	0.686	19
1	(CLF 1, CLF 2, CLF 3, CLF 4, CLF 5, CLF 6, CLF 7, CLF 8, CLF 9, CLF 10, CLF 11, CLF 12, CLF 13, CLF 14, CLF 15, CLF 16 , CLF 17 , CLF 18, CLF 19, CLF 20)	30.4	0.684	51

Hence, the SPL configuration/solution for the AS feature model after the consideration of feature dependencies and the business objectives can be seen in Figure 14.

The GA resulted in the following near optimal solution:

(CLF 4, CLF 8), (CLF 2, CLF 6, CLF 13, CLF 14, CLF 16), (CLF 1, CLF 15, CLF 17, CLF 18, CLF 20), (CLF 3, CLF 5, CLF 19), (CLF 9, CLF 10), (CLF 7, CLF 11, CLF 12)

Chromosome Representation:

(13 22 34 41 54 62 76 81 95 105 116 126 132 142 153 162 173 183 194 203 | 1 2 3 4 5 6)

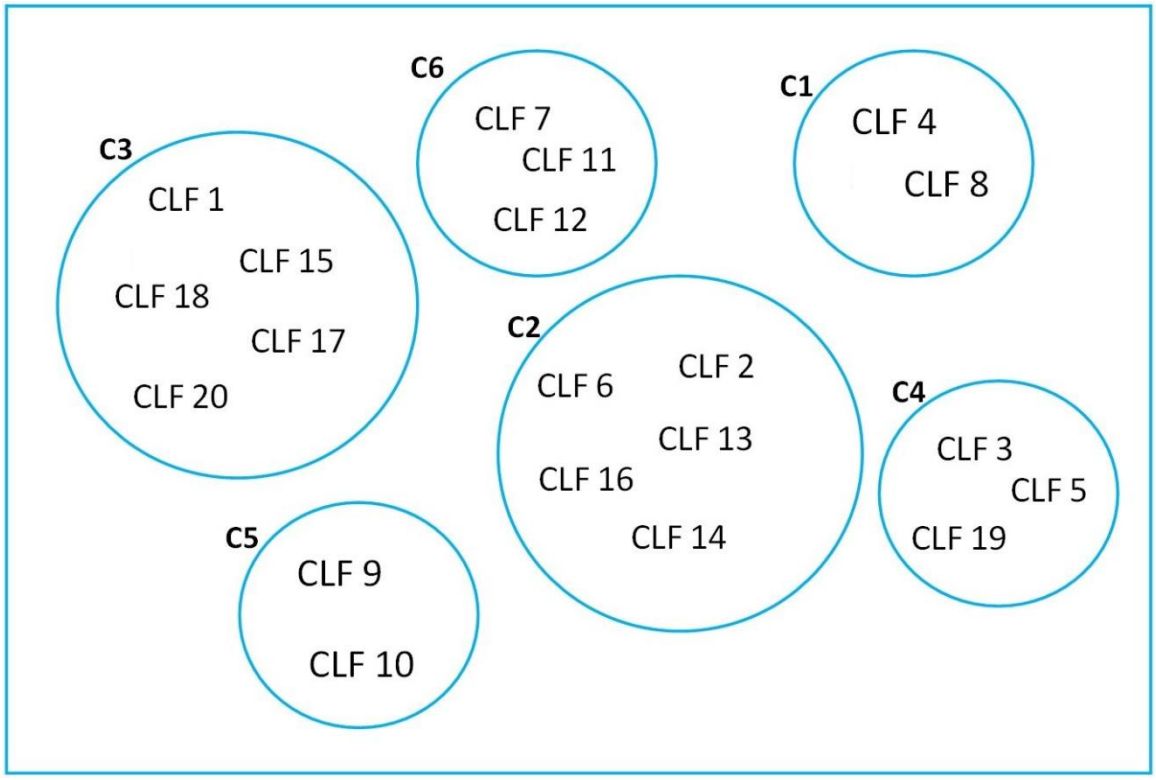

Figure 14 - SPL configuration representation after running GA

4.2 Case Study 2 – Life Cycle Assessment software tool (LCA)

For the purpose of the case study a suitable feature model was formulated for a Life Cycle Assessment Tool. It is a software tool used by architects, structural and Environmental Engineers and many others, to help them automate an exhaustive manual assessment known as Life-cycle assessment (LCA).

Life-cycle assessment is a technique to assess environmental impacts associated with all the stages of a product's life from cradle to grave [49]. Designers use this process to help critique their products. LCAs can help avoid a narrow outlook on environmental concerns by:

- Compiling an inventory of relevant energy and material inputs and environmental releases;
- Evaluating the potential impacts associated with identified inputs and releases;
- Interpreting the results to help make a more informed decision.

4.2.1 LCA - Feature Modeling

The feature model for the LCA tools was formulated based on the 4 phases that are part of the LCA process; namely (1) Goal and Scope Specification, (2) Inventory Analysis, (3) Impact Assessment and (4) Interpretation. Furthermore, to confirm the features and the variations that can be present in a LCA tool we used and inquired several commonly used LCA tools like OpenLCA, SimaPro and Gabi. This process resulted in a feature model that contained 39 features and 6 cross-tree constraints. These constraints were the basis upon which the relationships between the features were defined. The feature model

can be seen Figure 15; it depicts the core and optional features having a variety of relations among the features.

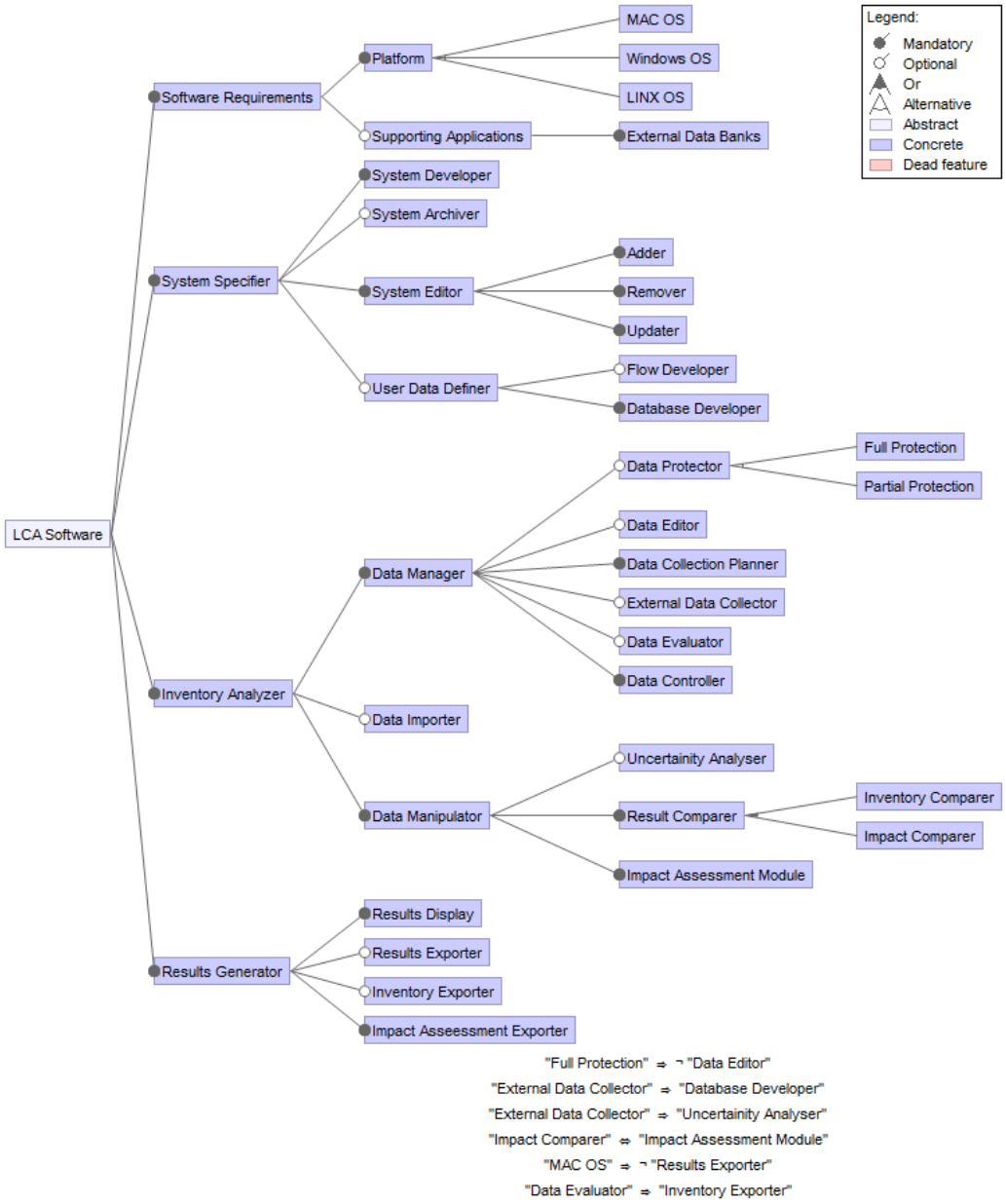

Figure 15 - Feature Model for LCA tool

4.2.2 LCA - Feature Dependency Analysis

The feature dependency analysis was performed over the feature model depicted in Figure 15. Like the previous case study the analysis considered the feature dependencies that existed among the concrete level features, as shown in Figure 3. The dependencies that existed among the CLFs of the LCA tool can be seen in Table 7.

Table 7 - Feature Dependency analysis for LCA tool

Concrete Level Features		Required Dependency / AND	Threat Dependency / Not / ALT	Structural Dependency OR	Usage Dependency
CLF 1	MAC OS		CLF 2, CLF 3, CLF 25		
CLF 2	Windows OS		CLF 1, CLF 3		
CLF 3	Linux OS		CLF 1, CLF 2		
CLF 4	External Data Banks				
CLF 5	System Developer	CLF 7, CLF 8, CLF 9			
CLF 6	System Archiver				
CLF 7	Adder	CLF 5, CLF 8, CLF 9			
CLF 8	Remover	CLF 5, CLF 7, CLF 9			
CLF 9	Updater	CLF 5, CLF 7, CLF 8			
CLF 10	Flow developer				
CLF 11	Database Developer				CLF 16
CLF 12	Full Protection		CLF 13, CLF 14		
CLF 13	Partial Protection		CLF 12		
CLF 14	Data Editor		CLF 12		
CLF 15	Data Collection Planner	CLF 18			
CLF 16	External Data Collector				CLF 11, CLF 20
CLF 17	Data Evaluator				CLF 26
CLF 18	Data Controller	CLF 15, CLF 23		CLF 21, CLF 22	
CLF 19	Data Importer				
CLF 20	Uncertainity Analyzer				CLF 16
CLF 21	Inventory Comparer			CLF 18, CLF 22	
CLF 22	Impact Comparer	CLF 23		CLF 18, CLF 21	
CLF 23	Impact Assessment Module	CLF 18			
CLF 24	Results Display	CLF 27			
CLF 25	Results Exporter		CLF 1		
CLF 26	Inventory Exporter				CLF 17
CLF 27	Impact Assessment Exporter	CLF 24			

4.2.3 LCA - Feature Dependency Graph (FDG)

The undirected FDG was created by developing a graph clustering algorithm defined in Chapter 3 of the thesis. The undirected FDG can be seen in Figure 16. The nodes of the graph represent the CLFs in the feature model, which was created for the LCA tool.

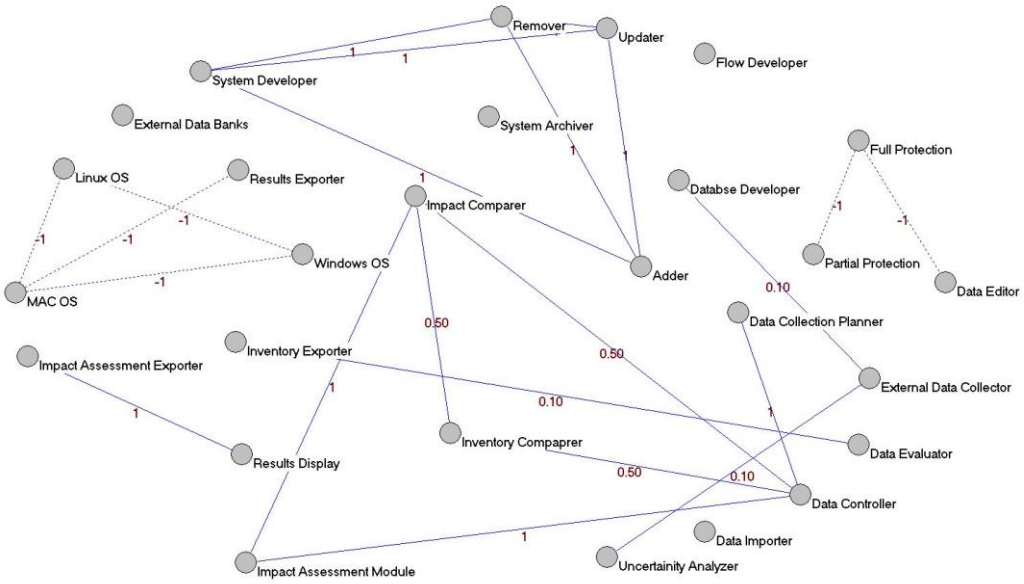

Figure 16 - FDG for the LCA FM using the feature dependencies

4.2.4 LCA - Clustering Results

Similar to the previous case study, the same method for graph clustering was used. The clusters were formed based on the relationships among the features. Clustering error closer to zero suggests that during clusters formation better compromises between the placement of features (based on the dependencies) into clusters, were made than the results produced with higher clustering error. The results after performing the graph clustering can be seen in Table 8.

Table 8 - Results after applying graph clustering on the LCA tool's FDG

Number of Clusters	Clusters	Clustering Error	Priority	Integrity
27	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5), (CLF 6), (CLF 7), (CLF 8), (CLF 9), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 18), (CLF 19), (CLF 20), (CLF 21), (CLF 22), (CLF 23), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	1.18	0.654	0
26	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 9), (CLF 6), (CLF 7), (CLF 8), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 18), (CLF 19), (CLF 20), (CLF 21), (CLF 22), (CLF 23), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	1.08	0.64	1
25	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5), (CLF 6), (CLF 7, CLF8, CLF 9), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 18), (CLF 19), (CLF 20), (CLF 21), (CLF 22), (CLF 23), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	0.88	0.62	6
24	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 18), (CLF 19), (CLF 20), (CLF 21), (CLF 22), (CLF 23), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	0.58	0.61	9
23	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 11, CLF 16), (CLF 6), (CLF 10), (CLF 12), (CLF 13), (CLF 14), (CLF 15),(CLF 17), (CLF 18), (CLF 19), (CLF 20), (CLF 21), (CLF 22), (CLF 23), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	0.57	0.615	9
22	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 22, CLF 23), (CLF 24, CLF 27), (CLF 6), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 18), (CLF 19), (CLF 20), (CLF 21), (CLF 25), (CLF 26)	0.38	0.575	11
21	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 18, CLF 21, CLF 22, CLF 23), (CLF 6), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 19), (CLF 20), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	0.23	0.579	15
20	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 16, CLF 20), (CLF 18, CLF 21, CLF 22, CLF 23), (CLF 6), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 17), (CLF 19), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	0.22	0.582	16
19	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15, CLF 18, CLF 22, CLF 23), (CLF 16) (CLF 17, CLF 26), (CLF 19), (CLF 20), (CLF 21), (CLF 24, CLF 27), (CLF 25)	0.12	0.547	17
18	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11, CLF 16), (CLF 12), (CLF 13), (CLF 14), (CLF 17, CLF 26), (CLF 15, CLF 18, CLF 22, CLF 23), (CLF 19), (CLF 20), (CLF 21), (CLF 24, CLF 27), (CLF 25)	0.11	0.55	17
17	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11), (CLF 12), (CLF 13, CLF 14), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 16), (CLF 17, CLF 26), (CLF 19), (CLF 20), (CLF 24, CLF 27), (CLF 25)	0.02	0.57	21
16	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11, CLF 16, CLF 20), (CLF 12), (CLF 13), (CLF 14), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 17, CLF 26), (CLF 19), (CLF 24, CLF 27), (CLF 25)	0	0.559	20
15	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11, CLF 16, CLF 20), (CLF 12, CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 13), (CLF 14), (CLF 17, CLF 26), (CLF 19), (CLF 24, CLF 27), (CLF 25)	0	0.572	20
14	(CLF 1, CLF 12), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6, CLF 25), (CLF10), (CLF 11, CLF 16, CLF 20), (CLF 13), (CLF 14), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 19), (CLF 17, CLF 26), (CLF 24, CLF 27)	0	0.58	20
13	(CLF 1), (CLF 2), (CLF 3, CLF 6, CLF 13), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9, CLF 12), (CLF 10), (CLF 11, CLF 16, CLF 20), (CLF 14), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 17, CLF 26), (CLF 19), (CLF 24, CLF 27), (CLF 25)	0	0.582	20
12	(CLF 1, CLF 13), (CLF 3), (CLF 4), (CLF 2, CLF 19, CLF 25), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11, CLF 16, CLF 20, CLF 24, CLF 27), (CLF 12), (CLF 14), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 17, CLF 26)	0	0.534	20
11	(CLF 1, CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 2, CLF 14), (CLF 3), (CLF 4, CLF 25), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6, CLF 13), (CLF 10), (CLF 11, CLF 16, CLF 20), (CLF 12), (CLF 17, CLF 26), (CLF 19, CLF 24, CLF 27)	0	0.555	20
10	(CLF 1, CLF 10), (CLF 2, CLF 13), (CLF 3, CLF 6), (CLF 4, CLF 25), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 11, CLF 16, CLF 20), (CLF 12, CLF 19), (CLF 14), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 17, CLF 24, CLF 26, CLF 27)	0	0.577	23
9	(CLF 1, CLF 11, CLF 16, CLF 20), (CLF 2, CLF 14), (CLF 3, CLF 4, CLF 19), (CLF 5, CLF 7, CLF8, CLF 9, CLF 13), (CLF 6, CLF 10, CLF 24, CLF 27), (CLF 12), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 17, CLF 26), (CLF 25)	0	0.545	22
8	(CLF 1, CLF 13, CLF 14, CLF 19), (CLF 2), (CLF 3, CLF 4, CLF 12), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6, CLF 10, CLF 24, CLF 27), (CLF 11, CLF 16, CLF 17, CLF 20, CLF 26), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 25)	0	0.64	29
7	(CLF 1, CLF 5, CLF 7, CLF8, CLF 9), (CLF 2, CLF 17, CLF 26), (CLF 3, CLF 6), (CLF 4, CLF 13, CLF 19), (CLF 10, CLF 14, CLF 24,CLF 25, CLF 27), (CLF 11, CLF 12, CLF 16, CLF 20), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23)	0	0.621	27
6	(CLF 1, CLF 24, CLF 27), (CLF 2, CLF 10), (CLF 3, CLF 6, CLF 11, CLF 13, CLF 16, CLF 20), (CLF 4, CLF 19), (CLF 5, CLF 7, CLF8, CLF 9, CLF 14, CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 12, CLF 17, CLF 25, CLF 26)	0	0.613	34
5	(CLF 1, CLF 5, CLF 7, CLF 8, CLF 9, CLF 10, CLF 12), (CLF 2, CLF 6, CLF 13, CLF 14), (CLF 3, CLF 17, CLF 26), (CLF 4, CLF 11, CLF 16, CLF 19, CLF 20, CLF 24, CLF 27), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23, CLF 25)	0	0.619	38
4	(CLF 1, CLF 4, CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 2, CLF 14, CLF 17, CLF 26), (CLF 3, CLF 6, CLF 10, CLF 13, CLF 19, CLF 25), (CLF 5, CLF 7, CLF 8, CLF 9, CLF 11, CLF 12, CLF 16, CLF 20, CLF 24, CLF 27)	0	0.607	31
3	**(CLF 1, CLF 4, CLF 13, CLF 19, CLF 24, CLF 27), (CLF 2, CLF 5, CLF 7, CLF 8, CLF 9, CLF 12, CLF 15, CLF 18, CLF 21, CLF 22, CLF 23, CLF 25), (CLF 3, CLF 6, CLF 10, CLF 11, CLF 14, CLF 16, CLF 17, CLF 20, CLF 26)**	**0**	**0.642**	**44**
2	(CLF 1, CLF 2, CLF 5, CLF 7, CLF 8, CLF 9, CLF 11, CLF 13, CLF 14, CLF 16, CLF 17, CLF 19, CLF 20, CLF 24, CLF 26, CLF 27), (CLF 3, CLF 4, CLF 6, CLF 10, CLF 12, CLF 15, CLF 18, CLF 21, CLF 22, CLF 23, CLF 25)	1.9	0.646	58
1	(CLF 1, CLF 2, CLF 3, CLF 4, CLF 5, CLF 6, CLF 7, CLF 8, CLF 9, CLF 10, CLF 11, CLF 12, CLF 13, CLF 14, CLF 15, CLF 16, CLF 17, CLF 18, CLF 19, CLF 20, CLF 21, CLF 22, CLF 23, CLF 24, CLF 25, CLF 26, CLF 27)	11.4	0.654	128

Hence, the optimum clustering error is 0.0 in our experiments. It remained constant at 0.0 for several different solutions. This behavior was expected as there were separate clusters seen in the FDG in Figure 16 prior to graph clustering. This meant the features in the LCA tool are less dependent upon each other. Hence, the features can be easily sorted out

into different clusters without compromising their interdependencies which are the crux of our approach.

Like in this case, if the error is same for several different solutions, the near optimal solution is the one that gives minimal error with the most number of clusters, as it gives more choices to distribute the product among the development teams. Therefore, the results in Table 8 suggest the optimal solution after applying the graph clustering algorithm is the SPL configuration/solution containing 16 clusters.

Therefore, the algorithm clustered the FDG into 16 clusters. The clustered FDG for the LCA tool can be seen in Figure 17, where features in same cluster carry the same color.

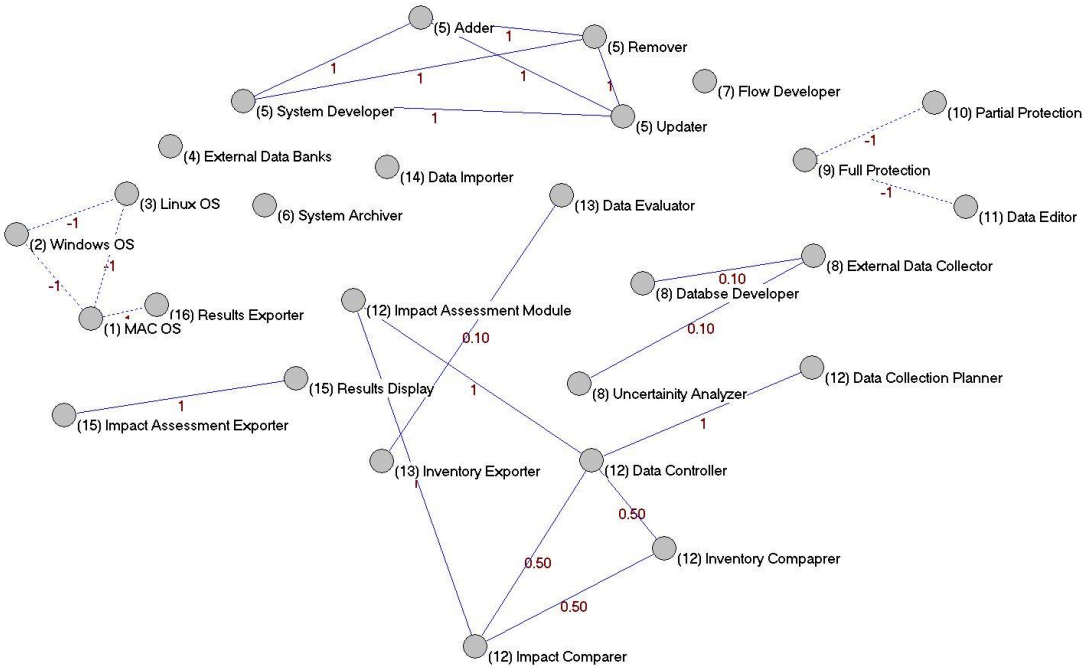

Figure 17 - A clustered FDG for LCA tool after running the Graph Clustering algorithm

4.2.5 Multi-objective feature selection using Genetic Algorithm

Similar to previous case study we applied GA algorithm to generate a near-optimal solution. The GA parameters that were used are mentioned in Table 9.

Table 9 - GA parameters

GA Parameters	Value
Population Size	20
Crossover probability	0.5
Mutation probability	0.1
Max Generation	1000
Selection Strategy	Tournament selection

To calculate the fitness value equation (3) is used. To compute the overall fitness value we need the computations of the PP, PI and Err or the clustering error is generated when we apply our graph clustering algorithm, PP is calculated by using equation (2) while PI is calculated using a technique called Nearest Common Predecessor, NCP. Calculation of PP and PI are discussed individually and then the resulting configuration is shown.

4.2.6 LCA – Calculating Product Priority, PP

As mentioned in Section 3.6.2, the assignment of priority was made based on a small survey where we asked the respondents, experts and developers, to rate the features based on the importance of the feature in the product. We then took the average value of the respondents' answers and assigned that value as the priority for a particular feature. Next, we inverted the priority for each feature as the higher the priority numbering the lower

the importance and vice versa (i.e. the feature having priority 1 is of most importance). This value was assigned to each individual feature as its priority and then used in the GA's objective function for calculating the product priority as discussed in Section 3.6.2.

The priorities for each concrete level feature in the 'Life Cycle Assessment software tool (LCA)' feature model can be seen in the Table 10.

Table 10 - Priorities of CLFs in LCA tools' feature model

Concrete Level Features		Priority	Priority Inverted
CLF 1	MAC OS	1	1
CLF 2	Windows OS	1	1
CLF 3	Linux OS	1	1
CLF 4	External Data Banks	2	0.5
CLF 5	System Developer	1	1
CLF 6	System Archiver	5	0.2
CLF 7	Adder	1	1
CLF 8	Remover	1	1
CLF 9	Updater	1	1
CLF 10	Flow developer	5	0.2
CLF 11	Database Developer	2	0.5
CLF 12	Full Protection	4	0.25
CLF 13	Partial Protection	4	0.25
CLF 14	Data Editor	4	0.25
CLF 15	Data Collection Planner	1	1
CLF 16	External Data Collector	2	0.5
CLF 17	Data Evaluator	3	0.333333333
CLF 18	Data Controller	1	1
CLF 19	Data Importer	2	0.5
CLF 20	Uncertainity Analyzer	2	0.5
CLF 21	Inventory Comparer	3	0.333333333
CLF 22	Impact Comparer	1	1
CLF 23	Impact Assessment Module	1	1
CLF 24	Results Display	1	1
CLF 25	Results Exporter	6	0.166666667
CLF 26	Inventory Exporter	6	0.166666667
CLF 27	Impact Assessment Exporter	1	1

4.2.7 AS – Calculating Product Integrity, PI

As mentioned in the previous chapter we used Nearest Common Predecessor (NCP) to calculate the integrity of the product. The NCP was calculated for every pair of concrete level features. To calculate the NCP of each pair of CLFs we converted the LCA tool's feature model to a similar sample that was seen in the previous chapter in Section 3.6.3 (Figure 6) and Section 4.1.7 (Figure 13). The modified feature model of LCA tool is shown in Figure 18.

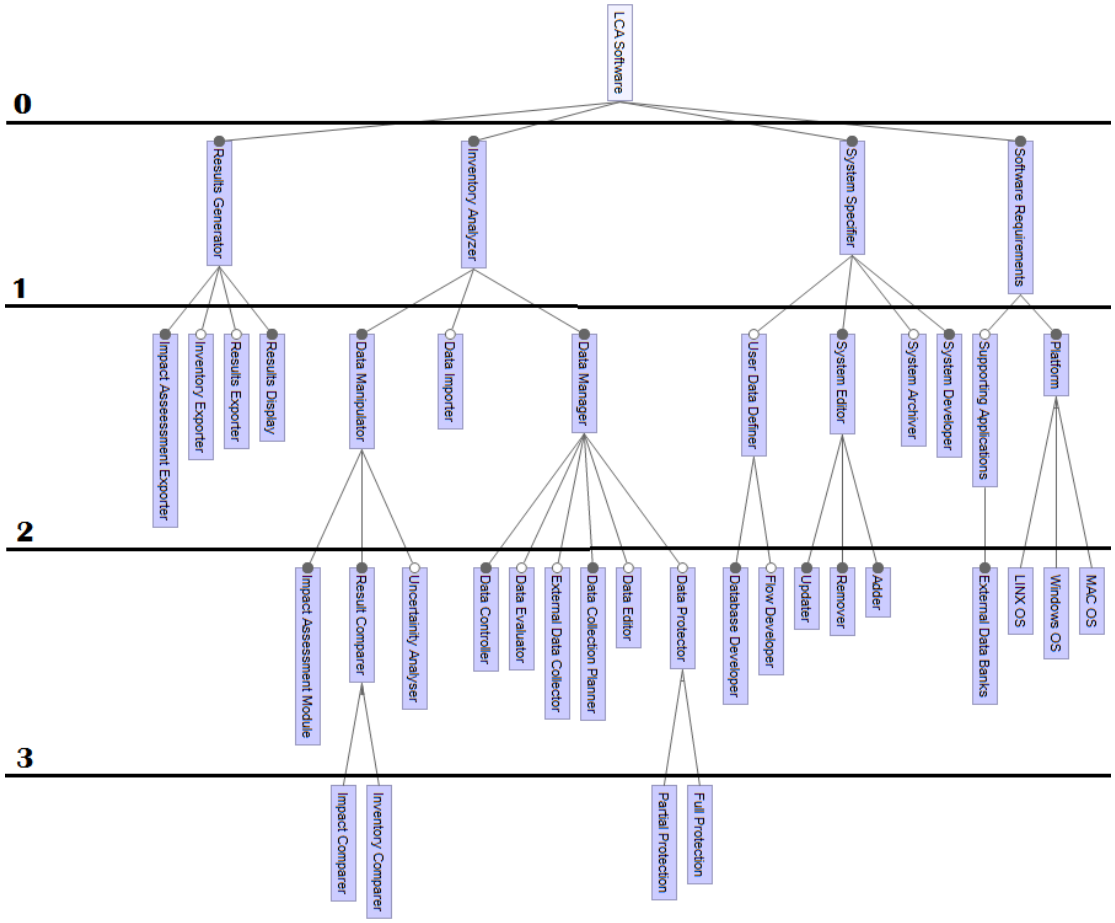

Figure 18 - Modified feature model to calculate NCP for LCA tool

Based on the Figure 13, the NCP calculations were carried out on LCA tool's feature model. The NCP value was calculated among each pair of concrete level features and then in the objective function for product integrity in the GA, the product integrity for each solution was calculated. The pair-wise NCP calculations can be seen in table 11.

Table 11 - NCP calculations for all CLFs in LCA tool's feature model

		MAC OS	Windows OS	Linux OS	External Data Banks	System Developer	System Archiver	Adder	Remover	Updater	Flow developer	Database Developer	Full Protection	Partial Protection	Data Editor	Data Collection Planner	External Data Collector	Data Evaluator	Data Controller	Data Importer	Uncertainity Analyzer	Inventory Comparer	Impact Comparer	Impact Assessment Module	Results Display	Results Exporter	Inventory Exporter	Impact Assessment Exporter
		CLF 1	CLF 2	CLF 3	CLF 4	CLF 5	CLF 6	CLF 7	CLF 8	CLF 9	CLF 10	CLF 11	CLF 12	CLF 13	CLF 14	CLF 15	CLF 16	CLF 17	CLF 18	CLF 19	CLF 20	CLF 21	CLF 22	CLF 23	CLF 24	CLF 25	CLF 26	CLF 27
MAC OS	CLF 1	**0**																										
Windows OS	CLF 2	2	**0**																									
Linux OS	CLF 3	2	2	**0**																								
External Data Banks	CLF 4	1	1	1	**0**																							
System Developer	CLF 5	0	0	0	0	**0**																						
System Archiver	CLF 6	0	0	0	0	1	**0**																					
Adder	CLF 7	0	0	0	0	1	1	**0**																				
Remover	CLF 8	0	0	0	0	1	1	2	**0**																			
Updater	CLF 9	0	0	0	0	1	1	2	2	**0**																		
Flow developer	CLF 10	0	0	0	0	1	1	1	1	1	**0**																	
Database Developer	CLF 11	0	0	0	0	1	1	1	1	1	2	**0**																
Full Protection	CLF 12	0	0	0	0	0	0	0	0	0	0	0	**0**															
Partial Protection	CLF 13	0	0	0	0	0	0	0	0	0	0	0	3	**0**														
Data Editor	CLF 14	0	0	0	0	0	0	0	0	0	0	0	2	2	**0**													
Data Collection Planner	CLF 15	0	0	0	0	0	0	0	0	0	0	0	2	2	2	**0**												
External Data Collector	CLF 16	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	**0**											
Data Evaluator	CLF 17	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	**0**										
Data Controller	CLF 18	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	**0**									
Data Importer	CLF 19	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	**0**								
Uncertainity Analyzer	CLF 20	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	**0**							
Inventory Comparer	CLF 21	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	**0**						
Impact Comparer	CLF 22	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	**0**					
Impact Assessment Module	CLF 23	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	**0**				
Results Display	CLF 24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**			
Results Exporter	CLF 25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**0**		
Inventory Exporter	CLF 26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	**0**	
Impact Assessment Exporter	CLF 27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	**0**

The results for the clustering errors, corresponding product priority and integrity values can be seen in Table 12.

Table 12 – Clustering Error, Priority and Integrity values for each resulting SPL solutions of LCA tool

Number of Clusters	Clusters	Clustering Error	Priority	Integrity
27	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5), (CLF 6), (CLF 7), (CLF 8), (CLF 9), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 18), (CLF 19), (CLF 20), (CLF 21), (CLF 22), (CLF 23), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	1.18	0.654	0
26	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 9), (CLF 6), (CLF 7), (CLF 8), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 18), (CLF 19), (CLF 20), (CLF 21), (CLF 22), (CLF 23), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	1.08	0.64	1
25	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5), (CLF 6), (CLF 7, CLF8, CLF 9), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 18), (CLF 19), (CLF 20), (CLF 21), (CLF 22), (CLF 23), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	0.88	0.62	6
24	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 18), (CLF 19), (CLF 20), (CLF 21), (CLF 22), (CLF 23), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	0.58	0.61	9
23	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 11, CLF 16), (CLF 6), (CLF 10), (CLF 12), (CLF 13), (CLF 14), (CLF 15),(CLF 17), (CLF 18), (CLF 19), (CLF 20), (CLF 21), (CLF 22), (CLF 23), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	0.57	0.615	9
22	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 22, CLF 23), (CLF 24, CLF 27), (CLF 6), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 18), (CLF 19), (CLF 20), (CLF 21), (CLF 25), (CLF 26)	0.38	0.575	11
21	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 18, CLF 21, CLF 22, CLF 23), (CLF 6), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 16), (CLF 17), (CLF 19), (CLF 20), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	0.23	0.579	15
20	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 16, CLF 20), (CLF 18, CLF 21, CLF 22, CLF 23), (CLF 6), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15), (CLF 17), (CLF 19), (CLF 24), (CLF 25), (CLF 26), (CLF 27)	0.22	0.582	16
19	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11), (CLF 12), (CLF 13), (CLF 14), (CLF 15, CLF 18, CLF 22, CLF 23), (CLF 16) (CLF 17, CLF 26), (CLF 19), (CLF 20), (CLF 21), (CLF 24, CLF 27), (CLF 25)	0.12	0.547	17
18	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11, CLF 16), (CLF 12), (CLF 13), (CLF 14), (CLF 17, CLF 26), (CLF 15, CLF 18, CLF 22, CLF 23), (CLF 19), (CLF 20), (CLF 21), (CLF 24, CLF 27), (CLF 25)	0.11	0.55	17
17	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11), (CLF 12), (CLF 13, CLF 14), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 16), (CLF 17, CLF 26), (CLF 19) , (CLF 20), (CLF 24, CLF 27), (CLF 25)	0.02	0.57	21
16	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11, CLF 16, CLF 20), (CLF 12), (CLF 13), (CLF 14), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 17, CLF 26), (CLF 19), (CLF 24, CLF 27), (CLF 25)	0	0.559	20
15	(CLF 1), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11, CLF 16, CLF 20), (CLF 12, CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 13), (CLF 14), (CLF 17, CLF 26), (CLF 19), (CLF 24, CLF 27), (CLF 25)	0	0.572	20
14	(CLF 1, CLF 12), (CLF 2), (CLF 3), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6, CLF 25), (CLF10), (CLF 11, CLF 16, CLF 20), (CLF 13), (CLF 14), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 19), (CLF 17, CLF 26), (CLF 24, CLF 27)	0	0.58	20
13	(CLF 1), (CLF 2), (CLF 3, CLF 6, CLF 13), (CLF 4), (CLF 5, CLF 7, CLF8, CLF 9, CLF 12), (CLF 10), (CLF 11, CLF 16, CLF 20), (CLF 14), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 17, CLF 26), (CLF 19), (CLF 24, CLF 27), (CLF 25)	0	0.582	20
12	(CLF 1, CLF 13), (CLF 3), (CLF 4), (CLF 2, CLF 19, CLF 25), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6), (CLF 10), (CLF 11, CLF 16, CLF 20, CLF 24, CLF 27), (CLF 12), (CLF 14), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 17, CLF 26)	0	0.534	20
11	(CLF 1, CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 2, CLF 14), (CLF 3), (CLF 4, CLF 25), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6, CLF 13), (CLF 10), (CLF 11, CLF 16, CLF 20), (CLF 12), (CLF 17, CLF 26), (CLF 19, CLF 24, CLF 27)	0	0.555	20
10	(CLF 1, CLF 10), (CLF 2, CLF 13), (CLF 3, CLF 6), (CLF 4, CLF 25), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 11, CLF 16, CLF 20), (CLF 12, CLF 19), (CLF 14), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 17, CLF 24, CLF 26, CLF 27)	0	0.577	23
9	(CLF 1, CLF 11, CLF 16, CLF 20), (CLF 2, CLF 14), (CLF 3, CLF 4, CLF 19), (CLF 5, CLF 7, CLF8, CLF 9, CLF 13), (CLF 6, CLF 10 , CLF 24, CLF 27), (CLF 12), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 17, CLF 26), (CLF 25)	0	0.545	22
8	(CLF 1, CLF 13, CLF 14, CLF 19), (CLF 2), (CLF 3, CLF 4, CLF 12), (CLF 5, CLF 7, CLF8, CLF 9), (CLF 6, CLF 10, CLF 24, CLF 27), (CLF 11, CLF 16, CLF 17, CLF 20, CLF 26), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 25)	0	0.64	29
7	(CLF 1, CLF 5, CLF 7, CLF8, CLF 9), (CLF 2, CLF 17, CLF 26), (CLF 3, CLF 6), (CLF 4, CLF 13, CLF 19), (CLF 10, CLF 14, CLF 24,CLF 25, CLF 27), (CLF 11, CLF 12, CLF 16, CLF 20), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23)	0	0.621	27
6	(CLF 1, CLF 24, CLF 27), (CLF 2, CLF 10), (CLF 3, CLF 6, CLF 11, CLF 13, CLF 16, CLF 20), (CLF 4, CLF 19), (CLF 5, CLF 7, CLF8, CLF 9, CLF 14, CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 12, CLF 17, CLF 25, CLF 26)	0	0.613	34
5	(CLF 1, CLF 5, CLF 7, CLF 8, CLF 9, CLF 10, CLF 12), (CLF 2, CLF 6, CLF 13, CLF 14), (CLF 3, CLF 17, CLF 26), (CLF 4, CLF 11, CLF 16, CLF 19, CLF 20, CLF 24, CLF 27), (CLF 15, CLF 18, CLF 21, CLF 22, CLF 23, CLF 25)	0	0.619	38
4	(CLF 1, CLF 4, CLF 15, CLF 18, CLF 21, CLF 22, CLF 23), (CLF 2, CLF 14, CLF 17, CLF 26), (CLF 3, CLF 6, CLF 10, CLF 13, CLF 19, CLF 25), (CLF 5, CLF 7, CLF 8, CLF 9, CLF 11, CLF 12, CLF 16, CLF 20, CLF 24, CLF 27)	0	0.607	31
3	**(CLF 1, CLF 4, CLF 13, CLF 19, CLF 24, CLF 27), (CLF 2, CLF 5, CLF 7, CLF 8, CLF 9, CLF 12, CLF 15, CLF 18, CLF 21, CLF 22, CLF 23, CLF 25), (CLF 3, CLF 6, CLF 10, CLF 11, CLF 14, CLF 16, CLF 17, CLF 20, CLF 26)**	**0**	**0.642**	**44**
2	(CLF 1, CLF 2, CLF 5, CLF 7, CLF 8, CLF 9, CLF 11, CLF 13, CLF 14, CLF 16, CLF 17, CLF 19, CLF 20, CLF 24, CLF 26, CLF 27), (CLF 3, CLF 4, CLF 6, CLF 10, CLF 12, CLF 15, CLF 18, CLF 21, CLF 22, CLF 23, CLF 25)	1.9	0.646	58
1	(CLF 1, CLF 2, CLF 3, CLF 4, CLF 5, CLF 6, CLF 7, CLF 8, CLF 9, CLF 10, CLF 11, CLF 12, CLF 13, CLF 14, CLF 15, CLF 16, CLF 17, CLF 18, CLF 19, CLF 20, CLF 21, CLF 22, CLF 23, CLF 24, CLF 25, CLF 26, CLF 27)	11.4	0.654	128

So the SPL configuration/solution for the LCA tool's feature model after the consideration of feature dependencies and the business objectives can be seen in Figure 19.

The GA resulted in the following near optimal solution:

(CLF 1, CLF 4, CLF 13, CLF 19, CLF 24, CLF 27), (CLF 2, CLF 5, CLF 7, CLF 8, CLF 9, CLF 12, CLF 15, CLF 18, CLF 21, CLF 22, CLF 23, CLF 25), (CLF 3, CLF 6, CLF 10, CLF 11, CLF 14, CLF 16, CLF 17, CLF 20, CLF 26)

Chromosome Representation:

(11 22 33 41 52 63 72 82 92 103 113 122 131 143 152 163 173 182 191 203 212 222 232 241 252 263 271 | 1 2 3)

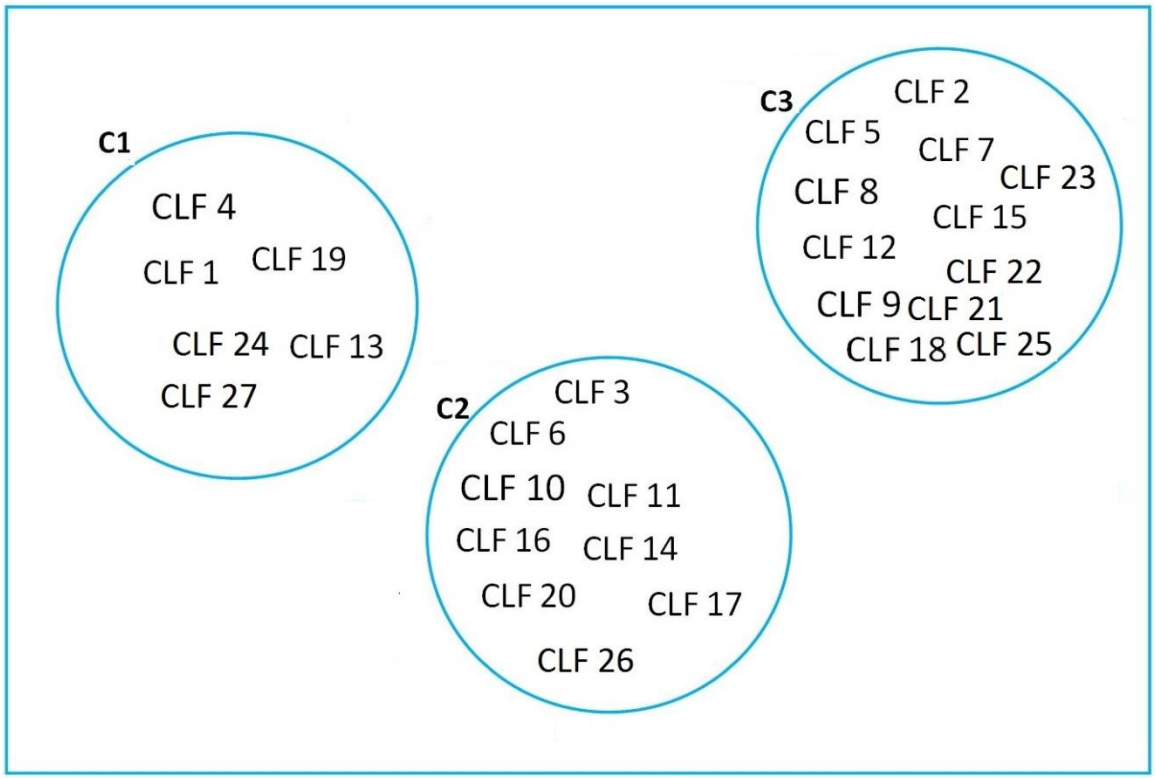

Figure 19 - SPL configuration representation for LCA tool after running GA

CHAPTER 5

DISCUSSION AND ANALYSIS

When we apply the graph clustering to both our case studies, they result in SPL solutions with variable clustering errors. These errors are representative of how much compliance the solution has with the dependencies that exist among the features. The closer the value to 0, the better the solution is; clustering error 0 suggests that all the feature dependencies are fulfilled. Hence, the more the dependencies among the features and the more distributive the dependencies are, the harder it is to come up with a solution closer to 0.

The best solution for AS (Case Study 1) is 0.45, which means that in producing a solution there is always a trade-off to be made while consideration of feature dependencies of different features. As seen in Table 2, there are a lot of dependencies that exist among the features due to the way the AS feature model is designed; and all these dependencies must be fully satisfied in order to achieve a cluster error of 0. In such complex situations it is hard to fulfill all dependencies, as fulfillment of one dependency might negatively affect the other. For instance, a feature A has a positive dependency with feature B, while feature B has a positive dependency with feature C and feature A has a negative dependency with feature C; in this case all three features must reside in a single cluster which requires some trade-off to be made in terms of consideration of the feature dependencies. So, some feature dependencies might be considered and some might not, depending upon the type of relationship that exists among each pair of features as each

type of dependency is assigned different edge weight in the feature dependency graph (FDG).

On the other hand, when we apply the graph clustering algorithm on LCA tool's feature model we get more solutions with the clustering error of 0. This means that there are several solutions available that fulfill all the feature dependencies among the features. This is due to the fact, and also can be seen in the feature dependency graph (FDG), Figure 16, that there are less distributive dependencies among the features; the FDG even before clustering seems to be clustered together. Hence, it is easier to satisfy the relationships among the features.

In real-world setting, however, we don't just have to comply with the feature dependencies but also with the business objectives which are required of the SPL solutions; as the purpose of using SPL is to be proactive in product evolution. Hence, SPL solutions are not of much importance if they don't accomplish the business objectives which are required out of them.

This need for conformity with the feature dependencies and the business objectives makes the pursuit of optimal SPL solution more complex. The problem leads to the second phase of the thesis where we apply the GA to produce near-optimal SPL solutions that observe the feature dependencies alongside the objectives, namely, product priority and product integrity.

As can be seen in Figure 20 and Figure 21, the SPL solutions produced before and after the application of our GA are different. The solutions on the right side of each Figure

show the SPL solutions that are near-optimal, fulfilling the objectives while conforming to the feature dependencies.

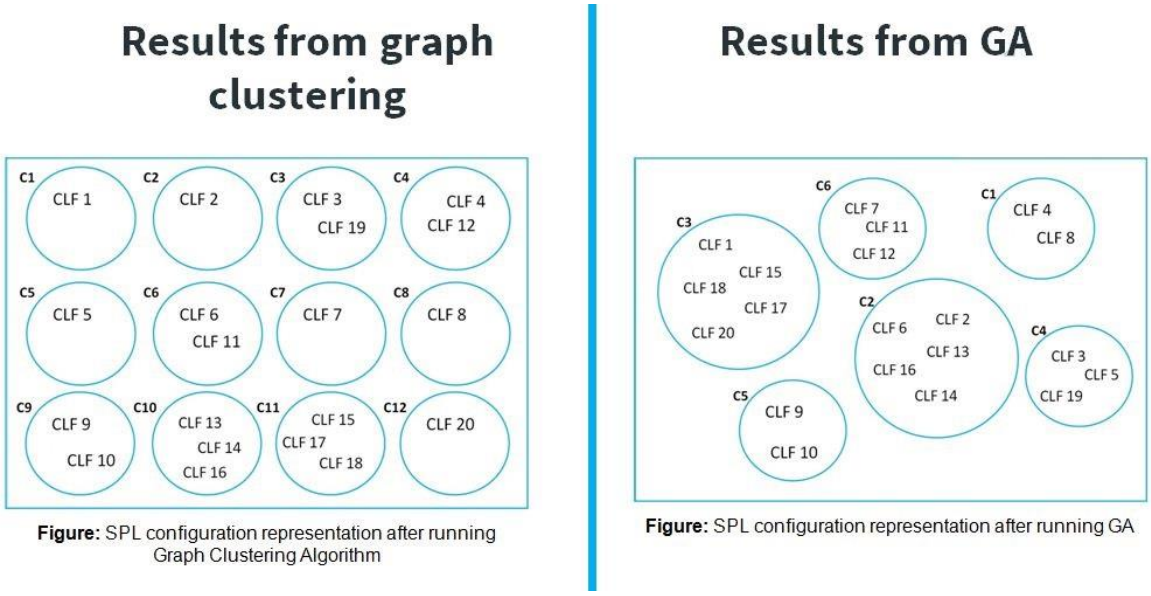

Figure 20 - Comparison of results after graph clustering and after applying GA, for AS

In Figure 20, the SPL configuration on the left shows the best possible solution we can get, when considering the relationships among the features only. However, this solution is less cohesive as there are several clusters having individual features, which shows the lack of synergy among the features. Hence, this SPL configuration has less integrity.

On the other hand, the solution on the left is the near-optimal solution which is a result of the GA we have developed. This solution has more cohesive components or clusters each having a collection of features that also comply with the dependencies in the best possible way, as both the solutions have the same clustering error of 0.45.

Moreover, as seen in Table 4 there were different priorities assigned to the each feature; our approach tries to preserve these priorities optimally while trying to keep a best trade-

off with the product integrity and clustering error. For instance, in the SPL solution on the left, the features in cluster ‘C6’ all have the same priority, all feature dependencies are considered except 1, which seems reasonable as the overall clustering error is 0.45, and all these features form a cluster with an integrity value of 1 which is greater than the overall integrity of the SPL configuration, suggesting that this component/cluster is highly cohesive.

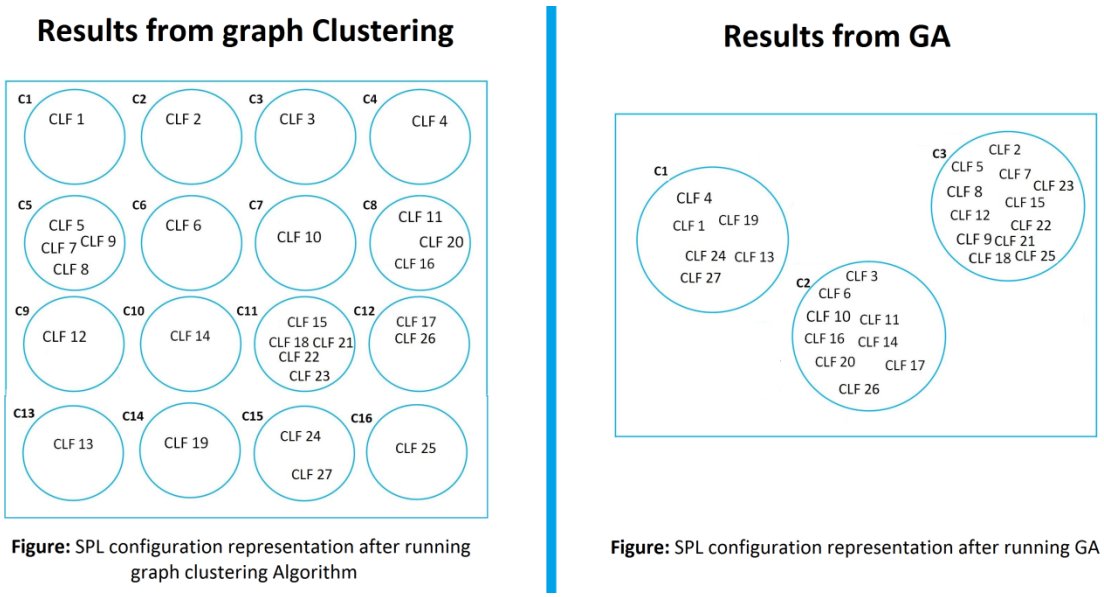

Figure 21 - Comparison of results after graph clustering and after applying GA, for LCA

Figure 21 shows the optimal SPL configuration before and after the application of the developed GA. On the left side we can see the best possible SPL solution which accords to the feature dependencies only. However, this solution too is perceived to be less cohesive as there are several clusters having individual features, which shows the lack of synergy among the features. Hence, this SPL configuration has less integrity.

The solution in discussion also has less product priority too. This is because the product priority is calculated using average of averages. In this case, if some feature with less priority is alone in a cluster, the cluster priority (average) will be less hence affecting the overall product priority negatively. However, if the same feature is placed in a cluster having high priority (average), this will surely decrease the priority (average) of the accepting cluster but the negative effect on the overall product priority will be less.

On the other hand, the solution on the left is the near-optimal solution which is a result of the GA we have developed. This solution has more cohesive components or clusters each having a collection of features that also comply with the dependencies in the best possible way, as both the solutions have the same clustering error of 0.

Moreover, as seen in Table 10 there were different priorities assigned to the each feature; the SPL solution produced after the full application of our approach preserves these priorities optimally while trying to keep a best trade-off with the product integrity and clustering error. For instance, the priority value of the cluster ‘C1’ in the SPL solution on the right side is 0.708 and the average of priorities from clusters (‘C1’, ‘C4’, ‘C13’, ‘C14’, ‘C5’) having the same features in the solution on the left side is 0.65. And as discussed in Section 3.6.2 the higher this value is the higher the priority.

This shows that our approach produces near-optimal solutions that conform to all the business objectives while fully considering the relationships among the features.

CHAPTER 6

CONCLUSION AND FUTURE DIRECTIONS

Due to the extensive use of software, the software industry and businesses require a strategic reuse technique like SPL to fulfill the expectations and the business needs. SPL however, requires the consideration of the interdependencies that exists among the core assets (features) of a SPL, while selecting the features for a valid product configuration. Furthermore, SPL also requires the fulfillment of the competing objectives/user preferences. In this thesis, an approach was developed that caters to both these requirements; by developing a dependency analysis technique for SPL features, then consolidating related features using signed graph clustering algorithm and finally, developing a genetic algorithm to balance the clustering error, product priority and product integrity (objectives) in a SPL to produce optimal (near-optimal) SPL configurations.

Moreover, we demonstrate the application of our graph based feature selection approach using two case studies, namely, Automotive System (AS) and LCA (Life Cycle Assessment) tool. The former is a common example to explain and easily understand the approach while demonstrating its concepts and applicability to similar environments. LCA tool however, is as engineering software. This shows that our approach can be potentially applied to different types of systems.

The results from the case studies suggest that only considering the feature dependencies may results in a different SPL configuration, which might actually not be an optimal

solution when other real-life business objectives are considered. Our approach, as depicted by the application of case studies, ensures that not only the relationships among the features are maintained but it also tries to give the near-optimal solution considering the business objectives.

In addition to that, the approach we formulated follows 'Design Science Research Guidelines/Framework', which ensures that the research work produced carries viable contributions, hence enabling technical as well as managerial personnel to readily make use of our approach.

Our Approach naturally is highly dependent upon the SPL specifications provided or elicited. Hence, it is of utmost importance that the specifications gathered are an exact idealization of the system under discussion. This affects the approach from the initial phases when we are forming the feature model for a given system. Hence, to formulate the feature model for the AS (case study 1), we carefully articulated the SPL specification and confirmed its viability from the experts in the industry. We also consulted the architectural engineering experts to form the feature model for the LCA tool (case study 2)

There are several directions in which this work can be extended further. For the future work we plan to rank the consolidated features (of the near-optimal solutions produced) using ranking algorithms to give a timeline to the SPL analyst in order to devise a way to easy and efficient SPL development during evolution. Another plan is to apply ACO algorithms to compare the results, which we believe will provide us further confidence in our technique.

References

[1] C. Henard, M. Papadakis, M. Harman, and Y. Le Traon, "Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines," pp. 517-528, 2015.

[2] R. Karimpour and G. Ruhe, "Bi-criteria Genetic Search for Adding New Features into an Exising Product Line," in *Proceedings of 1st International Workshop on Combining Modelling and Search-Based Software Engineering*, 2013, pp. 34-38.

[3] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*: Addison-Wesley, Boston, MA, USA, 2001.

[4] K. Pohl, G. Bockle, and F. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*: Springer, 2005.

[5] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," *Technical Report CMU/SEI-90-TR-021, Software Engineering Institute*, 1990.

[6] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "FORM: a feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. 5, pp. 143-168, 1998.

[7] K. C. Kang, J. Lee, and P. Donohoe, "Feature-oriented product line engineering," *IEEE Software*, vol. 19, pp. 58-65, 2002.

[8] J. Lee, K. C. Kang, P. Sawyer, and H. Lee, "A Holistic Approach to Feature Modeling for Product Line Requirements Engineering," *Requirements Engineering*, vol. 19, pp. 377-395, 2014.

[9] J. White, J. A. Galindo, T. Saxena, B. Dougherty, D. Benavides, and D. C. Schmidt, "Evolvoing Feature Model Configurations in Software Product Lines," *Journal of Systems and Software*, vol. 87, pp. 119-136, 2014.

[10] D. Batory, D. Benavides, and A. Ruiz-Cortes, "Automated analysis of feature models: challenges ahead," *Communications of the ACM*, vol. 49, pp. 45-47, 2006.

[11] 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*, vol. 84, pp. 2208-2221, 2011.

[12] A. S. Karatas and H. Oguztuzun, "Attribute-Based Variability in Feature Models," *Requirements Engineering,* vol. DOI 10.1007/s00766-014-0216-9, In Press.

[13] H. Cho, K. Lee, and K. C., "Feature Relationship and Dependency Management: An Aspect-Oriented Apporach," in *Proceedings of 12th International Software Product Line Conference*, 2008, pp. 3-11.

[14] S. Deelstra, M. Sinnema, and J. Bosch, "Experiences in software product families: problems and issues during product derivation," *Lecture Notes in Computer Science* vol. 3154, pp. 165-182, 2004.

[15] S. Deelstra, M. Sinnema, and J. Bosch, "Product derivation in software product families: a case study," *Journal of Systems and Software,* vol. 74, pp. 173-194, 2005.

[16] C. Henard, M. Papadakis, M. Harman, and Y. L. Traon, "Combining multi-objective search and constraint solving for configuring large software product lines," in *37th International Conference on Software Engineering (ICSE'15) - Volume 1*, 2015, pp. 517-528.

[17] D. Sprott and L. Wilkes, "Understanding service-oriented architecture," *The Architecture Journal,* vol. 1, pp. 10-17, 2004.

[18] M. A. Khan and S. Mahmood, "A graph based requirements clustering approach for component selection," *Advances in Engineering Software,* vol. 54, pp. 1-16, 2012.

[19] J. White, B. Dougherty, and D. C. Schmidt, "Selecting highly optimal architectural feature sets with Filtered Cartesian Flattening," *Journal of Systems and Software,* vol. 82, pp. 1268-1284, 2009.

[20] D. Benavides and A. Ruiz-Cortés, "First International Workshop on Analysis of Software Product Lines (ASPL'08)," *Software Product Line ...,* 2008.

[21] K. Czarnecki and A. Wasowski, "Feature diagrams and logics: There and back again," in *11th International Conference on Software Product Line*, 2007.

[22] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, " Automated reasoning on feature models," *Advanced Information Systems Engineering - Lecture Notes in Computer Science,* vol. 3520, pp. 491-503, 2005.

[23] D. Batory, " Feature models, grammars, and propositional formulas," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3714, pp. 7-20, 2005.

[24] E. Zitzler and K. Simon, "Indicator-Based Selection in Multiobjective Search," in *8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)* vol. 3242, ed, 2004, pp. 832-842.

[25] Y. Wang and J. Pang, " Ant colony optimization for feature selection in software product lines," *J. Shanghai Jiaotong Univ*, vol. 19, pp. 50-58, 2013.

[26] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "“Optimum feature selection in software product lines: Let your model and values guide your search,” " *1st Int. Work. Comb. Model. Search-Based Softw. Eng.* , pp. 22-27, 2013.

[27] X. Lian, "Optimized Feature Selection towards Functional and Non-functional Requirements in Software Product Lines," in *22nd IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2015, pp. 191-200.

[28] X. L. Lizhang, "An Evolutionary Methodology for Optimized Feature Selection in Software Product Lines," in *26th International Conference on Software Engineering & Knowledge Engineering (SEKE2014)*, 2015, pp. 2-5.

[29] A. S. Sayyad, T. Menzies, and H. Ammar, "On the value of user preferences in search-based software engineering: A case study in software product lines," in *35th Int. Conf. Softw. Eng*, 2013, pp. 492–501.

[30] K. De Jong, "Genetic algorithms: a 30 year perspective," *Perspectives on Adaptation in Natural and Artificial Systems*, vol. 11, 2005.

[31] E. Falkenauer, *Genetic Algorithms and Grouping Problems*: John Wiley \& Sons, Inc., 1998.

[32] Z. Michalewicz, *Genetic algorithms+ data structures= evolution programs*: Springer Science & Business Media, 2013.

[33] M. Quiroz-Castellanos, L. Cruz-Reyes, J. Torres-Jimenez, C. Gómez, H. J. F. Huacuja, and A. C. Alvim, "A grouping genetic algorithm with controlled gene transmission for the bin packing problem," *Computers & Operations Research*, vol. 55, pp. 52-64, 2015.

[34] R. Lewis and B. Paechter, "Application of the grouping genetic algorithm to university course timetabling," in *Evolutionary Computation in Combinatorial Optimization*, ed: Springer, 2005, pp. 144-153.

[35] E. C. Brown and R. T. Sumichrast, "CF-GGA: A grouping genetic algorithm for the cell formation problem," *International Journal of Production Research*, vol. 39, pp. 3651-3669, 2001.

[36] S. H. Razavi, E. O. M. Ebadati, S. Asadi, and H. Kaur, "An efficient grouping genetic algorithm for data clustering and big data analysis," in *Computational Intelligence for Big Data Analysis*, ed: Springer, 2015, pp. 119-142.

[37] O. Sobeyko and L. Mönch, "Grouping genetic algorithms for solving single machine multiple orders per job scheduling problems," *Annals of Operations Research*, vol. 235, pp. 709-739, 2015.

[38] L. E. Agustı´n-Blas, S. Salcedo-Sanz, S. Jiménez-Fernández, L. Carro-Calvo, J. Del Ser, and J. A. Portilla-Figueras, "A new grouping genetic algorithm for clustering problems," *Expert Systems with Applications*, vol. 39, pp. 9695-9703, 2012.

[39] B. W. Zulawinski, W. F. P. Iii, E. D. Goodman, and E. Falkenauer, "The grouping genetic algorithm (gga) applied to the bin balancing problem," 1995.

[40] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Quarterly*, vol. 28, pp. 75-105, 2004.

[41] F. Ahmed and L. F. Capretz, "Managing the business of software product line: An empirical investigation of key business factors," *Information and Software Technology*, vol. 49, pp. 194-208, 2007.

[42] L. M. Northrop, "Software product lines: reuse that makes business sense," *Australian Software Engineering Conference (ASWEC'06)*, vol. 2006, pp. 1 pp.-3, 2006.

[43] P. Doreian and A. Mrvar, "Partitioning signed social networks," *Social Networks*, vol. 31, pp. 1-11, 2009.

[44] T. Zaslavsky, "A mathematical bibliography of signed and gain graphs and allied areas," *The Electronic Journal of Combinatorics*, pp. 2009-2012, 2012.

[45] J. A. Davis, "Structural Balance, Mechanical Solidarity, and Interpersonal Relations," *American Journal of Sociology*, vol. 68, pp. 444-462, 1963.

[46] P. Doreian, V. Batagelj, and A. Ferligoj, "Generalized Blockmodeling, Structural Analysis in the Social Sciences 25," ed: Cambridge University Press, 2005.

[47] K. B. F. C. Takahiro, "The power of product integrity," *Harvard Bisiness Review*, vol. 68, p. 5, Nov, 1990 1990.

[48] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Systems*, vol. 9, pp. 193-212, 1995.

[49] S. Schaltegger, *Life Cycle Assessment (LCA)—Quo vadis?*: Springer Science & Business Media, 1996.

Vitae

Name	: Mohammad Ahsan Javed
Nationality	: Pakistani
Date of Birth	:11/6/1990
Email	: ahsanjaved09@gmail.com
Address	: P.O.BOX 221526, Riyadh 11311
Academic Background	: **M.S (Software Engineering)**
	May, 2016
	King Fahd University of Petroleum and Minerals
	Dhahran, Saudi Arabia
	B.S (Computer Science)
	May, 2013
	National University of Computer and Emerging Sciences
	Islamabad, Pakistan