UNED Escuela de Doctorado

TESIS DOCTORAL

2016

BDD Algorithms to Perform Hard Analysis Operations on Variability Models

D. Héctor José Pérez Morago

MÁSTER EN INGENIERÍA DEL SOFTWARE Y SISTEMAS

INGENIERÍA DE SISTEMAS Y CONTROL

DR. RUBEN HERADIO GIL

DR. DAVID FERNÁNDEZ AMORÓS

UNED Escuela de Doctorado

TESIS DOCTORAL

2016

BDD Algorithms to Perform Hard Analysis Operations on Variability Models

D. Héctor José Pérez Morago

MÁSTER EN INGENIERÍA DEL SOFTWARE Y SISTEMAS

INGENIERÍA DE SISTEMAS Y CONTROL

DR. RUBEN HERADIO GIL

DR. DAVID FERNÁNDEZ AMORÓS

Abstract

To compete in the global marketplace, manufacturers try to differentiate their products by focusing on individual customer needs. Fulfilling this goal requires companies to shift from mass production to mass customization. In the context of software development, software product line engineering has emerged as a cost-effective approach to developing families of similar products by supporting high levels of mass customization.

Variability models are often used to specify the common and variable features of the products in one such family. Moreover, they model the inter-feature constraints which must be satisfied to guarantee the validity of the derived products. Despite the benefits of variability models, constructing and maintaining them can be a laborious task, especially in product lines with a large number of features and constraints. As a consequence, the study of automated techniques to reason on variability models has become an important research topic for the product line community.

The aim of most automated techniques can be grouped in two classes: (i) ensuring the correctness of variability models, and (ii) providing guidance to derive products from a variability model. The former is usually put in the practise by means of analysis operations, which can be performed by black box reusing logic engines, such as SAT-solvers and binary decision diagram libraries. Unfortunately, such kind of reuse implies long computation times, and for some operations that need calling the logic engine an excessive amount of times, the approach does not scale. To overcome this problem, this thesis proposes new algorithms that directly deal with the binary decision decision diagram data structure encoding a variability model. In particular, our algorithms are specifically designed to detect those features that need to be included in all legal products, those ones that do not belong to any legal product, the number of products that include a particular feature, as well as the set of features that a particular feature needs to include or exclude to be part in any legal product.

The second problem this thesis faces is related to product derivation. In complex variability models, deriving a valid product is not trivial task at all, since

a lot of constraints between the features must be taken into account. To speed up product derivation, this thesis proposes a new approach that tries to minimise the number of configuration steps required on average to derive a whole product. Our approach, based on the information theory concept of entropy, takes advantage of the fact that, due to the inter-feature constraints, some decisions may be automatically derived from other decisions previously made.

Resumen

Como consecuencia de la globalización de los mercados, los fabricantes necesitan adaptar sus productos a las necesidades específicas de cada cliente. Alcanzar este objetivo requiere de las mismas un cambio en sus modelos de producción, pasando de la producción en masa a la producción personalizada de productos. En el contexto del desarrollo de software, la ingeniería de líneas de productos software ha emergido como un enfoque efectivo en costes que se centra en el desarrollo de familias de productos similares, soportando al mismo tiempo un alto grado de personalización.

Los modelos de variabilidad son utilizados para especificar las características comunes y variables a la familia de productos. Además, permiten modelar las restricciones entre características que todo producto debe satisfacer para ser considerado válido. A pesar de los beneficios de los modelos de variabilidad, construir y mantener dichos modelos puede ser una tarea compleja y laboriosa, especialmente para aquellos con un gran número de características y restricciones. Como resultado, el estudio de técnicas de razonamiento automático sobre modelos de variabilidad ha pasado a ser uno de los temas de investigación más importantes para la comunidad de las líneas de productos.

Las técnicas automáticas de análisis puede ser clasificadas en dos grupos atendiendo al objetivo para el que han sido diseñadas: aquellas que se encargan de garantizar la corrección del modelo y aquellas que se encargan de dar soporte a la configuración de los distintos productos. Las primeras son generalmente llevadas a la práctica por medio de operaciones de análisis, las cuales son realizadas por medio de funciones predefinidas en motores lógicos, tales como SAT-solvers o librerías de diagramas de decisión binarios. El principal problema de esta estrategia consiste en que tal tipo de reutilización implica el consumo de grandes cantidades de tiempo. Este hecho unido al hecho de que algunas de estas operaciones requieren de múltiples llamadas a dichas funciones deriva en que esta estrategia presente graves problemas de escalabilidad. Con el fin de dar respuesta a este problema, en esta tesis proponemos nuevos algoritmos que directamente interactúan con la estructura de datos interna del diagrama de decisión binario

utilizado para codificar el modelo de variabilidad. En concreto, nuestros algoritmos están específicamente diseñados para detectar aquellas caracteristicas que son incluidas en todos lo productos válidos, aquellas otras que no pertenecen a ningún producto válido, el número de produtos que incluyen una determinada caracteristicas, así como los conjuntos de caracteristicas que una caracteristica particular necesita incluir o excluir para ser incluida en algún producto válido.

El segundo problema que abordamos en esta tesis tiene que ver con la configuración de productos. La configuración de productos en modelos de variabilidad no es una tarea trivial ya que requiere de una gran cantidad de decisiones acerca de qué características deben ser incluidas y cuales no, teniendo en cuenta que estas se encuentra interrelacionadas. Con el fin de acelerar este proceso, en esta tesis proponemos un nueva estrategia que trata de minimizar el numero de decisiones a tomar en promedio para configurar un producto. Esta estrategia, basada en el concepto de entropía de la teoría de la información, aprovecha el hecho de que ciertas decisiones pueden ser automáticamente derivadas de las decisiones tomadas previamente.

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

CHAPTER

1

Introduction

To increase variety, improve customer satisfaction, reduce lead-times, and shorten costs, many companies have shifted from *mass production* to *mass customization* [SSJ05]. This shift of paradigm enriches the mass production economies of scale with custom manufacturing flexibility by developing families of related products instead of single products. From this perspective, designing a product family is the process of capturing and modeling multiple product variants to satisfy different market niches. Figure 1.1 depicts an example of how mass customization is being applied in the textile industry. In this kind of applications, customers may choose those options that best fit their needs.

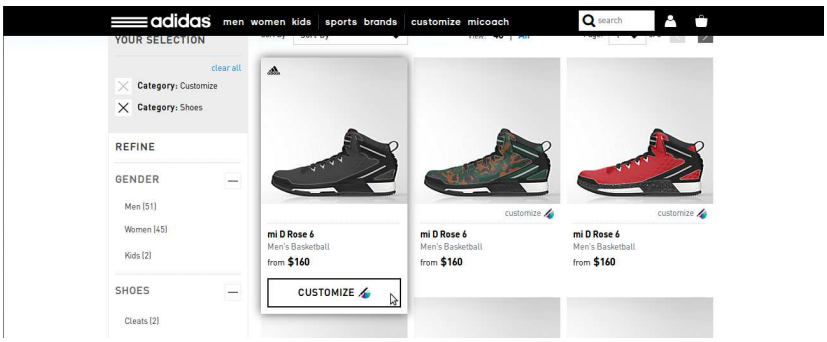


Figure 1.1: Web application to customize Adidas shoes

1. INTRODUCTION

Software Product Lines (SPLs) are a particular case of mass customization focused on the production of software. A SPL looks for the efficient development of whole portfolios of software products [CN01]. The basis of the approach is that products, instead of being developed from scratch one by one, are built from a core asset base (CAB), i.e., a collection of artifacts that have been designed specifically for use across the portfolio.

The SPL approach brings the benefits of economies of scale to software engineering, since less time and effort are needed to produce a greater variety of products. Many companies have exploited the concept of SPLs to increase the resources that focus on highly differentiating functionality and thus improve their competitiveness with higher quality and reusable products while decreasing the time-to-market condition. For instance, van der Linden et al. [vdLSR07] summarize experience reports from ten different companies working on diverse domains (e.g., Bosch on Gasoline Systems, Nokia on Mobile Phones, Philips on Consumer Electronics Software for Televisions, Siemens on Medical Solutions, etc.).

Under this approach, variability models are essential to represent the common and variable features that products may include. Here, products are specified as a unique combination of these features, where a feature is defined in the literature as "any prominent and distinctive aspect or characteristic that is visible to various stakeholders" [KCH+90].

As Berger et al. note [BSL+13], the practical significance of variability modeling is reflected in the rise of industrial tools for SPLs, such as *pure::variants* by Pure Systems GmbH and *Gears* by Big Lever Software Inc., in the current development of the Common Variability Language (CVL) standard by the Object Management Group (OMG), and, above all, by the fact that important open-source projects, as the Linux Kernel and the eCos operating system, are indeed managing huge variability models with thousands of selectable features.

On the other hand, not all feature combinations are valid. There may be feature incompatibilities (e.g., "it is not possible to choose more than one target architecture in the eCos operating system"), feature dependencies (e.g. "the TCP/IP stack could specify that it needs per-thread data support"), etc. Hence, there is a need to provide an automated support to deal with such entangled large-scale variability models.

Existing analyses of variability models fall into two main classes: correctness checking and derivation support [MWC09]. The usual way to carry out correctness checking is by means of analysis operations. According to the systematic literature review carried out by Benavides et al. [BSRC10], more than 30 operations have been reported, such as detecting core features (i.e., features that need to be included in all legal products) and dead features

(i.e., features that do not belong to any legal product). To do so, most approaches proceed as follows: first, models are translated into propositional logic formulas, then they are processed using off-the-self logic engines, such as SAT-solvers, Binary Decision Diagrams (BDDs) or CSP solvers, among others. In general, current approaches for variability model reasoning adopt a "black-box" strategy, that is, without seeing, knowing nor modifying the internals of the underlying logic engine. In practice, these approaches are efficient enough for some important operations [LGRC15, PLP11], because they only require calling once the logic engine. For instance, checking if a given combination of features is valid (i.e., the combination does not violate any of the feature dependencies), or detecting void models (i.e., models that, due to feature dependencies, do not represent any valid product), among others. However, it is important to note that other interesting operations require repeatedly calling the logic engine, and therefore, their response time can increase dramatically. In addition, it is well known that determining the satisfiability of a boolean formula is a NP-complete problem (i.e., the algorithm might not be feasible), whereas good BDD variable orderings produce compact BDDs and bad orderings may generate huge BDDs that eventually will become intractable.

Automatic analyses are also used to support derivation of products, i.e., the process of selecting or not features to obtain a product according to the model. In most cases, deriving a product involves one or more end users, also called *decision makers*, that translate their requirements into decisions on how the variability should be handled. However, it is important to note that this is not always a straightforward task because of two main reasons. First, the number of features that a model may contain can be very large, so the number of decisions that a user has to make can be huge. Second, the dependencies between features constrain the selection process. To overcome those difficulties, some tools, commonly called configurators, are used [PH04]. Some examples of commercial configurators are Configit[1], SAP Product Configurator[2], Oracle Configurator[3], etc. Given the high number of decisions needed to configure a complex product, a novel problem not supported by commercial tools is guiding the decision maker to minimize the number of configuration steps required. This is one of the problems this thesis faces.

[1] http://configit.com/
[2] https://scn.sap.com/docs/DOC-25224
[3] http://www.oracle.com/us/products/applications/046986.pdf

1.1 Objectives

For the most part, this thesis contributes to the SPL research field. In particular, it focuses on two of the most important topics in the area: the automated analysis of varibility models and guidance support for product derivation. Therefore, the general objective of this thesis is twofold. On the one hand, it provides new algorithms to perform hard analysis operations (i.e, those ones that require repeatedly calling a logic engine). On the other hand, it provides a new heuristic to speed up the product derivation process by reducing the number of decisions that users have to make.

In particular, the following specific objectives are addressed in this thesis:

- Designing an approach to speed up the product derivation process on variability models.
- Implementing algorithms to efficiently compute core/dead features on variability models.
- Implementing algorithms to efficiently compute the impact/exclusion set of each feature.
- Implementing algorithms to efficiently compute feature commonality.
- Redefining core/dead and impact/exclusion measures, applying the concepts of feature commonality and feature conditional commonality. Such redefinitions unveil critical information that current measures cannot detect.
- Implementing algorithms to support our measure redefinition.
- Identifying the main research topics, the evolution of the interest in those topics and the relationships among topics for the SPL research area.

These objectives are elaborated on each part of the thesis as described in the following section.

1.2 PhD thesis outline

This work can be divided in three well-differentiated parts. The first one analyzes the literature on SPLs using bibliometric techniques. The second one is focused on reviewing and providing solutions for the automated analysis research topic. The final part of this thesis focuses on the product derivation task, where we provide a new heuristic for user guidance. Further details of each chapter are given below.

Chapter 2

This chapter provides the background information necessary for reading this thesis, including the subjects of variability models, automated analysis, product derivation and BDDs.

Chapter 3

This chapter presents the state-of-the-art on the SPL literature by means of a bibliometric analysis. As a result of this analysis, the most researched topics and how the interest in these topics has evolved over the time are provided. In addition, the state-of-the-art of two of the most researched topics, which will be covered in this thesis, are reviewed in detail: the automated analysis of variability models and product derivation.

Chapter 4

This chapter introduces our approaches to compute hard operations identified in Section 3.2. Furthermore, an algorithm for each operation is proposed. The chapter ends up comparing the theoretical complexity of our approaches with existing ones.

Chapter 5

This Chapter introduces our heuristic to minimize the number of decisions that users have to make in order to derive a product from a variability model. In addition, an algorithm to support our approach and its corresponding implementation are presented. Finally, our approach is theoretically compared with alternative methods proposed in the related work.

Chapter 6

This chapter reports the results of empirical experiments to evaluate approaches proposed in this thesis. First, the models that our benchmark includes are presented. Then, the result of the experiments are discussed in order to test the validity of our approaches to compute hard operations. Finally, we report the results of empirical experiments of our heuristic for user guidance.

Chapter 7

This chapter wraps up the thesis, highlighting its main contributions and proposing future lines of work.

1.3 Main contributions

This section briefly summarizes the contributions of the thesis, distinguishing between software developed and published papers.

1.3.1 Software implementations

The particular results of this Ph. D. thesis include the development of algorithms to support analysis operations and product derivation guidance. The most important deliverables are:

- A c++ implementation for the BuDDy library of our algorithm for the efficient identification of core and dead features in variability models is provided. In addition this implementation and the benchmark used to validate it are freely available at `http://hperez30.github.io/CoreAndDeadFeatures/`
- A c++ implementation for the BuDDy library of our algorithm for the efficient computation of the commonalities of the variables of a boolean formula encoded as a BDD.
- A c++ implementation for the BuDDy library of our algorithm for the efficient computation of the impact and exclusion sets of the variables of a boolean formula encoded as a BDD.

1.3.2 Publications

The research undertaken in this thesis has been published in 4 journal papers and 2 conference papers:

- Efficient Identification of Core and Dead Features in Variability Models. **Hector Perez-Morago**, Ruben Heradio, David Fernandez-Amoros, Roberto Bean, Carlos Cerrada. IEEE Access. 2015, Volume 3, Pages 2333-2340. doi:10.1109/ACCESS.2015.2498764. JCR Q2.

- A Bibliometric Analysis of 20 Years of Research on Software Product Lines. Ruben Heradio, **Hector Perez-Morago**, David Fernandez-Amoros, Francisco Javier Cabrerizo, Enrique Herrera-Viedma. Information and Software Technology. 2016, Volume 72, Pages 1 - 15.doi:10.1016/j.infsof.2015.11.004. JCR Q1.

- Augmenting Measure Sensitivity to Detect Essential, Dispensable and Highly Incompatible Features in Mass Customization. Ruben Heradio, **Hector Perez-Morago**, Mauricio Alférez, David Fernandez-Amoros, Germán H Alférez. European Journal of Operational Research. Volume 248, Issue 3, 1 February 2016, Pages 1066–1077. doi:10.1016/j.ejor.2015.08.005. JCR Q1.

- Speeding up Derivative Configuration from Product Platforms. Ruben Heradio, David Fernandez-Amoros, **Hector Perez-Morago**, Antonio-Adan. Entropy. June 2014, Volume 16(6), Pages 3329-3356. doi:10.3390/e16063329. JCR Q2.

- Binary Decision Diagram Algorithms to Perform Hard Analysis Operations on Variability Models. Ruben Heradio, **Hector Perez-Morago**, David Fernandez-Amoros, Roberto Bean, F. Javier Cabrerizo, Carlos Cerrada, and Enrique Herrera-Viedma. Intelligent Software Methodologies, Tools and Techniques: 15th International Conference, SoMet 2016, Larnaca (Cyprus), September 12-14, 2016. CORE B.

- A Science Mapping Analysis of The Literature on Software Product Lines. Ruben Heradio, **Hector Perez-Morago**, David Fernandez-Amoros, Francisco Javier Cabrerizo, Enrique Herrera-Viedma. Intelligent Software Methodologies, Tools and Techniques: 14th International Conference, SoMet 2015, Naples (Italy), September 15-17, 2015. doi:10.1007/978-3-319-22689-7_1. CORE B.

1.3.3 Research visits

Throughout the development of these doctoral studies we carried out some research activities in collaboration with the Johannes Kepler University (Linz, Austria). A research visit to the Institute for Software Systems Engineering (ISSE) for the period from 1 June to 31 August 2016, which was supported by Estancias Breves 2016 funded by Universidad Nacional de Educación a Distancia (UNED).

1.3.4 Research projects

This work was supported by a pre-doctoral scholarship funded by Universidad Nacional de Educación a Distancia (UNED), the Comunidad de Madrid under the RoboCity2030 excellence research network under grant 52013/MIT-2748, and the Spanish Ministry of Economy and Competitiveness through the CICYT. Project DPI-2013-44776-R.

CHAPTER

2

Background

In this chapter, the background information necessary for reading this thesis is provided. Section 2.1 introduces variability models. Section 2.2 shows how they can be translated into equivalent propositional formulas. Section 2.3 introduces SAT Solvers and shows how they can be used to reason on variability models. Section 2.4 reviews reduced ordered BDDs, a well-known data structure, and shows how they can be used to reason on variability models and to undertake product derivation. Finally, the data structure for BDDs used by the algorithms described throughout this thesis is presented in Section 2.5.

2.1 A Brief Introduction to Variability Models

A *variability model*, also known as feature model, decision diagram, configuration model, etc, is often used to specify the features supported by a product line and their inter-relations [PBL05]. There are in the literature several notations for variability models. However, as we will see, our work deals with the logic representation of variability models. For that reason, providing detailed survey on the different variability model languages is out of the scope of this thesis. We refer the reader to [SHTB07] for a detailed review and comparison between them.

We present now the running example which will be used throughout this thesis. Figure 2.1 represents a variability model written as a directed graph where the nodes represent features and the edges represent constraints related to dependencies or incompatibilities between features. A dashed-line edge depicts that the two connected features are incompatible, while a solid-line edge from a feature to another represents that the first feature requires the second one[1].

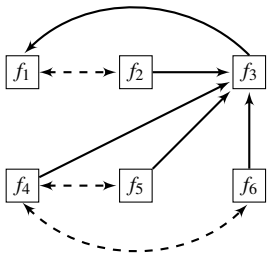


Figure 2.1: A variability model example

A product in a SPL can be specified by a selection of features. Hence, the product $P_1 = \{f_1, \neg f_2, \neg f_3, \neg f_4, \neg f_5, \neg f_6\}$ describes a unique product for the variability model depicted in Figure 2.1. P_1 is valid since it does not violate any of the relations in the model. However, $P_e = \{f_1, \neg f_2, f_3, f_4, f_5, \neg f_6\}$ is discarded because it violates the constraint $f_4 \dashleftarrow\!\dashrightarrow f_5$. From all possible combinations of 6 features, i.e., 2^6, the set of valid products is reduced to the six ones summarized in Table 2.1.

	Valid Products
1	$f_1, \neg f_2, \neg f_3, \neg f_4, \neg f_5, \neg f_6$
2	$f_1, \neg f_2, \quad f_3, \neg f_4, \neg f_5, \neg f_6$
3	$f_1, \neg f_2, \quad f_3, \quad f_4, \neg f_5, \neg f_6$
4	$f_1, \neg f_2, \quad f_3, \neg f_4, \quad f_5, \neg f_6$
5	$f_1, \neg f_2, \quad f_3, \neg f_4, \neg f_5, \quad f_6$
6	$f_1, \neg f_2, \quad f_3, \neg f_4, \quad f_5, \quad f_6$

Table 2.1: Valid products for Figure 2.1

From here on, products will be expressed enumerating only those features which are included in them. For instance, $\{f_1, f_3\}$ and $\{f_1, f_3, f_4\}$ expresses products P_2 and P_3 in Table 2.1, respectively.

[1]Note that, this is a simple example and more complex relation between features may occur.

2.2 Reasoning on Variability Models

As mentioned in the previous section, a number of different notations on variability models are available. For instance, Feature Diagrams (FD) [KCH+90], the Configit language[1], the SAP Product Configurator language [2], etc. Interestingly, most of those notations are semantically equivalent [SHTB07]. In fact, instead of processing models directly, automated tools for variability management usually translate them into a propositional logic representation, such as a logic formula in conjunctive normal form (CNF), a BDD, etc. That logic representation is then processed using off-the-self tools, such as SAT solvers, BDD engines, etc. [HFAPMA14].

To sum up, the general approach proceeds as follows [BSRC10]:

1. Each feature of the variability model maps to a variable of the propositional formula.

2. Each relationship of the model is mapped into one or more formulas depending on the type of relationship. Note that, in this step, some auxiliary variables can appear.

3. The resulting formula is the conjunction of all the resulting formulas of step 2.

As an example, Equation 2.1 shows the Boolean encoding of Figure 2.1. The first row means that at least one of the six features has to be selected. The second and third rows encode constraints between those features. The second row encodes five dependencies, for example $\neg f_2 \vee f_3$ means that f_2 requires f_3. The third row encodes three incompatibilities, for example $\neg f_1 \vee \neg f_2$ means that f_1 is incompatible with f_2. We refer the reader to [Bat05] for a more detailed explanation of the model-to-logic conversion.

$$\begin{aligned}\psi \equiv\ & (f_1 \vee f_2 \vee f_3 \vee f_4 \vee f_5 \vee f_6) \wedge \\ & (\neg f_2 \vee f_3) \wedge (\neg f_3 \vee f_1) \wedge (\neg f_4 \vee f_3) \wedge (\neg f_5 \vee f_3) \wedge (\neg f_6 \vee f_3) \wedge \\ & (\neg f_1 \vee \neg f_2) \wedge (\neg f_4 \vee \neg f_5) \wedge (\neg f_4 \vee \neg f_6)\end{aligned} \tag{2.1}$$

[1] http://configit.com/
[2] https://scn.sap.com/docs/DOC-25224

2.3 Boolean Satisfiability Problems

Once a variability model is translated into a propositional logic representation, a SAT solver or a BDD engine can be used to reason about it. In this section, we focus on SAT Solvers, showing how they can be used to reason about variability models. Later, BDD engines are reviewed in detail in Section 2.4.

SAT solvers support determining the satisfiability of a boolean formula in conjunctive normal form (CNF). A boolean formula is in CNF if it represents a conjunction of clauses (i.e., the constraints in a variability model) in which a clause is a disjunction of literals and a literal is a variable (i.e., the features in a variability model) or its negation. For instance, Equation 2.1 is in CNF.

SAT solvers only return one of the two following results: *satisfiable* or *unsatisfiable*. A CNF formula is satisfiable iff there is an assignment for all its variables such that the CNF formula evaluates to true. Otherwise, it is unsatisfiable.

Although, SAT-solvers only return satisfiable/unsatisfiable, the reasoning they support can be quite complex. For instance, by analyzing the CNF formula or changing assumptions one can find out if variables always have to be either true or false to make the formula satisfiable, e.g. given the formula in Equation 2.1, a SAT solver can be use to deduce that f_1 always needs to be true to make this CNF satisfiable.

Unfortunately, SAT solvers have two main issues. On the one hand, the SAT problem is known to be NP-complete [Coo71]. On the other hand, it is important to take into account that the logic representation of the model may not be in CNF. Therefore, this formula has to be transformed into an equivalent formula in CNF. However, the equation size of the translated formula may grow exponentially. To overcome this problem there are in the literature some methods [Tse83, dlT92]. But, such methods usually introduce additional variables which increase the gap between SAT solver's solution and its interpretation in the application domain [IST13].

2.4 Reduced Ordered Binary Decision Diagrams

BDDs have been widely used as a way of representing and manipulating boolean functions in many practical applications during the last decades. For instance, they have been used in logic synthesis, verification, configuration, constraint satisfaction and optimization [Men09].

BDDs are a way of representing boolean functions. They are rooted, directed, acyclic graphs, which consist of several decision nodes and terminal nodes [Bry86]. There are two types of terminal nodes called 0-terminal and 1-terminal. Each decision node v is labeled by a boolean variable f and has two child nodes called *low* and *high* (which are usually depicted by dashed and solid lines, respectively). The edge from node v to a low (or high) child represents an assignment of f to 0[1] (resp. 1).

A BDD is said to be *ordered* and *reduced* if it has the following two properties: on the one hand, it is ordered if different variables appear in the same order on all paths from the root. On the other hand, it is reduced if the following rules are applied to its graph [HR04]:

R1 *Removal of duplicate terminals*. If a BDD contains more than one terminal *0*-node, then all edges which point to such a *0*-node are redirected to just one of them. The same rule is applied to terminal nodes labelled with 1.

R2 *Removal of redundant tests*. If both outgoing edges of a node v_i point to the same node v_j, then v_i is eliminated, sending all its incoming edges to v_j.

R3 *Removal of duplicate non-terminals*. If two distinct nodes v_i and v_j in the BDD are the roots of structurally identical subBDDs, then one of them is eliminated, say v_j, and all its incoming edges are redirected to the other one.

In popular usage, the term BDD almost always refers to Reduced Ordered Binary Decision Diagram [HR04]. From here on, we will follow that convention as well. According to [Bry86] BDDs are a canonical representation for a particular boolean function and variable order. This property makes the test for equivalence checking inexpensive.

A path from the root to 1-terminal node represents a valid assignment of values to variables. For example, the path $v_8 \rightarrow v_7 \dashrightarrow v_6 \dashrightarrow v_4 \dashrightarrow v_3 \dashrightarrow v_2 \dashrightarrow 1$ in Figure 2.2 represents a valid product P_1 in Table 2.1.

Figure 2.2 is the BDD representation of Equation 2.1 using the variable ordering $f_1 \prec f_2 \prec f_3 \prec f_4 \prec f_5 \prec f_6$. Note that a logic formula may be encoded with different BDDs according to the variable ordering used to synthesize the BDD.

It is well-known that the size of a BDD depends on the variable ordering used to build them [Bry86]. For instance, Figure 2.3 is the BDD representation of Equation 2.1 using the

[1] throughout this dissertation false/true and 0/1 are used interchangeably

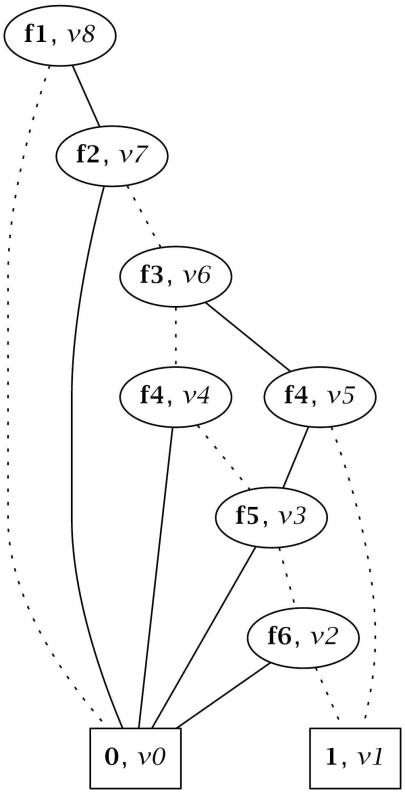


Figure 2.2: BDD for Eq. 2.1 according to the var ordering $f_1 \prec f_2 \prec f_3 \prec f_4 \prec f_5 \prec f_6$.

variable ordering $f_4 \prec f_5 \prec f_6 \prec f_3 \prec f_2 \prec f_1$. Note that this BDD is semantically equivalent to the one in Figure 2.2. However, it is has more nodes, and so it is a less memory-efficient encoding than the BDD in Figure 2.2.

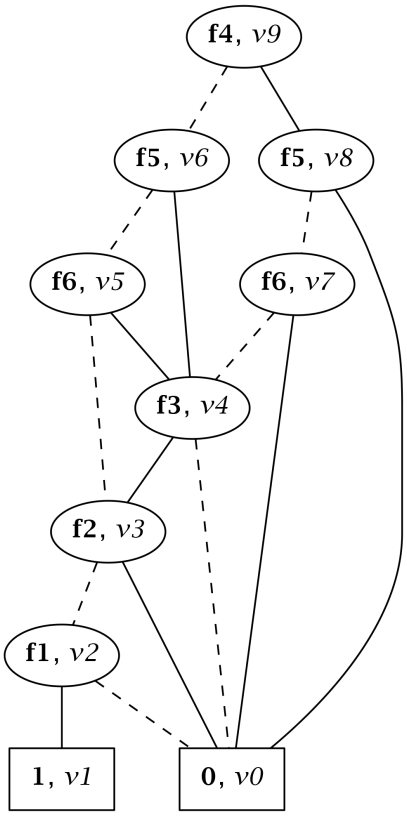


Figure 2.3: BDD for Eq. 2.1 according to the var ordering $f_4 \prec f_5 \prec f_6 \prec f_3 \prec f_2 \prec f_1$.

Unfortunately, it is also well-known that finding an optimal ordering is an NP-complete problem [BW96]. Providing a heuristic to find a good variable ordering is out of the scope of this thesis. Nevertheless, some heuristics specifically designed to deal with variability models are available [Men09, MT98, NW07].

2.5 Data structures

Following the directions given by Bryant [Bry86], a data structure to represent BDDs which will be used throughout this dissertation is going to be defined. In particular, the following data structures represent a BDD that has m nodes and encodes a boolean formula with n variables.

- The variable ordering used to synthesize the BDD is represented by an array declared as follows:

```
var_ordering: array[0..n-1] of string
```

- Each node is represented by a record declared as follows:

```
type node = record
    index: 0..n
    low, high: node
    mark: boolean
end
```

Where:

1. *index* is the index of the variables in the ordering. The terminal nodes of the BDD (i.e., 0 and 1) have index n.
2. *low* and *high* are the low and high node successors.
3. *mark* is used to mark which nodes have been visited during a traversal of the graph. As we will see, all our algorithms are called at the top level with the root node as argument and with the mark fields of the nodes being either all true or all false. They then systematically visit every node in the graph by recursively visiting the subgraphs rooted by the two children *low* and *high*. As they visit a node, they complement the value of the *mark* field, so that they can later determine whether a child has already been visited by comparing the two marks.

- The BDD is represented by an array declared as follows:

```
bdd: array[0..m] of node
```

The terminal nodes of the BDD, 0 and 1, are stored at positions 0 and 1 of the *bdd* array, respectively.

For instance, Tables 2.2 and 2.3 represent the content of *bdd* and *var_ordering* for the BDD in Figure 2.2, respectively.

position	index	low	high	mark
0	6	nil	nil	false
1	6	nil	nil	false
2	5	1	0	false
3	4	2	0	false
4	3	3	0	false
5	3	1	3	false
6	2	4	5	false
7	1	6	0	false
8	0	0	7	false

Table 2.2: Content of the *bdd* array for Figure 2.2

position	content
0	"f_1"
1	"f_2"
2	"f_3"
3	"f_4"
4	"f_5"
5	"f_6"

Table 2.3: Content of the *var_ordering* array for Figure 2.2

2.6 Summary

In this chapter, a common way to represent SPLs, called variability models, have been introduced. Moreover, the most common approaches to reason on SPLs have been reviewed.

As we have seen, this process is usually carried out using logic engines. Therefore, the most popular logic engines for reasoning on variability models: SAT solvers and BDDs, have been presented. Finally, this chapter has concluded by providing the data structure to store BDDs, which will be used in the rest of this thesis.

CHAPTER

3

Related Work

This chapter is divided into two well-differentiated parts. The first one summarizes the state-of-the-art on SPL literature. To do so, a bibliometric analysis has been carried out by using a science mapping bibliometric technique in Section 3.1. The main goal of this analysis is to identify which are the most important topics in the SPL research field and their evolution over time. From this analysis, we conclude that the automated analysis of variability models and product derivation are two of the most researched topics in the area for the period 2010-2014. Therefore, in the second part of this chapter, we focus on these important topics. In particular, both of them are reviewed in detail in Sections 3.2 and 3.3, respectively. The chapter is concluded by reviewing the most common approaches to these topics.

3.1 Science Mapping Analysis

The goal of this section is to analyze, using a science mapping analysis, the literature on SPL for the last twenty years. From this analysis, the main topics and trends of this research area are provided. In particular, we provide information regarding the following issues: the most impacting papers for a given topic along a certain period of time, the main topics in the area and their evolution over time. To achieve the above goals, 2845 bibliographic records

retrieved from ISI Web of Science (ISI-WoS) and Scopus have been processed by means of a technique called co-word analysis.

3.1.1 Methodology

This section describes in detail the process used to carry out the science mapping analysis. In particular, we have followed the workflow proposed in [BCB03, CLHHVH11b], which is composed of the following steps:

1) Data retrieval. The data processed in this section come from ISIWoS and Scopus, which are the most reliable bibliographic databases at the moment [GJGJMO14, VG09]. On September 2015, the following query[1] was made on Scopus and the ISIWoS Core Collection for the time span 1995-2014:

```
TOPIC =
  "software product line*" or (
    (
      "product line*" or "mass customization" or "product famil*" or
      "program famil*" or "software factor*" or "product platform*"
    ) and (
      "domain engineering" or "application engineering" or
      "feature model*" or "feature diagram*" or "decision model*" or
       "decision diagram*" or (software and variabilit*)
    )
  )
```

The Venn diagram in Figure 3.1 depicts the number of publication records provided by each database.

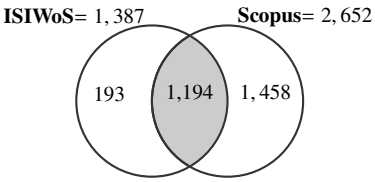


Figure 3.1: Number of records retrieved from ISIWoS and Scopus

[1] the asterisk pattern character means zero to many characters; it is used in our query to catch the noun plurals

2) Data aggregation. The scatter plot in Figure 3.2 shows the number of citations of the records common to ISIWoS and Scopus, including the corresponding regression line as well. ISIWoS has a more selective procedure to include bibliographic references than Scopus. Therefore, Scopus provides more records than ISIWoS, and the citation counts tend to be higher as well.

The Pearson's correlation coefficient of the citation counts is 0.87. So, the information provided by both databases is rather consistent.

To combine the records, the citation count for the common records was computed as the maximum of the citations given by ISIWoS and Scopus.

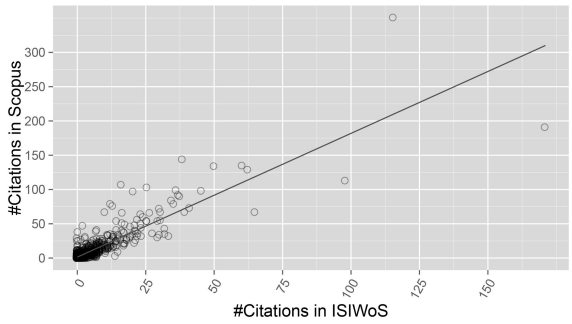


Figure 3.2: Number of citations for publications common to ISIWoS and Scopus

3) Preprocessing. Data retrieved from bibliographic databases usually have errors. For instance, references may be duplicated, authors' names may appear in different ways, etc. So, it is necessary to preprocess the data before carrying out any analysis.

To track the evolution of the SPL research area, we have used an approach that requires analyzing publication keywords: Co-Word Analysis. Hence, we have performed a laborious preprocessing procedure to:

1. *Correct invalid citations*; e.g., the technical report [BC05] appears cited in the raw data gathered from ISIWoS as "Bachman F., 2005, CMUSEI2005 TR012" and "Bachmann F., 2005, CMUSEI2005TR012".
2. *Standardize keywords*. From the ISIWoS records, a set of 2,000 keywords was available: 1,667 were authors' keywords and 333 were words provided by ISIWoS *KeyWords Plus* (index terms created by Thomson Reuters from significant,

frequently occurring words in the title field of a cited article references). The Scopus records included a set of 9,308 keywords. As Figure 3.3 summarizes, the initial aggregated set of 11,308 keywords was progressively reduced by applying the following steps:

(a) Keywords were converted to uppercase, leading and trailing white-spaces were removed, and inner white-spaces were replaced by the character '-'. After that, the repeated keywords were removed and plurals were grouped.

(b) Keywords useless to identify research topics inside the SPL area were discarded. For example, SOFTWARE-PRODUCT-LINE, PRODUCT-FAMILY, SOFTWARE-ENGINEERING, etc. are applicable to all the records and thus they cannot be used to distinguish particular topics.

(c) Keywords were grouped according to their meaning. To improve the interpretability of the co-word analysis results, the set of keywords was reduced by grouping those words that refer to the same topic. For instance, STAGED-CONFIGURATION, AUTOMATED-CONFIGURATION, etc. were grouped as PRODUCT-DERIVATION.

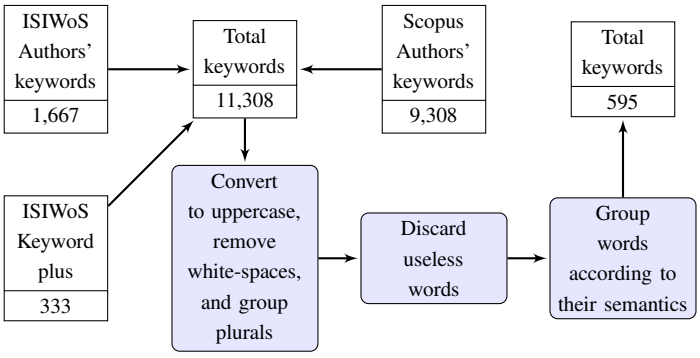


Figure 3.3: Summary of the keyword standardization

4) Analysis. Finally, bibliographic data have been examined using science mapping methodology. The next section provides an overview of the results of this analysis. For a detailed description of the work performed we refer to the reader to [HPMFA+16].

To preprocess and analyze the bibliographic data, the open source software tool SciMAT [CLHHVH12] which is freely available at: http://sci2s.ugr.es/scimat/, has been used.

3.1.2 Science Mapping and Longitudinal Study for Software Product Lines

Science mapping has been used to get three kind of maps: *strategic diagrams*, *thematic networks*, and *maps of conceptual evolution*.

First of all, to analyze the structure and dynamics of the SPL research area, the bibliographic data were divided into four consecutive periods of time: 1995-1999, 2000-2004, 2005-2009, and 2010-2014. To detect the main topics of each period, the algorithm of simple centers [CCL91] was run using the parameters summarized in Table 3.1. Out of a total of 2,845 documents, 23 were published in Period 1, 208 in Period 2, 902 in Period 3, and 1,712 in Period 4. Note that periods 1 and 2 have rather less documents than periods 3 and 4. As recommended by [CMK98, CLHHVH11a], parameters $\min_{occurrences}$ and $\min_{co\text{-}occurrences}$ were reduced for those periods to accommodate the lesser volume of data.

Parameter	Period 1 (1995-1999)	Period 2 (2000-2004)	Period 3 (2005-2009)	Period 4 (2010-2014)
$\min_{occurrences}$	2	3	4	4
$\min_{co\text{-}occurrences}$	2	3	3	3
$\min_{keywords}$	2	2	2	2
$\max_{keywords}$	5	5	5	5

Table 3.1: Parameters for simple centers algorithm

3.1.2.1 Strategic diagrams

Strategic diagrams offer a global representation of the simple center algorithm outcomes. In this kind of map, the detected topics in each period are arranged according to their centrality and density. Topics are depicted as nodes whose volume is proportional to the number of associated publications. The role that a topic plays in a research area is characterized according to the quadrant where it is placed in the map [CCL91, TR91]:

Top-right quadrant includes topics both well developed and important for the research field. Due to their high centrality and density, those topics are usually known as *motor topics*.

Bottom-right quadrant contains important but weakly structured topics. They could be transversal topics or underdeveloped topics of considerable significance for the entire research field [CLHHVH11a].

Bottom-left quadrant has both weakly developed and marginal topics, representing emerging or disappearing topics.

Top-left quadrant includes topics that have well developed internal links but unimportant external ties. Since those topics are internally well structured, they indicate that a constituted social group is active in them. Nevertheless, they should be considered peripheral to the work being performed in the global research field [He99].

From here on we refer to the above quadrants as follows: top-right as q_1, bottom-right as q_2, bottom-left as q_3 and top-left as q_4.

Figure 3.4 displays the detected topics on SPL literature for all the periods.

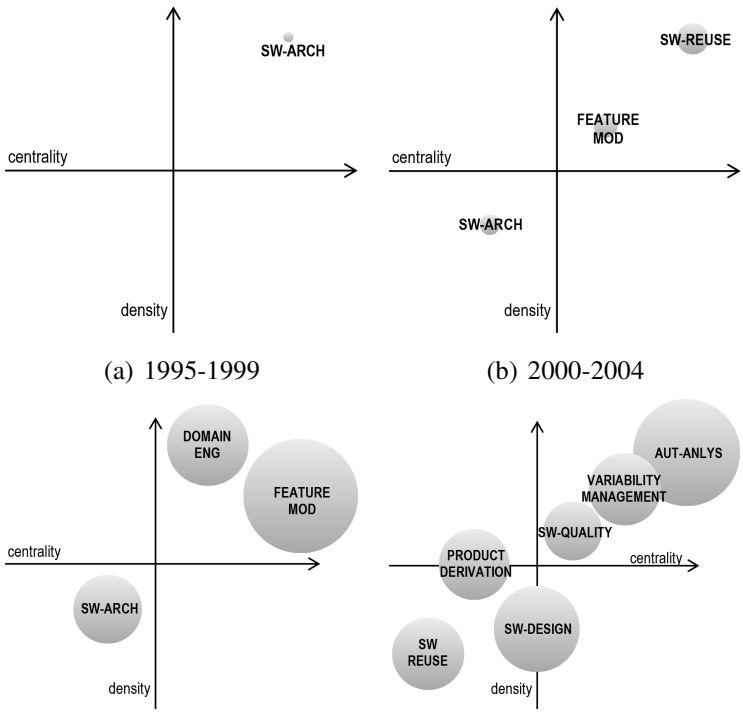
3.1.2.2 Thematic networks

A particular science mapping approach, known as co-word analysis, has been used to identify the main topics of a scientific field and their interrelationship. It measures the association strengths of terms representative of the publications in the field by analyzing the co-ocurrence frequency of pairs of keywords.

We use again the simple centers clustering algorithm jointly with the equivalence index to identify research topics by looking for groups of strongly linked keywords [CLR86, CCL91, KUU10]. In such a way, each detected topic is modeled by a cluster of interrelated keywords known as a thematic network.

3.1.2.3 Thematic network for Period 1

Period 1 includes just one topic: Software Architecture (SW-ARCH). Figure 3.5 shows the structure of its associated cluster as a graph. Keywords are represented as nodes whose volume is proportional to their associated number of publications. The equivalence index



(a) 1995-1999 (b) 2000-2004

(c) 2005-2009 (d) 2009-2014

Figure 3.4: Strategic diagrams for periods 1, 2, 3, and 4

of two keywords *A* and *B* is depicted by the thickness of the edge that links *A* to *B*. Table 3.2 summarizes the performance of the period, i.e., the number of publications on topic SW-ARCH, the number of citations they received in total and on average, the H-index for SW-ARCH, and those publications that should be considered as classics for SW-ARCH.

According to Figure 3.5 and Table 3.2, the SPL field comes from a confluence of research on SW-ARCH, Domain Engineering (DOMAIN-ENG), Requirements Engineering (REQUIREMENTS-ENG), and Object Orientation (OBJECT-ORIENTATION). From 1995 to 1999 there was a reduced number of published papers with a high number of citations on average, shaping a topic with high density and centrality that acted as motor for the research carried out in the subsequent periods.

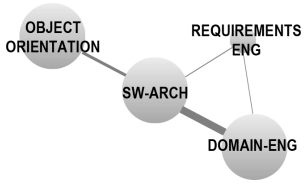


Figure 3.5: Thematic network for Period 1 (1995-1999)

Topic	#Publications	#Citations	Average Citations	H-index	H-core
SW-ARCH	13	157	12.08	7	[CHW98, SB99, Lam98, DKO+97, RS98] [KKLL99, LHM+98]

Table 3.2: Topic Performance for Period 1 (1995-1999)

3.1.2.4 Thematic network for Period 2

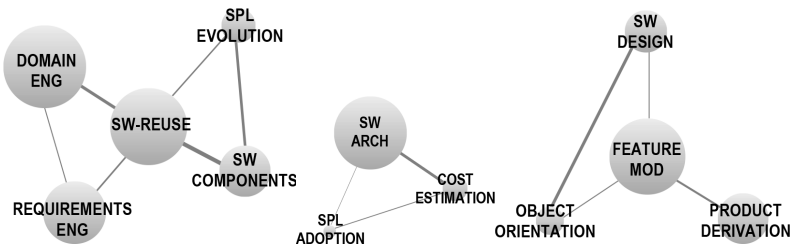
Period 2 encompasses three topics: Software Reuse (SW-REUSE), SW-ARCH, and Feature Modeling (FEATURE-MOD). Figure 3.6 shows the structure of their associated clusters. Table 3.3 sums up the performance of the period.

According to Figure 3.4.b, SW-REUSE and FEATURE-MOD replaced SW-ARCH as motor topic. In particular, SW-ARCH fell into q_3, becoming a declining topic.

From 2000 to 2004, most published papers where about SW-REUSE, a topic which bound together research on DOMAIN-ENG, REQUIREMENTS-ENG, SPL Evolution, SW-REUSE, and Software Components.

Topic	#Publications	#Citations	Average Citations	H-index	H-core
SW-REUSE	68	678	9.98	12	[Sch02, Nor02, LK04, ADH+00, vGBS01, PKS04] [Lut00, TNK04, ZJY03, GS03, BBMY04, GA01]
FEATURE-MOD	52	562	10.81	12	[CHU04, LKL02, BPSP04, LK04, PKS04, GA01] [Mat04, SPR04, Gri00, HSVM00, DSB04, CPRS04]
SW-ARCH	47	433	9.21	9	[SB02, BCM+04, Sch02, BHJ+03, Kru02] [GS03, BBMY04, SPR04, FV03]

Table 3.3: Topic Performance for Period 2 (2000-2004)



(a) SW-REUSE (b) SW-ARCH (c) FEATURE-MOD

Figure 3.6: Thematic networks for Period 2 (2000-2004)

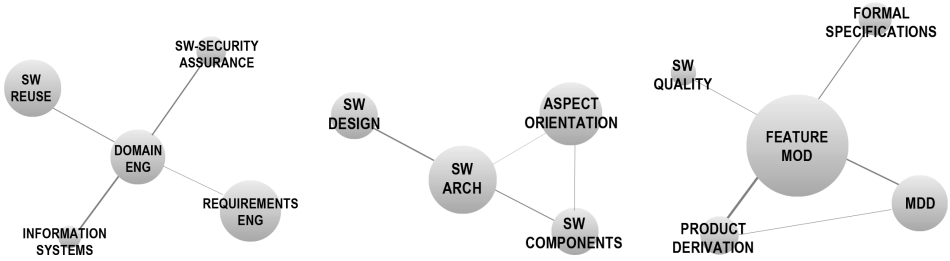
In addition, FEATURE-MOD emerged as a motor topic that grouped research on Software Design, Feature Modeling, Object Orientation, and Product Derivation on SPLs.

Notice that the relation between papers and topics is $N : M$, that is, a paper may talk about many topics, and a topic may be covered by more than one paper. For instance, reference [PKS04] is a core paper of both SW-REUSE and FEATURE-MOD (see Table 3.3).

3.1.2.5 Thematic network for Period 3

Period 3 has three main topics: DOMAIN-ENG, SW-ARCH, and FEATURE-MOD. Figure 3.7 shows the structure of their associated clusters. Table 3.4 summarizes the performance of the period.

From 2005 to 2009, FEAUTURE-MOD became the most important topic in SPL research, not only in terms of quantity (being the topic about which more papers were published), but also in terms of quality (being the topic with the most citations).



(a) DOMAIN-ENG (b) SW-ARCH (c) FEATURE-MOD

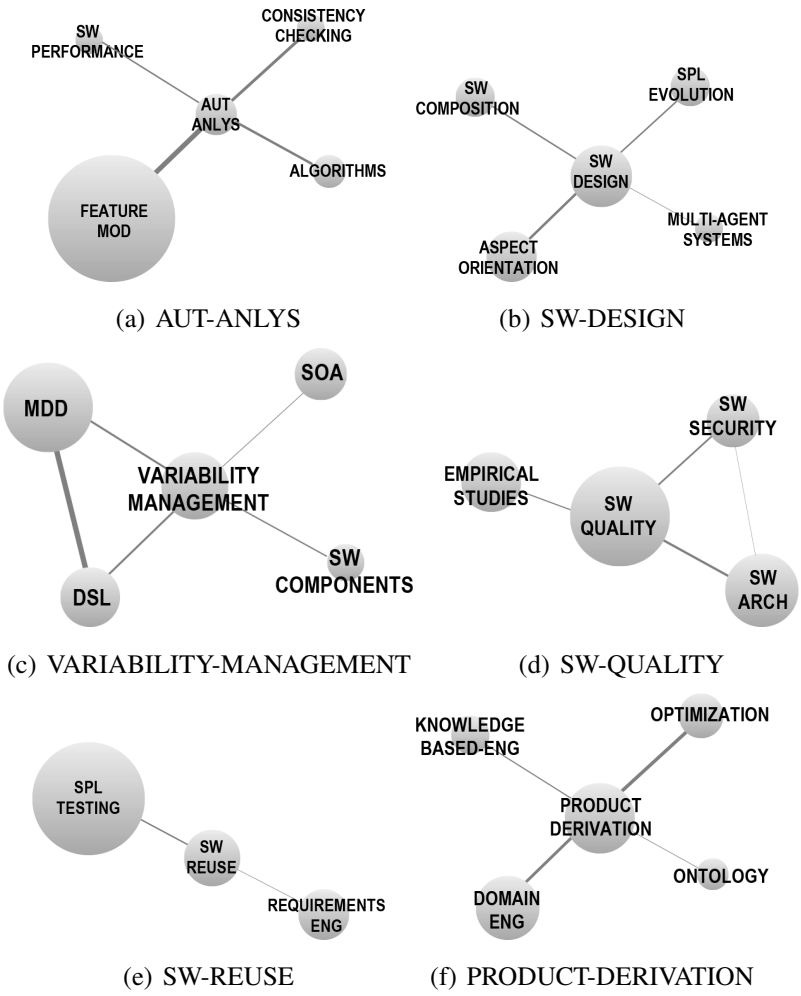
Figure 3.7: Thematic networks for Period 3 (2005-2009)

Topic	#Publications	#Citations	Average Citations	H-index	H-core
DOMAIN-ENG	198	1,865	9.42	12	[MYC05, vO05, AMS07, LDL07, RBS09] [RW07, KPSY07, AC08, KG09, BLP05] [WHG+09, PL05]
FEATURE-MOD	282	2,923	10.36	23	[CHE05a, CHE05b, Bat05, SHTB07, BTRC05] [SHT06, CW07, TBKC07, ATLS08, SvGB05] [CP06, MPH+07, TBK09, VG07, KAB07] [BBRC06, HKW08, MMLP09, AGM+06, MBC09] [LDL07, CZZM05, TBD+08]
SW-ARCH	162	1,180	7.28	11	[KAK08, HHPS08, FCS+08, ATLS08, SvGB05] [VG07, KAB07, FUB06, vO05, TBD07, LHBC05]

Table 3.4: Topic Performance for Period 3 (2005-2009)

3.1.2.6 Thematic network for Period 4

Period 4 displays an explosion of topics. It has twice as many topics as Periods 2 and 3: Automated Analysis of Variability Models (AUT-ANLYS), SW-DESIGN, VARIABILITY-MANAGEMENT, SW-QUALITY, SW-REUSE, and PRODUCT-DERIVATION. Figure 3.8 shows the structure of their associated clusters. Table 3.5 summarizes the performance of the period.



(a) AUT-ANLYS (b) SW-DESIGN

(c) VARIABILITY-MANAGEMENT (d) SW-QUALITY

(e) SW-REUSE (f) PRODUCT-DERIVATION

Figure 3.8: Thematic networks for Period 4 (2010-2014)

According to Figure 3.4.c, from 2010 to 2014, three topics have played a motor role: AUT-ANLYS, VARIABILITY-MANAGEMENT, and SW-QUALITY. As Figure 3.8.a shows, FEATURE-MOD is the keyword with most associated publications (not only for its cluster,

Topic	#Publications	#Citations	Average Citations	H-index	H-core
AUT-ANLYS	264	1,451	5.50	13	[BSRC10, CHS+10, SLB+11, AKGL10, CHSL11] [DGR11, BSL+10, CBH11, SHBRC11, DDH+13] [MC10, ACLF09, ACLF13]
SW-DESIGN	213	796	3.74	11	[SBDT10, BBS10, AKGL10, CB11, LBL11] [KGM10, DGRN10, SRC+12, NTS+11, TDR+11] [URG10]
VARIABILITY MANAGEMENT	182	583	3.20	9	[SBDT10, LAL+10, CB11, DGRN10, TDR+11] [RB10, LMN10, ZSS+10, GWT+14]
SW-QUALITY	146	310	2.12	5	[BG11, MFMP10, RFBRC+12, MAaI12, MGH+11]
PRODUCT DERIVATION	175	524	2.99	9	[DGR11, MC10, RGD10, WBS+10, LK10] [GWW+11, DYS11, BBG+10, SIMA13]
SW-REUSE	186	693	3.72	10	[ER11, dMSNdCMM+11, SHBRC11, ANAc10] [OMR10, HBG11, PSK+10, KBK11, POS+12] [UKB10]

Table 3.5: Topic Performance for Period 4 (2010-2014)

but for all the clusters of Period 4). The high equivalence index between FEATURE-MOD and AUT-ANALYS reflects the works on formal methods and algorithms to support the automated analysis of feature diagrams. According to Figure 3.8.c, VARIABILITY-MANAGEMENT binds together research on Software Components, Domain Specific Languages (DSL), Model Driven Development (MDD), and Service-Oriented Architecture (SOA). Finally, as Figure 3.8.d shows, SW-QUALITY groups research on SW-ARCH, Software Security, and Empirical Studies where the SPL paradigm is validated.

According to Figures 3.6.c and 3.7.c, the derivation of particular products from a SPL platform has been strongly associated to research on feature modeling from 2000 to 2009. Nevertheless, Figure 3.8.f and the position of PRODUCT-DERIVATION in the strategic diagram in Figure 3.4.c show that this subject is gaining independence and, in the current period, it binds together research on Ontologies, Domain Engineering, Optimization problems and Knowledge-Based Engineering.

In Figure 3.4.d, SW-DESIGN is halfway between q_3 and q_2. Jointly with Figure 3.8.b, it could be interpreted as that, from 2010 to 2014, a considerable effort has been made to apply general research on Software Development (Software Design, Aspect Orientation, Multi-Agent Systems, Software Composition) to SPLs.

Finally, it is important to highlight that the most performing topic in the last years has been the automated analysis of feature models, according to Table 3.5.

3.1.2.7 Longitudinal Analysis of Topic Evolution

The analysis of co-citation clusters over consecutive periods of time can be used to track the emergence and growth of research areas, and predict their near term change [Sma06]. In particular, the movement of topics throughout the quadrants of strategic diagrams in successive periods provides information regarding their evolution. For instance, when a topic passes from q_3 to q_1, it means that an emerging topic has been developed and become central for the field.

The inclusion index between two sets S and S' of words is computed as the number of common words to both sets divided by the number of words of the smallest set.

Figure 3.9 describes topic evolution. In this figure, topics are represented by nodes (vertically aligned by periods) whose volume is proportional to the number of publications they have associated. The inclusion index of two topics T and T' is represented by the thickness of the edge that links T to T'. Whenever the keyword that gives name to T' is also a keyword in cluster T, the edge is represented by a solid line. Otherwise, the edge is depicted as a dashed line.

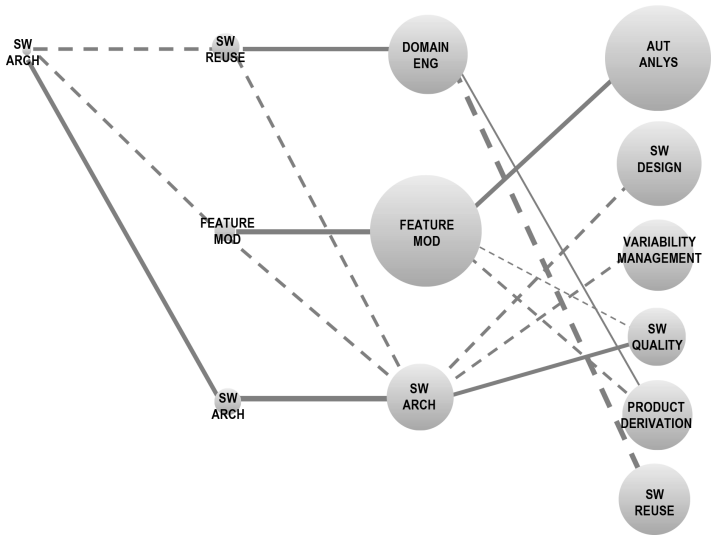


Figure 3.9: Topic evolution between periods

According to Figure 3.9, the evolution of the SPL area has behaved properly, growing smoothly and continuously. The number of publications has increased in each period.

Moreover, no topic has abruptly disappeared; on the contrary, original topics have progressively been consolidated and, especially in the last five years, they have been branched out to more specific research themes.

Feature modeling has been the most important topic in the whole SPL research area. It presents the best evolution behavior and the best quality indicators (see the H-index of FEATURE-MOD, AUT-ANLYS, SW-QUALITY, and PRODUCT-DERIVATION in Tables 3.3, 3.4, and 3.5). Research on software architectures and software reuse has also been essential for the development of the area. In particular, the following conclusions may be drawn from Figures 3.4 and 3.9:

1. According to the movement of SW-ARCH through the quadrants of the strategic diagrams in Figure 3.4, SW-ARCH was the initial motor topic, which inspired research on SPLs. Nevertheless, it moved from q_1 in Period 1, to q_3 in Periods 2 and 3, becoming a peripheral topic.

2. From Period 2, feature modeling has behaved as an essential motor topic for the development of the SPL field. As the size of the nodes in Figure 3.9 shows, the number of publications on this topic has grown dramatically. In the last five years research on feature modeling has spread out to several subareas: AUT-ANLYS, PRODUCT-DERIVATION, and SW-QUALITY (two of them playing a motor role).

3. A main goal for the SPL paradigm is shifting from opportunistic to systematic reuse of software. Accordingly, SW-REUSE and DOMAIN-ENG have worked as motor topics from 2000 to 2009. However, nowadays SW-REUSE may be considered a peripheral topic.

To sum up, according to the above explanation the Automated Analysis of Variability Models and Product Derivation have been two of the most researched topics in the area for the period 2010-2014. Therefore, in the remainder of this dissertation, we focus on these important topics.

3.2 Automated Analysis of Variability Models

As product lines grow and evolve, variability models become bigger and harder to understand. So, there is a need for an automated mechanism that provides information regarding which role each feature plays according to the variability model. This automated process is carried out by means of analysis operations. Several analysis operations have been reported in the literature [BSRC10, Boe11], which are put into practice by using the process mentioned in Section 2.2. This process, as already mentioned, is composed of the following two steps. First, models are translated into equivalent propositional logic formulas. Then, the logic representation is processed using off-the-self logic engines, such as SAT-solvers and BDDs. In general, the engines are black-box reused, that is, without seeing, knowing nor modifying the engine internals. In practice, this approach is efficient enough for some important operations [LGRC15, PLP11], such as checking if a given combination of features is valid (i.e., the combination does not violate any of the feature dependencies), or detecting void models (i.e., models that, due to feature dependencies, do not represent any valid product). However, we show that black-box reuse is inadequate for other interesting operations because it imposes calling many times the logic engine. For those operations, it is more efficient to glass-box reuse the BDD libraries. In particular, in this dissertation we focus on the following hard operations:

1. Computation of feature *commonality*. Let $\mathcal{P}$ and $\mathcal{P}_f$ denote the sets all of valid products and products that include a particular feature f, respectively. Then, the commonality of a feature f is $\frac{\#\mathcal{P}_f}{\#\mathcal{P}}$. As discussed in [FAHCC14], feature commonality provides descriptive statistics to account for the standardization/parameterization balance of variability models, and it is useful to improve the accuracy of product line economic models to estimate the Return On Investment (ROI).

2. Detection of core and dead features, i.e., those features that appear in all or none of the valid products [BSRC10]. Note that those features are absolutely essential or dispensable in a product line, respectively. It can be seen as a particular case of the above measure, where a feature is core or dead if its commonality is 1 or 0, respectively.

3. Computing the feature *impact* and *exclusion sets*. The *impact set* of a feature f is composed of all the features f' that require f to be enabled whenever they are included

in a valid product, i.e.,

$$\text{Impact Set}(f) = \{f' \cdot f' \Rightarrow f\}$$

The *exclusion set* of a feature f is composed of all the features f' that are required to be disabled whenever f is included in a valid product, i.e.,

$$\text{Exclusion Set}(f) = \{f' \cdot f \Rightarrow \neg f'\}$$

Impact/exclusion sets are useful to quantify how necessary/incompatible are the features in a model [Boe11].

For the sake of clarity, we repeat again the small example introduced in Section 2.1.

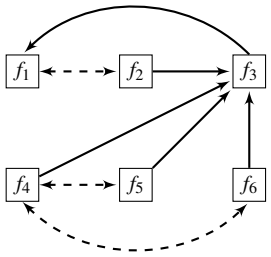


Figure 3.10: A variability model example (rep. Figure 2.1)

	Valid Products
1	$f_1, \neg f_2, \neg f_3, \neg f_4, \neg f_5, \neg f_6$
2	$f_1, \neg f_2, \ f_3, \neg f_4, \neg f_5, \neg f_6$
3	$f_1, \neg f_2, \ f_3, \ f_4, \neg f_5, \neg f_6$
4	$f_1, \neg f_2, \ f_3, \neg f_4, \ f_5, \neg f_6$
5	$f_1, \neg f_2, \ f_3, \neg f_4, \neg f_5, \ f_6$
6	$f_1, \neg f_2, \ f_3, \neg f_4, \ f_5, \ f_6$

Table 3.6: Valid products for Figure 3.10 (rep. Table 2.1)

In the following subsections, we review the above operations and show how they are usually computed. The reminder of this section is structured as follows: first, the most common approaches to compute feature commonalities are reviewed in Subsection 3.2.1. Then, the most common approaches to identify absolutely essential and dispensable features are reviewed in Subsection 3.2.2. Finally, this section concluded by reviewing the most common approaches to compute highly required and incompatible features in Subsection 3.2.3.

3.2.1 Measures to Compute Feature Commonality

As mentioned before, the *commonality* of a feature f is $\frac{\#\mathcal{P}_f}{\#\mathcal{P}}$, where $\mathcal{P}$ and $\mathcal{P}_f$ denote the sets all of valid products and products that include a particular feature f, respectively. As an example, according to Table 3.6, f_3 is included in five of the six valid products, so the commonality of f_3 is $\frac{5}{6}$. Table 3.7 summarizes feature commonalities for Figure 3.10. In the literature, the commonality of a feature is also referred to as its probability. In this thesis, both terms are used as synonyms.

Feature	f_1	f_2	f_3	f_4	f_5	f_6
Pr(f)	1	0	$\frac{5}{6}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$

Table 3.7: Feature commonalities for Figure 3.10

Two main approaches are found in the literature to compute the commonality of a feature: the former is based on *#SAT* whereas the latter is performed by using the sat_count function provided by BDD libraries.

1. The first approach consists of repeatedly calling to #SAT model counter. A #SAT model counter is a program that computes the number of satisfying assignments of a boolean formula. Some examples of these kinds of tools are cachet[1], relsat[2], sharpSAT[3], among others. Let ψ the logic representation of the model, Kübler et al. [KZK10] propose computing $\#\mathcal{P}$ and $\#\mathcal{P}_f$ by calling a #SAT counter with ψ and $f \wedge \psi$ as input parameters, respectively.

   In [FAHCC14], it has been experimentally shown that calling repeatedly a #SAT solver only scales for small variability models. Note that while the SAT problem is known to be NP-complete [Coo71], it is widely believed that the #SAT problem is even harder [BHvM+09]. To overcome such problem [FAHCC14] propose an alternative approach to compute commonality, which takes advantage of the tree structure of feature models (a popular notation for variability modelling). Unfortunately, this approach only works for feature models.

[1] http://www.cs.rochester.edu/u/kautz/Cachet/index.htm
[2] http://www.bayardo.org/resources.html
[3] https://sites.google.com/site/marcthurley/sharpsat

2. The second approach consists of repeatedly calling the sat_count function in a BDD library. This function returns the number of satisfying assignments of a boolean formula. Let ψ the logic representation of the model. The commonality of a feature f can be computed by calling a sat_count function with ψ and $f \wedge \psi$ as input parameters, respectively.

Algorithm 1 shows how the above approaches proceed.

Algorithm 1: get_feature_commonalities *(Commonality brute force approach)*

```
1  Input boolean formula ψ; set F of all features
2  Output a list including all commonality features
3  var feature_commonalities: list; count_f, count: int;
4  begin
5      feature_commonalities = {}
6      count = sat_count(ψ)
       // or by using a #SAT solve(ψ)
7      forall f ∈ F do
8          count_f = sat_count(ψ ∧ f)
           // or by using a #SAT solve(ψ ∧ f)
9          feature_commonalities.insert(count_f/count)
10     return feature_commonalities
```

3.2.1.1 Computational cost

Commonality features	
#SAT solver	sat_count apch.
$NP - complete$	$O(m.n)$

Table 3.8: Computational complexity for Algorithm 1.

Let m the number of nodes of the BDD and n the number of variables (features) of the boolean formula, Algorithm 1 requires calling the sat_count function $n + 1$ times. In addition, as sat_count has complexity $O(m)$, this approach has a complexity $O(m.n)$. Table 3.8 summarizes the computational cost for computing commonality features in a variability model.

3.2.2 Measures to Identify Absolutely Essential and Dispensable Features

Features in a product line usually have varying degrees of importance. Some features may be highly demanded by the market and so most products should include them. Other features may become dispensable as the market demand evolves. In particular, features that appear in all or none of the valid products are absolutely essential or dispensable, respectively [vDK02, PHRCT06, ZMZ06, BSTRC07, ZYZJ08, TBD+08, MWC09, TC09].

According to Table 3.6, f_1 is included in all products and so it is a core feature. As f_2 is missing in every products, it is a dead feature. The following three main approaches are found in the literature to identify them:

- The first one consists of repeatedly calling a SAT-solver to enumerate all valid products, and then inspecting the products to identify what features appear in all and none of them, respectively. Unfortunately, the number of products represented by a variability model grows exponentially with the number of features. For instance, a model with 260 independent optional features (i.e., features without dependencies with the remaining ones) represents more combinations than the number of atoms in the observable universe [FKC12]. So, this method, which is used for instance by the FAMA tool [TBRC+08], has serious scalability limitations[1].

- The second one proceeds as follows: let ψ a boolean formula encoding a variability model, a feature f is core iff $\psi \rightarrow f$ is a tautology or, in other words, iff $\neg f \wedge \psi$ is unsatisfiable [HR04]. Similarly, f is dead iff $f \wedge \psi$ is unsatisfiable. Thus, all core and dead features can be detected by repeatedly calling a SAT-solver (or a BDD engine), which is the method proposed in [ZK10, Tar13]. This approach is described by Algorithm 2. As Lesta et al. [LSW15] note, this approach can be improved for the case of dead features by reducing the number of checks as follows: whenever $f \wedge \psi$ is satisfiable, the SAT-solver not only returns "satisfiable", but also a valid assignment; all features f' that are true in that assignment cannot be dead (since, at least, they are

[1] Note that, this operation can also be computed using all_sat function provided by BDD libraries. However, as all_sat function has a complexity $O(2^m.n)$, where m and n are the number of nodes and variables (features), respectively, it has the same scalability limitations.

included in the product returned by the solver). Therefore, checking the satisfiability of $f' \wedge \psi$ can be avoided.

Although Lesla et al.'s work does not deal with core feature detection, their reasoning can be applied to that kind of features as well: whenever $f \wedge \psi$ or $\neg f \wedge \psi$ are satisfiables, none of the features f' that are false in the satisfying assignment returned by the solver can be core. This approach is described by Algorithm 3.

- The third one consists of computing commonality features[1]. In this case, a feature f is core iff the number of valid products that include f is the same as the number of valid products represented by the model. Similarly, a feature f is dead iff the number of valid products that include f is zero [vDK02, PHRCT06, ZMZ06, BSTRC07, ZYZJ08, TBD+08, MWC09, TC09, HFACC11]. This approach, where ψ is the logic representation of the model with n features, is described by Algorithm 4. The key here is to repeatedly call the sat_count function, once for each feature f, to get the number of satisfying assignments of $\psi \wedge f$.

3.2.2.1 Computational cost

Let m the number of nodes of the BDD and n the number of variables (features) of the boolean formula. The straightforward approach (Algorithm 2) and Lesla et al.'s approach (Algorithm 3) require calling apply function n times, as that function has complexity $O(m)$, these approaches have complexity $O(m.n)$. Finally, as we discussed in Section 3.2.1, Algorithm 4 has complexity $O(m.n)$. Note that, although the time complexity for Algorithm 3 is the same that for Algorithm 4, in practice it often saves steps and thus runs faster. Table 3.9 summarizes the complexities for Algorithms 2, 3 and 4.

[1] See Sections 3.2.1 and 3.2.1 for a complete description of feature commonality and reviewing the most common approaches to compute feature commonalities.

Core & dead features		
Strainghtforward apch.	Lesla et al.'s apch.	Commonality apch.
$O(m.n)$	$O(m.n)$	$O(m.n)$

Table 3.9: Computational complexity for Algorithms 2, 3 and 4.

Algorithm 2: get_core_and_dead_features *(straightforward core/dead approach)*

```
1  Input boolean formula ψ; set ℱ of all features
2  Output two lists including all core and dead features
3  var core_features, dead_features: list;
4  begin
5      core_features = {}
6      dead_features = {}
7      forall f ∈ ℱ do
8          if ψ ∧ f ≠ bdd_false then
9              if ψ ∧ ¬f == bdd_false then
10                 core_features.insert(f)
11         else if ψ ∧ ¬f ≠ bdd_false then
12             dead_features.insert(f)
13     return core_features, dead_features
```

Algorithm 3: get_core_and_dead_features *(Lesla's core/dead apch.)*

```
1  Input boolean formula ψ; set ℱ of all features
2  Output two lists including all core and dead features
3  var core_features, dead_features, true_features, true_features: list;
4  begin
5      core_features = {}
6      dead_features = {}
7      assignment = sat_one(ψ) // get a satisfying feature assignment
8      true_features = get_features_with_true_value(assignment)
9      false_features = get_features_with_false_value(assignment)
10     forall f ∈ ℱ do
11         if f ∈ true_features then
12             if ψ ∧ ¬f == bdd_false then
13                 core_features.insert(f)
14         else if ψ ∧ f == bdd_false then
15             dead_features.insert(f)
16     return core_features, dead_features
```

Algorithm 4: get_core_and_dead_features *(Commonality core/dead approach)*

```
1  Input boolean formula ψ; set ℱ of all features
2  Output two lists including all core and dead features
3  var core_features, dead_features: list; count_f, count: int;
4  begin
5      core_features = {}
6      dead_features = {}
7      count = sat_count(ψ)
8      forall f ∈ ℱ do
9          count_f = sat_count(ψ ∧ f)
10         if count == count_f then
11             core_features.insert(f)
12         else if count_f == 0 then
13             dead_features.insert(f)
14     return core_features, dead_features
```

3.2.3 Measures to Identify Highly Required and Incompatible Features

Valuable information can be extracted from variability models merely by looking at how features interact with each other in all valid products[ADCBZ09, DCB10, Boe11]. On the one hand, if a feature is required by many others, it can be said that it is highly necessary. On the other hand, if a feature excludes many others, it is highly incompatible.

In particular, Boender [Boe11] provides the following measures to quantify feature necessity and incompatibility:

1. The *impact set* of a feature f is composed of all the features f' that require f to be enabled whenever they are included in a valid product, i.e.,

$$\text{Impact Set}(f) = \{f' \cdot f' \Rightarrow f\}$$

2. The *exclusion set* of a feature f is composed of all the features f' that are required to be disabled whenever f is included in a valid product, i.e.,

$$\text{Exclusion Set}(f) = \{f' \cdot f \Rightarrow \neg f'\}$$

3. The *necessity* and *incompatibility* of a feature f are the cardinal of its impact and exclusion set, respectively, divided by the total number of features $\#\mathcal{F}$, i.e.,

$$\text{Necessity}(f) = \frac{\#\text{Impact Set}}{\#\mathcal{F}},\ \text{Incompatibility}(f) = \frac{\#\text{Exclusion Set}}{\#\mathcal{F}}$$

For instance, Table 3.10 summarizes feature co-ocurrences in Figure 3.10. According to Table 3.6, all products that include f_4, f_5, or f_6, also include f_3. So the *impact set* of f_3 is $\{f_3, f_4, f_5, f_6\}$, and the *necessity* of it is $\frac{4}{6}$ (it is required by 4 of the 6 features).

Feature	Impact set	Necessity	Exclusion set	Incompatibility
f_1	$\{f_1, f_3, f_4, f_5, f_6\}$	$\frac{5}{6}$	$\{f_2\}$	$\frac{1}{6}$
f_2	$\emptyset$	0	$\{f_1, f_2, f_3, f_4, f_5, f_6\}$	1
f_3	$\{f_3, f_4, f_5, f_6\}$	$\frac{4}{6}$	$\{f_2\}$	$\frac{1}{6}$
f_4	$\{f_4\}$	$\frac{1}{6}$	$\{f_2, f_5, f_6\}$	$\frac{3}{6}$
f_5	$\{f_5\}$	$\frac{1}{6}$	$\{f_2, f_4\}$	$\frac{2}{6}$
f_6	$\{f_6\}$	$\frac{1}{6}$	$\{f_2, f_4\}$	$\frac{2}{6}$

Table 3.10: Summary of feature co-ocurrences in Figure 3.10

It is interesting to note that essential, dispensable and highly required or incompatible features are interconnected by the following relations, where $\mathcal{F}$ is the set of all features:

1. Impact_Set(f) = $\mathcal{F} \Leftrightarrow f$ is core
2. Exclusion_Set(f) = $\mathcal{F} \Leftrightarrow f$ is dead

The following two main approaches are found in the literature to identify highly required or incompatible features: both of them are based on repeatedly calling the bdd sat_count function. Note that, the second one proceeds in a similar way that Lesla's et al. approach.

- Algorithm 5 shows how the first one proceeds to compute the impact and exclusion sets. It checks the following two conditions for all pairwise combinations of features f and f':

  1. If sat_count($\psi \wedge f$) and sat_count($\psi \wedge f \wedge f'$) coincide, then f belongs to the f' impact set. Similarly, a SAT solver can be used. In such case, if solver($\psi \wedge f$) is satisfiable and solver($\psi \wedge f \wedge \neg f'$) is unsatisfiable then f belongs to the f' impact set respectively.
  2. If sat_count($\psi \wedge f \wedge f'$) is zero then f' belongs to the f exclusion set. With a solver, it is enough to check if solver($\psi \wedge f \wedge f'$) is unsatisfiable.

  The problem remains tractable if one just wants to check whether a feature requires or excludes another. However, if one wants to get the exclusion and impact set for all the features in a variability model, this algorithm requires calling the sat_count function $n \cdot (n-1)$ times. Therefore, if the computation is performed with a BDD, the complexity is $O(m \cdot n^2)$. Since there are reported models in the literature with thousands of features, this algorithm is not feasible [BSL+10, BSL+13].

- Boender [Boe11] proposes Algorithm 6, which is similar to Algorithm 3. It includes the following shortcut: for each feature f, a satisfying assignment a of $\psi \wedge f$ is computed using the sat_one function (line 9), which has time complexity O(m) [Bry86]. If a feature f' is true in a, it cannot belong to f exclusion set (i.e., since there is at least one product that includes both f and f', they cannot be incompatible). So line 18 avoids the unnecessary computation sat_count($\psi \wedge f \wedge f'$) for such f'. Likewise, if a feature f' is false in a, f' cannot be part of f impact set, and so line 22 avoids calling sat_count($\psi \wedge f \wedge f'$). Although the time complexity for Algorithm 6 is the same that for Algorithm 5, in practice it usually saves steps and thus runs faster.

Algorithm 5: get_impact_and_exclusion_sets *(Commonality impact/exclusion apch.)*

```
1  Input boolean formula ψ; set ℱ of all features
2  Output two hash tables including for each feature its impact and exclusion sets
3  var impact_sets, exclusion_sets: hash; count, count': int;
4  begin
5      impact_sets = Hash.new
6      exclusion_sets = Hash.new
7      forall f ∈ ℱ do
8          count = sat_count(ψ ∧ f)
9          if count == 0 then
10             impact_sets[f] = {}
11             exclusion_sets[f] =ℱ
12         else
13             impact_sets[f] = {f}
14             exclusion_sets[f] = {}
15             forall f' ∈ ℱ\{f} do
16                 count' = sat_count(ψ ∧ f ∧ f')
17                 if count' == 0 then
18                     exclusion_sets[f].insert(f')
19                 else if count == count' then
20                     impact_sets[f'].insert(f)
21     return impact_sets, exclusion_sets
```

Algorithm 6: get_impact_and_exclusion_sets *(Boender's apch.)*

```
1  Input boolean formula ψ; set ℱ of all features
2  Output two hash tables including for each feature its impact and exclusion sets
3  var impact_sets, exclusion_sets: hash; count, count': int; true_features, true_features: list;
4  begin
5      impact_sets = Hash.new
6      exclusion_sets = Hash.new
7      forall f ∈ ℱ do
8          count = sat_count(ψ ∧ f)
9          assignment = sat_one(ψ ∧ f) // get a satisfying feature assignment
10         true_features = get_features_with_true_value(assignment)
11         false_features = get_features_with_false_value(assignment)
12         if count == 0 then
13             impact_sets[f] = {}
14             exclusion_sets[f] = ℱ
15         else
16             impact_sets[f] = {f}
17             exclusion_sets[f] = {}
18             forall f' ∈ ℱ\({f} ∪ exclusion_sets[f] ∪ true_features) do
19                 count' = sat_count(ψ ∧ f ∧ f')
20                 if count' == 0 then
21                     exclusion_sets[f].insert(f')
22                     exclusion_sets[f'].insert(f)
23             forall f' ∈ ℱ\({f} ∪ false_features) do
24                 count' = sat_count(ψ ∧ f ∧ f')
25                 if count == count' then
26                     impact_sets[f'].insert(f)
27     return impact_sets, exclusion_sets
```

3.2.3.1 Computational cost

Table 3.9 summarizes the complexities for Algorithms 5 and 6. Let m the number of nodes of the BDD and n the number of features (variables) of the boolean formula. The straightforward approach (Algorithm 5) requires calling apply function $n \cdot (n-1)$ times, as that function has complexity $O(m)$, this approaches has complexity $O(m \cdot n^2)$. On the other hand, although Bohender's approach (Algorithm 6) is often faster in practice than Algorithm 5, it also has complexity $O(m \cdot n^2)$.

Impact & exclusion sets	
Strainghtforward apch.	Boender's apch.
$O(m.n^2)$	$O(m.n^2)$

Table 3.11: Computational complexity for Algorithms 5 and 6

3.3 Product Derivation

Product derivation, also known as *product configuration* or *decision-making process*, can be defined as the process of selecting features in the variability model in order to build valid products or specifications for a product line system. In this context, products are also called derivatives, specifications, configurations, etc.

As Chen et al. [CE11] point out, the benefits of the SPL paradigm highly depends on how efficient the derivation process is, given that whenever a new product is required the derivation process is used.

It is well known that variability models provide a big number of interrelated features. So, to derive a valid product, all constraints between its constituent features must be satisfied. Checking by hand product validity is infeasible for all, but the most trivial variability models. Thus, an automated mechanism to support this process is needed.

In practice, the derivation process is assisted by automated tools commonly called configurators. Research on automated configurators is mainly focused on *consistency checking* and *optimization* [SW98, Jun06]. For example, reasoning engines such as BDD libraries, SAT solvers and Logic-Truth Maintenance systems (LTMS) have been used to detect invalid products (i.e., those which violate some feature interdependencies) [DGR11, SHN+07, Men09]; to provide explanations for derivation flaws [WBS+10, Jan10], to optimize products (i.e., to find products whose cost is less or equal than a given one) [HHRV11, SAH+11, SRK+11], etc.

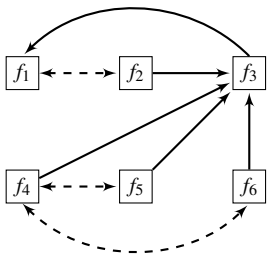


Figure 3.11: A variability model example. (rep. Figure 2.1)

In most cases, configuring a product involves one or more end users, also called decision makers, that translate the requirements they have on the product into decisions about how

Valid Products	
1	$f_1, \neg f_2, \neg f_3, \neg f_4, \neg f_5, \neg f_6$
2	$f_1, \neg f_2, \ f_3, \neg f_4, \neg f_5, \neg f_6$
3	$f_1, \neg f_2, \ f_3, \ f_4, \neg f_5, \neg f_6$
4	$f_1, \neg f_2, \ f_3, \neg f_4, \ f_5, \neg f_6$
5	$f_1, \neg f_2, \ f_3, \neg f_4, \neg f_5, \ f_6$
6	$f_1, \neg f_2, \ f_3, \neg f_4, \ f_5, \ f_6$

Table 3.12: Valid products for Figure 3.11. (rep. Table 2.1)

variability should be handled. As an example, to derive the product P_6=$\{f_1$, f_3, f_5, $f_6\}$ from our example in Figure 3.11, we need to answer the following questions:

1. is f_1 in the product? Yes. Note that, questions about f_2 are not needed because current automated configurators guarantee the derivation of valid products ensuring the satisfaction of all model constraints. So, when the first question is answered, the configurator deduces that the product being configured necessarily excludes f_2.

2. f_3 ? No.

3. f_4 ? No.

4. f_5 ? Yes.

5. f_6 ? Yes.

This way, the configurator is indirectly saving the decision-maker from answering the irrelevant question, "is f_2 in the configuration?".

The problem is finding an optimal question ordering that maximizes the number of decisions automatically derived from other questions previously answered. In other words, reducing the number of questions that a decision maker needs to answer.

Despite the importance of the interactive question ordering problem that this thesis tackles, which was pointed out by Steinberg more than thirty years ago [Ste80], there is little research on it.

A straightforward approach to get such optimal question ordering is computing for each valid product all possible orderings and, thus, finding the ordering with less questions on average for every product.

orderings ($n!$)	products ($\leq 2^n$)						avg number of questions
	$\{f_1\}$	$\{f_1, f_3\}$	$\{f_1, f_3, f_4\}$	$\{f_1, f_3, f_5\}$	$\{f_1, f_3, f_6\}$	$\{f_1, f_3, f_5, f_6\}$	
$f_1 \prec f_2 \prec f_3 \prec f_4 \prec f_5 \prec f_6$	5	5	5	5	5	5	30/6
$f_1 \prec f_3 \prec f_2 \prec f_4 \prec f_5 \prec f_6$	5	5	5	5	5	5	30/6
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$f_6 \prec f_5 \prec f_4 \prec f_3 \prec f_2 \prec f_1$	6	4	3	2	2	2	19/6

Table 3.13: Brute force approach to compute the optimal ordering on average

Table 3.13 sums up the needed computations. For instance, the next-to-last column summarizes the number of questions needed for derivative $P_6 = \{f_1 , f_3 , f_5 , f_6\}$. Ordering $f_6 \prec f_5 \prec f_4 \prec f_3 \prec f_2 \prec f_1$ needs two questions, $f_1 \prec f_2 \prec f_3 \prec f_4 \prec f_5 \prec f_6$ needs five, and so on. Afterwards, the average number of questions for each ordering is computed. Using this approach in the previous example, ordering $f_6 \prec f_5 \prec f_4 \prec f_3 \prec f_2 \prec f_1$ would be selected as an optimal one. As a result, the question sequence for product $P_6 = \{f_1 , f_3 , f_5 , f_6\}$ would be shortened to (1) is f_6 in the product? yes. Then, the configurator deduces that the product being configured necessarily excludes f_4. In addition, as f_6 requires f_3 and f_1, the configurator deduces that the product necessarily includes f_3 and f_1. Finally, as f_1 excludes f_2, the configurator excludes it. (2) is f_5 in the product? yes, and the process ends. In this case the configurator has saved four questions.

Unfortunately, this approach requires $m \cdot n!$ computations, where n is the number of features of the variability model and m is a number $\leq 2^n$. So it is extremely expensive in computational terms and does not scale except for the most trivial variability models. To overcome the scalability limitations of the former approach, two recent approaches that specifically deal with this problem are provided by Chen et al. [CE11] and Mazo et al. [MDSD14]. In the following lines, both of them are described:

Chen et al.'s approach Proposes to minimize the number of configuration steps by sorting the features according to their *probability*[1] of being included in a product. (See Section 3.2.1 for a complete description of probability).

[1] in their original paper and with a fully equivalent meaning, Chen et al. use the term *selectivity* instead of *probability*. As it will be discussed in Chapter 5 our approach follows an entropy driven heuristic and, the Information Theory concept of entropy is defined in terms of probability (see Section 5.1), we have preferred to use *probability* throughout this thesis.

Mazo et al.'s approach Proposes the following heuristics for ordering configuration questions:

Heuristic 1 Features with the smallest domain first: choose first the feature with the smallest domain. The domain of a feature is the set of possible values that the feature can take according to its domain definition and the constraints in which the feature is involved.

Heuristic 2 The most constrained features first: choose the feature that participates in the largest number of constraints.

Heuristic 3 Features appearing in most products first. This heuristic is exactly the same as Chen et al's approach.

Heuristic 4 Automatic completion when there is no choice. This heuristic "provides a mechanism to automatically complete the configuration of features where only one value of their domain is possible [...] it also works when a feature has several values on its domain but only one is valid". In ascending order of computational cost and descending order of constraint propagation capability, Mazo et al. summarize three approaches to implement Heuristic 4 when the variability model is encoded as a Predicate Logic formula: (i) forward-checking, (ii) partial look-ahead, and (iii) full look-ahead (i.e., whereas forward-checking is the fastest algorithm but produces the most limited form of constraint propagation during search, full look-ahead is the most expensive approach but gets the best results). In this thesis, variability models are encoded as BDDs, where full constraint propagation is computationally feasible [Men09]. Instead of considering Heuristic 4 apart, we will use it as a complement to the remaining heuristics by running constraint propagation after every configuration step.

Heuristic 5 Features required by the latest configured feature first: choose the feature that has the largest number of constraints in common with already configured features.

Heuristic 6 Features that split the problem space in two first: set first the features that divide the problem space in two parts of approximately the same size. Unfortunately, Mazo et al. do not provide a way to implement this heuristic which takes into account all model constraints. In particular, Mazo et al. propose a

simplification by just using the tree structure of a FD, or the variation points of an orthogonal variability model [PBL05], but not processing the cross-tree constraints.

3.4 Summary

As a result of our bibliometric analysis, we have identified two of the most prominent topics in the SPL research area in recent years: the automated analysis of variability models and product derivation. According to Figure 3.4.c, from 2010 to 2014, automated analysis of variability models has played a motor role. In addition, this is the most performing topic in this period, as Table 3.5 shows. On the other hand, the derivation of particular products has gained importance in the fourth period, as Figure 3.8.f and the position of PRODUCT-DERIVATION in the strategic diagram in Figure 3.4.d show. Therefore, the rest of this chapter was dedicated to review these important topics, which will be the area of action of this thesis.

Since variability models specify how features can be combined to get valid products, it is possible to compute and identify the commonality, the incompatibility and the relative importance of the features by directly inspecting the models. Nevertheless, existing approaches to carry out such computations have the following limitations:

1. *Algorithms to compute the measures are inefficient*. As we have seen, the usual way to reason on variability models (by repeatedly calling logic engines) has poor time performance and so it hinders user interactivity.

2. *Measures are rigid*. Although some measures have been proposed [BSRC10, DCB10, Boe11] for feature incompatibility and relative importance, their sensitivity is not adjustable and thus they are often too rigid in practice. For instance, the dead measure is commonly used to detect if a feature is expendable for a product line [BSRC10]. Its traditional definition is: "a feature is dead if, due to its dependencies and incompatibilities with the remaining features, it cannot be included in any product". However, imagine a feature that can only be included in 1 percent of the products. The high dispensability of such feature would go unnoticed for the current definition of dead feature.

To overcome such limitations, in Chapter 4 we will provide new **ad-hoc algorithms** that directly interact with the data structure of the BDD that encodes a variability model, and thus they are more efficient than the black-box reuse of logic engines.

On the other hand, as we have discussed in Section 3.3 the process for building a product is not a trivial task and it requires a lot of work in order to translate the user requirements into the decisions needed to properly derive the product. So, there is a need for guidance support for product derivation. In particular, in Chapter 5 we will focus on the interactive question ordering problem and how to determine the optimal path through a series of related questions to any possible product.

CHAPTER

4

Automated Analysis of Variability Models

As argued in Chapter 3, the usual way to compute analysis measures is inefficient for some important ones such as computing features commonalities, impact/exclusion sets of features, and core/dead features. To overcome such limitations, an algorithm based on the glass-box reuse of BDD libraries for each hard operation is proposed. In particular, this dissertation supports hard operations by providing algorithms that directly interact with the data structure of the BDD that encodes a variability model. Furthermore, as we have also commented, impact/exclusion sets of features and core/dead features are too rigid to identify essential, dispensable, and highly incompatible features. For instance, a feature which is only included in one valid product is highly dispensable. However, it would not fit into the current definition of dead feature. We proposed an algorithm, based on the concept of feature commonality, which adds sensitivity to exist measures to identify essential, dispensable, and highly incompatible features. The remainder of this chapter is structured as follows: first, an algorithm to efficiently compute features commonalities of a variability model codified into BDD is presented in Section 4.1. Later, two algorithms to identify essential, dispensable and highly incompatible features are proposed in Sections 4.2 and 4.3, respectively. Section 4.4 presents our algorithm to support our measure redefinition. The chapter ends up by summarizing the computational cost of our approaches w.r.t the related work in Section 4.5.

4.1 Commonality

In this section we provide an algorithm to compute feature commonalities of a Boolean formula encoded as a BDD. First, some definitions required to understand our algorithm are given. Then, the algorithm is presented.

Let us use our small example introduced in Section 2.1 again. Figure 4.2 is its BDD representation using the variable ordering $f_1 \prec f_2 \prec f_3 \prec f_4 \prec f_5 \prec f_6$[1].

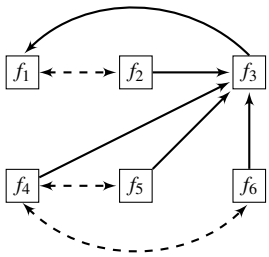


Figure 4.1: A variability model example (rep. Figure 2.1)

$$\begin{aligned}\psi \equiv\ & (f_1 \vee f_2 \vee f_3 \vee f_4 \vee f_5 \vee f_6) \wedge \\ & (\neg f_2 \vee f_3) \wedge (\neg f_3 \vee f_1) \wedge (\neg f_4 \vee f_3) \wedge (\neg f_5 \vee f_3) \wedge (\neg f_6 \vee f_3) \wedge \\ & (\neg f_1 \vee \neg f_2) \wedge (\neg f_4 \vee \neg f_5) \wedge (\neg f_4 \vee \neg f_6)\end{aligned} \tag{4.1}$$

	Valid Products
1	$f_1, \neg f_2, \neg f_3, \neg f_4, \neg f_5, \neg f_6$
2	$f_1, \neg f_2, f_3, \neg f_4, \neg f_5, \neg f_6$
3	$f_1, \neg f_2, f_3, f_4, \neg f_5, \neg f_6$
4	$f_1, \neg f_2, f_3, \neg f_4, f_5, \neg f_6$
5	$f_1, \neg f_2, f_3, \neg f_4, \neg f_5, f_6$
6	$f_1, \neg f_2, f_3, \neg f_4, f_5, f_6$

Table 4.1: Valid products for Figure 4.1 (rep. Table 2.1)

[1] note that a logic formula may be encoded with different BDDs according to the variable ordering used to synthesize the BDD. Likewise, our algorithm produces the same results for equivalent BDDs (i.e., BDDs that encode the same formula)

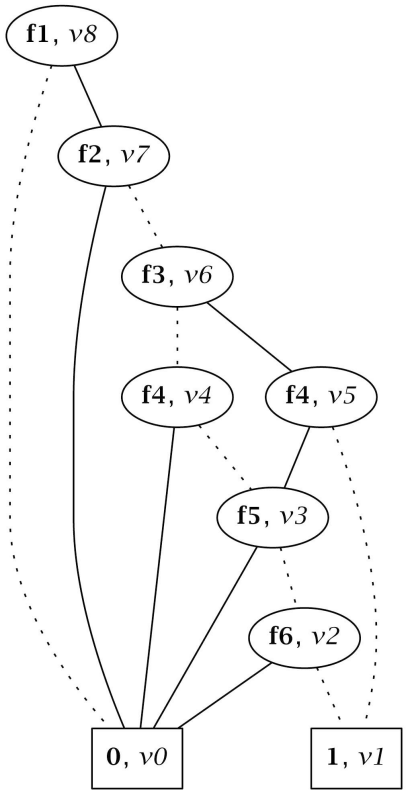


Figure 4.2: BDD for Eq. 4.1 according to the var ordering $f_1 \prec f_2 \prec f_3 \prec f_4 \prec f_5 \prec f_6$.

4.1.1 Theoretical basis of our approach

The commonality of a feature f can be defined as its satisfying probability, i.e., let ψ be a boolean formula that encodes a variability model, and n the number of features of the model, the commonality of a feature is the number of satisfying assignments of ψ where f is true divided to the number of satisfying assignments of ψ.

Definition 1 *The satisfying set of a boolean formula $\psi(f_1, ..., f_n)$, denoted S_ψ, is defined by Equation 4.2.*

$$S_\psi = \{(f_1, ..., f_n) \cdot \psi(f_1, ..., f_n) \equiv \text{true}\} \tag{4.2}$$

Definition 2 *The satisfying set of the variable f_i of a boolean formula $\psi(f_1, f_2, ..., f_n)$, denoted $S_{\psi|_{f_i}}$, is defined by Equation 4.3.*

$$S_{\psi|_{f_i}} = \{(f_1, ..., f_{i-1}, f_{i+1}, ..., f_n) \cdot \psi(f_1, ..., f_{i-1}, f_{i+1}, ..., f_n) \equiv \text{true}\} \tag{4.3}$$

For instance, according to Table 4.1, $\#S_\psi = 6$ since there are 6 valid products, i.e., 6 rows where ψ evaluates to true[1], and $\#S_{\psi|f_4} = 1$ because $f_4 = 1$ in 1 of the 6 rows where ψ is evaluated to true.

Definition 3 *The satisfying probability of a boolean formula $\psi(f_1, ..., f_n)$, denoted $\Pr(\psi)$, is defined by Equation 4.4.*

$$\Pr(\psi) = \frac{\#S_\psi}{2^n} \tag{4.4}$$

Definition 4 *The satisfying marginal probability of a variable f_i in a boolean formula $\psi(f_1, f_2, ..., f_n)$, denoted $\text{MPr}(\psi, f_i)$, is defined by Equation 4.5.*

$$\text{MPr}(\psi, f_i) = \frac{\#S_{\psi|_{f_i}}}{2^n} \tag{4.5}$$

Definition 5 *The satisfying probability of a variable f_i in a boolean formula $\psi(f_1, f_2, ..., f_n)$, denoted $\Pr(\psi|_{f_i})$, is defined by Equation 4.6.*

$$\Pr(\psi|_{f_i}) = \frac{\#S_{\psi|_{f_i}}}{\#S_\psi} \tag{4.6}$$

[1] throughout this section 0/1 and false/true are used interchangeably

For instance, looking at Table 4.1, it is easy to see that $\Pr(\psi) = \frac{6}{2^6}$, $\text{MPr}(\psi|_{f_4}) = \frac{1}{2^6}$, and $\Pr(\psi|_{f_4}) = \frac{1}{6}$.

For convenience, from here on we denote $\Pr(\psi|_{f_i})$ and $\text{MPr}(\psi|_{f_i})$ as $\Pr(f_i)$ and $\text{MPr}(f_i)$, respectively, which is consistent with the notation introduced in Section 3.2 (i.e., if a variability model is encoded by ψ then: $\#S_{\psi|_{f_i}}$ represents the number of products than include f_i, $\#S_{\psi}$ is the total number of products, and consequently, $\Pr(\psi|_{f_i}) = \Pr(f_i)$).

4.1.2 Algorithm to Compute Feature Commonalities

$\Pr(f_i)$ is computed jointly by Algorithms 7, 8 and 9. Figure 4.4 summarizes the computations for the BDD in Figure 4.2. Let us examine how our approach proceeds:

Algorithm 7 computes $\Pr(f_i)$ as $\Pr(f_i) = \frac{\text{MPr}(f_i)}{\Pr(\psi)}$ by calling the auxiliary Algorithms 8 and 9.

Algorithm 8 computes $\Pr(\psi)$. A nice mental picture to understand Algorithm 8 is thinking in pouring 1 liter of water from the BDD root to the terminal nodes. 1 liter goes through the root, then half a liter goes through the low branch and half a liter through the high branch. This procedure advances until the water reaches the leaves. Hence, $\Pr(\psi)$ is the amount of water that node 1 has.

In Figure 4.4, through node v_8 goes 1 liter (i.e., formula_sat_prob[8][1] = 1). Half of it goes to v_7 and the other half to 0-terminal. The $\frac{1}{2}$ liter of water that reaches v_7 is divided in two, half of it goes to 0-terminal and the other half to v_6, and so on. Thus, formula_sat_prob is recomputed at each node: it starts being 1 at v_8, at v_7 is $\frac{1}{2}$, at v_6 is $\frac{1}{4}$, ..., at v_2 is $\frac{1}{16}$, and finally at the 1-terminal node is $\frac{3}{32}$, so $\Pr(\psi) = \frac{3}{32}$.

Algorithm 9 computes $\text{MPr}(f_i)$ by following the procedure proposed by Bryant [Bry86] for traversing a BDD and performing some operation on its nodes. The algortithm is called at the top level with the root node as argument and with the mark fields of the nodes being all false. It then systematically visits every node in the graph by recursively visiting the subgraphs rooted by the two children. As it visits a node, it sets to true the value of the mark field, so that it can later determine whether a child has already been visited.

[1] according to Tables 2.2 and 2.3, the root node has label v_8 and it is in the position 8 of the *bdd* array

In particular, let us examine how it computes MPr(f_5). There are two valid products when when f_5 is true, i.e, ψ evaluates to true when f_5 is true two times:

1. When the call get_marginal_prob(3, ...) is made, lines 10-23 compute the marginal probability of f_5 for the explicit path $v_3 \rightarrow v_2$. The probabilities due to the low and high branches of v_i are stored into the prob_low and prob_high variables, respectively. As $bdd[v_3].low \neq 0$, a recursive call is made to compute the total probability due to the low descendants of v_3 (i.e., get_marginal_prob(2, ...)). As a result: total_prob[v_2] = $\text{prob\_low}_{v_2} + \text{prob\_high}_{v_2} = \frac{1}{32} + 0 = \frac{1}{32}$

   prob_low = prob_low = $\frac{\text{total\_prob}[v_2] \cdot \frac{\text{formulal\_sat\_prob}[v_3]}{2.0}}{formula\_sat\_prob[v_2]} = \frac{\frac{1}{32} \cdot \frac{1}{2}}{\frac{1}{16}} = \frac{1}{32}$

   Since $bdd[v_3].high = 0$, prob_high is 0.

   Finally: prob[bdd[v_3].index] = prob_high = 0

2. In addition, the following implicit paths $v_5 \dashrightarrow v_3 \rightarrow v_2 \dashrightarrow 1 - terminal$ and $v_5 \dashrightarrow v_3 \rightarrow v_2 \rightarrow 1 - terminal$ that have been removed from the reduced BDD (Figure 4.3) should be considered to compute the marginal probability of f_5. Lines 24-31 account for that kind of implicit paths, adjusting the marginal probability with the variables omitted in the paths. For instance, when the algorithm is called for v_5, the marginal probability of f_5 is updated with half the prob_low of v_5.

To sum up:

$$\text{MPr}(f_5) = \text{MPr}(v_5 \dashrightarrow 1 - terminal) + \text{MPr}(v_3 \rightarrow v_2) = \frac{\text{prob\_low}_{v_5}}{2} + 0 = \frac{\frac{1}{16}}{2} = \frac{1}{32}$$

$$\Pr(f_5) = \frac{\text{MPr}(f_5)}{\Pr(\psi)} = \frac{\frac{1}{32}}{\frac{3}{32}} = \frac{1}{3}$$

Figure 4.4 summarizes the computations for the BDD in Figure 4.2.

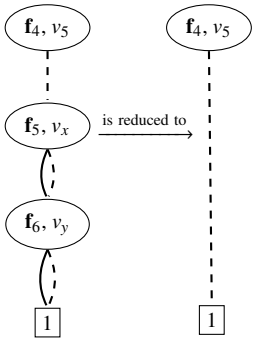


Figure 4.3: Algorithm 9 to update MPr(f) for those features whose nodes has been removed from the BDD due to Reduction R2.

Algorithm 7: get_prob

```
1  Input bdd and var_ordering arrays
2  Output an array which stores Pr(f_i) in position i
3  var formula_sat_prob, total_prob: array[0..length(bdd)-1] of float;
4      prob: array[0..length(var_ordering)-1] of float; i: int;
5  begin
6      for (i=0; i < length(bdd); i++) do
7          total_prob[i] = 0.0
8      for (i=0; i < length(var_ordering); i++) do
9          prob[i] = 0.0
10     formula_sat_prob = get_formula_sat_prob(bdd)
11     get_marginal_prob(length(bdd)-1, total_prob, formula_sat_prob, prob, bdd, var_ordering)
12     for (i=0; i < length(var_ordering); i++) do
13         prob[i] = prob[i] / formula_sat_prob[1]
14     return prob
```

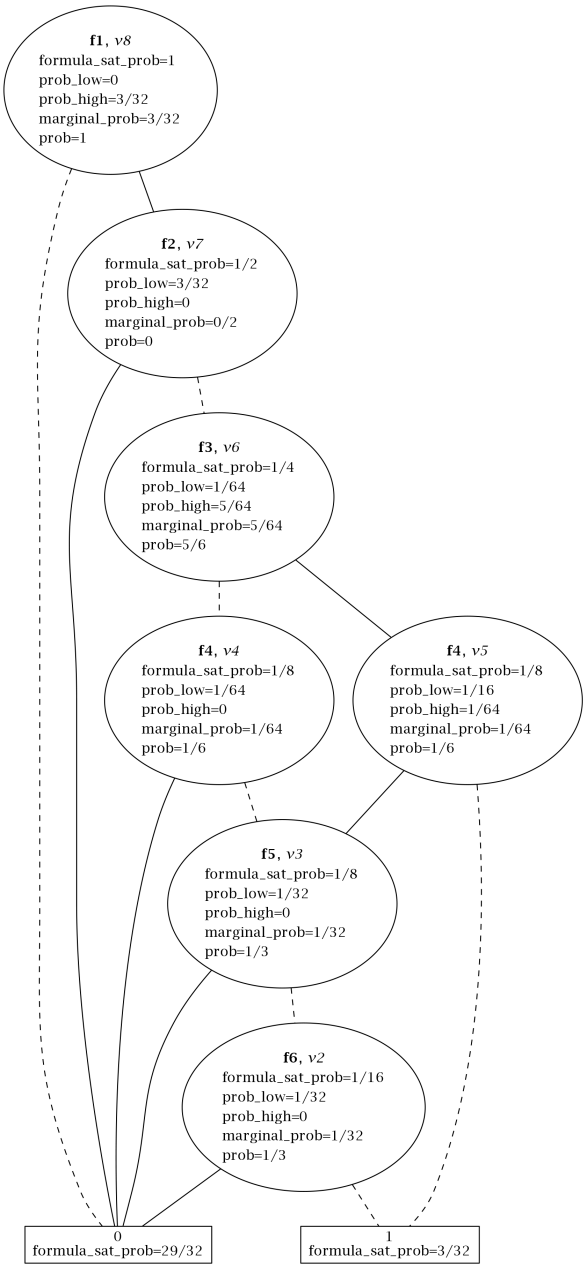


Figure 4.4: Commonality computation for the BDD in Figure 4.2

Algorithm 8: get_formula_sat_prob

```
1  Input bdd array
2  Output an array which in position 1 stores Pr(ψ)
3  var formula_sat_prob: array[0..length(bdd)-1] of float; i: int
4  begin
5      for (i=0; i < length(bdd)-1; i++) do
6          formula_sat_prob[i] = 0.0 // non-root nodes prob is initialized to 0
7      formula_sat_prob[i] = 1.0 // root node prob is 1
8      i=length(bdd)-1
9      while i > 1 do // for all non-terminal nodes
10         formula_sat_prob[bdd[i].low] += formula_sat_prob[i] / 2.0
11         formula_sat_prob[bdd[i].high] += formula_sat_prob[i] / 2.0
12         i -= 1
13     return formula_sat_prob
```

Algorithm 9: get_marginal_prob

```
1  Input v: 0..length(bdd)-1; total_prob, formula_sat_prob: array[0..length(bdd)-1] of float;
2      prob: array[0..length(var_ordering)-1] of float; bdd and var_ordering arrays
3  Output prob is passed by reference and, at the end of the algorithm execution,
4        it stores MPr(f_i) in position i
5  var prob_low, prob_high: float; i: int
6  begin
7      prob_low = 0.0
8      prob_high = 0.0
9      bdd[v].mark = not bdd[v].mark
       // explicit path recusive traversal
10     if bdd[v].low == 1 then
11         prob_low = formula_sat_prob[v] / 2.0
12     else if bdd[v].low ≠ 0 then
13         if bdd[v].mark ≠ bdd[bdd[v].low].mark then
14             get_marginal_prob(bdd[v].low, total_prob, formula_sat_prob, prob, bdd, var_ordering)
15         prob_low = (total_prob[bdd[v].low] · formula_sat_prob[v] / 2.0) / formula_sat_prob[bdd[v].low]
16     if bdd[v].high == 1 then
17         prob_high = formula_sat_prob[v] / 2.0
18     else if bdd[v].high ≠ 0 then
19         if bdd[v].mark ≠ bdd[bdd[v].high].mark then
20             get_marginal_prob(bdd[v].high, total_prob, formula_sat_prob, prob, bdd, var_ordering)
21         prob_high = (total_prob[bdd[v].high] · formula_sat_prob[v] / 2.0) / formula_sat_prob[bdd[v].high]
22     total_prob[v] = prob_low + prob_high
23     prob[bdd[v].index] += prob_high
       // implicit path iterative traversal
24     i = bdd[v].index + 1
25     while i<bdd[bdd[v].low].index do
26         prob[i] += prob_low / 2.0
27         i +=1
28     i = bdd[v].index + 1
29     while i<bdd[bdd[v].high].index do
30         prob[i] += prob_high / 2.0
31         i +=1
```

4.2 Automated Approach to Detect absolutely Essential and Dispensable Features

In this section we provide an algorithm to detect absolutely essential and dispensable features in variability models encoded as BDDs. First, some definitions are provided. Then, the algorithm is presented.

4.2.1 Theoretical basis of our approach

A *valuation* of a formula ψ is an assignment of each variable in ψ to a truth value. For instance, a possible valuation for Equation 4.1 is $\{f_1 = 1, f_2 = 0, f_3 = 0, f_4 = 0, f_5 = 0, f_6 = 0\}$, which evaluates ψ to true. If ψ encodes a variability model, the valid products of the model are represented by the valuations that make ψ true [Bat05]. For example, the former valuation represents the product P_1 in Table 4.1, which just includes the feature f_1.

In BDDs, valuations are represented as paths from the root to the terminal nodes. For instance, the BDD in Figure 4.2 encodes the aforementioned valuation as $v_8 \rightarrow v_7 \dashrightarrow v_6 \dashrightarrow v_4 \dashrightarrow v_3 \dashrightarrow v_2 \dashrightarrow 1$. A valuation results true if its corresponding path ends in the 1-terminal node.

Let $f \rightarrow_+ 1$ and $f \dashrightarrow_+ 1$ be two predicates. $f \rightarrow_+ 1$ is true if the 1-terminal node is reachable from the root traversing the high edge of some node labelled with f; $f \dashrightarrow_+ 1$ is true if the 1-terminal node is reachable through a low edge of some node labelled with f. For example, in Figure 4.2, $f_3 \rightarrow_+ 1$ is true due to the path $v_8 \rightarrow v_7 \dashrightarrow v_6 \rightarrow v_5 \dashrightarrow 1$. Nevertheless, $f_1 \dashrightarrow_+ 1$ is false because v_8 is the only node labeled with f_1 and, once the low edge of v_8 is traversed, it is not possible to reach the 1-terminal node.

Lemma 1 *A feature f is core iff $f \rightarrow_+ 1$ is* true *and $f \dashrightarrow_+ 1$ is* false.

Lemma 2 *A feature f is dead iff $f \rightarrow_+ 1$ is* false *and $f \dashrightarrow_+ 1$ is* true.

Proof: By definition: a feature f (i) is core iff all valid products include f, and (ii) it is dead iff no valid product includes f. That is, when determining if f is core or dead, only the valid products are of interest or, in BDD terms, the paths from the root to the 1-terminal node. So, we are just concerned about the cases where f is traversed and the 1-terminal node is reached, i.e., when $f \rightarrow_+ 1$ is true or $f \dashrightarrow_+ 1$ is true:

1. If $f \rightarrow_+ 1$=true and $f \dashrightarrow_+ 1$=true, the 1-terminal node may be reached traversing high and low edges of nodes labelled with f, or f is omitted in any path where 1-terminal node may be reached due to Reduction R2 (see Section 2.4). That is, f is included in some valid products, but not in another valid ones. Therefore, f neither is core, nor is dead.

2. If $f \rightarrow_+ 1$=true and $f \dashrightarrow_+ 1$=false, the 1-terminal node cannot be reached traversing low edges of nodes labelled with f (due to $f \dashrightarrow_+ 1$=false). All the paths that end in the 1-terminal node go across a high edge of f. In other words, all valid products include f and so f is core.

3. If $f \rightarrow_+ 1$=false and $f \dashrightarrow_+ 1$=true, the 1-terminal node cannot be reached traversing high edges of nodes labelled with f (due to $f \rightarrow_+ 1$=false). All the paths that end in the 1-terminal node go across a low edge of f. As, f is disabled in all valid products, f is dead.

4.2.2 Algorithm to detect core and dead features

Algorithm 10 computes the core and dead features from an input formula ψ, which represents a given variability model. To do so, the algorithm builds the BDD that encodes ψ according to an input variable ordering (line 10). Two arrays named through_high_array and through_low_array are used to store in position i-1 the values of $f_i \rightarrow_+ 1$ and $f_i \dashrightarrow_+ 1$, respectively. The computation of such arrays is performed by Algorithm 12 (lines 13-14). For instance, after executing Algorithm 12 for Figure 4.2,

$$\text{through\_high\_array} = [\text{true}, \text{false}, \text{true}, \text{true}, \text{true}, \text{true}]$$
$$\text{through\_low\_array} = [\text{false}, \text{true}, \text{true}, \text{true}, \text{true}, \text{true}]$$

So,

$$\text{through\_high\_array}[1] = f_2 \rightarrow_+ 1 = \text{false}$$
$$\text{through\_low\_array}[1] = f_2 \dashrightarrow_+ 1 = \text{true}$$

Then, Algorithm 10 applies Lemmas 1 and 2 to identify the core and dead features according to through_high_array and through_low_array (lines 15-19). For instance, since $f_2 \rightarrow_+ 1 = \text{false}$ and $f_2 \dashrightarrow_+ 1 = \text{true}$, Algorithm 10 determines that f_2 is dead.

Algorithm 10: get_core_and_dead_features *(our approach)*

```
1  Input bdd and var_ordering arrays
2  Output two lists : one with all the core features, and
3       another one with all the dead features
4  var core_features, dead_features : list; i: int;
5       through_high_array, through_low_array :
6            array[0..n-1] of boolean;
7       it_reaches_the_1_terminal_array : array[0..m − 1] of boolean
8  begin
9       core_features = {}; dead_features = {};
10      through_high_array = [false, false, . . ., false]
11      through_low_array = [false, false, . . ., false]
12      it_reaches_the_1_terminal_array = [false, false, . . ., false]
13      does_it_reach_the_1-terminal?(length(bdd)-1,
14           through_high_array, through_low_array,
15                it_reaches_the_1_terminal_array)
16      for (i=0; i < length(var_ordering); i++) do
17           if through_high_array[i] ∧ ¬through_low_array[i] then
18                core_features.insert(var_ordering[i])
19           else if ¬through_high_array[i] ∧ through_low_array[i] then
20                dead_features.insert(var_ordering[i])
21      return core_features, dead_features
```

Algorithm 11: update_reduced_vertices

```
1  Input v: 0..m-1; direction : string ∈ {"high", "low"};
2       through_high_array, through_low_array : array[0..n-1] of
3            boolean;
4  Output through_high_array and through_low_array are passed
5       by reference
6  var i: int;
7  begin
8       if direction == "high" then
9            for (i = bdd[v].index + 1; i < bdd[bdd[v].high].index; i++) do
10                through_high_array[i] = true; through_low_array[i] = true;
11      else
12           for (i = bdd[v].index + 1; i < bdd[bdd[v].low].index; i++) do
13                through_high_array[i] = true; through_low_array[i] = true;
```

Algorithm 12: does_it_reach_the_1-terminal?

```
1  Input  v: index of a node in the bdd (i.e., 0..m-1);
2      through_high_array, through_low_array : array[0..n-1] ofboolean;
3      it_reaches_the_1_terminal_array :array[0..m − 1] of boolean
4  Output  the algorithm returns true if v can reach the 1-terminal. Otherwise, it returns false;
5      through_high_array and through_low_array are passed by reference
6  begin
7      if ¬bdd[v].mark then
8          bdd[v].mark = true
           // does v reach the 1-terminal through high?
9          if bdd[v].high == 1 then                          // the 1-terminal is reached
10             through_high_array[bdd[v].index] = true
11             it_reaches_the_1_terminal_array[v] = true
12             update_reduced_nodes(v, "high",
13                 through_high_array, through_low_array)
14         else if bdd[v].high ≠ 0 then                                  // keep searching
15             it_reaches_the_1_terminal_array[v] =
16                 does_it_reach_the_1-terminal?(
17                     bdd[v].high,
18                         through_high_array, through_low_array,
19                             it_reaches_the_1_terminal_array)
20             if it_reaches_the_1_terminal_array[v] then
21                 through_high_array[bdd[v].index] = true
22                 update_reduced_nodes(v, "high", through_high_array, through_low_array)
           // does v reach the 1-terminal through low?
23         if bdd[v].low == 1 then                           // the 1-terminal is reached
24             through_low_array[bdd[v].index] = true
25             it_reaches_the_1_terminal_array[v] = true
26             update_reduced_nodes(v, "low", through_high_array, through_low_array)
27         else if bdd[v].low ≠ 0 then                                   // keep searching
28             it_reaches_the_1_terminal_array[v] =
29                 does_it_reach_the_1-terminal?(
30                     bdd[v].low, through_high_array, through_low_array,
31                         it_reaches_the_1_terminal_array)
32         if it_reaches_the_1_terminal_array[v] then
33             through_low_array[bdd[v].index] = true
34             update_reduced_nodes(v, "low", through_high_array, through_low_array)
35     return it_reaches_the_1_terminal_array[v]
```

To compute $f \rightarrow_+ 1$ and $f \dashrightarrow_+ 1$, Algorithm 12 follows the procedure proposed by Bryant [Bry86] for traversing a BDD and performing some operation on its vertices. The algorithm is called at the top level with the root vertex as argument and with the mark fields of the vertices being all false. It then systematically visits every vertex in the graph by recursively visiting the subgraphs rooted by the two children. As it visits a vertex, it sets to true the value of the mark field, so that it can later determine whether a child has already been visited. For each vertex, it is checked if the 1-terminal node is reached. If so, Algorithm 11 is called to update arrays through_high_array and through_low_array to account for the nodes that have been removed from the BDD due to Reduction R2 (see Section 2.4). For instance, as Figure 4.5 shows, the edge $v_5 \dashrightarrow 1$ in Figure 4.2 is the result of erasing vertices v_x and v_y from the BDD because their high and low outgoing edges point to the same node. Taking that removal into account and as $f_4 \dashrightarrow_+ 1$ is true, then it follows that $f_5 \rightarrow_+ 1$, $f_5 \dashrightarrow_+ 1$, $f_6 \rightarrow_+ 1$ and $f_6 \dashrightarrow_+ 1$ are also true.

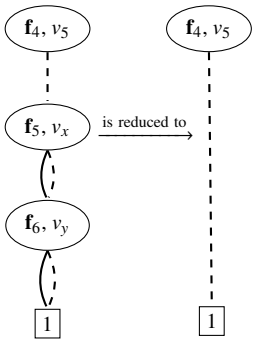


Figure 4.5: Algorithm 11 to update the *1*-reachability for features whose vertices has been removed from the BDD due to Reduction R2.

4.3 Automated Approach to Identify Highly Required and Incompatible Features

This section describes an algorithm to compute the impact and exclusion sets of the features represented by a variability model encoded as a BDD. In particular, our approach is based on repeatedly calling Algorithm 10. Algorithm 13 summarizes how our approach proceeds.

Algorithm 13: get_impact_and_exclusion_sets *(our approach)*

```
1  Input bdd and var_ordering arrays
2  Output two arrays of lists : one including the impact set for each
3       feature, and another one including the exclusion set
4            for each feature
5  var core, core′, dead, dead′ : list
6       impact_set, exclusion_set : array[0..n-1] of list; i, j, k : int
7  begin
8       impact_set = [{}, {}, . . ., {}]
9       core, dead = get_core_and_dead(bdd, var_ordering)
10      for (i=0; i < length(var_ordering); i++) do
11           if var_ordering[i] ∈ dead then
12                impact_set[i] = {}
13                exclusion_set[i] = var_ordering
14           else
15                if var_ordering[i] ∈ core then
16                     impact_set[i] = var_ordering
17                     exclusion_set[i] = dead
18                else
19                     core′, dead′ = get_core_and_dead(bdd ∧
20                          var_ordering[i], var_ordering)
21                     for (j=0; j < length(core′); j++) do
22                          k = index of core′[j] in var_ordering
23                          impact_set[k].insert(var_ordering[i])
24                          exclusion_set[i] = dead′
25      return impact_set, exclusion_set
```

4.4 Measure Flexibilization

The sensitivity of existing measures to identify essential, dispensable and highly incompatible features can be augmented using the concept of feature commonality, which accounts for the likeliness of a feature to be included in a product. First, some definitions required to understand our algorithm are given. Then, the algorithm is presented.

4.4.1 Theoretical basis of our approach

Table 4.2 summarizes feature commonalities for Figure 4.1. For instance, looking at Equation 4.1 it can be checked that feature f_3 is included in 5 of the 6 valid products, so its commonality is $\frac{5}{6}$.

Feature	f_1	f_2	f_3	f_4	f_5	f_6
Pr(f)	1	0	$\frac{5}{6}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$

Table 4.2: Feature commonalities for Figure 4.1

Let *sensitivity* be a number between 0 and 1. The measures this section deals with can be redefined as follows.

Definition 6 *A feature f is core or dead under sensitivity α iif Equation 4.7 or 4.8 holds, respectively.*

$$\Pr(f) \geq (1 - \alpha) \tag{4.7}$$

$$\Pr(f) \leq \alpha \tag{4.8}$$

For example, under sensitivity 0.2, f_3 is considered core (i.e., $\frac{5}{6} \geq 1 - 0.2$), and f_4 dead (i.e., $\frac{1}{6} \leq 0.2$).

Definition 7 *The commonality* $\Pr(f|f')$ *of a feature f conditioned to another feature f' is calculated as:*

$$\Pr(f) = \frac{\#\mathcal{P}_{f,f'}}{\#\mathcal{P}_{f'}}$$

where $\mathcal{P}_{f,f'}$ denote the set all of valid products that include f and f'. And $\mathcal{P}_f$ denote the set all of valid products that include f'.

For instance, from all the products summarized by Table 4.1, only the following two include f_5: P_4 and P_6. As just one of them includes f_6 (P_6), $\Pr(f_6|f_5) = \frac{1}{2}$.

Definition 8 *The impact set of a feature f under sensitivity α is composed of all the features f' that require f to be enabled whenever they are included in at least $1 - \alpha$ of the products, i.e,*

$$\text{Impact Set}_\alpha(f) = \{f' \cdot \Pr(f|f') \geq (1 - \alpha)\}$$

Table 4.3 summarizes the conditional commonalities for all features in Figure 4.1. Let us suppose sensitivity is 0.2, then the impact set of f_3 is $\{f_1, f_3, f_4, f_5, f_6\}$ (i.e., those features whose rows are ≥ 0.8 in column f_3).

		Commonality of					
		f_1	f_2	f_3	f_4	f_5	f_6
Conditioned to	f_1	1	0	$\frac{5}{6}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$
	f_2	0	0	0	0	0	0
	f_3	1	0	1	$\frac{1}{5}$	$\frac{2}{5}$	$\frac{2}{5}$
	f_4	1	0	1	1	0	0
	f_5	1	0	1	0	1	$\frac{1}{2}$
	f_6	1	0	1	0	$\frac{1}{2}$	1

Table 4.3: Feature conditional commonalities for Figure 4.1

Definition 9 *The* exclusion set *of a feature f under sensitivity α is composed of all the features f' that are required to be disabled in at least $1 - \alpha$ of the derivatives that include f, i.e.,*

$$\text{Exclusion Set}_\alpha(f) = \{f' \cdot \Pr(f'|f) \leq \alpha\}$$

For example, according to Table 4.3, the exclusion set of f_3 with sensitivity 0.2 is $\{f_2, f_4\}$ (i.e., those features whose columns are ≤ 0.2 in row f_3).

Note that Definitions 6, 8, and 9 coincide with the rigid ones given in Sections 3.2.2 and 3.2.3 for the extreme case when sensitivity is 0.

4.4.2 Algorithms to Detect Core & Dead Features and to Compute Impact & Exclusion Feature Sets

Our approach uses the algorithm 7 that computes the feature commonalities of a variability model encoded as a BDD, presented in Section 4.1.2. Using that algorithm:

1. The identification of all core and dead features in a variability model considering a given sensitivity is performed by Algorithm 14.

2. The impact and exclusion sets of all features in a variability model considering a given sensitivity is computed by Algorithm 15.

Algorithm 14: get_core_and_dead_features *(flexible apch.)*

```
1  Input boolean formula ψ; var_ordering: array[0..n-1] of string; sensitivity ∈ [0, 1]
2  Output two lists including all core and dead features
3  var core_features, dead_features: list; i: int;
4  begin
5      core_features = {}
6      dead_features = {}
7      bdd = get_bdd(ψ, var_ordering)
8      feature_probabilities = get_prob(bdd, var_ordering)
9      i = 0
10     while i < length(var_ordering) do
11         if feature_probabilities[i] ≥ (1 − sensitivity) then
12             core_features.insert(var_ordering[i])
13         else if feature_probabilities[i] ≤ sensitivity then
14             dead_features.insert(var_ordering[i])
15         i += 1
16     return core_features, dead_features
```

Algorithm 15: get_impact_and_exclusion_sets *(flexible apch.)*

```
1  Input boolean formula ψ; var_ordering: array[0..n-1] of string; sensitivity ∈ [0, 1]
2  Output two hash tables including for each feature its impact and exclusion sets
3  var impact_sets, exclusion_sets: hash; i, j: int;
4  begin
5      impact_sets = Hash.new
6      exclusion_sets = Hash.new
7      i = 0
8      while i < length(var_ordering) do
9          impact_sets[var_ordering[i]] = {}
10         exclusion_sets[var_ordering[i]] = {}
11         bdd = get_bdd(ψ ∧ var_ordering[i], var_ordering)
12         feature_probabilities = get_prob(bdd, var_ordering)
13         j = 0
14         while j < length(var_ordering) do
15             if feature_probabilities[j] ≥ (1 − sensitivity) then
16                 impact_sets[var_ordering[j]].insert(var_ordering[i])
17             if feature_probabilities[j] ≤ sensitivity then
18                 exclusion_sets[var_ordering[i]].insert(var_ordering[j])
19             j += 1
20         i += 1
21     return impact_sets, exclusion_sets
```

4.5 Computational cost

This section summarizes the computational cost of our approaches w.r.t the related work. Let us analyze the time complexity of Algorithm 7. Remember that this algorithm computes feature commonalities by calling Algorithms 8 and 9. Let m be the number of nodes of the BDD, and n the number of variables of the boolean formula. Algorithm 8 requires traversing all the nodes, so its computational complexity is $O(m)$. Algorithm 9 also traverses all the BDD nodes. In addition, to account for the implicit paths removed from the reduced BDD, the variables omitted on the edges that come from each node need to be traversed (which is done by lines 24-31). Table 4.4 summarizes those traversals for Figure 4.2. For instance, when v_5 is recursively traversed, the variables f_5 and f_6 need to be iteratively traversed because the edge $v_5 \dashrightarrow 1$ omits them (i.e., the variable encoded by node v_5, f_4, jumps directly to 1 omitting the intermediate variables f_5 and f_6 in the ordering $f_1 \prec f_2 \prec f_3 \prec f_4 \prec f_5 \prec f_6$). Table 4.4 helps noticing the savings our algorithm provides compared to the straightforward approach (Algorithms 2 and 5), which requires traversing all nodes for all variables (which in computational cost terms is equivalent to traversing all variables for every node). Therefore, Algorithm 9 does not traverse mn elements, but mn', where n' is strictly less than n. If $n' = n$, all nodes in the BDD should go directly to 0 or 1, jumping over all the variables. Nevertheless, as BDDs are organized in hierarchical levels according to the variable ordering, this is impossible (i.e., the nodes that encode a variable with position k in the ordering only can jump over the variables with positions $k + 1 \ldots n$).

Likewise, our approach to compute core and dead features has the same time complexity as Algorithm 7, i.e., $O(mn')$. The difference between them is that Algorithm 7 requires MPA calculations whereas Algorithm 10 woks on boolean variables.

On the other hand, as our approach to compute impact and exclusion sets calls algorithm 10 n times, its time complexity is $O(mn'n)$.

Finally, we are going to review the time complexity of our approaches to compute core/dead features and impact/exclusion sets under under sensitivity α. It follows that Algorithm 7 has computational complexity $O(mn')$, and thus, Algorithms 14 and 15 have complexity $O(mn')$ and $O(mn'n)$, respectively.

Table 4.8 summarizes the complexities for Algorithms 2, 5, 3, 14, and 15. Our approach has the best time complexity. Nevertheless, computational complexity O only provides an upper bound on the worst time required by the algorithms. As it will be empirically shown

node	arcs	omitted vars that are traversed
v_8	$v_8 \dashrightarrow 0 - terminal$	none
	$v_8 \rightarrow v_7$	none
v_7	$v_7 \dashrightarrow v_6$	none
	$v_7 \rightarrow 0 - terminal$	none
v_6	$v_6 \dashrightarrow v_4$	none
	$v_6 \rightarrow v_5$	none
v_5	$v_5 \dashrightarrow 1 - terminal$	f_5, f_6
	$v_5 \rightarrow v_3$	none
v_4	$v_4 \dashrightarrow v_3$	none
	$v_4 \rightarrow v_2$	none
v_3	$v_3 \dashrightarrow v_2$	none
	$v_3 \rightarrow 0 - terminal$	none
v_2	$v_2 \dashrightarrow 1 - terminal$	none
	$v_2 \rightarrow 0 - terminal$	none

Table 4.4: Variables iteratively traversed for BBD in Figure 4.2

Commonality features	
sat_count apch. (Alg. 1)	Our apch. (Alg. 7)
$O(m.n)$	$O(m.n')$ where $n' < n$

Table 4.5: Time complexity comparison of Algorithms 1 and 7.

Core & dead features			
Strainghtforward apch. (Alg. 2)	Lesla et al.'s apch. (Alg. 3)	Commonality apch. (Alg. 4)	Our apch. (Alg. 10)
$O(m.n)$	$O(m.n)$	$O(m.n)$	$O(m.n')$ where $n' < n$

Table 4.6: Time complexity comparison of Algorithms 2, 3, 4 and 10.

Impact & exclusion sets		
Strainghtforward apch. (Alg. 5)	Boender's apch (Alg. 6)	Our appch. (Alg. 13)
$O(m.n^2)$	$O(m.n^2)$	$m.n.n'$ where $n' < n$

Table 4.7: Time complexity comparison of Algorithms 5, 6 and 13.

in Section 6.2, in practice our approach is also the fastest for all the evaluated cases. In particular:

1. Algorithms 2 and 5 run always on the worst case, i.e., they require traversing mn and mn^2 for all variability models, respectively.

2. Thanks to its lines 18 and 22, Algorithm 3 saves a number of iterations, requiring in practice mnn'' steps where n'' is usually $< n$.

3. Algorithm 15 requires mnn' iterations and, on average requires less time than Algorithm 3, i.e., $n' \ll n''$.

Core & dead features		**Impact & exclusion sets**		
Strainghtforward apch. (Algorithm 2)	Flexible apch. (Algorithm 14)	Strainghtforward apch. (Algorithm 5)	Boender's apch. (Algorithm 3)	Flexible apch. (Algorithm 15)
$O(mn)$	$O(mn')$ where $n' < n$	$O(mn^2)$		$O(mn'n)$ where $n' < n$

Table 4.8: Time complexity comparison of rigid vs flexible measures.

To sum up, all of our approaches have better time complexity than related work. In addition, our measure flexibilization not only is more time-efficient than related work but also provides new functionality (i.e., it can take into account different levels of sensitivity).

CHAPTER

5

Product Derivation

This chapter presents our heuristic to minimize the number of steps required to derive a product from a variability model. Section 5.1 introduces the theoretical background of our approach. As we will see, our heuristic, in the same way that the other heuristics summarized in Section 3.3, requires computing features commonalities in a variability model. Section 3.2.1 discussed the scalability limitations of the approach commonly used to compute those probabilities. To overcome such limitations, we used the algorithm 7 proposed in Section 4.1.2, that provides an efficient commonality computation. After reviewing the foundations of our approach, Section 5.2 describes our heuristic in detail.

5.1 Information Theory

The following definitions were originally introduced by Shannon [Sha48]. Let us start with the concept of *entropy*.

Definition 10 *Let X be a discrete random variable with alphabet $\mathcal{X}$ and probability mass function $\Pr(x) = \Pr\{X = x\}$, $x \in \mathcal{X}$; the entropy H of X is defined by Equation 5.1:*

$$H(X) = -\sum_{x \in \mathcal{X}} \Pr(x)\Big(\log_2 \Pr(x)\Big) \tag{5.1}$$

Let us present the concept of *conditional entropy*, which is the entropy of a random variable conditional on the knowledge of another random variable.

Definition 11 *Let X and Y be two discrete random variables. The conditional entropy $H(X|Y)$ of X given Y is defined by Equation 5.2:*

$$H(X|Y) = \sum_{y \in \mathcal{Y}} \Pr(y) H(X|Y = y) \tag{5.2}$$

Finally, let us introduce the concept of *mutual information*, also called *information gain*, which represents the reduction in a variable uncertainty due to another random variable.

Definition 12 *Consider two random variables X and Y with a joint probability mass function $\Pr(x, y)$ and marginal probability mass functions $\Pr(x)$ and $\Pr(y)$. The mutual information $I(X; Y)$ is defined by Equation 5.3 as the relative entropy between the joint distribution and the product distribution $\Pr(x)\Pr(y)$:*

$$I(X; Y) = \sum_{x,y} \Pr(x, y) \log_2 \frac{\Pr(x, y)}{\Pr(x)\Pr(y)} \tag{5.3}$$

Entropy and mutual information satisfy the following properties that will be used throughout this chapter:

1. $H(X) \geq 0$
2. $H(X) \leq \log_2 \#\mathcal{X}$, with equality if and only if X is distributed uniformly over $\mathcal{X}$ (the number of elements of a set S is denoted as $\#S$)
3. $I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = I(Y; X)$

5.2 Entropy driven configuration

Let us return to the original problem this chapter tackles. Given a set of questions $\mathcal{Q}$, our goal is to sort it in such a way that the user has to answer as few questions as possible to complete the desired product. To find the optimal order of $\mathcal{Q}$, we propose to rank each question q according to its expected information gain, i. e., measuring how much uncertainty can be reduced on average when the engineer answers it. Such information gain is modeled as the

mutual information $I(\mathcal{F}; q)$, where $\mathcal{F}$ is the set of all valid products (i. e., the ones that satisfy all feature interdependencies).

When a product is completely configured, the entropy of every question q is zero. Since q has been answered, $H(q|\mathcal{F}) = 0$. Thus, it follows that $I(\mathcal{F}; q) = H(q)$, as Equation 5.4 demonstrates (see Property 3 in Section 5.1).

$$I(\mathcal{F}; q) = H(q) - H(q|\mathcal{F}) = H(q) \tag{5.4}$$

When we ask "is feature f in the configuration?", the entropy of the question $H(q)$ is computed by Equation 5.5, where $\Pr(f)$ is the probability that f is included in the product.

$$\begin{aligned} H(q) &= -\Pr(f)\log_2\Pr(f) - \Pr(\neg f)\log_2\Pr(\neg f) \\ &= -\Pr(f)\log_2\Pr(f) - (1 - \Pr(f))\log_2(1 - \Pr(f)) \end{aligned} \tag{5.5}$$

Our approach to guide the derivation of a product may be thought of as a binary search for the user desired product (in this point it is important to remember Heuristic 6 presented in Section 3.3). To successively divide the search space into subspaces of approximately the same size (i. e., where the pursued product is approximately with the same probability), the user answers the question that provides the most information about the product (i. e., the question with highest entropy). Thus, the derivation process advances iteratively, by performing the following activities, until the entropy of all features becomes zero:

1. Computing the feature probabilities from the input variability model. As the process advances, the derivation space gets narrower and, consequently, the feature probabilities change.

2. Computing the entropy value for each question.

3. Sorting the questions in descending order of entropy.

4. Asking the user for answering a question with entropy greater than zero. Note than when a question has zero entropy, it is because it has been answered in a previous step directly or indirectly (i. e., because of the question interdependencies).

5. Updating the set of answers and the variability model. Note that the customer answer may be "yes" or "no". Therefore, the boolean formula ψ that encodes the variability model has to be updated to $\psi \wedge f$ or $\psi \wedge \neg f$, respectively.

Entropy may also be used to measure how hard it is to derive a given model. From the "point of view" of an automated configurator, when the derivation process starts, the product desired by the customer is any f in $\mathcal{F}$ with the same probability. So the variability model uncertainty is calculated by Equation 5.6 (see Property 2 in Section 5.1).

$$H(\mathcal{F}) = \log_2 \#\mathcal{F} \tag{5.6}$$

5.2.1 Example

Coming back to the running example introduced in Section 2.1, let us see how our approach works.

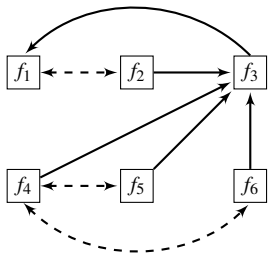


Figure 5.1: A variability model example (rep. Figure 2.1).

	Valid Products
1	$f_1, \neg f_2, \neg f_3, \neg f_4, \neg f_5, \neg f_6$
2	$f_1, \neg f_2, \ f_3, \neg f_4, \neg f_5, \neg f_6$
3	$f_1, \neg f_2, \ f_3, \ f_4, \neg f_5, \neg f_6$
4	$f_1, \neg f_2, \ f_3, \neg f_4, \ f_5, \neg f_6$
5	$f_1, \neg f_2, \ f_3, \neg f_4, \neg f_5, \ f_6$
6	$f_1, \neg f_2, \ f_3, \neg f_4, \ f_5, \ f_6$

Table 5.1: Valid products for Figure 5.1 (rep. Table 2.1).

Figures 5.2, 5.3 and 5.4 sum up the steps required to derive the product P_1=$\{f_1$, f_3 , f_5 , $f_6\}$ using the entropy heuristic. In particular, these figures show the entropy and probability values for each feature in each step required to derive the product P_1. In the first step (Figure 5.2), f_5 and f_6 are the features with highest entropy. So, the system asks the user if f_6 is

included in the product. Once the user answers affirmatively, the probabilities of the features are recomputed and so the entropies (e.g., the inclusion of f_6 implies the exclusion of f_4, so Pr(f_4)=0 and thus $H(f_4)$=0). Finally, the system asks the user if f_5 is included in the product. Once the user answers affirmatively, the derivation process is completed as Figure 5.4 shows.

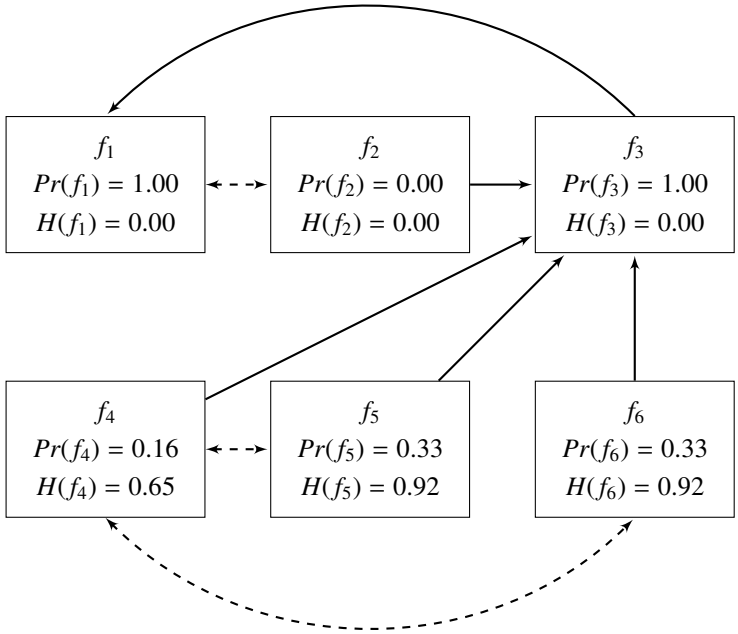


Figure 5.2: Deriving product P_1=$\{f_1$, f_3 , f_5 , $f_6\}$ using feature entropy. Step 0.

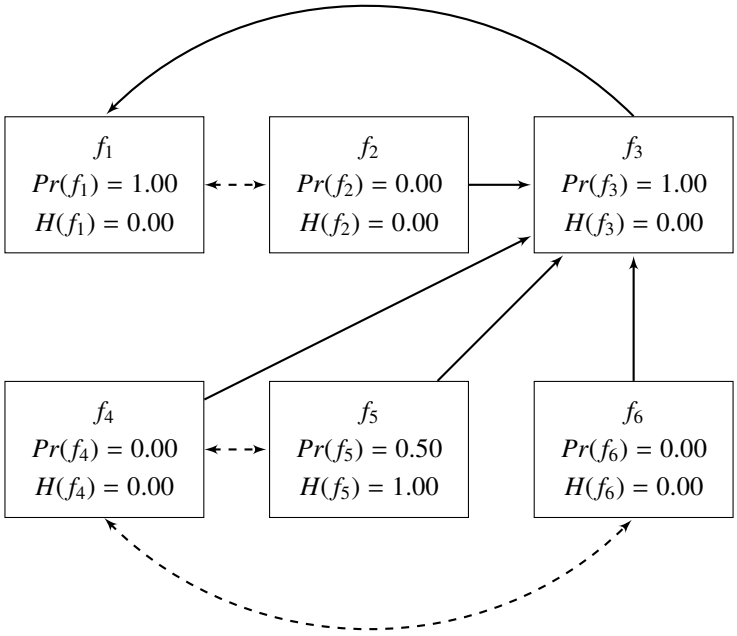


Figure 5.3: Deriving product P_1=$\{f_1$, f_3 , f_5 , $f_6\}$ using feature entropy. Step 1.

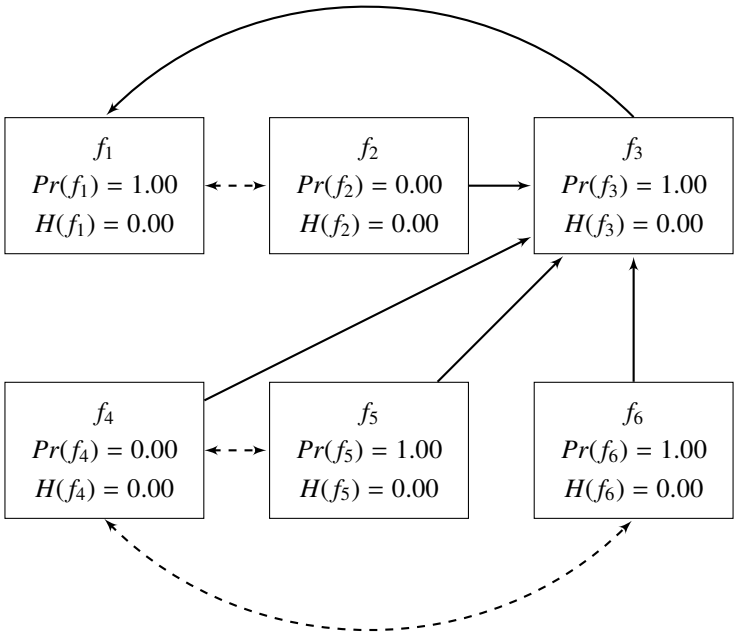


Figure 5.4: Deriving product P_1=$\{f_1$, f_3 , f_5 , $f_6\}$ using feature entropy. Step 2.

We remark here that our approach does not force the user to follow a fixed sequence of questions. In each configuration step, the user may decide not to answer the best entropy-ranked question, but the one she thinks is more convenient. After the question is answered, the entropies are recomputed and thus our approach adjusted accordingly to the user preferences in an interactive way.

CHAPTER

6

Experimental evaluation

In this chapter, the results of several empirical experiments carried out to test the validity of our approaches are reported. As we have shown in Section 4.5, all of our approaches present better time complexity than related work. So, in this chapter, our main goals are to experimentally check if: they are faster than others in practice, i.e., they have better time performance on real variability models, the measure flexibilization proposed really helps to detect essential, dispensable and highly incompatible features. Also, our heuristic for user guidance requires less steps to derive a valid product w.r.t the related work. In particular, Section 6.1 reports the results of comparing our algorithms for computing the rigid measures to traditional ones. Section 6.2 provides empirical evidence of the usefulness of our measure redefinition. This chapter is concluded by summarizing the results of our heuristic for user guidance in Section 6.3.

6.1 Measures to Identify Absolutely Essential and Dispensable Features

This section reports the time-performance comparison of our approach with respect to Tartler.'s [Tar13] and Lesta et al.'s [LSW15] procedures.

6.1.1 Experimental design

Table 6.1 summarizes the benchmark used to validate our approach. Variability models $m_1 - m_{10}$ come from the SPLOT repository[1], models $m_{11} - m_{14}$ come from She et al. [She13, BRN+13], and models $m_{15} - m_{19}$ come from Bak [Bak13], being $m_{16} - m_{19}$ randomly generated. All tests were conducted on an Intel© Core™ i7-3537v 2.00 GHz with 8GB RAM (although only one core was used). To improve the accuracy of the experimental results, we tried to minimize the operating system interference in the tests (e.g., due to interrupts to update internal OS kernel clock, automated task management and planing, memory garbage-collection, etc.) by repeating each experiment 50 times.

Table 6.1: Summary of the experimental results

Variability Model			Time-performance (in seconds)				
id.	Name	#features	Tartler. [Tar13]		Lesta et al. [LSW15]		Our apch.
			SAT	BDD	SAT	BDD	
m_1	Billing	88	0.02175	0.00143	0.01010	0.00070	0.00001
m_2	Coche ecologico	94	0.09131	0.00451	0.05008	0.00200	0.00002
m_3	UP estructural	97	0.08387	0.00321	0.05020	0.00170	0.00002
m_4	Xtext	137	0.07575	0.00513	0.04062	0.00270	0.00003
m_5	Battle of tanks	144	0.18828	0.00786	0.09803	0.00390	0.00003
m_6	FM Test	168	0.17512	0.01769	0.10501	0.00850	0.00009
m_7	Printers	172	0.12809	0.01177	0.06800	0.00510	0.00006
m_8	Banking software	176	0.19442	0.00921	0.09802	0.00530	0.00005
m_9	eShop	290	0.47985	0.04139	0.23400	0.02013	0.00031
m_{10}	EIS	366	0.74156	0.06534	0.44803	0.03404	0.00210
m_{11}	axTLS	108	0.07135	0.00228	0.03840	0.00121	0.00015
m_{12}	Fiasco	171	0.14856	0.00517	0.07864	0.00252	0.00024
m_{13}	uClibc	369	3.84395	0.42035	1.83082	0.18740	0.01362
m_{14}	BusyBox	881	10.04722	0.99665	4.93475	0.53460	0.04274
m_{15}	Android	88	0.79197	0.08849	0.38050	0.04031	0.00232
m_{16}	FM-500-50-1	500	27.70023	5.23822	14.42221	2.24335	0.18439
m_{17}	FM-1000-100-2	1000	70.11689	16.68317	37.82924	9.02403	0.29285
m_{18}	FM-2000-200-3	2000	325.68394	78.73022	175.9050	42.35382	2.80637
m_{19}	FM-5000-500-4	5000	1976.10194	406.55513	874.46547	226.93796	7.73044

Our algorithm was implemented as an extension of the BDD library *BuDDy*[2]. As the im-

[1] http://www.splot-research.org/

[2] http://buddy.sourceforge.net/manual/main.html

plementation of alternative approaches to ours relies on SAT-technology, just comparing the performance of our algorithm to that implementation could skew the experimental validation to a SAT versus BDD contest. To overcome this problem, we also evaluated the performance of a BDD implementation of related work as well. In particular, Tartler.'s and Lesta et al.'s approaches were implemented with the SAT-solver *minisat*[1] and BuDDy. Finally, the orderings of the variables for all BDDs were computed using the heuristic proposed by Narodytska et al. [NW07].

The implementation of our algorithm and the benchmark used to validate its performance are available at:

http://hperez30.github.io/CoreAndDeadFeatures/

6.1.2 Experimental results

Table 6.1 and Figure 6.1 summarize the results of the conducted experimental evaluation. Note that, as each test was run 50 times, both the table and the figure show averaged data.

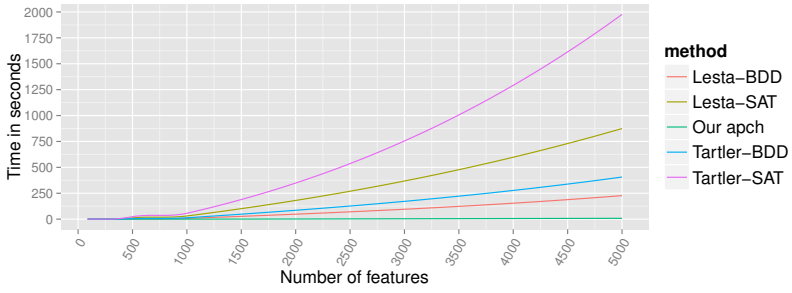


Figure 6.1: Graphical representation of the experimental results

According to the results, our approach outperforms related work in every variability model, being its benefits more apparent as models are larger. In addition, it should be noted that the BDD implementation of related work runs faster than the SAT implementation.

[1] http://minisat.se/

6.2 Measures to Identify Essential Dispensable and Highly Incompatible Features

This section reports the results of several empirical experiments carried out to test the validity of our approach. In particular, the goal of this section is twofold: to check if our approach is faster than others (i.e., if our approach has better time performance on real variability models in practice), and if the measure flexibilization is useful on real variability models.

6.2.1 Experimental design

The time performance of our approach and related work have been evaluated on a benchmark composed the following models:

1. A configuration model provided by the car manufacturing company Renault DVI, which deals with the configuration of a family of cars named Renault Megane. The model is written in the *Configit language* and can be downloaded from:

   http://www.itu.dk/research/cla/externals/clib/

2. A configuration model for laptops, which was reverse engineered from the DELL homepage on February 2009 by Nöhrer [NE11, NBE12, NE13]. The model is specified as a *decision model* and can be downloaded from the C2O website:

   http://www.sea.jku.at/tools/c2o

3. All *feature models* from the SPLOT repository including more than 100 features: Xtext, Battle of Tanks, FM Test, Printers, Banking Software, Electronic Shopping, and a Model for Decision-making for Investments on Enterprise Information Systems.

All tests were conducted on an Intel© Core™ 2 i3-4010U with 1.7 GHz and 4GB RAM (although only one core was used). To improve the accuracy of the experimental results, we tried to minimize the operating system interference in the tests (e.g., due to interrupts to update internal OS kernel clock, automated task management and planing, memory garbage-collection, etc.) by repeating each experiment 50 times.

To support the reliable comparison of the algorithms described in Sections 3.2.2 and 4.4, all of them have been implemented extending the BuDDy package for BDDs.

6.2.2 Time Performance

Table 6.2 summarizes the results of the performance tests. As we can seen, our approach (Algorithms 14 and 15 parameterized with sensitivity = 0) outperforms related work (Algorithms 2, 5, and 3) in all the experiments.

Variability model	Number of features	Dead & core features		Impact & exclusion sets		
		Alg. 2	Alg. 14	Alg. 5	Alg. 3	Alg. 15
Renault	398	5.07081	3.80750	823.38893	764.03980	587.12394
Dell	123	0.04938	0.03310	1.38710	1.31957	0.88028
Xtext	137	0.05547	0.00185	1.41291	1.67129	0.20609
Battle of Tanks	144	0.11169	0.00200	2.81473	2.59596	0.21417
FM Test	168	0.12117	0.04005	6.52654	6.67389	1.72019
Printers	172	0.12962	0.00271	2.53492	2.90606	0.21415
Banking Software	176	0.15372	0.00397	3.69472	5.16730	0.32365
Electronic Shopping	290	1.18944	0.37749	276.89974	224.35847	98.18766
Investments on Enterprise Information Systems	366	17.86456	4.50573	6,002.73245	5,120.10355	952.13433

Table 6.2: Performance in seconds of rigid and flexible measures

It is worth noting that Boender's shortcut to compute the impact and exclusion sets (Algorithm 3) is not always faster than the straightforward approach (Algorithm 5). Thus, in some cases the extra cost required due to the `sat_one` call in line 9 of Algorithm 3 does not payoff. In particular, it is slightly slower for the following SPLOT test cases: Xtext, FM Test, Printers, and Banking Software.

6.2.3 Benefits of Measure Sensitivity

The Renault Megane model is a benchmark widely used by the mass customization community (e.g., [AFM02] [Jen04] [OOF05] [HHOW05] [NW07] [HT07] [CO08] [Que11] [Kro12] [Gan12] [BFL13]). To the extent of our knowledge no author has reported dispensable nor highly incompatible features for the model. Nevertheless, the model includes 6 dead features, i.e., the 1.51% of the features cannot be included in any product.

If sensitivity is taken into account and so feature probabilities are computed, the perception of feature reusability changes dramatically. Figure 6.2 shows the histogram of feature probabilities. According to such figure, 53.27% of the features are dead at sensitivity 0.05

(see the first bar). In other words, more than half of the features can be reused at most in just 5% of the valid products!

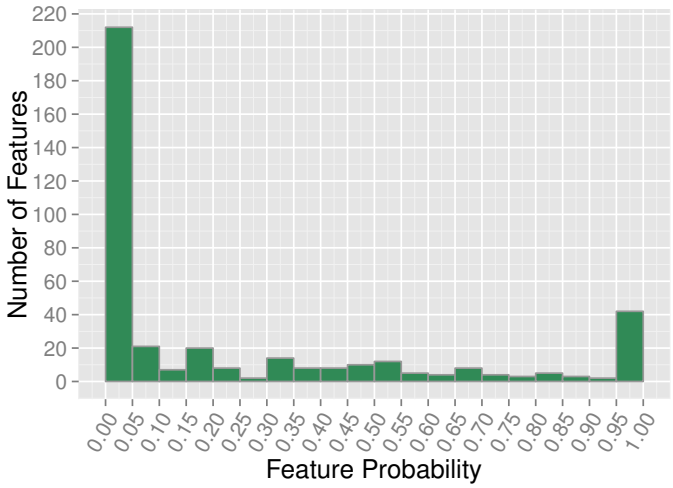


Figure 6.2: Histogram of feature probabilities for the Reanult Megane example

Figure 6.3 shows how the number of incompatible feature grows drastically with a small sensitivity increase. According to the rigid version of feature incompatibility, there are 33 features incompatible with at least 50% of the remaining ones (8.29% of the features). When sensitivity is set to 0.5, that number becomes 380 (95.48% of the features).

Table 6.3 summarizes the aforementioned outcomes, highlighting the benefits of using our flexible approach.

Rigid Measures (sensitivity=0)		Flexible Measures (sensitivity=0.05)	
Dead features	1.51%	53.27%	Features with reuse probability ≤ 0.05
Core features	1.76%	10.55%	Features with reuse probability ≥ 0.95
Features incompatible with more than 50% of the remaining features	8.29%	95.48%	Features incompatible in 95% of the valid products with more than 50% of the remaining features

Table 6.3: Comparison of rigid versus flexible outcomes for the Reanult Megane example

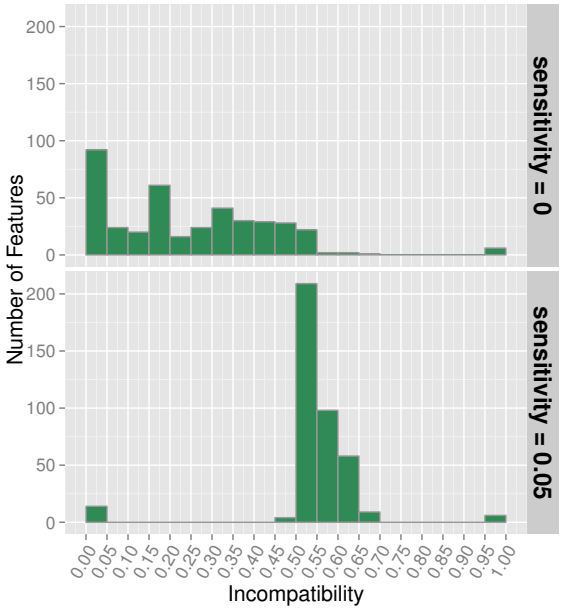


Figure 6.3: Histogram of feature incompatibilities for the Reanult Megane example

6.3 Our approach for user guidance

This section reports the results of several empirical experiments carried out to test the validity of our approach for user guidance. In particular, the goal of this section is to check if:

- Our approach produces better results than related work.
- The algorithm presented in Section 4.1.2 provides reasonable response times and thus support customer interactivity during the derivation process.

6.3.1 Experimental design

To test the validity of our approach, the following two case studies have been used:

1. The variability model provided by the car manufacturing company Renault DVI, presented in previous sections. This model has been selected because it illustrates the practical applicability of our approach (i.e., instead of using an example made up for academic purposes, our work is tested on a real configuration model that comes from the industry).

2. The Electronic Shopping model provided by Lau [Lau06], which deals with an electronic commerce platform that allows consumers to directly buy goods or services from a seller over the Internet using a web browser. This benchmark is widely used by the software product line community [Men09] [BG11] [POS+12].

On the other hand, we have created a test bed composed of 1000 random products for every variability model. As we will see, a sample of 1000 products is big enough to get results with high statistical power and significance.

To get efficient BDD sizes, the directions given by Narodytska et al. [NW07] have been followed. The BuDDy package has been used to guarantee the generation of valid products (i.e., derivatives that conform to the BDD).

The test bed is used to compare the following methods:

1. Mazo et al.'s Heuristics 1, 2 and 5[1].

2. *Probability* driven approach, i.e., the method proposed by Chen et al. and Mazo et al. (Heuristic 3).

3. *Entropy* driven approach, i.e., the method we propose in Chapter 5.

To compute the feature probabilities, which are required by the *entropy* and *probability* approaches, an implementation of the algorithm presented in Section 4.1 has been developed as an extension of the BuDDy package. All tests were conducted on an Intel© Core™ 2 i3-4010U with 1.7 GHz and 4GB RAM (although only one core was used).

6.3.2 Case study 1: Renault Megane

6.3.2.1 Results

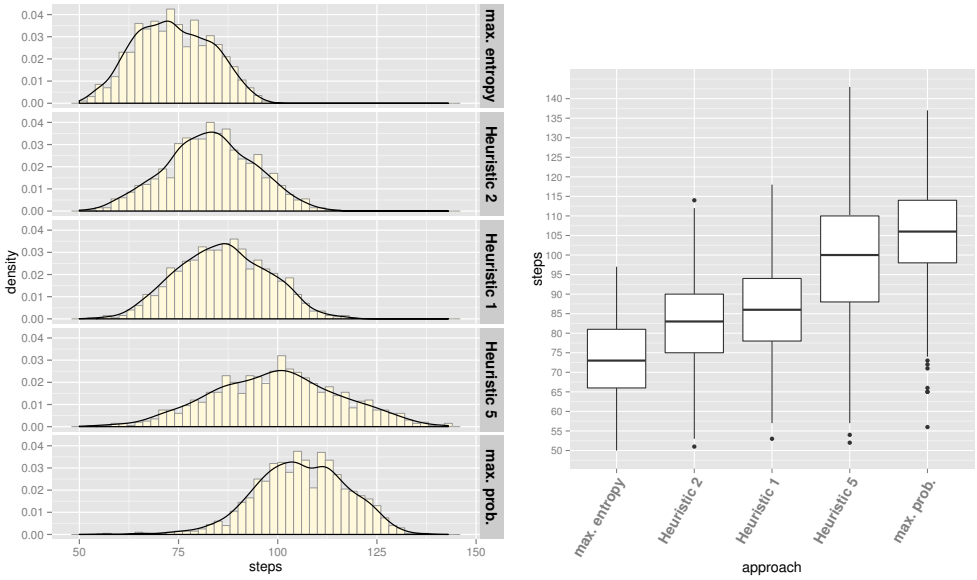
Table 6.4 summarizes the results of the experiments for the Renault Megane variability model. Histograms in Figure 6.4.a represent the number of steps needed to derive the 1000 products using Mazo et al.'s Heuristics 1, 2 and 5, and the *entropy* and *probability*

[1] remember that, strictly speaking, Mazo et al.'s Heuristic 4 is not a *heuristic*, but a propagation mechanism that all derivation systems should support. So we have included such mechanism in all the methods tested in this Section.

approaches. Figure 6.4.b complements the histogram representation of the results with a box plot[1].

approach	mean	std. deviation	median	min	max	range
entropy	73.49	9.5	73	50	97	47
probability	105.79	11.54	106	56	137	81
Heuristic 1	86.04	11.26	86	53	118	65
Heuristic 2	82.74	11.12	83	51	114	63
Heuristic 5	99.39	15.95	100	52	143	91

Table 6.4: Results for the Renault Megane example



(a) Histograms (b) Box plots

Figure 6.4: Number of derivation steps according to the used approach for the Renault Megane example

Using the Central Limit Theorem, the 95% Confidence Intervals (CI) of the population mean can be estimated (i.e., the range where, with a 95% guarantee, the mean of the number of steps required to derive every product of the Megane model lies). Table 6.5 summarizes

[1] "whiskers" in Figure 6.4.b start from the edge of the box and extend to the furthest data point that is within 1.5 times the inter-quartile range (i.e., the range that goes from the 25th percentile to the 75th percentile). Points that are past the ends of the whiskers have been considered outliers and displayed with dots.

the CIs for each approach, which are estimated as population mean CI = sample mean ± t(std. error, 95%, 999 degrees of freedom), where t stands for the Student's t-distribution.

entropy		probability		Heuristic 1		Heuristic 2		Heuristic 5	
std. error	95% CI	std. error	95% CI	std. error	95% CI	std. error	95% CI	std. error	95% CI
0.03	70.90-74.08	0.36	105.07-106.50	0.36	85.35-86.74	0.35	82.05-83.43	0.5	98.40-100.38

Table 6.5: 95% CI of the population mean the Renault Megane example

According to the summarized data, there is experimental evidence supporting that our approach produces better results than related work.

6.3.2.2 Statistical significance

To check the statistical significance of the results, an Analysis of Variance (ANOVA) test has been run on the experimental data. Table 6.6 summarizes the ANOVA outcomes. Since the p-value is less than 0.001 (in particular, p-value $< 2 \cdot 10^{-16}$), the experimental results are statistically highly significant.

	degrees of freedom	sum of squares	mean of squares	F-value	Pr($> F$)
approaches	4	676884	169221	1162	$< 2 \cdot 10^{-16}$
residuals	4995	727312	146		

Table 6.6: ANOVA test for the Renault Megane example

Table 6.7 summarizes the power analysis of the ANOVA test. Given the sample size and the high effect size (i.e., the high values of η^2 and Cohen's f^2), the ANOVA test has high statistical power.

effect size		power
eta squared η^2	Cohen's f^2	
0.48	0.93	≈ 1

Table 6.7: Power analysis for the Renault Megane example

Finally, to check the statistical significance of the pairwise comparison between the approaches, a Tukey Honest Significant Differences (HSD) has been run. According to the

results, summarized in Table 6.8, the difference between the number of steps required by any pair of approaches to derive a product is statistically highly significant[1].

	difference	95% CI	adjusted p-value
Heuristic 2 *vs* entropy	9.25	7.78-10.72	≈ 0
Heuristic 1 *vs* entropy	12.56	11.08-14.02	≈ 0
Heuristic 5 *vs* entropy	25.89	24.42-27.37	≈ 0
probability *vs* entropy	32.29	30.82-33.77	≈ 0
Heuristic 1 *vs* Heuristic 2	3.30	1.83-4.77	≈ 0
Heuristic 5 *vs* Heuristic 2	16.65	15.17-18.12	≈ 0
probability *vs* Heuristic 2	23.04	21.57-24.52	≈ 0
Heuristic 5 *vs* Heuristic 1	13.34	11.87-14.82	≈ 0
probability *vs* Heuristic 1	19.74	18.27-21.22	≈ 0
probability *vs* Heuristic 5	6.40	4.93-7.87	≈ 0

Table 6.8: Tukey HSD test for the Renault Megane example

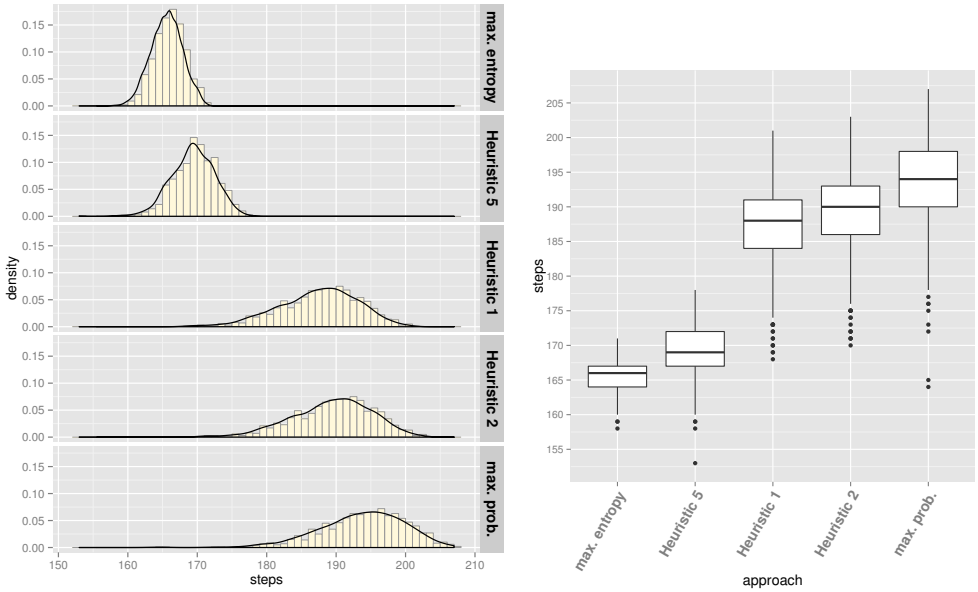
6.3.3 Case study 2: Electronic Shopping

Table 6.9 and Figure 6.5 summarize the results of the experiments for the Electronic Shopping variability model. Table 6.10 summarizes the CIs for each approach. According to the outcomes, there is experimental evidence supporting that our approach produces better results than related work.

approach	mean	std. deviation	median	min	max	range
entropy	165.57	2.23	166	158	171	13
probability	193.67	6.06	194	164	207	43
Heuristic 1	187.38	5.69	188	168	201	33
Heuristic 2	189.36	5.7	190	170	203	33
Heuristic 5	169.33	3.1	169	153	178	25

Table 6.9: Results for the Electronic Shopping example

[1] whereas the ANOVA test rejects the null hypothesis: "there is no difference between the five approaches (i.e., all of them produce approximately the same results)", Tukey HSD test rejects ten null hypotheses separately: "there is no difference between the Heuristic 2 and the entropy approach", "there is no difference between Heuristic 1 and the entropy approach, etc.".



(a) Histograms (b) Box plots

Figure 6.5: Number of derivation steps according to the used approach for the Electronic Shopping example

entropy		probability		Heuristic 1		Heuristic 2		Heuristic 5	
std. error	**95% CI**	**std. error**	**95% CI**	**std. error**	**95% CI**	**std. error**	**95% CI**	**std. error**	**95% CI**
0.07	165.43-165.71	0.19	193.29-194.04	0.18	187.03-187.73	0.18	189.01-189.72	0.1	169.14-169.52

Table 6.10: 95% CI of the population mean for the Electronic Shopping example

6.3.3.1 Statistical significance

Table 6.11 summarizes the ANOVA outcomes. Since the p-value is less than 0.001 (in particular, p-value $< 2 \cdot 10^{-16}$), the experimental results are statistically highly significant. Table 6.12 summarizes the power analysis of the ANOVA test. Given the sample size and the high effect size, the ANOVA test has high statistical power. Finally, Table 6.13 summarizes the outcomes of HSD, which show that the difference between the number of steps required by any pair of approaches to derive a product is statistically highly significant.

	degrees of freedom	**sum of squares**	**mean of squares**	**F-value**	**Pr($> F$)**
approaches	4	645314	161328	6950	$< 2 \cdot 10^{-16}$
residuals	4995	115944	23		

Table 6.11: ANOVA test for the Electronic Shopping example

effect size		**power**
eta squared η^2	**Cohen's f^2**	
0.85	5.57	≈ 1

Table 6.12: Power analysis for the Electronic Shopping example

	difference	**95% CI**	**adjusted p-value**
Heuristic 5 *vs* entropy	3.76	3.17-4.35	≈ 0
Heuristic 1 *vs* entropy	21.81	21.22-22.40	≈ 0
Heuristic 2 *vs* entropy	23.79	23.21-24.38	≈ 0
probability *vs* entropy	28.09	27.51-28.68	≈ 0
Heuristic 1 *vs* Heuristic 5	18.05	17.46-18.64	≈ 0
Heuristic 2 *vs* Heuristic 5	20.03	19.45-20.62	≈ 0
probability *vs* Heuristic 5	24.33	23.75-24.92	≈ 0
Heuristic 2 *vs* Heuristic 1	1.98	1.39-2.57	≈ 0
probability *vs* Heuristic 1	6.28	5.69-6.87	≈ 0
probability *vs* Heuristic 2	4.30	3.71-4.89	≈ 0

Table 6.13: Tukey HSD test for the Electronic Shopping example

6.4 Threats to Validity

The following points summarize the main threats to the validity of our work, and the strategies we have followed to try to overcome them:

1. **Bibliometric analysis**. The first one is referred to our bibliometric analysis presented in Chapter 3, and it is composed of the following main threats:

   - Bibliographic database. Although there are several freely available databases, such as Google Scholar (GS), CiteSeerX, Microsoft Academic Search, getCITED, etc., the paid subscription databases ISIWoS and Scopus are currently the most reliable [GJGJMO14, VG09].

     By Combining data from ISIWoS and Scopus, we have tried to achieve a good balance between:

     – *Retrospective coverage*: ISIWoS offers the most complete retrospective quality coverage for all scientific disciplines, which is specially appropriated for the rigorous science mapping analysis carried out in this thesis.

     – *Conference coverage*: the scientific community in Computer Science is highly driven by conferences, which are inadequately covered by ISIWoS. In contrast, Scopus includes most of the key conferences in SPLs, such as the International Workshop on Variability Modelling of Software-intensive System (VAMOS).

   - Keyword standardization. If the input to the simple centers algorithm included too many keywords, the output might be hard to interpret: there would be a high number of clusters, composed of many keywords interrelated with low equivalence indices. To overcome this problem, the standardization described in Section 3.1.1 was performed.

   - Period setting and parametrization of the simple centers algorithm. To get good performance in the analysis of co-citation clusters over consecutive periods, a balanced period length is needed: short enough to prevent smoothing excessively the data and long enough to include sufficient publications for the analysis [CLHHVH11a]. In our work, we decided to group the publications in periods of

five years, as [KUU10], and [CLHHVH11a] also do in similar science mapping analyses.

To ease the review of our work, the records retrieved from ISIWoS, all the raw keywords included in such records, the standardized keywords, a description on how keywords were grouped, and the sources where the papers where published are available on:

http://rheradio.github.io/SPL-Bib-Anlys/

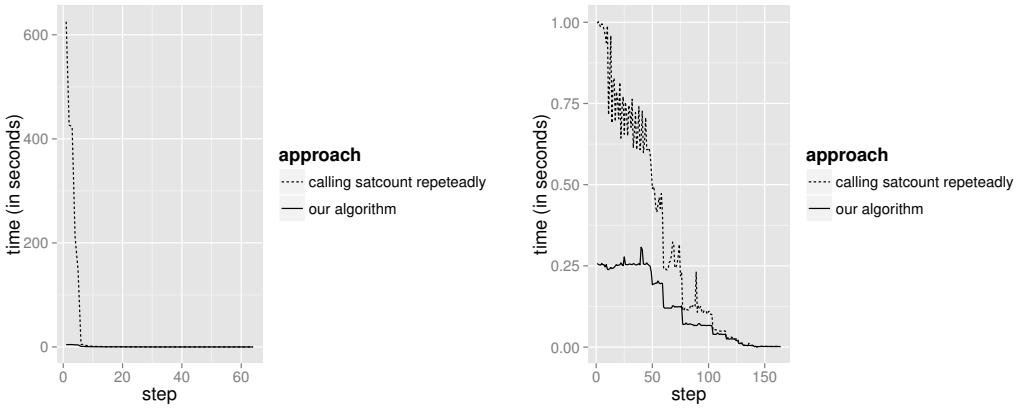
2. **BDD encoding**. The second thread refers to the starting point of all our approaches: The synthesis of a BDD for a variability model. Unfortunately, synthesising complex models may be extremely hard because the size of the BDDs highly depends on the chosen variable ordering [Bry86], and it has been already mentioned throughout this thesis it is a well-known fact that finding an optimal ordering is an NP-complete problem [BW96]. It has also been already mentioned that providing a heuristic to find a good variable ordering is out of the scope of this thesis. Nevertheless, some heuristics specifically designed to deal with variability models are available. For instance, Mendonça [Men09] proposes a static heuristic for feature models which has been proven to work efficiently for the models included in the SPLOT repository. Static heuristics produce a ordering which remains invariable during the whole course of the BDD construction. It is worth noting that this kind of heuristic sometimes is not appropriate since the optimal ordering at the end may be different at the beginning of the BDD building [MT98]. In such cases, it is better to improve the variable ordering dynamically as the BDD is being built. Narodytska et al. [NW07] propose three heuristics for reducing the time and space required to built a BDD for a variability model. The first heuristic attempts to limit the growth in the size of the BDD by providing an ordering in which constraints are added to the decision diagram; the second heuristic provides an initial ordering for the variables within the decision diagram; and the third heuristic groups variables together so that they can be reordered by a dynamic variable reordering procedure.

3. **Product derivation**. A threat to the validity of our approach for guiding the derivation process is the time required to compute the feature probabilities[1]. For the sake of

[1] which is also the Achilles' heel for Chen et al.'s approach and Mazo et al.'s Heuristic 3.

interactivity, configurators must provide customer guidance in a short-time and the usual way to compute the probabilities is highly time-consuming (see Section 3.2.1). To assess the response time of our algorithm (see Section 4.1), we determined the time needed to derive 1000 randomly generated products for the case studies 1 and 2 using our entropy-driven approach. Figure 6.6 compares the average times needed to completely derive the products by computing the feature probabilities using Algorithm 7, and calling repeatedly the BuDDy function *satcount*, i.e., calling Algorithm 1.

As Figure 6.6 shows, our algorithm greatly improves the probability computation time. For instance, it requires 4.54 seconds on average to compute all feature probabilities (and thus their entropy values) for the first configuration step in the Renault Megane example. In contrast, calling *satcount* repeatedly consumes 625.18 seconds.



(a) Renault Megane (b) Electronic Shopping

Figure 6.6: Time required to compute feature probabilities

Note that the first product steps are the most expensive in time. As the derivation process advances, the derivation space gets reduced and so the time needed to compute the probabilities. There is a point where both approaches converge and get response times close to 0.

CHAPTER 7

Conclusions and Future Work

This chapter sums up the main contributions of this thesis and opens the door for future research based on the ideas proposed throughout this dissertation. This final chapter is organised as follows: Section 7.1 provides the concluding remarks of the thesis. Finally, Section 7.2 presents research opportunities based on the ideas proposed in our work.

7.1 Summary

In this dissertation, research on automated analysis of variability models and product derivation in SPLs have been described. The first stage of this thesis was identifying the most interesting and impacting areas of work. To do so, a bibliometric analysis of the literature on SPLs was carried out. In particular, thanks to science mapping, the main researched topics, the evolution of the interest in those topics and the relationships among them have been identified. On the one hand, it has been detected that software architecture was the initial motor of research in SPLs, and the work on software systematic reuse has also been essential for the development of the area. On the other hand, feature modeling has been the most important topic for the last fifteen years, having the best evolution behaviour in terms of number of published papers and received citations. In addition, as we pointed out in Chapter 3, the

automated analysis of variability models and product derivation have gained importance becoming two of the most researched topics in the area for the period 2010-2014. As a result, we decided to work in these research topics.

Automated Analysis

The development and maintenance of SPLs require the management of complex variability models. The role each feature plays in such models is unclear to the naked eye and so automated support is needed to identify which features are essential, dispensable, highly required by other features and highly incompatible with the remaining features. We have exposed the drawbacks of existing approaches to provide that support, and we have shown that those strategies have poor time performance and so they hinder user interactivity. To overcome the scalability limitations of related work, we have proposed some algorithms for these important measures, based on the glass-box reuse of BDD libraries. As we have shown, our approaches directly interacts with the data structure of the BDD that encodes a variability model, and thus they exhibit better time-performance than related work, both theoretically and experimentally. In particular, the experimental validation reported in Chapter 6 reveals that our approaches provide a relative improvement of approximately 93% w.r.t related work.

Moreover, we have identified that current measures to detect essential, dispensable, highly required features are too rigid. Due to the rigidness of these measures to account for feature interrelations, relevant information is frequently overlooked. According to our experimental validation, the number of dead features for the Renault Megane benchmark is the 1.51%. Nevertheless, it grows until 53.27 % at sensitivity 0.05. In other words, more than half of the features can be reused at most in just 5 percent of the valid derivatives. Furthermore, it has been shown, both theoretically and experimentally, that our algorithms not only can take into account different levels of sensitivity, but also are more time-efficient than related work even for computing the rigid measures. In particular, the experimental validation reveals that our approaches provide a relative improvement of approximately 73% w.r.t related work.

Product Derivation

As already mentioned, the configuration of all but trivial derivatives involves considerable effort in selecting which features they should include, while avoiding violations of the inter- feature dependencies and incompatibilities. Our approach enriches existing automated configurators by reducing the number of steps required to derive a valid product. Applying

the Information Theory concept of entropy, our approach takes advantage of the fact that, due to the inter- features relations, some decisions may be automatically derived from other decisions previously made. So, the order in which decisions are made has a strong influence on the number of decisions required to complete a valid product. Moreover, our approach does not provide a static ordering that the customer is forced to follow. On the contrary, it suggests orderings dynamically, reacting to the customer decisions. In addition, we have used the algorithm that efficiently computes the variable probabilities of a boolean formula introduced in Chapter 4, supporting this way not only our approach but also other methods proposed in related work.

The Renault Megane and the Electronic Shopping configuration benchmarks have been used to test the applicability of our approach and its effectiveness. In particular, it has been shown that our approach produces better results than related work:

- It requires 30.52%, 14.58%, 11.18% and 26.06% less steps than Chen et al.'s approach, Heuristic 1, Heuristic 2 and Heuristic 5 for the Renault Megane test, respectively.
- It requires 14.51%, 11.64%, 12.56% and 2.60% less steps than Chen et al.'s approach, Heuristic 1, Heuristic 2 and Heuristic 5 for the Electronic Shopping test, respectively.

7.2 Future Work

In the following points, we discuss about research opportunities to extend the ideas proposed in this dissertation:

- It is well-known that some products are more demanded by customers than other ones. However, our heuristic to guide product derivation does not take into account such information. So, a redefinition of our approach that deals with additional information available form previously configured products might be developed.
- It is well-known that BDD size strongly depends on BDD ordering heuristic. It would be convenient to develop an R wrapper for the most popular BDD libraries, i.e, BuDDy and Cudd[1]. This implementation would be useful to get and analyze the effects that different input parameters, such as variable ordering or constraint ordering, have on the BDD size.

[1] http://vlsi.colorado.edu/~fabio/CUDD/

- As already mentioned, all of our heuristic have been implemented as an extension of the BuDDy library. However, it would be convenient to implement them in other popular bdd libraries such as Cudd, javaBdd[1], etc in order to seize other ordering heuristics supported by those libraries.

[1] http://javabdd.sourceforge.net/

Bibliography

[AC08] Faheem Ahmed and Luiz Fernando Capretz. The software product line architecture: An empirical investigation of key process activities. *Information and Software Technology*, (11):1098–1113, 2008. 28

[ACLF09] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. Composing feature models. In *Software Language Engineering Conference*, pages 62–81, Denver, CO, USA, October 2009. 30

[ACLF13] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming*, (6):657 – 681, 2013. 30

[ADCBZ09] P. Abate, R. Di Cosmo, J. Boender, and S. Zacchiroli. Strong dependencies between software components. In *3rd International Symposium on Empirical Software Engineering and Measurement*, pages 89–99, Buena Vista, Florida, USA, October 2009. 41

[ADH+00] M. Ardis, N. Daley, D. Hoffman, H. Siy, and D. Weiss. Software product lines: a case study. *Software - Practice & Experience*, (7):825–847, June 2000. 27

[AFM02] Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis. Consistency restoration and explanations in dynamic csps-application to configuration. *Artificial Intelligence*, 135(1-2):199–234, 2002. 89

BIBLIOGRAPHY

[AGM+06] Vander Alves, Rohit Gheyi, Tiago Massoni, Uirá Kulesza, Paulo Borba, and Carlos Lucena. Refactoring product lines. In *Int. Conference on Generative Programming and Component Engineering*, pages 201–210, New York, NY, USA, 2006. 28

[AKGL10] Sven Apel, Christian Kästner, Armin Gröβlinger, and Christian Lengauer. Type safety for feature-oriented product lines. *Automated Software Engineering*, (3):251–300, September 2010. 30

[AMS07] Timo Asikainen, Tomi Männistö, and Timo Soininen. Kumbang: A domain ontology for modelling variability in software product families. *Advanced Engineering Informatics*, (1):23–40, 2007. 28

[ANAc10] Vander Alves, Nan Niu, Carina Alves, and George Valença. Requirements engineering for software product lines: A systematic literature review. *Information and Software Technology*, (8):806– 820, 2010. 30

[ATLS08] Sven Apel, T. Thomas Leich, and Gunter Saake. Aspectual feature modules. *IEEE Transactions on Software Engineering*, (2):162–180, March 2008. 28

[Bak13] Kacper Bak. *Modeling and Analysis of Software Product Line Variability in Clafer*. PhD thesis, University of Waterloo, 2013. 86

[Bat05] Don Batory. Feature models, grammars, and propositional formulas. In *Software Product Line Conference*, pages 7–20, Rennes, France, September 2005. 11, 28, 63

[BBG+10] Marko Boskovic, Ebrahim Bagheri, Dragan Gasevic, Bardia Mohabbati, Nima Kaviani, and Marek Hatala. Automated staged configuration with semantic web technologies. *International Journal of Software Engineering and Knowledge Engineering*, (4):459–484, June 2010. 30

[BBMY04] B. Boehm, A.W. Brown, R. Madachy, and Ye Yang. A software product line life cycle cost estimation model. In *International Symposium on Empirical Software Engineering*, pages 156–164, Redondo Beach, CA, USA, Aug 2004. 27

[BBRC06] Don Batory, David Benavides, and Antonio Ruiz-Cortes. Automated analysis of feature models: Challenges ahead. *Commun. ACM*, (12):45–47, December 2006. 28

[BBS10] Jan Bosch and Petra Bosch-Sijtsema. From integration to composition: On the impact of software product lines, global development and ecosystems. *Journal of Systems and Software*, (1):67–76, 2010. 30

[BC05] Felix Bachmann and Paul C. Clements. Variability in software product lines. Technical report, Software Engineering Institute. CMU/SEI-2005-TR-012, 2005. 21

[BCB03] Katy Börner, Chaomei Chen, and Kevin W. Boyack. Visualizing knowledge domains. *Annual Review of Information Science and Technology*, (1):179–255, 2003. 20

[BCM+04] Gunter Böckle, Paul Clements, John D. McGregor, Dirk Muthig, and Klaus Schmid. Calculating roi for software product lines. *IEEE Software*, (3):23–31, May 2004. 27

[BFL13] Christian Bessiere, Hélène Fargier, and Christophe Lecoutre. Global inverse consistency for interactive constraint satisfaction. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 159–174. Springer, 2013. 89

[BG11] Ebrahim Bagheri and Dragan Gasevic. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, (3):579–612, 2011. 30, 92

[BHJ+03] A. Birk, G. Heller, I. John, K. Schmid, T. von der Massen, and K. Muller. Product line engineering, the state of the practice. *IEEE Software*, (6):52–60, November 2003. 27

[BHvM+09] Armin Biere, Marijn J.H. Heule, Hans van Maaren, Toby, and Walsh. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009. 35

[BLP05] S. Buhne, K. Lauenroth, and K. Pohl. Modelling requirements variability across product lines. In *International Conference on Requirements Engineering*, pages 41–50, Paris, France, Aug 2005. 28

[Boe11] Jaap Boender. *A formal study of Free Software Distributions*. PhD thesis, Ecole doctorale de Sciences Mathematiques de Paris Centre. Universite Paris Diderot, March 2011. 33, 34, 41, 42, 50

[BPSP04] Danilo Beuche, Holger Papajewski, and Wolfgang Schröder-Preikschat. Variability management with feature models. *Science of Computer Programming*, (3):333–352, 2004. 27

[BRN+13] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. A survey of variability modeling in industrial practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS' 13, pages 7:1–7:8, New York, NY, USA, 2013. ACM. 86

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986. 13, 16, 42, 57, 67, 99

[BSL+10] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wasowski, and Krzysztof Czarnecki. Variability modeling in the real: A perspective from the operating systems domain. In *International Conference on Automated Software Engineering*, pages 73–82, New York, NY, USA, 2010. 30, 42

[BSL+13] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki. A study of variability models and languages in the systems software domain. *IEEE Transactions on Software Engineering*, 39(12):1611–1640, Dec 2013. 2, 42

[BSRC10] David Benavides, Sergio Segura, and Antonio Ruiz-Cortes. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, (6):615–636, 2010. 2, 11, 30, 33, 50

[BSTRC07] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortes. Tooling a framework for the automated analysis of feature models. In *1st Int. Workshop on Variability Modelling of Softw. Intensive Syst.*, Limerick, Ireland, 2007. 37, 38

[BTRC05] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortes. Automated reasoning on feature models. In *International Conference on Advanced Information Systems Engineering*, Porto. Portugal, June 2005. 28

[BW96] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-Complete. *Trans. on Computers*, 45:993–1002, Sept. 1996. 15, 99

[CB11] Lianping Chen and Muhammad Ali Babar. A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology*, (4):344–362, 2011. 30

[CBH11] Andreas Classen, Quentin Boucher, and Patrick Heymans. A text-based approach to feature modelling: Syntax and semantics of tvl. *Science of Computer Programming*, (12):1130 – 1143, 2011. 30

[CCL91] M. Callon, J.P. Courtial, and F. Laville. Co-word analysis as a tool for describing the network of interactions between basic and technological research: The case of polymer chemsitry. *Scientometrics*, (1):155–205, 1991. 23, 24

[CE11] Sheng Chen and M. Erwig. Optimizing the product derivation process. In *15th International Software Product Line Conference*, pages 35–44, Munich, Germany, 2011. IEEE Computer Society. 46, 48

[CHE05a] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, (1):7–29, 2005. 28

[CHE05b] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, (2):143–169, 2005. 28

[CHS+10] Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, and Jean-François Raskin. Model checking lots of systems: Efficient verification of temporal properties in software product lines. In *International Conference on Software Engineering*, pages 335–344, New York, NY, USA, 2010. 30

[CHSL11] Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, and Axel Legay. Symbolic model checking of software product lines. In *International Conference on Software Engineering*, pages 321–330, New York, NY, USA, 2011. 30

[CHU04] Krzysztof Czarnecki, Simon Helsen, and Eisenecker Ulrich. Staged configuration using feature models. In *Software Product Line Conference*, pages 266–283, Boston, USA, September 2004. 27

[CHW98] James Coplien, Daniel Hoffman, and David Weiss. Commonality and variability in software engineering. *IEEE Software*, (6):37–45, November 1998. 26

[CLHHVH11a] M.J. Cobo, A.G. Lopez-Herrera, E. Herrera-Viedma, and F. Herrera. An approach for detecting, quantifying, and visualizing the evolution of a research field: A practical application to the fuzzy sets theory field. *Journal of Informetrics*, (1):146–166, 2011. 23, 24, 98, 99

[CLHHVH11b] M.J. Cobo, A.G. Lopez-Herrera, E. Herrera-Viedma, and F. Herrera. Science mapping software tools: Review, analysis, and cooperative study among tools. *Journal of the American Society for Information Science and Technology*, (7):1382–1402, 2011. 20

[CLHHVH12] M.J. Cobo, A.G. Lopez-Herrera, E. Herrera-Viedma, and F. Herrera. Scimat: A new science mapping analysis software tool. *Journal of the American Society for Information Science and Technology*, (8):1609–1630, 2012. 23

[CLR86] Michel Callon, John Law, and Arie Rip. *Mapping the Dynamics of Science and Technology*, chapter Qualitative Scientiometrics, pages 103–123. 1986. 24

[CMK98] Neal Coulter, Ira Monarch, and Suresh Konda. Software engineering as seen through its research literature: A study in co-word analysis. *Journal of the American Society for Information Science*, (13):1206–1223, 1998. 23

[CN01] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. 2001. 2

[CO08] Hadrien Cambazard and Barry O'Sullivan. Reformulating positive table constraints using functional dependencies. In *14th International Conference on Principles and Practice of Constraint Programming*, pages 418–432, Sydney, Australia, 2008. Springer. 89

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM. 12, 35

[CP06] Krzysztof Czarnecki and Krzysztof Pietroszek. Verifying feature-based model templates against well-formedness ocl constraints. In *Int. Conference on Generative Programming and Component Engineering*, pages 211–220, New York, NY, USA, 2006. 28

[CPRS04] V. Cechticky, A. Pasetti, O. Rohlik, and W. Schaufelberger. Xml-based feature modelling. In *International Conference on Software Reuse: Methods, Techniques and Tools*, pages 101–114, Madrid, Spain, July 2004. 27

[CW07] K. Czarnecki and A. Wasowski. Feature diagrams and logics: There and back again. In *Software Product Line Conference*, pages 23–34, Kyoto, Japan, September 2007. 28

[CZZM05] Kun Chen, Wei Zhang, Haiyan Zhao, and Hong Mei. An approach to constructing feature models based on requirements clustering. In *International Conference on Requirements Engineering*, pages 31–40, Paris, France, Aug 2005. 28

[DCB10] Roberto Di Cosmo and Jaap Boender. Using strong conflicts to detect quality issues in component-based complex systems. In *Proceedings of the 3rd India Software Engineering Conference*, ISEC '10, pages 163–172, New York, NY, USA, 2010. ACM. 41, 50

[DDH+13] Jean-Marc Davril, Edouard Delfosse, Negar Hariri, Mathieu Acher, Jane Cleland-Huang, and Patrick Heymans. Feature model extraction from large collections of informal product descriptions. In *Joint Meeting on Foundations of Software Engineering*, pages 290–300, New York, NY, USA, 2013. 30

[DGR11] Deepak Dhungana, Paul Grünbacher, and Rick Rabiser. The dopler meta-tool for decision-oriented variability modeling: A multiple case study. *Automated Software Engineering*, (1):77–114, March 2011. 30, 46

[DGRN10] Deepak Dhungana, Paul Grünbacher, Rick Rabiser, and Thomas Neumayer. Structuring the modeling space and supporting evolution in software product line engineering. *Journal of Systems and Software*, (7):1108–1122, 2010. 30

[DKO+97] D. Dikel, D. Kane, S. Ornburn, W. Loftus, and J. Wilson. Applying software product-line architecture. *Computer*, (8):49–55, August 1997. 26

[dlT92] Thierry Boy de la Tour. An optimality result for clause form translation. *Journal of Symbolic Computation*, 14(4):283–301, 1992. 12

[dMSNdCMM+11] Paulo Anselmo da Mota Silveira Neto, Ivan do Carmo Machado, John D. McGregor, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. A systematic mapping study of software product lines testing. *Information and Software Technology*, (5):407 – 423, 2011. 30

[DSB04] Sybren Deelstra, Marco Sinnema, and Jan Bosch. Experiences in software product families: Problems and issues during product derivation. In *Software Product Line Conference*, pages 165–182, Boston, MA, USA, August 2004. 27

[DYS11] Ming Dong, Dong Yang, and Liyue Su. Ontology-based service product configuration system modeling and development. *Expert Systems with Applications*, (9):11770 – 11786, 2011. 30

[ER11] Emelie Engstrom and Per Runeson. Software product line testing - a systematic mapping study. *Information and Software Technology*, (1):2–13, 2011. 30

[FAHCC14] D. Fernandez-Amoros, R. Heradio, J.A Cerrada, and C. Cerrada. A scalable approach to exact model and commonality counting for extended feature models. *IEEE Transactions on Software Engineering*, (9):895–910, Sept 2014. 33, 35

[FCS+08] Eduardo Figueiredo, Nelio Cacho, Claudio Sant'Anna, Mario Monteiro, Uira Kulesza, Alessandro Garcia, Sérgio Soares, Fabiano Ferrari, Safoora Khan, Fernando Castor Filho, and Francisco Dantas. Evolving software product lines with aspects: An empirical study on design stability. In *International Conference on Software Engineering*, pages 261–270, Leipzig, Germany, May 2008. 28

[FKC12] Rick Flores, Charles Krueger, and Paul Clements. Mega-scale product line engineering at general motors. In *16th International Software Product Line Conference*, pages 259–268, New York, NY, USA, 2012. ACM. 37

[FUB06] Dario Fischbein, Sebastian Uchitel, and Victor Braberman. A foundation for behavioural conformance in software product line architectures. In *Workshop on Role of Software Architecture for Testing and Analysis*, pages 39–48, New York, NY, USA, 2006. 28

[FV03] D. Faust and C. Verhoef. Software product line migration and deployment. *Software - Practice & Experience*, (10):933–955, August 2003. 27

[GA01] Critina Gacek and Michalis Anastasopoules. Implementing product line variabilities. In *Symposium on Software reusability: putting software reuse in context*, New York, NY, USA, May 2001. 27

BIBLIOGRAPHY

[Gan12] Graeme Keith Gange. *Combinatorial Reasoning for Sets, Graphs and Document Composition*. PhD thesis, Department of Computing and Information Systems. The University of Melbourne, 2012. 89

[GJGJMO14] Valentin Gomez-Jauregui, Cecilia Gomez-Jauregui, Cristina Manchado, and Cesar Otero. Information management and improvement of citation indices. *International Journal of Information Management*, (2):257 – 271, 2014. 20, 98

[Gri00] M. L. Griss. Implementing product-line features by composing aspects. In *Software Product Line Conference*, pages 271–288, Denver, Colorado, August 2000. 27

[GS03] Jack Greenfield and Keith Short. Software factories: Assembling applications with patterns, models, frameworks and tools. In *ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, pages 16–27, New York, NY, USA, 2003. 27

[GWT+14] M. Galster, D. Weyns, D. Tofan, B. Michalik, and P. Avgeriou. Variability in software systems - a systematic literature review. *IEEE Transactions on Software Engineering*, 40(3):282–306, March 2014. 30

[GWW+11] Jianmei Guo, Jules White, Guangxin Wang, Jian Li, and Yinglin Wang. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software*, (12):2208–2221, 2011. 30

[HBG11] A. Hervieu, B. Baudry, and A. Gotlieb. Pacogen: Automatic generation of pairwise test configurations from feature models. In *International Symposium on Software Reliability Engineering*, pages 120–129, Hiroshima, Japan, Nov 2011. 30

[He99] Quin He. Knowledge discovery through co-word analysis. *Library Trends*, (1):133–159, 1999. 24

[HFACC11] R. Heradio, D. Fernandez-Amoros, J.A. Cerrada, and C. Cerrada. Supporting commonality-based analysis of software product lines. *IET Software*, (6):496 –509, dec. 2011. 38

[HFAPMA14] Ruben Heradio, David Fernandez-Amoros, Hector Perez-Morago, and Antonio Adan. Speeding up derivative configuration from product platforms. *Entropy*, (6):3329–3356, 2014. 11

[HHOW05] Emmanuel Hebrard, Brahim Hnich, Barry O'Sullivan, and Toby Walsh. Finding diverse and similar solutions in constraint programming. In *20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference*, Pittsburgh, Pennsylvania, USA, 2005. AAAI Press / The MIT Press. 89

[HHPS08] Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and K. Schmid. Dynamic software product lines. *Computer*, (4):93–95, April 2008. 28

[HHRV11] Abel Hegedus, Akos Horvath, Istvan Rath, and Daniel Varro. A model-driven framework for guided design space exploration. In *26th International Conference on Automated Software Engineering*, pages 173–182, Washington, DC, USA, 2011. IEEE Computer Society. 46

[HKW08] Florian Heidenreich, Jan Kopcsek, and Christian Wende. FeatureMapper: Mapping Features to Models. In *International Conference on Software Engineering*, pages 943–944, New York, NY, USA, May 2008. 28

[HPMFA+16] Ruben Heradio, Hector Perez-Morago, David Fernandez-Amoros, Francisco Javier Cabrerizo, and Enrique Herrera-Viedma. A bibliometric analysis of 20 years of research on software product lines. *Information and Software Technology*, 72:1 – 15, 2016. 22

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. 2004. 13, 37

BIBLIOGRAPHY

[HSVM00] Andreas Hein, Michael Schlick, and Renato Vinga-Martins. Applying feature models in industrial settings. In Patrick Donohoe, editor, *Software Product Lines*, The Springer International Series in Engineering and Computer Science, pages 47–70. 2000. 27

[HT07] Esben Rune Hansen and Peter Tiedemann. Compressing configuration data for memory limited devices. In *22nd National Conference on Artificial Intelligence*, Vancouver, British Columbia, Canada, 2007. AAAI Press. 89

[IST13] Markus Iser, Carsten Sinz, and Mana Taghdiri. Minimizing models for tseitin-encoded sat instances. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing*, SAT'13, pages 224–232, Berlin, Heidelberg, 2013. Springer-Verlag. 12

[Jan10] Mikolas Janota. *SAT Solving in Interactive Configuration*. PhD thesis, Department of Computer Science. University College Dublin, 2010. 46

[Jen04] R. M. Jensen. CLab: a C++ library for fast backtrack-free interactive product configuration. In *10th International Conference on Principles and Practice of Constraint Programming*, Toronto, Canada, 2004. Springer. 89

[Jun06] Ulrich Junker. *Handbook of Constraint Programming*, chapter Configuration, pages 837–868. Francesca Rossi and Peter van Beek and Toby Walsh, 2006. 46

[KAB07] Christian Kästner, Sven Apel, and Don Batory. A case study implementing features using aspectj. In *Software Product Line Conference*, pages 223–232, Washington, DC, USA, 2007. 28

[KAK08] Christian Kästner, Sven Apel, and Martin Kuhlemann. Granularity in software product lines. In *International Conference on Software Engineering*, pages 311–320, New York, NY, USA, May 2008. 28

[KBK11] Chang Hwan Peter Kim, Don S. Batory, and Sarfraz Khurshid. Reducing combinatorics in testing product lines. In *International Conference on Aspect-oriented Software Development*, pages 57–68, New York, NY, USA, 2011. 30

[KCH+90] Kyo Kang, Sholom Cohen, James Hess, William Novak, and Spencer Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-21, Softw. Eng. Institute, 1990. 2, 11

[KG09] Mahvish Khurum and Tony Gorschek. A systematic review of domain analysis solutions for product lines. *Journal of Systems and Software*, (12):1982–2003, 2009. 28

[KGM10] Jörg Kienzle, Nicolas Guelfi, and Sadaf Mustafiz. Crisis management systems: A case study for aspect-oriented modeling. In Shmuel Katz, Mira Mezini, and Jörg Kienzle, editors, *Transactions on Aspect-Oriented Software Development VII*, Lecture Notes in Computer Science, pages 1–22. 2010. 30

[KKLL99] Kyo Chul Kang, Sajoong Kim, Jaejoon Lee, and Kwanwoo Lee. Feature-oriented engineering of pbx software for adaptability and reuseability. *Software - Practice & Experience*, (10):875–896, 1999. 26

[KPSY07] Minseong Kim, Sooyong Park, Vijayan Sugumaran, and Hwasil Yang. Managing requirements conflicts in software product lines: A goal and scenario based approach. *Data & Knowledge Engineering*, (3):417–432, 2007. 28

[Kro12] Christian Kroer. Sat and smt-based interactive configuration for container vessel stowage planning. Master's thesis, IT University of Copenhagen, 2012. 89

[Kru02] C. Krueger. Eliminating the adoption barrier. *IEEE Software*, (4):29–31, July 2002. 27

[KUU10] Vasileios Kandylas, S. Phineas Upham, and Lyle H. Ungar. Analyzing knowledge communities using foreground and background clusters. *ACM Transactions on Knowledge Discovery from Data*, (2010), May 2010. 24, 99

[KZK10] Andreas Kübler, Christoph Zengler, and Wolfgang Küchlin. Model counting in product configuration. In *1st International Workshop on Logics for Component Configuration*, pages 44–53, Edinburgh, UK., July 2010. 35

[LAL+10] Jörg Liebig, Sven Apel, Christian Lengauer, Christian Kästner, and Michael Schulze. An analysis of the variability in forty preprocessor-based software product lines. In *International Conference on Software Engineering*, pages 105–114, New York, NY, USA, 2010. 30

[Lam98] W. Lam. A case-study of requirements reuse through product families. *Annals of Software Engineering*, pages 253–277, January 1998. 26

[Lau06] Sean Quan Lau. Domain analysis of e-commerce systems using feature-based model templates. Master's thesis, Dept. Electrical and Computer Engineering, University of Waterloo, Canada, 2006. 92

[LBL11] Jing Liu, Samik Basu, and Robyn R. Lutz. Compositional model checking of software product lines using variation point obligations. *Automated Software Engineering*, (1):39–76, 2011. 30

[LDL07] Jing Liu, Josh Dehlinger, and Robyn Lutz. Safety analysis of software product lines using state-based modeling. *Journal of Systems and Software*, (11):1879–1892, November 2007. 28

[LGRC15] Jia Hui Liang, Vijay Ganesh, Venkatesh Raman, and Krzysztof Czarnecki. Sat-based analysis of large real-world feature models is easy. In *Software Product Line Conference*, Nashville, TN USA, July 2015. 3, 33

[LHBC05] Roberto E. Lopez-Herrejon, Don Batory, and William Cook. Evaluating support for features in advanced modularization technologies. In *European Conference on Object-Oriented Programming*, pages 169–194, Glasgow, UK, 2005. 28

[LHM+98] R.R. Lutz, G.G. Helmer, M.M. Moseman, D.E. Statezni, and S.R. Tockey. Safety analysis of requirements for a product family. In *International Conference on Requirements Engineering*, pages 24–31, Colorado Springs, Colorado, USA, 1998. 26

[LK04] K. Lee and K.C. Kang. Feature dependency analysis for product line component design. In *International Conference on Software Reuse*, pages 65–69, Madrid, Spain, July 2004. 27

[LK10] Kwanwoo Lee and Kyo C. Kang. Usage context as key driver for feature selection. In *Software Product Line Conference*, Lecture Notes in Computer Science, pages 32–46, Jeju Island, South Korea, 2010. 30

[LKL02] K. Lee, K.C. Kang, and J. Lee. Concepts and guidelines of feature modeling for product line software engineering. In *International Conference on Software Reuse*, pages 62–77, Austin, Texas, April 2002. 27

[LMN10] Jaejoon Lee, Dirk Muthig, and Matthias Naab. A feature-oriented approach for developing reusable product line assets of service-based systems. *Journal of Systems and Software*, (7):1123–1136, 2010. 30

[LSW15] Uwe Lesta, Ina Schaefer, and Tim Winkelmann. Detecting and explaining conflicts in attributed feature models. In *Workshop on Formal Methods and Analysis in SPL Engineering*, pages 31–43, London, UK, April 2015. 37, 85, 86

[Lut00] Robyn R. Lutz. Extending the product family approach to support safe reuse. *Jounal of Systems and Software*, (3):207–217, September 2000. 27

[MAaI12] Sonia Montagud, Silvia Abrahão, and Emilio Insfran. A systematic review of quality attributes and measures for software product lines. *Software Quality Control*, (3-4):425–486, September 2012. 30

[Mat04] M. Matinlassi. Comparison of software product line architecture design methods: Copa, fast, form, kobra and qada. In *International Conference on Software Engineering*, pages 127–136, Edinburgh, Scotland, UK, May 2004. 27

[MBC09] Marcilio Mendonca, Moises Branco, and Donald Cowan. S.p.l.o.t.: Software product lines online tools. In *ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, pages 761–762, New York, NY, USA, 2009. 28

[MC10] Marcilio Mendonca and Donald Cowan. Decision-making coordination and efficient reasoning techniques for feature-based configuration. *Science of Computer Programming*, (5):311 – 332, 2010. 30

[MDSD14] R. Mazo, C. Dumitrescu, C. Salinesi, and D. Diaz. *Recommendation Systems in Software Engineering*, chapter Recommendation Heuristics for Improving Product Line Configuration Processes. Springer-Verlag Berlin Heidelberg, 2014. 48

[Men09] Marcilio Mendonça. *Efficient Reasoning Techniques for Large Scale Feature Models*. PhD thesis, University of Waterloo, 2009. 12, 15, 46, 49, 92, 99

[MFMP10] Daniel Mellado, Eduardo Fernández-Medina, and Mario Piattini. Security requirements engineering framework for software product lines. *Information and Software Technology*, (10):1094 – 1117, 2010. 30

[MGH+11] Bardia Mohabbati, Dragan Gašević, Marek Hatala, Mohsen Asadi, Ebrahim Bagheri, and Marko Bošković. A quality aggregation model for service-oriented software product lines based on variability and composition patterns. In *International Conference on Service-Oriented Computing*, pages 436–451, Paphos, Cyprus, 2011. 30

[MMLP09] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl. Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In *ICSE Workshop on Principles of Engineering Service Oriented Systems*, pages 18–25, May 2009. 28

[MPH+07] A. Metzger, K. Pohl, P. Heymans, P. Schobbens, and G. Saval. Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In *Int. Requirements Engineering Conference*, pages 243–253, New Delhi, India, October 2007. 28

[MT98] Christoph Meinel and Thorsten Theobald. *Algorithms and data structures in VLSI design: OBDD-foundations and applications*. Springer Science & Business Media, 1998. 15, 99

[MWC09] M. Mendonça, A. Wasowski, and K. Czarnecki. Sat-based analysis of feature models is easy. In *13th International Software Product Line Conference*, pages 231–240, San Francisco, CA, USA, Aug. 2009. 2, 37, 38

[MYC05] Mikyeong Moon, Keunhyuk Yeom, and Heung Seok Chae. An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line. *IEEE Transactions on Software Engineering*, (7):551–569, July 2005. 28

[NBE12] Alexander Nöhrer, Armin Biere, and Alexander Egyed. Managing sat inconsistencies with humus. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 83–91, New York, NY, USA, 2012. ACM. 88

[NE11] Alexander Nöhrer and Alexander Egyed. Optimizing user guidance during decision–making. In *15th International Software Product Line Conference*, Munich, Germany, August 2011. IEEE Computer Society. 88

[NE13] Alexander Nöhrer and Alexander Egyed. C2o configurator: a tool for guided decision-making. *Automated Software Engineering*, 20(2):265–296, 2013. 88

[Nor02] L.M. Northrop. Sei's software product line tenets. *IEEE Software*, (4):32–40, July 2002. 27

[NTS+11] Laís Neves, Leopoldo Teixeira, Demóstenes Sena, Vander Alves, Uirá Kulezsa, and Paulo Borba. Investigating the safe evolution of software product lines. In *International Conference on Generative Programming and Component Engineering*, pages 33–42, New York, NY, USA, 2011. 30

[NW07] Nina Narodytska and Toby Walsh. Constraint and variable ordering heuristics for compiling configuration problems. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 149–154, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. 15, 87, 89, 92, 99

[OMR10] Sebastian Oster, Florian Markert, and Philipp Ritter. Automated incremental pairwise testing of software product lines. In *Software Product Line Conference*, pages 196–210, Jeju Island, South Korea, September 2010. 30

[OOF05] Barry O'Sullivan, Barry O'Callaghan, and Eugene C. Freuder. Corrective explanation for interactive constraint satisfaction. In *19th International Joint Conference on Artificial Intelligence*, pages 1531–1532, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc. 89

[PBL05] Klaus Pohl, Gunter Bockle, and Frank Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005. 9, 50

[PH04] F. K. Pil and M Holweg. Mitigating product variety's impact on the value chain. *Interfaces*, 34(5):394–403, 2004. 3

[PHRCT06] J. Pena, M. Hinchey, A. Ruiz-Cortes, and P. Trinidad. Building the core architecture of a multiagent system product line: with an example from a future nasa mission. In *7th International Workshop on Agent Oriented Software Engineering*, Hakodate, Japan, May 2006. 37, 38

[PKS04] Sooyong Park, Minseong Kim, and Vijayan Sugumaran. A scenario, goal and feature-oriented domain analysis approach for developing software product lines. *Industrial Management & Data Systems*, (4):296–308, 2004. 27

[PL05] Prasanna Padmanabhan and Robyn R. Lutz. Tool-supported verification of product line requirements. *Automated Software Engineering*, (4):447–465, October 2005. 28

[PLP11] R. Pohl, K. Lauenroth, and K. Pohl. A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models. In *International Conference on Automated Software Engineering*, pages 313–322, Kansas, USA, Nov 2011. 3, 33

[POS+12] Gilles Perrouin, Sebastian Oster, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves le Traon. Pairwise testing for software product lines: comparison of two approaches. *Software Quality Journal*, (3-4):605–643, 2012. 30, 92

[PSK+10] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. Le Traon. Automated and scalable t-wise test case generation strategies for software product lines. In *International Conference on Software Testing, Verification and Validation*, pages 459–468, Paris, France, April 2010. 30

[Que11] Matthieu Queva. *A Framework for Constraint-Programming based Configuration*. PhD thesis, Technical University of Denmark, 2011. 89

[RB10] Iris Reinhartz-Berger. Towards automatization of domain modeling. *Data & Knowledge Engineering*, (5):491–515, 2010. 30

[RBS09] Iris Reinhartz-Berger and Arnon Sturm. Utilizing domain models for application design and validation. *Information and Software Technology*, (8):1275–1289, August 2009. 28

[RFBRC+12] Fabricia Roos-Frantz, David Benavides, Antonio Ruiz-Cortés, André Heuer, and Kim Lauenroth. Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Control*, (3-4):519–565, September 2012. 30

[RGD10] Rick Rabiser, Paul Grunbacher, and Deepak Dhungana. Requirements for product derivation support: Results from a systematic literature review and an expert survey. *Information and Software Technology*, (3):324–346, 2010. 30

[RS98] David C Rine and Robert M Sonnemann. Investments in reusable software. a study of software reuse investment success factors. *Journal of Systems and Software*, (1):17–32, 1998. 26

[RW07] Mark-Oliver Reiser and Matthias Weber. Multi-level feature trees: A pragmatic approach to managing highly complex product families. *Requirements Engineering*, (2):57–75, May 2007. 28

[SAH+11] Samaneh Soltani, Mohsen Asadi, Marek Hatala, Dragan Gasevic, and Ebrahim Bagheri. Automated planning for feature model configuration based on stakeholders' business concerns. In *26th International Conference on Automated Software Engineering*, pages 536–539, Washington, DC, USA, 2011. IEEE Computer Society. 46

[SB99] Mikael Svahnberg and Jan Bosch. Evolution in software product lines: Two cases. *Journal of Software Maintenance*, (6):391–422, November 1999. 26

[SB02] Yannis Smaragdakis and Don Batory. Mixin layers: An object-oriented implementation technique for refinements and collaboration-based designs. *ACM Trans. Softw. Eng. Methodol.*, (2):215–255, April 2002. 27

[SBDT10] Ina Schaefer, Lorenzo Bettini, Ferruccio Damiani, and Nico Tanzarella. Delta-oriented programming of software product lines. In *Software Product Line Conference*, pages 77–91, Jeju Island, South Korea, 2010. 30

[Sch02] K. Schmid. A comprehensive product line scoping approach and its validation. In *International Conference on Software Engineering*, pages 593–603, Orlando, FL, USA, May 2002. 27

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948. 77

[SHBRC11] Sergio Segura, Robert M. Hierons, David Benavides, and Antonio Ruiz-Cortés. Automated metamorphic testing on the analyses of feature models. *Information and Software Technology*, (3):245 – 258, 2011. 30

[She13] Steven She. *Feature Model Synthesis*. PhD thesis, University of Waterloo, 2013. 86

[SHN+07] Carsten Sinz, Albert Haag, Nina Narodytska, Toby Walsh, Esther Gelle, Mihaela Sabin, Ulrich Junker, Barry O'Sullivan, Rick Rabiser, Deepak Dhungana, Paul Grünbacher, Klaus Lehner, Christian Federspiel, and Daniel Naus. Configuration. *IEEE Intelligent Systems*, 22:78–90, 2007. 46

[SHT06] P. Schobbens, P. Heymans, and J.-C. Trigaux. Feature diagrams: A survey and a formal semantics. In *Int. Conference on Requirements Engineering*, pages 139–148, Minneapolis, MN, USA, Sept 2006. 28

[SHTB07] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. Generic semantics of feature diagrams. *Computer Networks*, (2):456–479, 2007. 9, 11, 28

[SIMA13] A.S. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Scalable product line configuration: A straw to break the camel's back. In *International Conference on Automated Software Engineering*, pages 465–474, Silicon Valley, CA, USA, Nov 2013. 30

[SLB+11] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki. Reverse engineering feature models. In *International Conference on Software Engineering*, pages 461–470, New York, NY, USA, 2011. 30

[Sma06] Henry Small. Tracking and predicting growth areas in science. *Scientometrics*, (3):595–610, 2006. 31

[SPR04] P. Sochos, I. Philippow, and M. Riebisch. Feature-oriented development of software product lines: Mapping feature models to the architecture. In *Int. Conference on Object-Oriented and Internet-Based Technologies*, pages 138–152, Erfurt, Germany, September 2004. 27

[SRC+12] Ina Schaefer, Rick Rabiser, Dave Clarke, Lorenzo Bettini, David Benavides, Goetz Botterweck, Animesh Pathak, Salvador Trujillo, and Karina Villela. Software diversity: state of the art and perspectives. *International Journal on Software Tools for Technology Transfer*, (5):477–495, 2012. 30

[SRK+11] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. Spl conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, pages 1–31, June 2011. 46

[SSJ05] Timothy W. Simpson, Zahed Siddique, and Jianxin Roger Jiao. *Product Platform and Product Family Design: Methods and Applications*. Springer, 2005. 1

[Ste80] Louis Steinberg. Question ordering in a mixed intiative program specification dialogue. In *1st Annual National Conference on Artificial Intelligence*, Stanford University, August 1980. AAAI Press. 47

[SvGB05] Mikael Svahnberg, Jilles van Gurp, and Jan Bosch. A taxonomy of variability realization techniques. *Software - Practice & Experience*, (8):705–754, july 2005. 28

[SW98] Daniel Sabin and Rainer Weigel. Product configuration frameworks-a survey. *IEEE Intelligent Systems*, 13(4):42–49, July 1998. 46

[Tar13] Reinhard Tartler. *Mastering Variability Challenges in Linux and Related Highly-Configurable System Software*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2013. 37, 85, 86

[TBD07] S. Trujillo, D. Batory, and O. Diaz. Feature oriented model driven development: A case study for portlets. In *International Conference on Software Engineering*, pages 44–53, Minneapolis, MN, USA, May 2007. 28

[TBD+08] P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, and M. Toro. Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software*, (6):883–896, 2008. 28, 37, 38

[TBK09] T. Thüm, D. Batory, and C. Kästner. Reasoning about edits to feature models. In *International Conference on Software Engineering*, pages 254–264, Washington, DC, USA, May 2009. 28

[TBKC07] Sahil Thaker, Don Batory, David Kitchin, and William Cook. Safe composition of product lines. In *Int. Conference on Generative Programming and Component Engineering*, pages 95–104, New York, NY, USA, 2007. 28

[TBRC+08] P. Trinidad, D. Benavides, A. Ruiz-Cortes, S. Segura, and A. Jimenez. Fama framework. In *Software Product Line Conference*, pages 359–359, Limeric, Ireland, Sept 2008. 37

[TC09] P. Trinidad and A. Ruiz Cortes. Abductive reasoning and automated analysis of feature models: How are they connected? In *3rd International Workshop on Variability Modelling of Software-Intensive Systems*, Sevilla, Spain, January 2009. 37, 38

[TDR+11] Leonardo P. Tizzei, Marcelo Dias, Cecília M.F. Rubira, Alessandro Garcia, and Jaejoon Lee. Components meet aspects: Assessing design stability of a software product line. *Information and Software Technology*, (2):121 – 136, 2011. 30

[TNK04] Anne Taulavuori, Eila Niemelä, and Päivi Kallio. Component documentation-a key issue in software product lines. *Information and Software Technology*, (8):535–546, 2004. 27

[TR91] W.A. Turner and F. Rojouan. Evaluating input/output relationships in a regional research network using co-word analysis. *Scientometrics*, (1):139–154, 1991. 23

[Tse83] Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning*, pages 466–483. Springer, 1983. 12

[UKB10] E. Uzuncaova, S. Khurshid, and D. Batory. Incremental test generation for software product lines. *IEEE Transactions on Software Engineering*, (3):309–322, May 2010. 30

[URG10] Muhammad Irfan Ullah, Gïther Ruhe, and Vahid Garousi. Decision support for moving from a single product to a product portfolio in evolving software systems. *Journal of Systems and Software*, (12):2496 – 2512, 2010. 30

[vDK02] A. van Deursen and P. Klint. Domain-specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10(1):1–17, 2002. 37, 38

[vdLSR07] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action. The Best Industrial Practice in Product Line Engineering*. 2007. 2

[VG07] Markus Voelter and Iris Groher. Product line implementation using aspect-oriented and model-driven software development. In *Software Product Line Conference*, pages 233–242, Washington, DC, USA, 2007. 28

[VG09] Elizabeth S. Vieira and Josa A.N.F. Gomes. A comparison of scopus and web of science for a typical university. *Scientometrics*, (2):587–600, 2009. 20, 98

[vGBS01] J. van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In *Working IEEE/IFIP Conference on Software Architecture*, pages 45–54, Amsterdan, Netherlands, August 2001. 27

[vO05] Rob van Ommering. Software reuse in product populations. *IEEE Transactions on Software Engineering*, (7):537–550, July 2005. 28

[WBS+10] J. White, D. Benavides, D.C. Schmidt, P. Trinidad, B. Dougherty, and A. Ruiz-Cortes. Automated diagnosis of feature model configurations. *Journal of Systems and Software*, (7):1094–1107, 2010. 30, 46

[WHG+09] J. White, James H. Hill, J. Gray, S. Tambe, A.S. Gokhale, and D.C. Schmidt. Improving domain-specific language reuse with software product line techniques. *IEEE Software*, (4):47–53, July 2009. 28

[ZJY03] Hongyu Zhang, Stan Jarzabek, and Bo Yang. Quality prediction and assessment for product lines. In *International Conference on Advanced Information Systems Engineering*, pages 681–695, Klagenfurt, Austria, 2003. 27

[ZK10] Christoph Zengler and Wolfgang Küchlin. Encoding the linux kernel configuration in propositional logic. In *European Conference on Artificial Intelligence. Workshop on Configuration*, Lisbon, Portugal, 2010. 37

[ZMZ06] Wei Zhang, Hong Mei, and Haiyan Zhao. Feature-driven requirement dependency analysis and high-level software design. *Requirements Engineering*, 11(3):205–220, 2006. 37, 38

[ZSS+10] Steffen Zschaler, Pablo Sánchez, João Santos, Mauricio Alférez, Awais Rashid, Lidia Fuentes, Ana Moreira, João Araújo, and Uirá Kulesza. Vml* - a family of languages for variability management in software product lines. In *International Conference on Software Language Engineering*, pages 82–102, Eindhoven, The Netherlands, 2010. 30

[ZYZJ08] W. Zhang, H. Yan, H. Zhao, and Z. Jin. A bdd-based approach to verifying clone-enabled feature models' constraints and customization. In *10th International Conference on Software Reuse*, Beijing, China, May 2008. 37, 38

ACRONYMS

BDD Binary Decision Diagram

CNF Conjunctive Normal Form

CSP Constraint Satisfaction Problem

CTCR Cross-Tree Constraint Ratio

CVL Common Variability Language

DSL Domain-Specific Language

FD Features Diagram

FODA Feature Oriented Domain Analysis

MDD Model Driven Development

OMG Object Management Group

ROBDD Reduced Ordered Binary Decision Diagram

ROI Return On Investment

SOA Software Oriented Architecture

SPL Software Product Line