17th International Configuration Workshop

Proceedings of the

17th International Configuration Workshop

Edited by

Juha Tiihonen, Andreas Falkner, and Tomas Axling

September 10-11, 2015

Vienna, Austria

Organized by

UNIVERSITY OF HELSINKI

FACULTY OF SCIENCE

University of Helsinki
Department of Computer Science
Faculty of Science
P.O. 68, FI-00014 UNIVERSITY OF HELSINKI
FINLAND

ISSN 1613-0073

Chairs

Juha Tiihonen, Helsinki University, Finland
Andreas Falkner, Siemens AG Österreich, Austria
Tomas Axling, Tacton, Sweden

Program committee

Michel Aldanondo, Toulouse University, Mines Albi, France
Claire Bagley, Oracle Corporation, USA
Andreas Falkner, Siemens AG, Austria
Alexander Felfernig, Graz University of Technology, Austria
Gerhard Friedrich, University of Klagenfurt, Austria
Cipriano Forza, University of Padova
José A. Galindo, University of Seville, Spain
Albert Haag, SAP, Germany
Alois Haselböck, Siemens AG, Austria
Mikko Heiskala, Aalto University, Finland
Lothar Hotz, University of Hamburg, HiTeC, Germany
Lars Hvam, Technical University of Denmark, Denmark
Dietmar Jannach, University of Dortmund, Germany
Thorsten Krebs, encoway GmbH, Germany
Varvana Myllärniemi, Aalto University, Finland
Tomi Männistö, Helsinki University, Finland
Mikko Raatikainen, Aalto University, Finland
Rick Rabiser, Johannes Kepler University, Austria
Florian Reinfrank, Graz University of Technology, Austria
Stefan Reiterer, Graz University of Technology, Austria
Markus Stumptner, University of South Australia, Australia
Juha Tiihonen, Helsinki University, Finland
Elise Vareilles, Toulouse University, Mines Albi, France
Franz Wotawa, Graz University of Technology, Austria
Linda Zhang, IESEG Business School Paris, France
Markus Zanker, University of Klagenfurt, Austria

Local arrangements

Andreas Falkner, Siemens AG, Austria

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

Preface

Configuration problems are among the most fruitful domains for applying and developing advanced artificial intelligence (AI) techniques. Powerful knowledge-representation formalisms are required to capture the great variety and complexity of configuration problems. Efficient reasoning is required to provide intelligent interactive behavior in contexts such as solution search, satisfaction of user preferences, personalization, optimization, and diagnosis.

The main goal of the workshop is to promote high-quality research in all technical areas related to configuration. The workshop is of interest both for researchers working in the various fields of Artificial Intelligence as well as for industry representatives interested in the relationship between configuration technology and the business problem behind configuration and mass customization. It provides a forum for presentation of original methods and the exchange of ideas, evaluations, and experiences especially related to the use of AI techniques in the configuration context.

This year's workshop is a standalone two day event that continues the series of 16 successful Configuration Workshops started at the AAAI'96 Fall Symposium and continued at IJCAI, AAAI, and ECAI conferences since 1999.

A total of 21 papers were selected for presentation on the Configuration workshop. The themes of the technical sessions are Strategy, Long-term management, Collaboration, Solving, Diagnosis, and Analytics.

The 17th International Configuration Workshop introduced the concept of Best Paper Award. The best paper was selected in a two-phase audience vote: three best papers (actually four due to an equal number of votes) of the first round entered the second round to select the best paper and a runner-up. The Best Paper Award winner was 'Column oriented compilation of variant tables' by Albert Haag. Two runner-ups (with an equal number of votes) were 'Impact on cost accuracy and profitability from implementing product configuration system – a case study' by Anna Myrodia, Katrin Kristjansdottir, and Lars Hvam; and 'Coupling two constraint-based systems into an on-line facade-layout configurator' by Andrés Felipe Barco Santa, Elise Vareilles, Paul Gaborit, Jean-Guillaume Fages, and Michel Aldanondo.

Juha Tiihonen, Andreas Falkner and Tomas Axling

Juha Tiihonen, Andreas Falkner, and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

Juha Tiihonen, Andreas Falkner, and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

Contents

Juha Tiihonen, Andreas Falkner, and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

Juha Tiihonen, Andreas Falkner, and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

Market-oriented Variant Management (position paper)

Dr. Thorsten Krebs and **Christoph Ranze**[1]

Abstract. The mega trend individualization drives product manufacturers towards offering more and more variants of their products. As seen in mass customization scenarios, product configuration based on a modular strategy is an enabler for this trend.

The key idea of variant management is to optimize the number of product variants that can be offered to a specific market segment, i.e. outer variety, while reducing the complexity of product development and manufacturing. Production costs are typically kept low by producing a small amount of modules that are generic and common for multiple products within the modular strategy, i.e. inner variety.

Classic variant management is driven by the idea of reducing costs between the fields of product design / engineering and manufacturing / logistics. Hence, we see this as efficiency. We apply the same method of optimizing the interplay between outer variety and inner variety to the sales-level; i.e. between the fields of logistics processing and sales / marketing. The product variants that are offered to a specific market segment should be aligned with the market needs. Hence, we see this as effectiveness. We show how the two views on variant management complement one another and how they relate to business economics, namely the economies of scale and the economies of scope.

We suggest using a combination of both efficiency and effectiveness to assess the capability of change of product manufacturers that are based on modular systems. Market-oriented variant management involves a number of business processes. A management-ready presentation of the potential that changing these processes has, can significantly influence a company's willingness to invest in such a change.

1 INTRODUCTION

The mega trend *individualization* is the main reason why the interest in mass customization strategies is continuously growing. This mega trend is supported by the fact that nowadays it is easier than ever before to get information about products and to compare them. The *digital product representation* enables lots of services like product selection, configuration or comparison. Using this digital product representation it is possible to get all the relevant information for deciding which products best fit the customers' needs; without going from one shop to another. It is this information that can also be used for market-oriented variant management.

[1] encoway GmbH, Bremen, Germany, email: { krebs | ranze }@encoway.de

Variant management is a holistic approach to control and optimize product diversification with respect to production costs and market strategy (see e.g. [1]). The term variant management has been around for quite some time. The key idea of variant management is to optimize the number of product variants that can be manufactured, i.e. *outer variety*, while reducing the complexity of product development and manufacturing. Production costs are typically kept low by manufacturing only a small amount of different modules that are common and recurring for multiple products, i.e. *inner variety*. These modules can then be manufactured in large scales.

Within mass customization, product configuration is seen as the key enabler for being able to communicate product variety into the market (see e.g. [2]). *Product configuration* describes the task of composing a product from a set of pre-defined modules; the modular system. In this sense a product configurator is a tool for managing the interaction between the inner variety, i.e. the modules, and the outer variety, i.e. the products.

The first occurrences of the term variant management stem from the area of product design, engineering and production. The base idea is to separate the development and manufacturing of recurring modules from the manufacturing of products that are based on these modules. In this sense the product manufacturing process is separated into two parts. The first part is an order-neutral process: pre-manufacturing the modules. The second part is an order-specific process: combining modules.

Defining the scope of outer variety, i.e. defining the right amount of product variants, is one of the major activities for effective sales. Optimal product diversification therefore must be based on the market's demand.

This is why *market-oriented variant management* goes one step further and uses the idea of modular strategy on another level: between manufacturing, or: logistics in general, and sales / marketing. In this sense, optimizing product diversification for a given market segment is managing the outer variety (from the sales view). At the same time, optimizing product development and manufacturing is managing the inner variety (from the logistics view).

The remainder of this paper is organized as follows. Chapter 2 describes our understanding of market-oriented variant management in general as well as the distinction between the logistics view and the sales view. Chapter 3 describes work-in-progress on how the capability of change of business processes related to variant management can be measured and assessed in general and applies these ideas to market-oriented variant management. Chapter 4 discusses related work and finally, Chapter 5 gives a conclusion and discusses future work.

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

2 MARKET-ORIENTED VARIANT MANAGEMENT

We have already explained that market-orient variant management distinguishes two levels of optimizing product diversification: the logistics view and the sales view. In principle, this distinction can be made on an arbitrary number of levels, e.g. between product design and engineering, between engineering and manufacturing, between manufacturing and logistics, between logistics and sales. As we will see later in Chapter 3, distinguishing between the logistics view and the sales view has an important impact on companies' business strategy. This impact also affects the most crucial business processes of companies that are based on modular systems: new product development, quote generation and order processing. Therefore, we will focus on these two views in the following.

Before we detail the two views on variant management, we give a short insight into relevant aspects of business economics; namely the economies of scale and economies of scope.

2.1 Economies of scale and economies of scope

The *economies of scale* describe reducing engineering and production costs per unit as fixed costs are spread out over more units of output [3]. This is the base principle of mass production: the price per unit decreases with larger lot sizes. The *economies of scope*, on the other hand, are based on the common and recurrent use of modules. Thus, they describe lowering average costs by sharing production costs or recurring resources, for example sales or marketing activities, over a variety of products [4], [5]. When economies of scope are based upon the common and recurring use of proprietary knowhow or specialized and indivisible physical assets, the product diversification is an efficient way of organizing economic activity [6].

With a large outer variety of products that is based on a small inner variety of modules, the recurring modules can be pre-produced order-neutrally in large scales. In a second step these modules are assembled specifically for one customer order. Thus, mass customization strategy benefits from both, the economies of scale and the economies of scope.

While the economies of scale do plateau in an efficient state which delivers high-margin revenues, economies of scope may never reach such a state. But still, it is worth trying (see e.g. [7]). Managing the ongoing scope-learning process is one of the most essential activities in business strategy, in particular for companies manufacturing products that are based on modular systems.

While optimizing the scope of modularization belongs to the logistics view on variant management, finding the right scope for product diversification is an activity within the sales view on variant management.

2.2 Logistics view on variant management

The logistics view on variant management focuses on an optimal interplay between outer variety and inner variety from product design and engineering via manufacturing towards logistics; i.e. supply chain management, shipping and so on. The major activity within this view is optimizing the scope of modularization. This means that we want to benefit from both, the economies of scale (by producing modules in large scales) and the economies of scope (by sharing production costs and other related resources over a variety of products).

This view is the "classic" variant management approach that is around in literature for quite some time already. Therefore we refer the interested reader to [1] and in the following focus on the sales view.

2.3 Sales view on variant management

The key aspect of the sales view on variant management is to offer exactly those product variants that a specific market segment desires. Not less but also not more than those. The simple case of designing the range of product variants can be described as portfolio management. In this sense a company aligns its product portfolio according to the markets needs and has market-driven product development and manufacturing processes.

However, offering optimal product variety can be more complex than this. European product manufacturers are currently under pressure in order to compete with the low-price mass production in countries from Asia. This is why a lot of *component manufacturers* that have been *component vendors* turn into *system vendors*. With the term *component* we describe products that are used in larger contexts: the component itself has no direct benefit for the customer's application but a combination of components that complement one another builds up functionality with extra benefit. Both terms *product combination* and *system* can be encountered within the business strategy of companies that manufacture components. For the purpose of this paper we treat both terms as synonyms and in the following stick to the term system.

A system may consist of discrete products, configurable products or a combination of both, possibly together with components that are not sold independently. *Discrete products* are non-configurable products that are described by and selected from a set of characteristics and that do not offer customization options. *Configurable products* on the other hand are customizable products that are available in a large variety. Typically, configurable products are based on a modular strategy and need to be configured in order to obtain a sellable product.

The challenge of advertising, configuring and selling systems is not to be underestimated. The most important part of selling systems is to support the customer in the buying decision process. Identifying the right combination of components is even more complex than configuring a single product.

Additionally, customers often do not know which products they need. However, what a customer does know is the application problem for which he needs a solution. This is why we see a solution configuration as one of the major improvements in sales-oriented variant management. The term *solution configuration* describes a configuration process that is started with a problem definition for which a solution is sought. The main difference compared to usual product configuration is that the customer does not decide on the product's characteristics but enters characteristics of the application. Selecting the best-fitting product and inferring the products characteristics from the application characteristics is hidden from the user.

In order to achieve this encapsulation, the configuration model is separated into layers: on top of the technical layer containing configuration knowledge about buildability or the sales layer

containing configuration knowledge about sales-oriented customization options, an *application layer* containing configuration knowledge about the product's application domain is added. This application layer guides the customer during product configuration. Thus, the customer can focus on describing his problem situation and is not distracted with technical details he does not know about.

2.4 Interplay between the logistics view and the sales view

In the previous sections we distinguished the logistics view and the sales view on variant management. Furthermore, we have described their influence on optimizing the scope of modularization and finding the right scope of product diversification, respectively.

Obviously, both activities of scoping influence one another. Reducing the set of modules which are manufactured order-neutrally in order to increase the gain of the economies of scale, i.e. reducing the modular system, has impacts on the possible product variety. Vice versa, broadening the variety of products that are offered to a specific market segment in order to increase the gain of the economies of scope, i.e. enlarging product diversification, has impacts on the modular system.

Nevertheless, it is important that both scoping activities are addressed individually:

- The activity of scoping modularization affects the efficiency of related business processes. The efficiency of modularization and of product manufacturing in general lies in being able to provide the expected outcome, i.e. the products, with the least possible use of resources. Note that besides low production costs this also includes fast time-to-market and delivery times.
- The activity of scoping product diversification affects the effectiveness of related business processes. The effectiveness of product diversification for product manufacturers that are based on modular systems lies in being able to provide exactly those products that the addressed market segment desires; at the right time and at the right place and most importantly at the right price.

Both scoping activities are carried out by different business units and can be initially set up, maintained, assessed or optimized individually. But companies manufacturing products that are based on modular systems will only be successful when addressing both of the scoping activities. Only then it will be possible to deliver the right products at reasonable prices but also to generate high-margin revenues.

3 CAPABILITY OF CHANGE

Variant management is an approach to control and optimize product diversification. In this sense, variant management significantly influences business processes like the new product development, quote generation and order processing. Business processes are crucial for companies and are typically not changed unless really necessary. The decision to change a business process therefore needs management-ready analysis and presentation of the change's potential.

In the following we present work-in-progress on how the *capability of change* for business processes of product manufacturers can be assessed and improved. The first step towards this goal is being able to measure business processes. We present first ideas on how to do that in general (Section 3.1) and apply these ideas to market-oriented variant management (Section 3.2). The second step towards this goal is being able to define a metric which compares the current state of a business process with its target state (Section 3.3). Such a metric can be used to describe the potential impact of a change in the business process and significantly influence the willingness to invest.

3.1 Measuring business processes in general

The efficiency of a process describes how good *the things are done right*. A processes efficiency can be measured using several criteria including but not limited to total processing time, resource utilization per unit of output, non-value added cost, non-value added time, cost of quality, and so on [8]. Furthermore, any deficiency in training or skills of the workers or any delay from the related processes that provide inputs for the measured process will also show up.

The effectiveness of a process describes how *the right things are done*. Measuring processes effectiveness begins with outlining the customers' expectations and needs in detail. These expectations would then be converted into measurable targets. Customer expectations are not readily available or clearly specified. This is what makes it hard to set up a quantifiable measurement [8]. Typical customer expectations are, among others, product quality, frequency of new products or updates, quality of service and the overall customer experience.

To the current state of our work-in-progress we do not have a process to generate quantified numbers or formulae, but here are some first thoughts:

- If we do exactly one thing in a perfect way, then we assume to have an efficiency of 100%. If we do the same thing with half the efficiency, then we expect to have an efficiency of 50%.
- If we do one perfectly right thing, then we assume to have an effectiveness of 100%. If we do one thing that is half as effective, then we assume to have an effectiveness of 50%.
- If we do one perfectly right thing with half the efficiency, then we assume to have an overall process performance of 50%. Vice versa, the same holds for doing a thing with half the effectiveness, but doing it perfectly efficient.

Hence, in order to measure the performance of a business process we need to measure both its efficiency and its effectiveness and then we need to generate a reasonable overall measurement including the input of both values.

3.2 Measuring market-oriented variant management

Classic variant management focuses on efficiency: optimizing the scope of modularization. The main reason for variant management on the logistics level is cost reduction. Managing the sales view focuses on effectiveness: finding the right scope for product diversification. The main reason for variant management on the sales level is selling more products. While effectiveness is most important from the point of view of external customers, efficiency is most important internally.

Efficiency and effectiveness of market-oriented variant management relate to different business processes. Efficiency is related to new product development and setting up the required tools for logistics processing, quote generation and order processing. Effectiveness, on the other hand is related to marketing and sales strategies and their influence on a customer's buying decision.

3.3 The potential of a change in the business process

The ideal performance of a process is assumed to be 100% – we do the right things and we do them right. But nevertheless, a desired target state for a business process may be less than 100%, e.g. when reaching the ideal status is expansive and a company wants to invest in smaller amounts. In such a case it may be viable to set a target state of, for example, pareto-optimal 80%.

The potential, that the process optimization can raise, is the distance between the current state and the target state. A management-ready presentation of this potential can significantly influence the decision whether to change the company's business process or not. Low potential will lead to low willingness to invest whereas high potential may also lead to investments although there are risks along that way.

4 RELATED WORK

In this paper we make use of business economics. Namely, these are the economies of scale [7] and the economies of scope [6], [8], [9]. The interplay between these two economies also has been researched earlier (see e.g. [10]).

There is also earlier research in the area of managing product variety and manufacturing complexity [9], [10], [11]. Also the research fields of variant management (see e.g. [1]) and mass customization (see e.g. [2]) give a lot of input on managing variety and scoping product diversification.

However, there are only few articles on using the economies of scale and economies of scope to explain or analyze economics for product manufacturers that are based on modular systems. One notable article is [12] in which a performance measurement system for modular product platforms is proposed. However, this article focuses on measuring the setting up and the maintenance of modular product platforms based on a set of criteria that is defined during the measurement process.

This paper presents a novel approach with the term *market-oriented variant management*, i.e. adding a sales view as another level on top of the logistics view that classic variant management deals with. Another novelty in this paper is relating the scoping activities of modularization and product diversification to the terms efficiency and effectiveness, respectively, and using them to measure the performance of business processes and to calculate the potential of changing business processes.

5 CONCLUSION AND FUTURE WORK

We introduced the term market-oriented variant management as a combination of classic variant management, which we see as the logistics view and an additional sales view that both use the same method of optimizing the relation between inner variety and outer variety. We have shown how the two views on variant management complement one another and how they relate to business economics, namely the economies of scale and the economies of scope.

Furthermore we introduced an approach that supports calculating the potential of changes to business processes that are related to market-oriented variant management. Such a potential in a management-ready form can significantly influence the decision of changing the company's most crucial business processes.

It is up to future work to fully understand the relation and interplay between the well-established research areas product configuration, mass customization, variant management and business economics.

Also, we need to research how the potential of a business process can be presented in meaningful numbers. Obviously, a calculated number in terms of percentage would significantly improve the statement of potential. But in order to get there, we have to go some steps: defining how to measure both the efficiency and the effectiveness of a business process, doing this for both, the current state and a target state of the relevant processes and then defining a metric for presenting the potential of the actual change. Another open topic is the influence of different products on the measurement. Perhaps it is necessary to define a weighting for the different products, for example based on sales numbers or revenue.

REFERENCES

[1] B. Avak, *Variant Management of Modular Product Families in the Market Phase* VDI-Verlag, Düsseldorf, ISBN 3-18-318016-2 (2006).

[2] F. Piller, *Mass Customization: Reflections on the State of the Concept*, International Journal of Flexible Manufacturing Systems (16.4), 313-334, (2004).

[3] A. O'Sullivan and S. M. Sheffrin, *Economics: Principles in Action.* The Wall Street Journal: Classroom Edition (2nd ed.), Prentice Hall, New Jersey, ISBN 0-13-063085-3 (2003).

[4] J. D. Goldhar and M. Jelinek, *Plan for Economies of Scope*, Harvard Business Review (November 1983).

[5] T. Hindle, *Guido to Management Ideas and Gurus (Economist)*, Profile Books, ISBN 978-1846681080 (2008).

[6] D.J. Teece, *Economies of Scope and the Scope of the Enterprise*, Journal of Economic Behaviour & Organization, Volume 1, Issue 3, Pages 223-247 (1980).

[7] V. Rao, *Economies of Scale, Economies of Scope*. Ribbonfarm: http://www.ribbonfarm.com/2012/10/15/economies-of-scale-economies-of-scope/, (October 15, 2012; last visited: June 2015).

[8] Management Study Guide: *Articles on Business Process Improvement*, http://www.managementstudyguide.com/business-process-improvement-articles.htm (last visited: June 2015).

[9] H. El'Maraghy.,G. Schuh., W. ElMaraghy, F. Piller, P. Schonsleben, M. Tseng, *Product Variety Management*, CIRP Annals - Manufacturing Technology, 629-652 (2013).

[10] S.J. Hu, X. Zhu, H. Wang, Y. Koren, *Product Variety and Manufacturing Complexity in Assembly Systems and Supply Chains*, CIRP Annals - Manufacturing Technology, 57, 45-48 (2008).

[11] J.P. MacDuffie, K. Sethuraman K., M.L. Fisher, *Product Variety and Manufacturing Performance – Evidence From the International Automotive Assembly Plant Study*, Management Science Vol. 42, No. 3, 350-369 (1996).

[12] G. Schuh, S. Rudolf and T. Vogels, *Performance Measurement of Modular Product Platforms*, Variety Management in Manufacturing — Proceedings of the 47th CIRP Conference on Manufacturing Systems, Volume 17, Pages 266–271 (2014).

An empirical study on product configurators' application: Implications, challenges and opportunities

Linda L. ZHANG[+] and **Petri T. HELO**

Abstract. As computer systems, product configurators have been widely applied in practice for configuring a right amount of product variety. While studies have been reported to shed light on how product configurators' application achieves time reduction and quality improvement in fulfilling customer orders, empirical investigations addressing the implications of product configurators' application for companies' business activities are very limited. However, understanding the implications is very important for companies because with such an understanding, they can better plan actions and make changes to embrace the implementation of product configurators. Thus, based on a survey, this study investigates the implications of product configurators' application. The results indicate (i) how product configurators' application affects companies' business activities, (ii) the difficulties in designing, developing, and using product configurators, and (iii) the potential barriers preventing companies from effectively applying product configurators in the future. With the results, we further highlight several improvement areas for companies to investigate in order to reap, to the largest extend, the benefits of implementing product configurators.

1 INTRODUCTION

Since the early 90's, product configurators have been receiving continuous interests and investigations from both the academia and industrial alike. Resulting from these efforts, countless articles have been published to present solutions to diverse configurator related issues. Many of these articles address configuration knowledge representation and modeling, methods and approaches for product configurator design from a theoretical point of view (e.g., Haug, 2010; Pitiot et al., 2013; 2013; Zhang et al., 2013). In comparison, a relatively small number of articles deal with the practical issues related to product configurators' application. Among them, some articles empirically investigate how product configurators achieve lead-time reduction and quality improvement (Trentin et al., 2011 & 2012; Haug et al., 2011); some articles use single cases to show (i) how product configurators' development can be facilitated (Haug et al., 2010; Hvam et al., 2003; Hvam & Ladeby, 2007), (ii) how product configurators contribute to variety management (Forza & Salvador, 2002), and (iii) the suitable product configurator development strategies (Haug et al., 2012). Due to the relatively less investigation, several important issues related to configurators' application are unclear. They include (i) how product configurators' applications affect companies' business activities, (ii) the difficulties in designing, developing/maintaining, and using product configurators, and (iii) the barriers potentially preventing companies from effectively using configurators in the future. However, it is important to have a clear understanding of all these issues. This is because such an understanding can help companies better plan actions for optimally applying product configurators, thus realizing the benefits of product configurators' application to the largest extent.

This study tries to fill this gap by investigating the implications of product configurators' application. It is done using a quantitative research method, as described in the next section. Based on the data collected, we present, in Section 3, the results and analysis. The results are discussed with respect to (i) how product configurators' application affects companies' business activities, (ii) the difficulties in designing, developing/maintaining, and using product configurators, and (iii) the potential barriers influencing the effective application of product configurators in the future. Built upon the insights in Section 3, we further discuss the possible changes and improvements that companies may undertake.

2 METHODS

As the aim of this study is to identify the implications of product configurators' application for companies' business activities, an empirical study was performed. In doing so, a questionnaire was developed for collecting data. The collected data was computed and analyzed for revealing the application implications and for identifying the related opportunities as well.

In designing the questionnaire, we included such questions like the functions that product configurators perform, the business process and IT system changes caused by product configurators' application, the difficulties in implementing product configurator projects, the potential barriers preventing the effective application of product configurators in the future, and the performance improvements resulting from product configurators' application. Besides these questions, there are also general questions such as position titles, the application time of configurators. By considering the explorative nature of this study, we used nominal scales by presenting alternative choices for each question. The alternatives were determined based on the literature and our experiences of working with companies, where product configurators are applied. Besides alternatives, we included the opportunity for respondents to choose "Other" and give details. In this way, we avoid the possibility of missing potential alternatives. To verify the initial questionnaire with respect to the sufficiency and appropriateness of questions, we pretested it in 5 companies, with which we have collaborations. In

+: Department of Management, IESEG School of Management (LEM-CNRS), France, email: l.zhang@ieseg.fr

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

the pretest, 5 company representatives filled up the questionnaire and provided comments. In addition, we made phone contacts for clarifications of the comments and additional remarks. Based on the feedback, some questions were revised, leading to the finalized questionnaire. (Appendix A summarizes the questionnaire.)

With the finalized questionnaire, survey was conducted in a research panel involving managers responsible for IT investments in US companies in October and November 2013. These companies were mainly from the computer, telecommunication system, and industrial machinery industries. Panel members were invited by a commercial research data collection company: EMPanel Online. Data was collected by sending invitation by email and using an on-line survey tool. To ensure that the respondents know about configurators and were responsible for configurator projects, a qualifying question was asked. In addition, another two questions regarding industry type and company size were asked. By considering these questions, we obtained and analyzed 64 completed questionnaires, which had balanced distributions with respect to the company size, industry, and time duration (in year) of product configurators' application.

In analyzing the data, we computed the total occurrence of each alternative, which was selected by the respondents, and the corresponding percentage. In this regard, we analyzed the distribution of the selected alternatives.

3 RESULTS

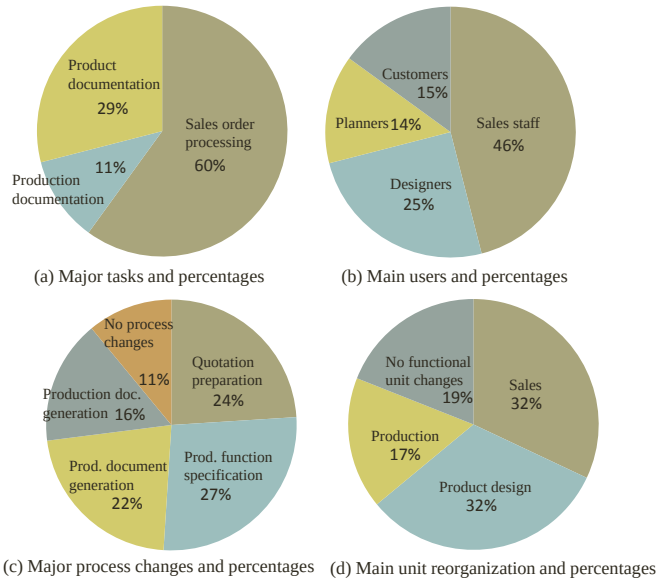
In accordance with the questionnaire, the collected data is analyzed with respect to (i) how the application of product configurators affects companies' business activities, (ii) the difficulties in designing, developing/maintaining, and using product configurators, and (iii) the issues potentially influencing the effective application of product configurators in the future.

3.1 How product configurators' application affects companies' business activities

In studying how companies' business activities are affected by the application of product configurators, it is essential to understand the major tasks that product configurators perform. This is because these tasks contribute to many of companies' activities for designing, producing, and delivering products. It is equally important to understand the major users of product configurators. In accordance with the configurators' tasks and users, there might be changes to companies' business processes, functional units, IT systems, the number of employees, and performances.

3.1.1 Major tasks and users of product configurators

Being consistent with the literature, the survey results suggest that a configurator performs multiple tasks and has multiple users. As shown in Fig. 1(a), configurators in 60% of the respondents (i.e., the responding companies) perform sales order processing, such as quotation preparation, sales order specification, and product specification; configurators in 29% and 11% of the respondents carry out product documentation: BOM and drawing generation and manufacturing documentation: routing and process plan generation, respectively. (Note that these figures were calculated based on the total number of tasks, instead of that of companies. In this regard, they are not mutually exclusive with respect to companies. This is the same with the calculation of all the rest figures.) Consistent with the tasks that configurators perform, the major users include sales staff (in 46% of the respondents), designers (in 25% of the respondents), customers (in 15% of the respondents), and production planners (in 14% of the respondents), as shown in Fig. 1(b). While published articles provide anecdotal evidences, there is no study presenting such a complete distribution of configurators' functions and users.



(a) Major tasks and percentages (b) Main users and percentages

(c) Major process changes and percentages (d) Main unit reorganization and percentages

Figure 1: Some results

3.1.2 Functional units reorganized

While product configurators take over tasks, which are performed previously in different functional units, their application does not bring many changes to the functional units. As shown in Fig. 1(c), both the sales and product design/development units in only 32% of the respondents are affected, thus reorganized; the production unit in 17% of the respondents is reorganized. It is interesting to see that product configurator application does not bring any changes to the functional units in 19% of the respondents. Our experiences of working with companies show that this might happen in the situations, where customers are the major users of product configurators. However, this needs to be further confirmed in the future investigations.

3.1.3 Business process changes

As product configurators automatically perform many activities, which are carried out manually in the past, their application might incur business process changes as well. In the survey, many respondents indicate that their business processes have been changed. These changes include (i) the original manual quotation preparation is done automatically (in 24% of the respondents), (ii) the sales unit is responsible for the task of specifying product functions (in 27% of the respondents), (iii) product technical details, e.g., BOMs, drawings, are dealt with automatically (in 22% of the respondents), (iv) manufacturing documents are generated automatically (in 16% of the respondents), as shown in Fig. 1(d). It

is worth mentioning: as pointed out by 11% of respondents, that product configurators' application does not bring changes to their business processes. This seems consistent with what we have found above: 19% of respondents indicate that product configurators' application does not cause changes to the functional units. When there are no changes to companies' functional units, there may not be the changes to the business processes. However, this needs to be further tested in the future.

3.1.4 Changes to companies' legacy systems

In performing tasks, product configurators interact with companies' other IT systems for receiving inputs and/or sending outputs. Consequently, product configurators' application may cause changes to companies' legacy systems. The results confirm this. As shown in the results, (i) design systems are modified to be linked with configurators (in 33% of the respondents), (ii) production systems, such as product planning and control systems and material resources planning systems, are modified to be linked with product configurators (in 48% of the respondents), and (iii) accounting systems are modified to be linked with product configurators (in 9% of the respondents). At last, 10% of the respondents indicate that there are no changes to their legacy systems. When product configurators are built-in modules of ERP systems, it might be possible that companies do not need to modify their legacy systems.

3.1.5 Changes to the number of employees

As product configurators perform automatically many activities, which are performed manually in the past, intuitively, product configurators' application should reduce the number of full time employees. However, it is surprising to see that 63% of the respondents hire full time employees, whereas only 6% indicate that they lay off employees. 31% point out that there are no changes to the number of their employees. As indicated in the following results (Subsections 3.2 and 3.3), companies do not have sufficient good IT system designers and developers. In this regard, the application of product configurators may lead to the recruitment of new employees.

3.1.6 Performance improvements

The available literature reports diverse performance improvements resulting from the application of product configurators. This study finds similar results, thus supporting the literature. For all the respondents, the improvements include (i) increased sales volume, (ii) increased correct sales orders, (iii) reduced production rework, (iv) increased customer orders, (v) reduced order processing time, and (vi) reduced sales delivery time. As indicated by the results, 47%/44%/ 44%/31% of respondents achieve 30%-50% increase of sales volume/increased correct sales orders/reduced production rework/increased customer orders accepted, respectively. Seen from the results, it is difficult for companies to achieve higher improvements in these performance measures. For example, only 3% of the respondents achieve 80% increase of sales volume. It is interesting to see that companies reduce much order processing time, whereas they do not achieve equivalent improvement in the sales delivery lead time. (50% of the respondents indicate higher than 50% reduced order processing time; 30% and 22% of the respondents achieve 0-30% and 30%-50% delivery lead time reduction, respectively.) This might be caused by the fact that there are more interactions among different functional units in the entire cycle of delivering sales orders. These interactions may result in lower improvements in sales delivery lead time.

3.2 Difficulties in designing, developing/maintaining, and using product configurators

As shown in the results, most companies experience difficulties in designing, developing/maintaining, and using product configurators. 50% of the respondents indicate that it is rather difficult for them to design product configurators. The two major reasons are (i) the lack of IT system designers (in 50% of the respondents) and (ii) IT system designers and product designers cannot communicate well (in 45% of the respondents). With our experiences of working with companies and based on the literature, these results are understandable. Manufacturing companies normally hire IT engineers for maintaining systems in support of their core business activities: design and production. In this regard, the IT engineers may not possess sufficient skills and capabilities for designing product configurators. The early literature points out that due to the differences in communication languages, configurator designers and product experts have difficulties in making effective communications (Haug et al., 2010). The results found in this study are consistent with the literature.

Similarly, most respondents have difficulties in developing/maintaining product configurators. The biggest challenge for companies to develop/maintain product configurators is the high complexity of product configurators, as supported by 52% of the respondents. The other two main difficulties include (i) the lack of good IT system developers (in 24% of the respondents) and (ii) the continuous evolution of products and the resulting high product complexity (in 24% of the respondents). While product complexities do not appear as a main difficulty in designing configurators, they do cause difficulties in developing and maintaining product configurators. This is because in accordance with product complexities, the product configurator design is complex too. It is understandable that complex product configurators are difficult to develop.

In using product configurators, companies also have difficulties. These are caused by (i) un-user friendly interfaces (in 44% of the respondents), (ii) the inefficient communications for getting required inputs (in 31% of the respondents), (iii) the high complexity of product configurators (in 12.5% of the respondents), and (iv) the lack of sufficient training (in 12.5% of the respondents). In processing customer orders, configuring products, and generating product/manufacturing documents, configurators require diverse inputs. These inputs are from customers, sales staff, designers, etc. In many cases, the input providers are from different offices or even different companies. This location dispersion may hinder the effective communications for required inputs. In addition, even in the situation where the input providers are in the same location, due to, e.g., other tasks that they need to deal with, the input providers may not be able to supply required inputs on-time. In our view, the other three difficulties are interconnected with one another. First, complex product configurators may have many interrelated modules and procedures. More training is required to understand and use these modules and procedures. However, companies are busy with dealing with daily operations activities and may not give enough

training time to users. As shown in practice, e.g., the fail of SAP project in Avon (http://blogs.wsj.com/cio/2013/12/11/avons-failed-sap-implementation-reflects-rise-of-usability/), caused by design difficulties, complex IT systems tend to have un-user friendly interfaces. In this regard, the un-user friendly interfaces may be, at least, partially related to product configurator complexities.

3.3 Issues potentially influencing the effective application of product configurators

One question in the questionnaire asked respondents for the barriers, which may potentially prevent them from effectively applying product configurators in the future. The results here are consistent with these discussed earlier. The earlier results show that companies have difficulties in designing and developing product configurators because of the lack of technical IT staff. Similarly, the lack of IT staff also appears to be a major barrier for companies to effectively use product configurators in the future. The fact that products keep evolving is one of the three difficulties in using product configurators is also acknowledged as one barrier for future use. Two additional barriers, including (i) the unclear customer requirements and (ii) the unsafe feeling of employees, are brought up. Due to the linguistic origins, customer requirements are normally imprecise and ambiguous (Jiao & Zhang, 2005), In addition, they often conflict with one another. As product configurators need articulated customer requirements, the ambiguous and conflicting requirements will negatively affect the effective application of product configurators. As a matter of fact, during the initial questionnaire pretest, some of the 5 company representatives indicated this. As product configurators execute activities, which are carried out earlier by the employees, the affected employees perceive product configurators as a menace to their positions inside the companies (Forza & Salvador, 2002). In this regard, the unsafe feeling of employees may become an obstacle for the effective application of product configurators in the future.

In summary, the largest barrier is the continuous evolution of products, as pointed out by 75% of the respondents. The other barriers include (i) the lack of technical IT staff for maintaining the configurators (seen by 47% of the respondents), (ii) the unclear customer requirements (perceived by 47% of the respondents), and (iii) the unsafe feeling from the employees because of the possibilities of losing jobs (agreed by 34% of the respondents).

4 DICUSSIONS

Along with the benefits achieved, product configurators' application brings many additional requirements and changes to companies' existing way of doing business, as shown in this study. While the changes and requirements may not be perceived beneficial, they open up opportunities for companies to improve the new way of doing business, which involves product configurators. Based on data analysis and results, this study highlights three areas for investigation: (i) IT capacity and capability enhancement, (ii) organization redesign, and (iii) top down support and company-wide engagement.

IT capacity and capability enhancement. Product configurators are basically IT systems. The optimal design and development of these systems will bring many advantages to companies, such as the configuration of optimal products, the cut-down of configuration time, the reduction of configuration errors, the easy application, the reduction of training time, etc (Haug et al., 2012). Such design and development demands sufficient system designers and developers with high skills and experiences. However, the results indicate that many companies do not have sufficient good designers and developers. In this regard, it will be beneficial to companies, especially these that design and develop configurators in house, for having sufficient well-trained system designers and developers. These designers and developers bring companies additional IT capacities and capabilities. Developing such IT capacities and capabilities can be also justified by other issues. The fact that products keep evolving necessitates continuous maintenance and upgrading to be performed (Section 3.2). Caused by its complexity, configurator maintenance and upgrading are not easy tasks and difficult to perform. In addition, if they are not well performed on-time, companies may delay product configuration, production, and delivery. This may, in turn, cause companies to lose customers. In this regard, sufficient, well-trained system designers and developers can also contribute to configurators' continuous maintenance and upgrading.

Organizational redesign. Product configurators' application brings many changes to companies existing activities, processes, and functional units (Sections 3.1.2 & 3.1.3). While simply reorganizing the affected units, as what the practice does (Section 3.1.2), may to certain degree facilitate product configurators' application, it is insufficient for companies to realize the full benefits of product configurators (Salvador & Forza, 2004). In fact, the communication difficulties (Section 3.2) lend themselves to this point. In accordance with the tasks and functions that product configurators perform, companies should reorganize their business processes and structures by reallocating the responsibilities of each individual employee and functional unit. The reorganization should be performed such that each employee has a clear vision for his activities, tasks, and responsibilities. This is the same for functional units. Besides, information exchange protocol and procedures need to be (re)designed such that communication difficulties in applying product configurators can be eliminated. At last, as one of the potential barriers for effective configurator application in the future lies in unclear customer requirements, some efforts in organization redesign may be directed to the suitable tools, techniques, systems, etc and the related issues for obtaining clear customer requirements.

Top-down support and company-wide engagement. As with the implementation of any new technologies, the implementation of product configurators needs continuous support and commitment from all levels, especially the top management level, in a company. The support and commitment is very important for completing the necessary organization changes (see above) and for successfully implementing product configurator projects. The literature shows that the lack of long-term commitment is one of the main reasons for the failure of many technology implementation projects (Bergey et al., 1999). As the employees including the middle management level have a tendency to resist changes (Paper & Chang, 2005), regular encouragement and incentives from the top management level are required to remove employees' hostile attitude towards the application of product configurators. Once the employees positively look at product configurators' application, they are willing to accept and implement organization changes. Perceived by companies, the employees' unsafe feeling for losing jobs is one of the important barriers potentially preventing companies from effectively applying configurators in the future (Section 3.3). To encourage the employees and remove their unsafe feelings, the top management

level may create more training activities. With these training activities, the employees may master additional skills. They may also involve employees in company's important meetings, share with employees company's daily or weekly news and development, etc. All these supports may help employees regain their confidence and develop correct attitudes towards configurators' application.

5 CONCLUSIONS

In view of the lasting interests that product configurators receive, this study set out to investigate the implications of product configurators' application for companies' business activities. The belief is that it is beneficial for companies to understand the difficulties and challenges before embarking upon a product configurator project. As shown in the results, product configurators' application brings many changes and difficulties along with performance improvements. The changes together with the difficulties highlight a number of areas to be investigated if companies want to achieve the optimal benefits from using product configurators.

In conducting the survey, we used nominal scales by considering the explorative nature of this study. While the nominal scale permits an easy understandable questionnaire, it makes analysis less exact than a Likert scale. In another words, it may not be able to identify the causal relationships among the interesting elements involved in the implications of configurators' application. In this regard, this study highlights an interesting future research topic. An extended quantitative method involving data analysis based on Likert scales might be conducted to reveal these causal relationships. In addition, as an initial study, the survey treated configurator development and maintenance as one issue, instead of two. As computer system development and maintenance are two different projects, the extended future study might take this into consideration. Moreover, the survey results reveal that many interesting questions need to be investigated in the future, such as in accordance with the major configurator tasks, who should be responsible for a configurator project and how they can better lead such a project, what are the configurator maintenance tasks and how can these tasks be better performed. At last, while the initial results in this study show some trends, further work is needed to investigate differences between small and large companies and between business-to-business companies and business-to-consumer companies.

REFERENCES

[1] J. Bergey, D. Smith, S. Tiley, N. Weiderman and S. Woods, Why reengineering projects fail, *Carnegie Mellon Software Engineering Institute-Product Line Practice Initiative*, **1**, 1-30, (1999).

[2] C. Forza and F. Salvador, Managing for variety in the order acquisition and fulfillment process: the contribution of product configuration systems, *International Journal of Production Economics*, **76**, 87-98, (2002).

[3] A. Haug, A software system to support the development and maintenance of complex product configurators, *International Journal of Advanced Manufacturing Technology*, 49, 393-406, (2010).

[4] A. Haug, L. Hvam and N.H. Mortensen, A layout technique for class diagrams to be used in product configuration projects, *Computers in Industry*, **61**, 409-418, (2010).

[5] A. Haug, L. Hvam and N.H. Mortensen, The impact of product configurators on lead times in engineering-oriented companies, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **25**, 197-206, (2011).

[6] A. Haug, L. Hvam and N.H. Mortensen, Definition and evaluation of product configurator development strategies, *Computers in Industry*, **63**, 471-481, (2012).

[7] http://blogs.wsj.com/cio/2013/12/11/avons-failed-sap-implementation-reflects-rise-of-usability/

[8] L. Hvam and K. Ladeby, An approach for the development of visual configuration systems, *Computers & Industrial Engineering*, **53**, 401-419, (2007).

[9] L. Hvam, J. Riis and B.L. Hansen, CRC cards for product modeling, *Computers in Industry*, **50**, 57–70, (2003).

[10] J. Jiao and Y. Zhang, Product portfolio identification based on association rule mining, *Computer-Aided Design*, **37**, 149-172, (2005).

[11] D. Paper and R. Chang, The state of business process reengineering: a search for success factors, *Total Quality Management*, **16**, 121-133, (2005).

[12] P. Pitiot, M. Aldanondo, E. Vareilles, P. Gaborit, M. Djefel and S. Carbonnel, concurrent product configuration and process planning: Towards an approach combining interactivity and optimality, *International Journal of Production Research*, **51**, 524-541, (2013).

[13] F. Salvador and C. Forza, configuring products to address the customization-responsiveness squeeze: A survey of management issues and opportunities, *International Journal of Production Economics*, **91**, 273-291, (2004).

[14] A. Trentin, E. Perin and C. Forza, Overcoming the customization-responsiveness squeeze by using product configurators: beyond anecdotal evidence, *Computers in Industry*, **62**, 260-268, (2011).

[15] A. Trentin, E. Perin and C. Forza, Product configurator impact on product quality, *International Journal of Production Economics*, **135**, 850-859, (2012).

[16] L. Zhang, E. Vareilles and M. Aldanondo, Generic bill of functions, materials, and operations for SAP2 configuration, *International Journal of Production Research*, **51**, 465-478, (2013).

Appendix: A summary of the questionnaire

General issues:

1: How long have the configurator been in use.

2. How many types of products are the configurator used for.

Product configurator applications:

1. Which tasks are performed by the configurator.
2. Who are the users.
3. Which functional units were reorganized.
4. Which business process were changed.
5. Which changes were made to other computer systems.
6. Did you layoff/hire full time employees.

Performance of product configurator applications:

1. The percentage of increased sales volume.
2. The percentage of increased correct sales orders.
3. The percentage of reduced production rework.
4. The percentage of increased customer orders accepted.
5. The percentage of reduced order processing time.
6. The percentage of reduced sales delivery time.

Difficulties in implementing product configurator project:

1. The difficulties in designing the product configurator.
2. The difficulties in developing/maintaining the configurator.
3. The difficulties in using the product configurator.

Barriers preventing companies from continuously applying product configurator:

1. The continuous evolution of products.
2. The lack of technical staff for maintaining the configurator.
3. Customer requirements are rather unclear.
4. The unsafe feeling from the employees because of the possibility of losing jobs.

Impact on cost accuracy and profitability from implementing product configuration system – A case-study

Anna Myrodia and **Katrin Kristjansdottir** and **Lars Hvam**

Abstract. This article aims at analyzing the impacts from implementing a product configuration system (PCS) on company's profitability and improved cost estimations in the sales phase. Companies that have implemented PCSs have achieved substantial benefits in terms of being more in control of their product assortment, making the right decisions in the sales phase and increasing sales of optimal products. Those benefits should have a direct impact on improved profitability in terms of increased contribution ratios and more accurate cost estimations used to determine the price in the budgetary quotations. As the literature describes, there are various benefits from implementing a PCS, however the effects on the company's profitability have not been discussed in detail. This study analyzes the impact from implementing a PCS on the accuracy of calculations in the quotations and the impact on the relative contribution ratios of products. For that reason, a configure-to-order (CTO) manufacturing company has been investigated. A longitudinal study is performed where both the accuracy of the cost calculations and the profitability is analyzed before and after the implementation of a PCS. The comparison reveals that increased profitability and accuracy of the cost estimation in the sales phase can be achieved from implementing a PCS.

1 INTRODUCTION

In today's business environment companies are forced to offer customized solutions without compromising delivery time, quality and cost [1]. In order to respond to those challenges mass customization strategies have received increasing attention over the years, both from practitioners and researchers. Mass customization refers to the ability to make customized products and services that fit every customer through flexibility and integration at cost similar to mass-produced products [2]. Utilizing product configuration systems (PCSs) is one of the key success factors in order to achieve the benefits from the mass customization approach [2][3].

PCSs are used to support design activities throughout the customization process, where a set of components along with their connections are pre-defined and where constrains are used to prevent infeasible configurations [4]. This is one of the reasons why configurations systems are considereded to be among the most successful applications of artificial intelligence [5].

Once implemented, the PCS usually supports the sales and engineering processes in various dimensions, which can lead to numerous benefits such as; shorter lead-times, more on-time deliveries, improved quality of the product specifications, less rework and increased customer satisfaction. Besides, its supportive function PCS enables improved decision making in the early phases of engineering and sales processes [6]. Furthermore, the system can be used as a tool that allows the salesperson to offer custom-tailored products within the boundaries of standard product architectures, thereby giving companies the opportunity to be more in control of their product assortment [7]. As the various benefits are described from implementing a PCS, it can be concluded that those benefits will have direct impact on the company's profitability in terms of increased contribution ratios and more accurate cost estimations in the sales phase. However the link between implementing a PCS and its effects on the company's profitability has not received much attention from researchers, even though it is one of the most critical factors during the planning phase of such a system. Ergo this article focuses on assessing the impact of the implementation of a PCS on companies' profitability by analysing the accuracy of the cost calculations in the sales phase and the profitability of the products in terms of their contribution ratios. Based on this, the following propositions are developed:

Proposition 1 *The accuracy of the cost calculations in the quotations is increased by the implementation of a PCS.*

Proposition 2 *The contribution ratio of products is increased when they are included in a PCS.*

Aiming to investigate these effects, a longitudinal case study was performed. In 2009, an analysis of the product's profitability and accuracy of the cost calculations in the quotations generated in the sales phase was conducted. The results from that analysis indicated that the performance of the sales processes could be improved by the implementation of a PCS. That recommendation was adopted by the company; hence a PCS was developed and implemented in 2011. Three years later, the same analysis was performed in order to determine the impacts on the company's profitability that could be related to the implementation of the PCS. The comparison of the results before and after the implementation of the configurator is assessed and discussed in relevance to the propositions.

This paper is structured in 5 sections. In section 2 the relevant literature will be analyzed in terms of PCSs and the benefits that can be achieved from implementing such a system. In section 3 a case study will be presented where the influence on company's profitability and the accuracy of the cost calculations from implementing a PCS will be analyzed. Then, in section 4 the conclusions from the case study are discussed. Finally, in section 5 discussion about the findings of this research work and future research are presented.

[1] Engineering Management Department, Technical University of Denmark, Denmark

Email: annamyr@dtu.dk, katkr@dtu.dk, lahv@dtu.dk

Juha Tiihonen, Andreas Falkner, and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

2 LITERATURE REVIEW

In this section the theoretical background of the present research is reported. In order to find the relevant publications a literature review has been performed in the research area of PCSs.

Forza and Salvador have performed extensive research in this field. The authors present a case-study of an CTO manufacturing company, that identifies the benefits of the implementation of a product configuration tool [8]. The benefits discussed are reduction of delivery time, improved customers' relationships and elimination of errors in Bill-Of-Material (BOM). They quantified the impact that the use of a configuration tool has on lead time, as reduction of manned activities in the tendering process (tendering lead-time from 5–6 days to one day). In addition to that, the errors in the products' BOM, misunderstandings between salespersons and customers are eliminated, while at the same time the level of correctness of product information has been increased, reaching almost 100%. In a related study [7], the authors present a case-study of a manufacturer, who's production strategy is based on customers' orders. After the implementation of a product configurator, one of the benefits identified is reduction to almost zero of the errors in the configurations released by the sales office, in addition to savings in man-hours. There have also been noted significant benefits in production, including manufacturing and assembly processes, due to the fact that by using the configuration tool correct information are received in the production. Furthermore, there has been an increase in technical productivity, both regarding product documentation release and design activities.

To this end, the benefits of using a PCS in the sales process are further investigated [9]. One of the main advantages discussed is that the configuration tool describes the possible configurations of the product in a way that they are simple and understandable by the customer. In that way, it can be ensured that there are no contradicting requirements, no missing specifications and that product configurations produced are valid. Moreover, since the configurator deals with real time information, it helps reducing dialogue time between salespersons and customers. Finally, it is highlighted that any kind of miscommunication between the salespersons and the customers is eliminated and possible errors are reduced. The reason for that is a possible source of errors in the quotations due to the sales personnel lack of technical knowledge.

The use of a product configurator and its effect on product quality is discussed by Trentin et al. [10]. The authors performed a survey in order to verify the relationship between the use of a configurator and the quality of products. The results show a positive effect on product quality by using a product configurator.

Haug et al. [11] discuss possible development strategies for product configurator and evaluate the concluding benefits. Advantages identified are, firstly, the convenience of evaluating information included in the configurator software before implementation and ease of altering implemented product information. Facilitating the communication between product experts and configurator software experts is essential in order to build a configuration tool. This results in minimizing use of resources on documentation work and handovers of information, and rapid implementation of gathered information. The benefits realized are both fewer chances of misunderstandings and errors, and faster processes. Haug et al. [12] performed another study in 14 ETO companies, where the impact of implementing a PCS on lead times is quantified. The results indicate significant improvement for the companies, as it has been measured 75% to 99.9% reduction of quotation lead time.

Another case study performed by Hvam [13] discusses and quantifies the impact from the implementation of a PCS in the electronics' industry. The main benefits from the use of configuration tools are considerably lower costs for specification processes and production. The reason for that is that when specifications are generated in the PCSs, the actual working time for preparing offers and production instructions tends to be near zero. The benefits in that case-study are quantified and show that the fixed production costs have been reduced by 50%. Additionally, the variable production costs have been reduced by 30%. On top of that quality has been improved, and is realized as a reduction from 30% to less than 2% in the number of assembly errors, as well as delivery time has been reduced from 11 – 41 days to one day. After-sales services and installation are also positively affected by the configurator. For instance, the time for replacing a battery has been reduced from 5–6 hours to 20–30 minutes.

Hvam et al. [14] performed another case measuring the impact of implementing a PCS in the ordering process of a CTO manufacturer. It is noted that only a 0.45% of the specification process time is value adding. As a result the non-value adding time spent on making the specifications can be reduced by the use of a configurator. By automating the process fewer errors occur, the productivity of employees is improved and the quality of information and documents is increased. That is due to both reducing the standard deviation of the duration of the processes and avoiding errors in quotations.

Similarly to the previous, another case-study is performed by Hvam et al. [15] in an ETO provider of cement plants. The benefits of the implementation of an IT-supported product configuration in the quotation process of complex products are aligned to those discussed above. In detail, a reduction in lead time from 15-25 days to one-two days for the generation of quotations is noted. An increase in the quality of quotes as it is made possible to optimize the cement plants satisfying better the customer's needs, and making less errors in the specifications made in the PCS. Resources consumption for making quotations is reduced in the engineering department from five man-weeks to one to two man-days.

Aldanondo et al. [16] identify the main benefits as the reduction of cost and cycle time for highly customizable products. That is due to the fact that without the support of PCSs iterative procedures occur in sales and design processes. These activities result in generating longer cycle time and increasing costs.

Slater [17] analyses the benefits of a web-based configurator in CTO environments. By using a PCS the company is able to offer the right product from the very beginning to each customer. The PCS assists the sales personnel to have an overview of the valid configurations and, therefore, avoid mistakes in the communication with the customers. This results in eliminating re-works on the customers' order. The same knowledge embedded in the configurator is used to provide unique manufacturing instructions and to make the rules for the correct configuration accessible to the engineers.

Gronalt et al. [18] outline the benefits of the implementation of a PCS, such as personalized customer support, representation of knowledge and distributed reusability of consolidated product configurations. To this end, Hong et al. [19] discusses the use of a configurator so as to reduce the information and attributes used to configure a product variant in One-of-a-kind production (OKP).

Fleischanderl et al. [20] provide empirical evidence showing that a configurator supporting the product development and production process can reduce the cost of the product's life cycle up to 60%.

The implementation of a PCS in the sales and marketing process has direct effects, such as number of errors, pricing, accuracy, time and cost to reworks, time to validate, reduce order cycle time, improve salespersons' morale, improve customers' satisfaction [21].

Empirical evidence from a built-to-order manufacturer claim benefits such as increase on customer satisfaction, lower costs and higher productivity. In addition to these, an increase in the technical accuracy of orders entering manufacturing processes is noted, which also leads to cost and errors' reduction [22].

Tenhiala and Ketokivi [23] performed a survey in make-to-order (MTO) manufacturing companies and found support to the hypothesis that the use of product configurator software in MTO production processes is positively associated with product conformance. Additional findings indicate that in general the use of configuration management practices in MTO production processes is positively associated with product conformance and delivery performance, among custom assemblers and producers.

Another problem that product configurators should focus on, according to Blecker et al. [5], is the customer perspective. They claim that designers of configurators mainly concentrate on the back-end technical aspects. By process simplification and personalization the wrong interpretation of the customer requirements by the supplier can be avoided. PCSs can therefore be beneficial both for the sales and engineering processes.

Tiihonen et al. [24] conduct a survey in Finish manufacturers with modular-based products. Their findings indicate that in extreme cases 80% of the sales specifications are either incomplete or inconsistent. At the same time, less than 20% of the total working time in the order processing is used for value-adding work. By implementing PCSs there is a reduction to the number of errors related to quality and to quality costs. Moreover, the sales' specification produced by the configurator can be directly used as an input for production, as it will not contain errors. Finally, the configuration can assist the representation of products and product families that are often differentiated in different market areas, and also the transfer of up-to-date product configuration knowledge to the sales units and to enforce its proper use.

Summarizing the findings from the literature review, it can be seen that the implementation of a PCS provides various benefits to the manufacturers, in terms of resources' reduction, reduction of lead time, better communication with customers and product quality. However, there is a lack of empirical evidence on quantifying the impact of the use of a product configuration tool on improved profitability and more accurate cost estimation. This work contributes to that fact, by providing a longitudinal case study, comparing the economic performance of the products and the accuracy of quotations before and 4 years after the implementation of a PCS in a CTO manufacturer.

3 CASE STUDY

In order to examine the propositions a case study is performed. The purpose of this case study is to illustrate the difference between cost estimation accuracy and product profitability, when using a PCS and without one. Therefore, a longitudinal case study is performed, by making similar analysis in 2009, when the company was not using a PCS, and in 2014, 4 years after the implementation of the configurator. The reason for selecting that specific period is to ensure that the PCS has been fully integrated into the business processes of the company, therefore increasing the validity of the comparison. The data for this research was gathered from company's internal databases and was verified with experts from the company.

3.1 Background

The company analysed in this case study is a Nordic company in the building industry that operates worldwide. In the year 2014 the company had around 100 employees and yearly turnover of approximately 15 million Euro. On average the company has around 50 projects per year where the average turnover per project is approximately 300.000 Euro. The company manufactures pre-made structural elements for buildings. The product portfolio consists mainly of six products; A, B, C, D, E and F. The first four products have standard product architecture that can be adjusted to the customers' needs. Products of type E denote to all the non-standardized solutions and products of type F relate to additional features or to the parts that can be added to the standard solutions.

In 2009 the process of making budgetary quotation and the accuracy of the cost estimation were analysed, which revealed the company's performance of accurate cost estimations could be improved. The analysis also revealed that the company's procedure of using Excel sheets to make the calculations of estimating the prices resulted in many errors that could be traced to human mistakes. Based on this analysis the company decided to invest 150.000 Euro to develop a PCS to improve the processes of generating budgetary quotations. The PCS used at the company is a commercial PCS, which builds on constrains propagation. In addition to that, the company also made process improvements and changes in the product assortment that aimed to increase standardisation. The implementation of the PCS also ensured that the salespersons are going to provide the customers with valid configurations from the standard product architecture.

The development of the PCS took place in the period 2009 – 2010, and in beginning of 2011 the company had developed a PCS able to handle most of the budgetary quotations. Only products of type E, which are categorized as non-standard solutions, have not been included in the system. However, due to insufficient change management, not all employees were willing to change their work procedures and therefore they still used Excel sheets to make the cost calculations for making the budgetary offers.

In this case study the impact from implementing the PCS on the company's ability to make accurate price estimations for the budgetary offers and the company's profitability will be assessed. The analyses were done both before implementing the system and for the period of its utilization over, the past 4 years (2011-2014). Thereafter the accuracy of the calculations made by using the Excel sheets and the PCS will be compared.

3.2 Analysis of the Company's Performance Before and After Implementation of Product Configuration System

In order to compare the performance before implementing the PCS (2009) and after the implementation (2011-2014), contribution margins (CM) and contribution ratio (CR) were calculated for each

project that had been carried out at the company within the timeframe of this research. Those calculations where both based on the estimations of the budgetary quotations and both the real cost and sales prices calculated after each project had been closed. CM and CR are calculated as follows [25]:

$$CM = Sales\ Price - Cost\ Price \quad (1)$$

$$CR = CM / Sales\ Price \quad (2)$$

Finally, the deviation in the CR is then calculated as actual CR minus the estimated CR.

$$DEV = CR_{actual} - CR_{estimated} \quad (3)$$

If the real cost of the project is higher than the estimated cost, it will result in negative deviation of the CR. Respectively, if the real cost of the project is less than the estimated it will result in positive deviation in the CR. The data for the analysis was extracted from the company's internal database and verified with specialist at the company. The cost prices of the projects were calculated as the sum of the costs for expenses on construction site, subcontractors, materials and salaries. The projects included in the comparison are from 2009, when only the Excel sheets were used to calculate the cost, until 2014. For the period 2011-2014 the cost calculations are either done in the PCS or by using Excel sheets. In Table 1 the main results from the analysis are listed.

Table 1. Overall analysis of the PCS contribution in terms of CR before implementation (2009) and after implementation (2011-2014)

	Performance indicators			
Year	◊	□	○	◊◊
2009	-1.5%	5.4%	14.6%	25.0%
2011	-3.6%	7.7%	28.5%	25.5%
2012	-0.7%	4.9%	8.9%	27.6%
2013	-2.4%	4.9%	9.5%	28.8%
2014	-1.1%	3.9%	2.2%	28.6%

◊ Average difference in CR □ Average absolute deviation in CR ○ Percentages of projects with greater deviation than 10% ◊◊ Average CR per project

The analysis shows that the average CR has steadily increased from 25.0% in 2009 to 28.6% in 2014. However, the overall company's goal is to have projects with CR of 30%.

The deviations in the CR also show positive improvements as the average deviations have been reduced from -1.5% in 2009 to -1.1% in 2014. Regarding the absolute value of the CR, when analyzed, the deviations showed reduction from 5.4% to 3.9%. It should also be noted that the percentages of projects with greater deviation than 10% have been significantly reduced from 14.6% in 2009 to 2.2% in 2014, as the calculations of the absolute values in the CR indicate. However in 2011, which was the first year when the PCS was utilized, the deviations in the CR increased considerably. This increased in deviations can be traced to the fact that the system had not been fully tested before implementation and the users of the system were lacking training. But as the users became more experienced in using the system and errors had been fixed, the PCS started providing valuable results.

This analysis indicates that the calculations are now more precise than before the implementation of the PCS and the company is moving closer to the targeted CR, which is 30%.

3.2.1 Analysis of Cost Structure and Deviations

In this section the company's cost structure and the deviations in the estimated and actual values with regards to the main cost elements is analysed. The purpose of that analysis is to identify whether cost estimations have been improved after the implementation of the PCS by analysing the main cost elements. The cost elements that are included in the analysis represent the direct cost of making the product, which covers the expenses related to materials, salaries, subcontractors and for the construction site, e.g. renting machines.

In 2009 a cost analysis was performed in order to assess the economic benefit by implementing a PCS. This benefit is highly associated to the reduction of deviations in the cost estimation, as the application of a PCS would thereby improve the budgetary quotations. In Figure 1 the results from the analysis are shown for the deviation in the CM for the cost calculations of the main cost elements.

Figure 1. Deviations in cost estimations for the projects 2009 distributed between the main cost elements

The figure above shows great deviations and irregularities in the pattern, which indicate that only a few projects were realized at the same cost as calculated. This means that the salespersons have estimated significantly lower costs for the projects than the actual ones, as the deviations are mainly negative. That means that the sales persons underestimated the cost of the project. That refers to all different cost categories, as they all deviate towards the same direction (positive or negative). Based on this it can be concluded that the accuracy of the calculations for the budgetary proposals were inadequate in 2009.

In order to see whether the situation has been improved at the company since implementing the PCS, a similar analysis was repeated in 2014, three years after implementing the PCS. The results from the analysis are shown in Figure 2.

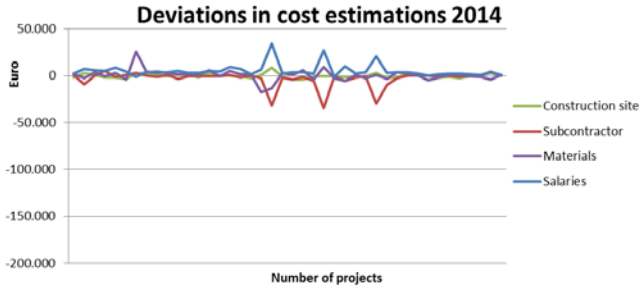


Figure 2. Deviations in cost estimations for the projects in 2014 distributed between the main cost elements

As illustrated in the above figure, when comparing the deviations in CMs of the projects from 2014, it can be clearly noted that there are far less fluctuations in the cost calculations than in 2009. Beside a few projects, the majority of them have rather low deviations from the calculated budget. By comparing the results from Figures 1 and 2, it can be realised that the accuracy of cost estimations has been improved since implementing the PCS. This is also supported by the fact that the deviations from 2009 for all the cost elements are not towards the same direction, meaning that in 2009 the deviations were negative in their overwhelming majority.

The costs in 2014 are closer to the baseline; nevertheless, three large negative spikes for the subcontractor category can clearly be seen, on the same projects with the positive spikes for the salaries. From an interview with a specialist at the company the reason for the large deviations in these three projects was due to the fact that the construction work was outsourced to subcontractors. This explains why the subcontractor category reveals large deviation and at the same time there is a positive deviation in the salary category, as the work was outsourced and therefore salaries to own employees could be reduced for the projects.

In the project where there is a large positive deviation in the material category. The explanation given was that from the time the proposal was given out and until it was finished it took several years and in the meantime a new steel structure was developed and implemented, which was much cheaper than the old structure. The large positive deviations in the cost estimations that were noticed in the 2014 analysis have therefore been reasonably explained. In the next section a more general explanation for the deviation is provided.

3.2.2 Reasons for the deviations

In order to gain a better understanding of the deviations in the cost estimations, a deeper analysis is performed, with the aim of clarifying why these deviations are still occurring at the company. The most significant deviations in the projects 2014 have been explained above, however in this section additional factors that influence the deviations will be further analysed.

The company aims to provide the customers with high quality service therefore if a customer wants to make changes to the specifications later in the process, the company will strive to adjust the solution to satisfy the customer's wishes. When such changes occur, the additional cost is added manually to the total price. This makes the actual cost of the project deviate from the initially calculated cost, but does not affect the profitability of the project.

Furthermore, the cost at the construction site is difficult to estimate since there are frequently unforeseen factors which have to be dealt with, such as difficulties to get the machines at the building side. That can result in increasing the time that the machines have to be rented and creating additional expenses due to salaries of subcontractors, which were not taken into account in the original calculation of the estimated cost. However, this threat could be reduced if technicians would examine the construction site in order to make more realistic estimations of the cost.

However, it worth mentioning that the highest peaks in the deviations of the cost calculations in 2014 are not caused by errors in the quotations, or additional costs due to unforeseen factors at the construction site but mainly because of the outsourcing work to subcontractors. Under certain circumstances, time can be limited and the company's employees might get closer to a deadline for a project and the construction team cannot finish on time. Then it might be necessary to outsource the work to subcontractors to finish on time and not delay the project, as that will also cause higher additional cost.

It can be concluded from the above analysis that the main reasons for the deviations in the cost calculations were not due to inconsistences of the PCS. Late changes from the customers, unforeseen circumstances and outsourcing are some of the main factors reported by the project managers of the company, which when occur, cause deviations in the cost calculations.

3.3 Comparison of Budgetary Proposals Made in Excel and PCS

In 2011 the PCS was first launched in the company. However, it has not been accepted by all sales representatives therefore some of them were still using Excel sheets for the calculations. The main reason for that is the lack of change management initiatives, which resulted in some employees resisting to use the PCS and therefore sticking to their old work habits. In this chapter the yearly turnover, CR of the projects and the deviations of the CR will be analysed in terms of whether they were generated by the Excel sheets or the by the PCS.

The turnover generated by projects sold through the PCS has been steadily increasing since 2011. In the year 2013 the point was reached, where slightly more proposals were generated by the use of the PCS. Figure 3 shows the yearly turnover for the proposals made in Excel and by use of the PCS.

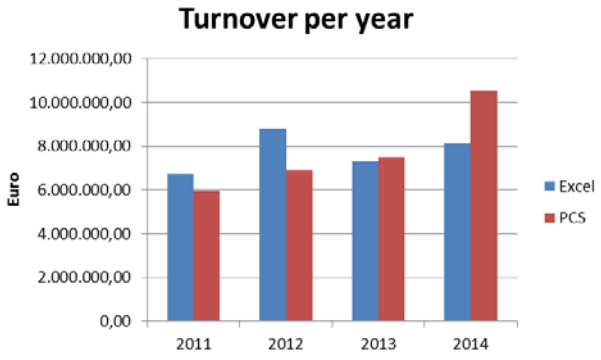


Figure 3. Comparison of turnover generated for proposals made in Excel and by use of the PCS

As can be seen from the above figure, in 2011 and 2012, the projects handled by the salespersons through the Excel sheets contributed more to the turnover of the company; even though the PCS has already been implemented in that period. This can be explained, firstly, by the reluctance that some of the salespersons showed towards including the new PCS in their working processes, as they still used Excel for the cost calculation and quotation generation. Additional, the period 2011-2012 was the initial introduction of the PCS at the company. However, the PCS did not include all products at that point of time; therefore its utilization was by definition limited. As a consequence, during the trial period the turnover contributed by the projects handled in Excel is higher than the one from the PCS, but this fact was significantly changed

in the following 2 years. So, in the period 2013-2014, when the company managed to take greater advantage of the PCS, and its utilization was strongly established, the turnover of the projects worked out by using the PCS outnumbered the ones worked out in Excel.

3.3.1 Sales Representatives and CR

The company's goal is that all projects should have a CR of 30%. An analysis of the overall performance of the company, (section 3.2) showed how the CR has been increasing since 2009. However, in order to confirm that this can be traced to the implementation of the PCS, a comparison of the CR for the budgetary proposal made by using the PCS and Excel is performed. In Figure 4, the actual CR is illustrated for the proposals made by use of the PCS and Excel.

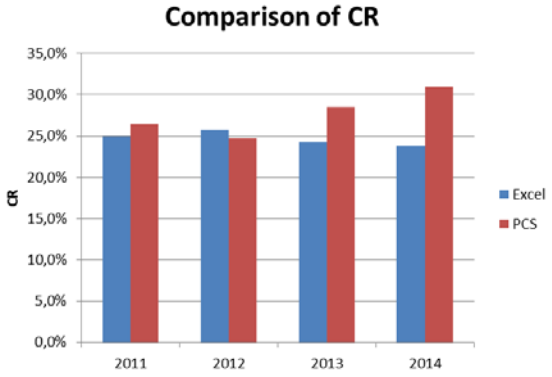


Figure 4. Comparison of CR for salespersons using Excel and PCS

From this it can be concluded that salespersons using the PCS contribute a higher CR, with an exception in the year 2012 where it was slightly lower. Furthermore, it can be seen that the gap between the CR is increasing between the salespersons using Excel and the PCS. In 2014 the average CR was 28.6%, where sale persons using the PCS had average CR 30.1% while sale persons using Excel had 23.8%. In other words, the salespersons using the PCS have managed to achieve the goal of 30%.

However as previously mentioned the products of type E, which are the non-standard products, are not included in the PCS therefore in order to make the price estimation Excel sheets are always used. Even though those products are not included in the calculations for the proposals made in Excel, when they were taken into account they only affected the CR for the proposals made in Excel by 0.2%. Therefore it can be concluded that the reason for lower CR cannot be traced to the special orders.

Another important factor is to compare the deviations in the CR between the proposals made in Excel and PCS. The results are illustrated in Figure 5.

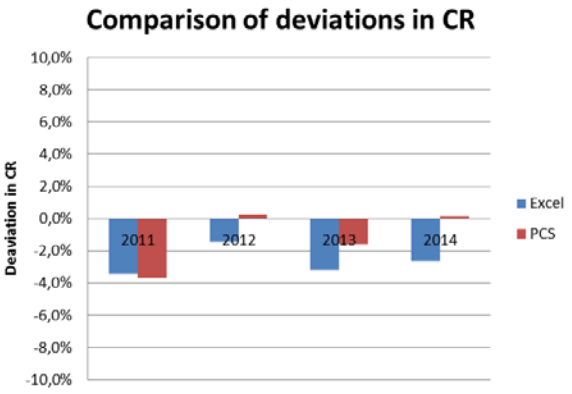


Figure 5. Comparison of deviations in CR for sale persons using Excel and PCS

As it can be seen from the above figure, the salespersons using the PCS have less deviation in their CR than the ones using Excel, with the exception of the year 2011 which was the first year the system was used in the company. Moreover, in 2012 and 2014 the deviations of the quotations made by the PCS are positive and close to 0.

The analysis of the performance of the salespersons using Excel and PCS therefore indicates that the PCS affects positively the accuracy of the cost estimation as well as the CR. Summarizing the results from this section, it can be seen that both the CR and the accuracy of estimations are improved by the utilization of the PCS. To this end, it can be concluded that the both propositions are supported by the results of the case study.

3.4 Future Initiatives

In order to improve the company's performance several factors have been identified that could reduce even further the deviations in the CR and increase the profitability of the company. The company intends to implement a check list in the end of each configuration in order to ensure that all the required information is both gathered in the sales phase and up-to-date. By implementing the check list, it is expected that errors made in the sales process will be reduced even further. Furthermore, the company is planning to enhance a higher degree of standardization in their product range, as well as move towards modular based product architecture. Finally, the company has decided to invest 140.000 Euro in further development of the PCS, in order to include more products and have greater benefits from its utilization.

4 CONCLUSIONS

The scope of this case study is to quantify the impact of implementing a PCS on product profitability and accuracy of cost estimation. The research results in significant improvements in both the CR of products sold through the PCS and the accuracy of quotations. These results confirm the propositions made in this paper. In detail, an improved performance of the margins of the products is recorded, as well as a reduction in the deviation of quotations. The comparison in the quotations' deviation is made between the same products, sold in 2009, before the company implemented the configurator, and in 2014, when the sales process

was supported by the PCS. Moreover, there is a comparison between the CR of products being sold with and without the use of the configurator for the period 2011 to 2014, when the PCS has been implemented and used to its full potential. Both comparisons lead to the same conclusion, that the contribution of the PCS is noteworthy, as the performance of the products included in the PCS is improved. Additionally, the deviations between the initial quotation provided to the customer by the PCS and the actual cost of each project are eliminated. Since the data used in the PCS is updated and all possible solutions are validated before making an offer, the quotation includes fewer errors and more accurate price estimation, when it is compared to the quotations of the products not included in the PCS.

5 DISCUSSION AND FUTURE RESEARCH

This work focuses on the benefits of implementing a PCS in a configure-to-order manufacturing company. The benefits widely discussed in the existing literature are directed towards customer satisfaction, cost reduction due to a better use of resources and elimination of errors, and improved product quality. The empirical evidence provided in this research is based on a single case study. However, the company is considered to be a typical example and highly representative in the configure-to-order industry.

This research is the first step in exploring the impact of a configurator on product's profitability. Hence, similar cases also need to be examined, in order to compare the profitability between projects going through the PCS and outside of it. By examining more cases, a deeper understanding can be gained and a more detailed explanation of the correlation between configuration tools and product profitability can be provided.

In this paper empirical evidence is provided by only one case study. Yet, the impact registered in this company indicates that there could be significant impacts from implementing a PCS, which have not been discussed in the literature previously. Therefore, this requires further research and additional case-studies in order to justify the underlying correlation between PCS and profitability increase.

REFERENCES

[1] C. Forza and F. Salvador, 'Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems', International Journal of Production Economics, 76, 87–98, (2002).

[2] B. J. Pine II, Mass Customization: The New Frontier in Business Competition, Harvard Business School Press, Boston, 1993.

[3] F. T. Piller and P. Blazek, Core capabilities of sustainable mass customization,107-120, Knowledgebased Configuration–From Research to Business Cases, Morgan Kaufmann Publishers, Waltham, 2014.

[4] A. Felfernig, G. E. Friedrich, and D. Jannach, 'UML as domain specific language for the construction of knowledge-based configuration system', *International Journal of Software Engineering and Knowledge Engineering*, 10, 449-469, (2000).

[5] T. Blecker, N. Abdelkafi, G. Kreuter and G. Friedrich, 'Product configuration systems: State-of-the-art, conceptualization and extensions' *in: A.B. Hamadou, F. Gargouri, M. Jmail (eds.): Génie logiciel & Intelligence artificielle, Eigth Maghrebian Conference on Software Engineering and Artificial Intelligencev (MCSEAI), Sousse, Tunesia, 2004*, 25–36, (2004).

[6] L. Zhang, 'Product configuration: a review of the state-of-the-art and future research', *International Journal of Production Research*, 52, 6381–6398, (2014).

[7] C. Forza and F. Salvador, 'Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems', *International Journal of Production Economics*, 76, 87–98, (2002).

[8] C. Forza and F. Salvador, 'Product configuration and inter-firm co-ordination: an innovative solution from a small manufacturing enterprise', *Computers in Industry*, 49, 37–46, (2002).

[9] C. Forza and F. Salvador, 'Application support to product variety management', *International Journal of Production Research*, 46, 817–836, (2008).

[10] A. Trentin, E. Perin, and C. Forza, 'Product configurator impact on product quality', *International Journal of Production Economics*, 135, 850–859, (2012).

[11] A. Haug, L. Hvam, and N. H. Mortensen, 'Definition and evaluation of product configurator development strategies', *Computers in Industry*, 63, 471–481, (2012).

[12] A. Haug, L. Hvam, and N. H. Mortensen, 'The impact of product configurators on lead times in engineering-oriented companies', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 25, 197–206, (2011).

[13] L. Hvam, 'Mass Customization in the electronics industry', *International Journal of Mass Customisation*, 1(04), 410 – 426, (2006).

[14] L. Hvam, M. Bonev, B. Denkena, J. Schürmeyer, and B. Dengler, 'Optimizing the order processing of customized products using product configuration', *Production Engineering*, 5, 595–604, (2011).

[15] L. Hvam, M. Malis, B. Hansen, and J. Riis, 'Reengineering of the quotation process: application of knowledge based systems', *Business Process Management Journal*, 10, 200–213, (2004).

[16] M. Aldanondo, 'Expert configurator for concurrent engineering: Cameleon software and model', *Journal of Intelligent Manufacturing*, 11, 127 – 134, (2000).

[17] P. J. P. Slater, 'Pconfig: a Web-based configuration tool for Configure-To-Order products', *Knowledge-Based Systems*, 12, 223–230, (1999).

[18] M. Gronalt, M. Posset, and T. Benna, 'Standardized Configuration in the Domain of Hinterland Container Terminals', *Series on Business Informatics and Application Systems Innovative Processes and Products for Mass Customization*, 3, 105-120, (2007).

[19] G. Hong, D. Xue, and Y. Tu, 'Rapid identification of the optimal product configuration and its parameters based on customer-centric product modeling for one-of-a-kind production' Computers in Industry, 61, 270-279, (2010).

[20] G. Fleischanderl, G. E. Friedrich, A. Haselböck, H. Schreiner and M. Stumptner, 'Configuring large systems using generative constraint satisfaction', *IEEE Intelligent Systems*,13, 59-68, (1998).

[21] J. Heatley, R. Agarwal and M. Tanniru, 'An evaluation of an innovative information technology—the case of Carrier EXPERT', *The Journal of Strategic Information Systems*, 4, 255-277, (1995).

[22] V. E. Barker, D. E. O'Connor, J. Bachant, and E. Soloway, 'Expert systems for configuration at Digital: XCON and beyond', *Communications of the ACM*, 32, 298–318, (1989).

[23] A. Tenhiälä and M. Ketokivi, 'Order Management in the Customization Responsiveness Squeeze*', *Decision Sciences*, 43, (2012).

[24] J. Tiihonen and T. Soininen, 'State of the practice in product configuration–a survey of 10 cases in the finnish industry', *Knowledge intensive CAD*, 1, 95-114, (1996).

[25] P. W. Farris, N. T. Bendle, P. E. Pfeifer and D. J. Reibstein, The Marketing Accountability - Marketing Metrics: The Definitive Guide to Measuring Marketing Performance, Pearson Education,Upper Saddle River, New Jersey, 2010.

Short paper

On Breaking The Curse of Dimensionality in Reverse Engineering Feature Models

Jean-Marc Davril and **Patrick Heymans**[1] and **Guillaume Bécan** and **Mathieu Acher**[2]

Abstract. Feature models have become one of the most widely used formalism for representing the variability among the products of a product line. The design of a feature model from a set of existing products can help stakeholders communicate on the commonalities and differences between the products, facilitate the adoption of mass customization strategies, or support the definition of the solution space of a product configurator (i.e. the sets of products that will be and will not be offered to the targeted customers). As the manual construction of feature models proves to be a time-consuming and error prone task, researchers have proposed various approaches for automatically deriving feature models from available product data. Existing reverse engineering techniques mostly rely on data mining algorithms that search for frequently occurring patterns between the features of the available product configurations. However, when the number of features is too large, the sparsity among the configurations can reduce the quality of the extracted model. In this paper, we discuss motivations for the development of dimensionality reduction techniques for product lines in order to support the extraction of feature models in the case of high-dimensional product spaces. We use a real world dataset to illustrate the problems arising with high dimensionality and present four research questions to address them.

1 Introduction

Feature Models (FMs) have been first introduced for representing the commonalities among the software systems of a software product line [15]. An FM specifies the features that form the potential products (also called configurations) of a product line and how these features can be combined to define specific products. In [4] Berger et al. survey the adoption of variability modeling techniques in industry and report that FMs are the most frequently observed notation.

Defining an FM over a set of existing configurations can bring valuable support in the adoption of *mass customization* strategies. Tseng and Jiao [18] define mass customization as the process of "*producing goods and services to meet individual customer's needs with near mass production efficiency*". A key phase in the development of a mass customization strategy is the definition of the *solution space* that should be offered by the provider [16] - that is, all the product configurations that should be made available in order to satisfy customer demand.

An FM is a concise representation of the solution space of a product line. It can be used to engineer mass customization systems, such as configurators [5, 9]. In this case the FM serves as the knowledge for the configuration system [10]. While deriving a configuration system solely from a set of existing products is not desired in practice, a reverse engineered FM can be used by stakeholders to collect valuable insights about the product domain and to assess the fit between the product line solution space and customer expectations. Depending on the complexity of the solution space and the richness of the product domain, the manual construction of an FM can prove to be time-consuming and prone to errors.

For these reason researchers have provided significant contributions in the development of techniques for automating the extraction of FMs from sets of existing products specifications (or *configurations*) - e.g. see [7, 19, 1, 14, 20, 2]. Existing approaches mostly rely on *logical techniques* that search for statistically significant relationships between the feature values. For instance, if two feature values never occur together in the configurations, one could infer a constraint stating that these values exclude each other. Similarly, two values that frequently occur together can imply a configuration dependency between the two features.

In this paper, we discuss the pitfalls related to the extraction of FMs from product configuration data. In particular, we highlight the limitation of applying logical approaches to *high dimensional* data (i.e. when the product space is defined by a very large number of features). When the number of features grows, logical methods require an increasing number of available configurations to maintain statistical significance. This means that while automatic support is desirable to help practitioners manage high dimensional product space, an FM extraction process that solely relies on logical techniques does not cope well with high dimensionality. We argue that, even when a large volume of configuration data is available, one cannot consistently assume that a logical pattern extracted from the data should be used to define the boundaries of the configuration space. It follows that while it is desirable to support practitioners with logical techniques, an FM extraction process should also consider available product domain knowledge. In the following sections, we highlight the need for formal methods that operate on both logical techniques and background domain knowledge.

The remainder of the paper is organised as follows. Section 2 describes feature modeling and prior work related to the synthesis of FMs. Section 3 illustrates the curse of dimensionality in the context of the FM synthesis problem with a real world example. In section 4 we elaborate a research plan with four research questions.

2 Feature Model Synthesis

An FM is an explicit representation of the underlying variability among the products of a product line. It defines the set of features that compose the products and how these features can be combined into products. An FM can be represented as a tree in which nodes are

[1] University Namur, Belgium, email: firstname.lastname@unamur.be
[2] Inria and Irisa, Université Rennes 1, France, email: firstname.lastname@inria.fr

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

features and edges between the features represent hierarchical relationships (see Figure 1). In the tree hierarchy, each parent-child decomposition constrains the valid combinations of features that can be found in product configurations. The FM in Figure 1 shows a *XOR-decomposition* for the feature `Screen` into the features `High definition` and `Basic` (i.e. the two child features form a *XOR-group*), which specifies that exactly one of the two child features has to be included in each configuration. Other usual decomposition types are *OR-groups* and *Mutex-groups*, which respectively define that when the parent feature is selected, all features, or at most one feature, must be included. As shown in Figure 1, filled circles and full circles represent mandatory and optional child features respectively. It is also possible to define cross-tree constraints such as the *implies* relationship in Figure 1.

An FM is *attributed* if there are typed attributes associated to its features. Figure 1 shows an attribute `size` of type *integer* under the feature `Screen`. Such FMs are referred hereafter as *Attributed Feature Models (AFM)*.

The semantics of an FM fm, noted $[\![fm]\!]$, is commonly defined as the sets of products (i.e. the sets of sets of features) that satisfy the constraints specified by fm [17]. Table 2 lists the three valid product configurations for the sample FM in Figure 1.

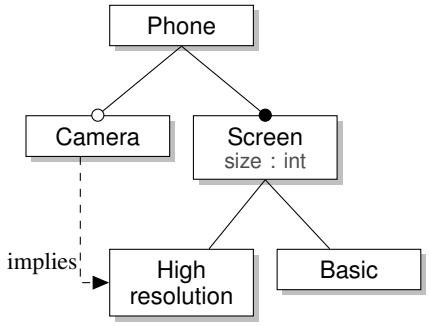


Figure 1. A sample FM for a product line of phones

Camera	High resolution	Basic
X	X	
	X	
		X

Table 1: The valid product configurations for the FM in Figure 1

The *FM synthesis problem* consists in the extraction of an FM from an existing set of products. The synthesis can be decomposed into two steps. First, a logical formula over the inclusion of features in products is mined from the set of configurations. Then, an FM is extracted from the logical formula [7]. Many different FMs can be built for the same logical formula [19]. Therefore, the FM extraction requires heuristics for guiding the selection of the edges that will form the tree hierarchy [19, 8, 20, 2].

The initial set of configurations can be represented as a *configuration matrix*, which presents the features that are included in the existing configurations, as well as the values for the feature attributes. Table 2 shows a possible initial *configuration matrix* from which the FM in Figure 1 could be synthesised.

There are multiple examples of prior work related to the synthesis of FMs from a logical formula, or from sets of formally defined configurations [7, 19, 1, 14, 20]. A major challenge in the synthesis

Camera	High resolution	Basic	size
X	X		8
X	X		7
	X		6
		X	7

Table 2: A potential product matrix that could lead to the extraction of the FM in Figure 1

of FMs is the elicitation of the FM hierarchy. She et al. [19] propose an interactive approach to recommend users with the likely parent candidates for specific features. In [8] we proposed to weight edges between features on the basis of both probabilistic dependencies between the features and similarity between their textual descriptions. We then considered the selection of the hierarchy as the computation of an optimum branching between the features [21]. The FM synthesis techniques proposed in [2] aim at producing FMs that convey a hierarchy that is conceptually sound w.r.t. ontological aspects. In [3], Bécan et al. use domain knowledge to distinguish features from attributes and propose a procedure to mine AFM.

Other works address the extraction of FMs from less structured artefacts such as textual product descriptions [23, 8, 11].

In [6] Czarnecki et al. use probabilistic logic to formalise the foundations of Probabilistic Feature Models (PFMs). The authors also propose an algorithm to build PFMs upon constraints extracted from sets of configurations. PFMs can contain *soft constraints* which express probabilistic dependencies between features.

3 The curse of dimensionality

A machine learning algorithm suffers from the effects of the so-called *curse of dimensionality* when it does not scale well with high-dimensional data. For example, performance issues can arise when the complexity of the algorithm is exponential in the number of dimensions of the dataset. High dimensionality can also impact the quality of results when some dimensions of the dataset are not relevant to the problem to be solved, and thus feed an algorithm with distracting information. Data are referred to as being high-dimensional when they are embedded into an high-dimensional space. In the context of the FM synthesis problem, the data are formed by the existing product configurations. Consequently, data dimensionality is defined by the number of features, the number of attributes, and the size of the domains for the values of these attributes.

3.1 High dimensionality in FM synthesis

We now list the variability structures that are commonly mined from configuration matrices by existing FM synthesis approaches.

- **Binary implications:** Binary implications indicate dependencies between the feature or attribute values in the configuration matrix.
- **Hierarchy:** A tree hierarchy is built from the binary implications between the features. Conceptually, the hierarchy of an FM organizes the features into different levels of increasing detail. It also defines that the selection of a child feature in a configuration always implies the selection of its parent feature.
- **Mandatory features:** Once the hierarchy has been found, for any binary implication from a parent feature to one of its child, the child has to be made mandatory.

- **Feature groups:** *OR-groups*, *XOR-groups* and *Mutex-groups* represent how sibling features can be combined together in product configurations.
- **Cross-tree constraints:** In addition to the constraints represented in the feature hierarchy, cross-tree constraints such as *requires* or *excludes* relationships are mined.

In order to illustrate the curse of dimensionality in the context of the FM synthesis problem, we have applied an AFM synthesis algorithm to a real world dataset extracted from the *Best Buy* product catalog. *Best Buy* is an American retailer that provides consumer electronics, and publishes its products data on the web through an API [3]. We built configuration matrices for the *Best Buy* data by merging extracted sets of products that have at least 75% of features and attributes in common. A description of the AFM synthesis algorithm is out of the scope of this paper and can be found in [3].

We have considered 242 extracted configuration matrices. Table 3 shows statistics about these matrices. The number of Configurations is the number of products in the matrix while the number of Variables is the number of columns (corresponding to features or attributes). Domain size is the number of distinct values in a column. The properties of the configuration matrices are quite representative of high dimensional product spaces. The number of products is low w.r.t. to the total number of distinct cell values. In our dataset, there is almost the same number of variables (columns) as configurations; and in average there are more than 5 values per variable. The application of the AFM synthesis algorithm to this dataset brings to light the need for further research efforts as summarised below.

	Min	Median	Mean	Max
Configurations	11	27.0	47.1	203
Variables (columns)	23	50.0	49.6	91
Domain size	1	2.66	5.45	47.18

Table 3: Statistics on the Best Buy dataset

Firstly, the *Best Buy* configuration matrices contain empty cells. The average proportion of empty cells in the matrices is 14.4%, and in the worst case, the proportion is 25.0% The problem with empty cells is that they do not have a clearly defined semantics in terms of variability. One might consider that an empty cells translates the absence of the corresponding feature in the configuration. However it is unsure whether the feature is really excluded, or if its value is simply unknown. This uncertainty is important because different interpretations of empty cells could lead to different synthesised FMs.

Another concern is the ability to distinguish features from attributes among the columns of the matrix. As for the empty cells, different heuristics for this task could result in very different synthesised FMs. Furthermore, each attribute should be associated to the appropriate parent feature, and automating the association resolution becomes harder as the number of features and attributes grows.

One possible direction for addressing the interpretation of empty cells and the distinction between features and attributes is to rely on the specification of domain knowledge by users. This strategy would require the design of user interactions that prevent users from being overwhelmed with huge volume of variability information, notwithstanding the large number of features and attributes in the dataset.

Another important concern is related to constraints. The number of constraints synthesised from the *Best Buy* matrices average 237 with a maximum of 8906. Such large numbers of constraints put into question the validity of the extracted constraints - that is whether these are legitimate configuration constraints w.r.t. to restrictions in the product line domain. When the data is sparse, it can be hard to evaluate whether the configurations just happened to exhibit the constraint. Moreover, when the number of constraints is high, many different FMs that fit the data equally well can be derived from them. A purely statistical synthesis approach is thus limited as it cannot be used to assess the quality of the candidate FMs. Therefore, it would be useful to automatically reduce the number of irrelevant constraints, or help users assess them. Several approaches can be considered to determine a readable subset of relevant constraints to present to users, e.g. prioritization, or minimisation [22].

Our example does not illustrate the synthesis of PFMs. While the constraints mined for crisp FMs have a confidence of 100% (i.e. they cannot be violated by any product), the constraints mined for PFMs have a confidence above a predefined threshold lower than 100%. PFMs can be useful to model *variability trends* among the products. Similar to the synthesis of FMs, a high dimensional matrix can lead to the computation of a very large number of constraints with a confidence above the predefined threshold, and thus make the elicitation of the PFM structure arduous.

3.2 Dimensionality reduction

In machine learning, the term *dimensionality reduction* denotes the process of reducing the number of attributes to be considered in the data for a particular task [12]. Dimensionality reduction techniques are commonly divided into two categories: *feature selection* and *feature extraction*.

In **feature selection**, a subset of the original data attributes is selected (see [13]). This typically involves the identification of filtering criteria on the attributes (*filter* methods) or the use of the machine learning algorithm itself for ranking the relevance of the attributes (*wrapper* methods). In an FM synthesis process, feature selection could be achieved by choosing a subset of the features to be considered during the elicitation of the FM hierarchy. Once an initial hierarchy would be computed from the core features, the filtered features could then be appended to it.

Feature extraction consists in defining a projection from the high-dimensional space of the data to a space of lower dimension. Let us consider a product line featuring the features `length`, `width` and `depth`. One could define a mathematical function over the values of these three features to replace them with a new attribute `size`, thus reducing the number of dimensions by mapping three features to a single one. The intended benefit is to reduce the cognitive effort when configuring since (1) less configuration variables are presented to users; (2) the configuration variables abstract details that are typically technical, making the promise of raising the level of abstraction for domain analysts or end-users of the engineered configurator.

4 Research Agenda

We aim at addressing dimensionality reduction in the synthesis of FMs in future research. To this end, we state four research questions:

- **RQ1: How should empty cells in configuration matrices be interpreted during the FM synthesis?** An empty cell can either represent the absence of a feature (resp. attribute) or translate a lack of knowledge for the value of a feature (resp. attribute). Acknowledging different semantics for empty cells can lead to different synthesis results. It would be interesting to investigate the

[3] http://developer.bestbuy.com/

use of complementary data, such as product descriptions or user knowledge, to set these missing values.

- **RQ2: How to use background-knowledge in the construction of FMs?** Current synthesis approaches commonly use only data for constructing FMs. However, when a configuration matrix is highly dimensional, it is possible to find many different FMs that fit the data equally well. Some researchers in the machine learning community have already considered that constructing models only from observed data is a bad practice, which has been referred to as *data fishing*. In order to select the right FM, we believe that practical synthesis procedures should be guided by existing knowledge about the application domain of the targeted FM (i.e. *background knowledge*). More specifically, background knowledge could be particularly useful for (1) differentiating the columns of a configuration matrix into features and attributes, and (2) selecting the tree hierarchy among the features of the FM.
- **RQ3: How to reduce dimensionality by modeling product qualities?** Configuration matrices represent products as sets of features, which refer to a large number of technical specifications (e.g. size, weight, battery life). However, customers usually communicate and reason about the products in terms of different abstractions we call *product qualities* (e.g. ease-of-use, portability, ergonomics). An interesting research direction is the design of formal methods for augmenting configuration matrices with product qualities. Projections of qualities onto the technical features could help define a new configuration space of lower dimensionality.
- **RQ4: How to assess the quality of extracted FMs?** One usually wants to elicit an FM with a set of specific tasks to solve in mind, which means that the quality of the FM should be assessed on the basis of the degree of support it provides w.r.t. these tasks. For instance, if an FM is extracted from a set of products with the aim of providing an overview of the underlying variability (domain analysis task), then the readability and the learnability of the FM are relevant quality criteria. In the case of the synthesis of an FM to model a solution space, the conformance of the FM to the existing products should be evaluated. A framework is thus required to identify the evaluation criteria of extracted FMs.

5 Conclusion

In this paper, we discussed the problems arising during the automatic synthesis of feature models from high-dimensional configuration matrices. We first framed concepts well-studied in the machine learning community, such as the *curse of dimensionality* and *dimensionality reduction*, in the context of the feature model synthesis problem. Using a real world dataset extracted from the *Best Buy* website, we then highlighted the steps in an attributed feature model synthesis process that require further investigations (e.g., when computing constraints). We stated research questions related to the application of FM synthesis algorithms to high dimensional configuration matrices.

The motivation for enabling domain experts to apply dimensionality reduction on configuration matrices is to synthesize variability information that is relevant w.r.t. to the intention of practitioners, and to produce more useful, readable resulting feature models.

REFERENCES

[1] Mathieu Acher, Anthony Cleve, Gilles Perrouin, Patrick Heymans, Charles Vanbeneden, Philippe Collet, and Philippe Lahire, 'On extracting feature models from product descriptions', in *Sixth International Workshop on Variability Modeling of Software-Intensive Systems*. ACM, (2012).

[2] Guillaume Bécan, Mathieu Acher, Benoit Baudry, and Sana Ben Nasr, 'Breathing ontological knowledge into feature model synthesis: An empirical study', *Empirical Software Engineering*, 51, (2015).

[3] Guillaume Bécan, Razieh Behjati, Arnaud Gotlieb, and Mathieu Acher, 'Synthesis of attributed feature models from product descriptions', in *19th International Software Product Line Conference*, Nashville, TN, USA, (2015).

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

[5] Quentin Boucher, Gilles Perrouin, and Patrick Heymans, 'Deriving configuration interfaces from feature models: A vision paper', in *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, pp. 37–44. ACM, (2012).

[6] Krzysztof Czarnecki, Steven She, and Andrzej Wasowski, 'Sample spaces and feature models: There and back again', in *Proceedings of the 12th International Software Product Line Conference*. IEEE, (2008).

[7] Krzysztof Czarnecki and Andrzej Wasowski, 'Feature diagrams and logics: There and back again', in *Proceedings of the 11th International Software Product Line Conference, 2007. SPLC'07*. IEEE, (2007).

[8] 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 *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, (2013).

[9] Deepak Dhungana, Andreas Falkner, and Alois Haselbock, 'Configuration of cardinality-based feature models using generative constraint satisfaction', in *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*. IEEE, (2011).

[10] Alexander Felfernig, 'Standardized configuration knowledge representations as technological foundation for mass customization', *IEEE Transactions on Engineering Management*, **54**(1), 41–56, (2007).

[11] Alessio Ferrari, Giorgio O Spagnolo, and Felice Dell'Orletta, 'Mining commonalities and variabilities from natural language documents', in *Proceedings of the 17th International Software Product Line Conference*, pp. 116–120. ACM, (2013).

[12] Imola K Fodor. A survey of dimension reduction techniques, 2002.

[13] Isabelle Guyon and André Elisseeff, 'An introduction to variable and feature selection', *The Journal of Machine Learning Research*, (2003).

[14] Evelyn Nicole Haslinger, Roberto Erick Lopez-Herrejon, and Alexander Egyed, 'On extracting feature models from sets of valid feature combinations', in *Fundamental Approaches to Software Engineering*, Springer, (2013).

[15] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson, 'Feature-oriented domain analysis (foda) feasibility study', Technical report, DTIC Document, (1990).

[16] FT Piller and P Blazek, 'Core capabilities of sustainable mass customization', *Knowledgebased Configuration–From Research to Business Cases. Morgan Kaufmann Publishers*, 107–120, (2014).

[17] P Schobbens, Patrick Heymans, and J-C Trigaux, 'Feature diagrams: A survey and a formal semantics', in *14th IEEE international conference on Requirements Engineering*, pp. 139–148. IEEE, (2006).

[18] J. Seng, M.M.and Jiao, 'Mass customization', in *Handbook of Industrial Engineering: Technology and Operations Management, Third Edition (ed G. Salvendy)*, (2001).

[19] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki, 'Reverse engineering feature models', in *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, pp. 461–470. IEEE, (2011).

[20] Steven She, Uwe Ryssel, Nele Andersen, Andrzej Wasowski, and Krzysztof Czarnecki, 'Efficient synthesis of feature models', *Information and Software Technology*, **56**(9), 1122–1143, (2014).

[21] Robert Endre Tarjan, 'Finding optimum branchings', *Networks*, **7**(1), 25–35, (1977).

[22] Alexander von Rhein, Alexander Grebhahn, Sven Apel, Norbert Siegmund, Dirk Beyer, and Thorsten Berger, 'Presence-condition simplification in highly configurable systems', in *Proceedings of the International Conference Software Engineering (ICSE)*, (2015).

[23] Nathan Weston, Ruzanna Chitchyan, and Awais Rashid, 'A framework for constructing semantically composable feature models from natural language requirements', in *13th International Software Product Line Conference*, pp. 211–220, (2009).

Customer buying behaviour analysis in mass customization

Tilak Raj Singh [1] and **Narayan Rangaraj** [2]

Abstract. Motivated by the importance of customer buying behaviour (such as correlation among product attributes/features of products configured in the past) in planning future configurations, this paper addresses the issue that product evolution (upgrades) usually render information gathered from past buying behaviour at least partially unusable. For instance, relations among features might have been changed, thus making it difficult to configure the same products again. The proposed approach aims to (1) find associations between product attributes based on the analysis of prior customer orders (2) apply configuration rules to prune attribute association rules which are not controlled by customers, and (3) check whether derived attribute association rules from past orders also work for the new upgraded product. Attribute associations consistent with the upgraded product are then used to predict configurations for production planning. We use machine learning algorithms and optimization techniques to address these issues.

1 Introduction

Mass customized products (e.g. Automobiles) involve a large number of product variants which are generated by combining different predefined features/attributes. Individual product attributes and attribute combinations control the final consumption of vehicle components and sub-assemblies during the production [13]. For example, the selection of features such as a specific gearbox or a sports package can decide which steering wheel will be used to build the vehicle. Thus, knowledge of customer buying behaviour (correlation between product attributes) is crucial for demand estimation of parts and subassemblies for future production.

One way to get customer buying behaviour is by extrapolating the configurations produced in the past. Due to the high degree of individualization and continuous changes in the product design, product evolution (upgrades) usually renders information gathered from past buying behaviour at least partially unusable. For instance, relations among features might have been changed, thereby not allowing configuration of the same products again. In the special case of a new product, information from existing models (having common features) is often used to prepare the initial production plan (set of vehicle configurations). As configuration rules of different products are not same, it is likely that *attribute associations* from the existing model may not be directly applicable to the new product. Thus, we need a mechanism to validate whether the derived attribute association rules from past orders also work for the new upgraded product. These consistent attribute association rules could then be used to predict future configurations for production planning [15].

Throughout this paper, we will use the terms *attribute association rules* or *attribute associations* to specify quantities reflecting the joint selection of attributes or conditional section of attributes. For example, figure 1 shows that 67% of configurations contain both attribute 1 and attribute 2.

In practice, the use of specific components in the final product assembly depends on 1) the way they are designed and 2) the way customers select them. Design or engineering related dependencies (or restrictions) are well documented in product's Bill-of-Material (BOM) or configuration rules. As the product development process starts well before the actual production, it is possible to know the product description for a future time (2-3 years in advance) from BOM [16]. However, attribute associations from the customer point of view are not known directly and can only be seen once customers have placed orders. Most manufacturers utilize information from product variants produced in the past to get estimates of customer demands. Customer behaviour can be expressed through associations between different attribute choices (e.g. joint selection) customers have made in the past.

In order to predict configurations for production planning the information from 1) configuration restrictions and 2) customer buying behaviour should be used together. Any conflicting information between these two sources points to an inconsistency in the planning information. Delay in the detection of such discrepancies may result in a wrong mix of parts being produced, thus hampering production efficiency.

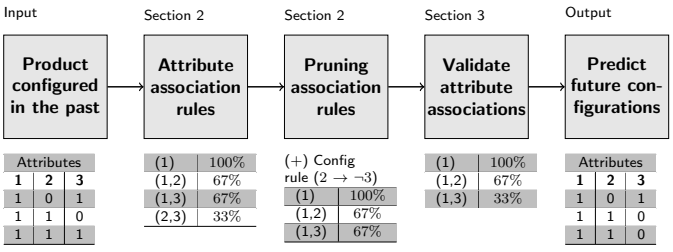


Figure 1. Predicting future configurations as per customer buying behaviour

Figure 1 shows a sequential block diagram for predicting future product configurations as per customer buying behaviour and configuration rules. First, customer prior demand is analysed to calculate attribute association rules (joint and conditional correlation between

[1] IT-Data Analytics, Mercedes-Benz R & D India, Bangalore, Email: tilak.singh@daimler.com
[2] Industrial Engineering and Operations Research, Indian Institute of Technology, Bombay, Powai Mumbai, India, email: narayan.rangaraj@iitb.ac.in

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

attributes). Then, association rules which conflict with configuration restrictions are pruned. In Section 2 we discuss a machine learning algorithm to calculate such association rules. In Section 3 we present optimization models which aim to build a set of future configurations by considering 1) configuration restriction and 2) attribute association rules simultaneously. If we are able to find such a configuration set which matches both of the input parameters, then the result can be used for future production planning. In case of conflicts between configuration restrictions and attribute association rules, further analysis is required and perhaps only a limited set of consistent association rules can be used to predict future configurations. In this case, our focus is to find such a consistent set of attribute association rules and use them to predict future configurations. Section 4 focuses on the system implementation followed by initial computational results, discussed in section 5.

2 Mining attribute association rules

A customizable product can be configured using different combinations of attributes (features). In an automobile, attribute could be body style, transmission type, sunroof or parking assistance. Customer buying behaviour can be studied by analysing how product attributes are associated with each other. Association rule mining has wide application in data mining for analysing and predicting customer behaviour [13]. In this section, we discuss a framework for mining association rules among product attributes from a given set of product configurations. As association rules are extracted from known product configurations, we first take a look at the characteristics of the configuration problem.

2.1 Product configurations

Let us define our product configuration problem as per [10, Definition 1]: the configuration problem C can be expressed through a triple (X, D, F), where:

- X is a set of product attributes (configuration variables) lets say $\{x_1, ...x_n\}$. Where n is the total number of attributes.
- D is the set of attributes finite domains $d_1, d_2, ..., d_n$.
- $F = \{f_1, f_2, ..., f_m\}$ is a set of propositional formulas (rules or restrictions) over attribute set X.

In this paper, we assume that the configuration variables $x_i \in X$ are boolean, hence domain $d_i \in \{0, 1\}, \forall i \in X$. A configuration is said to be feasible if an assignment for all attributes ($i \in X$) is found which fulfils each and every proposition in F. $X = x_1, ...x_n$ is a set and each d_i is a set. In this case, each d_i is a binary set. In other words, $\{d_1, ..., d_n\}$ is a collection of sets, whereas $\{x_1, ..., x_n\}$ are the elements of set X.

Example 1

Let us assume a car is configured using six attributes $X = \{1, 2, ..., 6\} \equiv \{$ Automatic Gearbox, Cruise control, Reverse camera, Sunroof, Keyless Go, Parktronic $\}$, $D \in \{0, 1\} \forall X$, and $F = \{f_1\}$ where

$f_1 = \{2 \rightarrow 1\}$: Cruise control requires Automatic Gearbox.

For a given set of boolean variables (attributes) and propositional formulas, finding a feasible configuration is a *Boolean Satisfiability* problem where the aim is to get an assignment (true or false value) of Boolean variables (X) which satisfies given configuration rules (F). Configurations from Table 1 can be treated as customer configurations and in the next section we will derive associations between different attributes from these. Table 1 contains a list of some feasible configurations which are created by combining given attributes and satisfying configuration rule F.

Order#	Automatic Gear-Box (AG)	Cruise Control (CC)	Reverse Camera (RC)	Sunroof (SR)	KeyLess Go (KG)	Parktronic (PA)
O001	1	1	1	1	1	0
O002	1	1	0	1	1	0
O003	1	0	1	0	1	0
O004	1	0	0	0	0	1
O005	1	1	1	1	0	1
O006	1	0	0	0	0	1
O007	1	0	1	1	1	1
O008	1	0	0	0	1	1
O009	0	0	1	0	1	1
O010	0	0	1	1	0	0

Table 1. Sample configurations (selected by customers) from Example 1

2.2 Attribute association rules and configuration restrictions

Association rule mining methodology is used to find the association between variables in large transactions [1]. In our case, each configuration is expressed over a subset of attributes, where attributes are feature/variables used to configure the product. An association between two disjoint sets of attributes p and q can be expressed using two numbers:

$Support(p \Rightarrow q)$: This is the proportion of configurations that contain both attribute sets p and q. In *Example1*, Support $(Sunroof,\ ReverseCamera)$= 4/10 = 0.4.
Confidence $(p \Rightarrow q)$: Given set of configuration which contains attributes set p, this is the proportion of configurations where attribute set q is also selected. In *Example1*, Confidence $(Sunroof \Rightarrow ReverseCamera)$ = 4/5 = 0.8. If support and confidence are greater than user specified thresholds, then we call that association rule "interesting".

After applying rule mining techniques, we get a large number of relations which satisfy our parameter of interestingness, although not all of them are customer driven. Because sales transactions do not explicitly state only customer selectable attributes, it is then our task to identify and remove attribute relations which are driven by the technical nature of the product. For example, in Table 2 (rule # 2) the association between two attributes $CruiseCtrl \Rightarrow AutomaticGearBox$ is given by Support = 0.3 and confidence = 1. The high confidence between Cruise control and Automatic Gearbox is not really driven by customer buying behaviour, but this is the only way a feasible configuration using attribute Cruise control can be created. This information is stated in the configuration rule ($F = \{f_1\}$) of Example 1.

In practical scenarios with hundreds of product attributes and thousands of configuration rules, identifying which attribute association rule is controlled by their technical dependencies is non-trivial. Also,

sr.	lhs		rhs	support	confidence
1	{CC}	⇒	{SR}	0.3	1
2	{CC}	⇒	{AG}	0.3	1
3	{PA}	⇒	{AG}	0.5	0.83
4	{KG}	⇒	{AG}	0.5	0.83
5	{SR}	⇒	{RC}	0.4	0.8
6	{SR}	⇒	{AG}	0.4	0.8
7	{CC}	⇒	{RC}	0.2	0.66
8	{CC}	⇒	{KG}	0.2	0.66
9	{RC}	⇒	{KG}	0.4	0.66
10	{RC}	⇒	{AG}	0.4	0.66
11	{SR}	⇒	{KG}	0.3	0.6
12	{PA}	⇒	{RC}	0.3	0.5
13	{PA}	⇒	{KG}	0.3	0.5
14	{SR}	⇒	{PA}	0.2	0.4
15	{CC}	⇒	{PA}	0.1	0.33

Table 2. Association rules between two attributes from Example 1

it is both error prone and time consuming to analyse and classify a large set of attribute associations manually. As the configuration problem is used to find feasible assignments of product attributes under configuration rules, we use this to state some direct dependencies among attributes. Any two attributes (let's say p and q) can be combined in a configuration based on the following relations:

$Sr\#$	Relation	p	q	Description
1	$\neg p \wedge \neg q$	0	0	Configuration without attribute p and q
2	$\neg p \wedge q$	0	1	Configuration with attribute q but not p
3	$p \wedge \neg q$	1	0	Configuration with attribute p but not q
4	$p \wedge q$	1	1	Configuration with both attribute p and q

Table 3. Possible relationships among two attributes in a configuration

Depending upon how many relations are satisfied from Table 3 any association rule can be classified in one of 2^4 possible cases. For example, if we consider Cruise control and Automatic gearbox from example 1, as attribute p and q, then with the given configuration rule, we will not be able to create a configuration which satisfies 3^{rd} relation $p \wedge \neg q$. If all the four relations are satisfied from Table 3 then we can say that the given attributes are independent of product's technical influence and any association derived from customer orders actually reflects their buying behaviour.

In the other case, let us assume that the product from Example 1 has been upgraded and new configuration restriction has been added i.e. $F = \{f_1, f_2\}$ where f_2= Parktronic comes with Reverse camera. Due to this new restriction configuration, O001, O003 and O010 from Table 1 will not be feasible for future product. Then, associations between different product attributes need to be validated against the new configuration rule. In the next section, we discuss a set of optimization models for validating attribute association rules with respect to configuration restrictions.

3 Validation of attribute association rules

In the task of validating attribute association rules for an upgraded product; our aim is to find one instance of future demand where all derived association rules are satisfied. The future demand estimate can be given in terms of a configuration set where correlation between attributes is controlled by predefined association rules.

If products are defined over sets of boolean attributes, the association rules can be expressed as boolean proposition formulas. For example, $support(p, q)$ can be modelled in a proposition formula to capture the joint selection of the attribute sets p and q. The value of support (p, q) indicates the fraction at which corresponding boolean propositional formula $(p \wedge q)$ evaluates as true in the configurations set. Thus, finding a consistent future demand estimate with respect to association rules is equivalent to satisfying a set of propositional formulas with some probability. For example, if $support(p, q) = 0.2$ then we want a clause $(p \wedge q)$ is true 20% of the time in final configurations. The resultant set of propositional formulas can be divided into two sets: 1) propositional formulas derived from the configuration problem (i.e. BOM) which has to evaluate to "true" for every demand instance 2) propositional formulas generated from association rules which are assigned a probability of being satisfied. If we are able to find a probability distribution on the truth assignments of the boolean variables corresponds to association rules that induces the given probabilities, then we will have an instance of future demand where all calculated association rules are satisfied.

For a given set of boolean variables and set of boolean clauses, determining whether it is possible to find a probability measure over truth assignments of the boolean variables that induce the given assessments is known as Probabilistic Satisfiability (PSAT) problem [9]. After modelling association rules as boolean propositional formulas and generating a solution instance where all association rules and configuration restrictions are satisfied simultaneously, we can say that derived association rules are consistent with the new configuration restrictions for the product. Our aim is to construct a configuration set such that it reflects the same support and confidence values from association rules as derived from past data. Support and confidence can be modelled as constraints in configuration problem and the associated quantity is then used to select the configuration set/ solution set to check the satisfiability [17].

3.1 The association rules verification model

In this section, we present the optimization model for evaluating the consistency among attributes association rules. Before formulating the mathematical model, let us discuss a small example to understand the underlying problem:

Example 2: *Let us assume we have discovered a customer behaviour from Table 1 that three different attributes (Reverse Camera (RC), Keyless Go (KG), Parktronic (PA)) are individually selected* 60% *of the time in prior demand. Now, new configuration restrictions (e.g. Upgraded product) specify that at least two of the attributes (out of three) have to be present in every feasible configuration. Now, is it feasible to assume that the attributes will be selected at the same rate as before?*

From given configuration rule in Example 2, at least two attributes have to be present in a feasible configuration, i.e. the following boolean clause has to be true: (RC $\vee$ KG), (KG $\vee$ PA) and (RC $\vee$ PA). If we are able to get a set of configurations where the above rules are satisfied and each attribute is selected 60% of the time in the configuration set, then we can say that derived attribute association rules and configuration restrictions are consistent with the upgraded product.

Before solving the problem of Example 2 let us formulate a general mathematical model to detect consistency among the association rules. The problem of validating association rules with respect to a new configuration rule can be defined as follows:

Let the index i refer to a logical association rule statement (Support or Confidence) defined over n Boolean variables $x_1, x_2, ...x_n$ using Boolean propositional formulas. As an Example from Table 2, x_1= {CC}, x_2= {SR}, the new variable π_1= Support(x_1,x_2) = $x_1 \wedge x_2$.

Let the index j refer to a configuration in the set J, the total configuration set (typically of very large size).

Data

π_i is the probability of i^{th} statement (attributes Support or Confidence) to be true. As an Example from Table 2, π_3=0.4

$$A_{i,j} = \begin{cases} 1 & \text{if } i^{th} \text{ statement is true in } j^{th} \text{ configuration} \\ 0 & \text{otherwise} \end{cases}$$

Decision variables

X_j = Fraction representing the proportion of j in the total configuration set, a real number between 0 and 1;
Z_i^+ = Positive deviation from target probability π_i, a real number between 0 and 1
Z_i^- = Negative deviation from target probability π_i, a real number between 0 and 1

Objective Function

$$OPT_1\textbf{: Minimize } \sum_i Z_i^+ + Z_i^- \tag{1}$$

Subject to

$$\sum_j A_{ij}X_j + Z_i^+ - Z_i^- = \pi_i \ldots \forall i \tag{2}$$

$$\sum_j X_j = 1 \tag{3}$$

$$0 \leq X_j, Z_i^+, Z_i^- \leq 1 \tag{4}$$

3.1.1 Support

Assigning support and confidence values to π_i in model OPT_1 is to control the joint probability and conditional probability among attributes. Support of any two attributes association (p $\wedge$ q) can be considered in the model OPT_1 by constraint 2 as follows:

$$\sum_j A_{p\wedge q,j}X_j + Z_{p\wedge q}^+ - Z_{p\wedge q}^- = \pi_{p\wedge q} \tag{5}$$

where $\pi_{p\wedge q}$ is equal to support (p $\Rightarrow$ q). $A_{p\wedge q,j}$ will take value one if both p and q are present in configuration j, otherwise zero.

3.1.2 Confidence

Confidence value from association rule mining can be controlled through conditional probability of given attributes. Confidence (p $\Rightarrow$ q) = $support(p \Rightarrow q)/support(q)$ = $\pi_{p|q}$.

$$\sum_j (A_{p\wedge q,j} - \pi_{p|q}A_{q,j})X_j + Z_{p|q}^+ - Z_{p|q}^- = 0 \tag{6}$$

Above equation controls the confidence (p $\Rightarrow$ q) by controlling the joint selection of p and q and individual selection of attribute q in the configuration.

Decision variables Z_i^+, Z_i^- are used to control the absolute deviation for each association rule. The matrix A lists the set of all feasible configurations which we use to match association rule quantity π_i. At this point let us say that A contains all feasible orders. From model OPT_1 if Z_i^+, Z_i^- are zero $\forall i$ then we can say the association rules are consistent among each other and also with configuration rules as we are only considering feasible configurations in the matrix A.

In Example 2, let us assume that x_1 = RC, x_2 = KG, x_3 = PA. Then the configuration constraint can be written as follows:

$x_1 + x_2 = \geq 1$, $x_1 + x_3 = \geq 1$ and $x_2 + x_3 = \geq 1$.

any combination of x_1, x_2, x_3 which satisfies the above constraint will be a possible configuration to use by model OPT_1 i.e. as a column of A-matrix. In this case, only four possible solutions are available so we can solve this example by explicitly enumerating all possible configurations.

$$OPT_{Example2}\textbf{ : Minimize } \sum_{i=1}^{3} Z_i^+ + Z_i^- \tag{7}$$

$$\underbrace{\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{pmatrix}}_{X} + \underbrace{\begin{pmatrix} Z_1^+ \\ Z_2^+ \\ Z_3^+ \end{pmatrix}}_{Z^+} - \underbrace{\begin{pmatrix} Z_1^- \\ Z_2^- \\ Z_3^- \end{pmatrix}}_{Z^-} = \underbrace{\begin{pmatrix} 0.6 \\ 0.6 \\ 0.6 \end{pmatrix}}_{\pi} \tag{8}$$

$$X_1 + X_2 + X_3 + X_4 = 1 \tag{9}$$

$$0 \leq X_j, Z_i^+, Z_i^- \leq 1 \tag{10}$$

By solving the optimization model $OPT_{Example2}$, at optimality we get objective function value 0.2 ($\neq$ 0). There are multiple optimal solutions and one is $X = 0.4, 0.2, 0.4, 0$ which means configuration 1, 2 and 3 are used 40%, 20% and 40% of the time respectively and configuration 4 is not used in the final solution. As the objective function value of $OPT_{Example2}$ model is not equal to zero, we can say that attribute association and configuration rules are not consistent with each other. However, one drawback of the model OPT_1 is that it does not explicitly specify how many association rules are satisfied. More specifically, we would like to be able to build a model which satisfies the maximum number of association rules in case of conflicting inputs as presented in $Example_2$. In the next subsection, we discuss one such model.

3.2 Maximum association rules fulfilment model

Complex products such as automobile undergo enormous changes through their life cycle. Thus, it is quite likely that some of the association rules derived from past orders may not be consistent with new configurations rules as discussed in our $Example_2$. A relevant question is whether we can maximize the coverage of association rules which can be fulfilled by an upgraded product. The OPT_1 model discussed in section 3.1 can only detect if all the association rules are satisfiable or not. If there are conflicts, we need to ideally find the minimum number of association rules, so that if we remove those, then all the remaining association rules become consistent.

With the same definitions of variables as in the OPT_1 model, let us formulate the problem as follows:

Decision variables:

$$y_i = \begin{cases} 1 & \text{if } i^{th} \text{ association rule statement is not satisfied} \\ 0 & \text{otherwise} \end{cases}$$

Objective Function

$$OPT_2 = \textbf{Minimize} \sum_i y_i \tag{11}$$

Subject to

$$\sum_j A_{ij}X_j + Z_i^+ - Z_i^- = \pi_i \ldots \forall i \tag{12}$$

$$\sum_j X_j = 1 \tag{13}$$

$$y_i \geq Z_i^+ + Z_i^- \ldots \forall i \tag{14}$$

$$0 \leq X_j, Z_i^+, Z_i^- \leq 1; y_i \in \{0, 1\} \tag{15}$$

y_i is a binary 0-1 decision variable associated with each association rule (confidence/ support). The variable y_i will take value 1 if there is some deviation between selection of attribute association $\sum_j A_{ij}X_j$ and the given rate π_i. For any association rule at a time only one variable Z_i^+ or Z_i^- will have a non zero value. Accordingly, y_i will take value 0 or 1 from constraint 14. If all association rules are consistent, y_i must be zero for all i.

In $Example_2$ the OPT_2 model will give objective function value 1 i.e. if we ignore one association rule then we can satisfy the remaining ones in the new product configuration. For example if we take only $\pi_2, \pi_3 = 0.6$ then we can build a set of feasible configurations which can satisfy both the association rules.

3.3 Solution procedure

Model formulation (OPT_1) and (OPT_2) are designed to express all possible configurations (X). This runs into the hundreds of millions! There are still two different problems with the optimization model (OPT_1) and (OPT_2):

1. How to consider all possible solutions? Very large number of decision variables.
2. How to build $A_{i,j}$ matrix?
   - Do we have to explicitly write all the columns of A?
   - Can we work with a small set of configurations and add more when needed?

The key for success here is that A_{ij} needs not to be stored or built explicitly. Based on the need, configurations can be added to A_{ij} to minimize the objective function. A solution for such a large scale optimization can be found using column generation [8]. We can start with a possible set of X_j variables. Solve OPT_1 or OPT_2 to decide which of those X_j's are included in the solution, and then try to generate a new configuration so as to improve the objective function value.

As discussed in figure 2, the overall problem is divided into two optimization problems. The first problem is to make a selection over known configurations (stored in $A_{ij} - Matrix$) such that attribute association rules can be matched as per objective function. In the next step, we want to know whether there is any feasible configuration (w.r.t. configuration rules) which improves the master problem objective function value. Let us assume that we have an 'Oracle' (sub-model) which will give us such a configuration each time we ask the question. If no such configuration is found, we conclude that there is no feasible configuration which can improve our current objective function value. The aim of the master problem in both model OPT_1 and OPT_2 is to find the optimal solution over known configurations (given columns of the A_{ij}-matrix) and the task of the sub model is to add new columns to the A_{ij}-matrix. We will repeat this procedure until no configuration is found that is worth considering. In the next section, we will discuss the formulation of sub model w.r.t. master problem OPT_1.

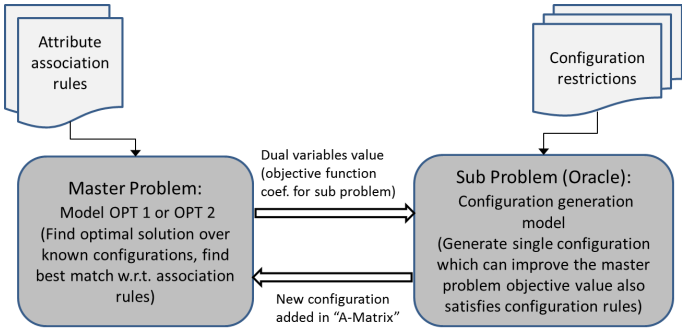


Figure 2. Column generation procedure to solve OPT_1 and OPT_2

3.4 The configuration generation model (Sub model)

In linear programming problems, in each iteration of the simplex algorithm we compute reduced costs to check if any non-basic variable can enter as part of the solution. To do so in model OPT_1, we have to evaluate if $-\sum w_i * [x_i]_j < 0$ for any configuration j. Where w_i is dual variable and $[x_i]_j$ is the new j^{th} configuration. As the value of w_i will be known from solving model OPT_1, if we know the coefficient of j^{th} order $[x_i]_j$ we can tell whether the given order will improve our objective function or not. Another way to look at this problem is to build the new j^{th} configuration so that $\sum w_i * [x_i]_j$ can be maximized.

The Sub-Problem:
Data: w_i= Dual variable from the model OPT_1, associated with constraints 2
B= Set of constraints derived from configuration rules
Decision Variable

$$[x_i]_j = \begin{cases} 1 & \text{if } i^{th} \text{ attributes association is true in new configuration j} \\ 0 & \text{otherwise} \end{cases}$$

$[x_i]_j$= new configuration for j^{th} column of configuration matrix $A_{i,j}$

Objective:

$$OPT1_{Sub} = \textbf{Maximize} \sum_i w_i * x_i \tag{16}$$

subject to:

$$B[x] \leq b \tag{17}$$

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

(i.e. x is a feasible configuration)

$$x_i = \{0, 1\} \tag{18}$$

In this formulation, we assume that configuration restrictions are modelled as a set of linear constraints as per Eq. 17 [15]. The sub-problem will generate a possible new configuration j. If this new configuration j satisfies Eq. 19 the configuration j enter the pool. This will be one column of A_{ij} matrix in $OPT1$ model. Each solution of $OPT1_{Sub}$ model will give a feasible configuration after satisfying all configuration rules from constraint 17. Configuration rules can be presented as propositional formulas and then transformed to sets of linear constraints [15].

$$\sum_i w_i * x_i \geq 0 \tag{19}$$

Dual costs are recomputed by solving the model $OPT1$ and the process terminates when no more configurations are found to be worth taking in. We use IBM ILOG Cplex engine to solve the sub-problem. The master ($OPT1$) and the subproblem ($OPT1_{Sub}$) may have to be solved multiple times before the terminating criteria is satisfied.

3.5 Column generation

The procedures discussed in section 3.1 and 3.4 works together to find a set of consistent association rules. Model $OPT1$ works with a predefined set of configurations and optimizes the current deviation with given association rules target. Model $OPT1_{Sub}$ is used to find a new configuration so that Model $OPT1$ objective function value improves. Implementing the column generation approach in association rule verification problem is done in the following way:

1 Solve the $OPT1$ model with current columns of A_{ij} matrix. In the first iteration, a feasible configuration can be used to initialize the A matrix. This iteration is used to get the dual variables which will be used in the new configuration generation model $OPT1_{Sub}$.
2 Get dual variable w_i from Eq. 2 of the $OPT1$ Model
3 Set up a sub problem as per Section 3.4
4 Get new column (order as 0-1 vector $[x]$ from solution of the sub problem discussed in section 3.4)
5 This generates a possible new configuration j with dual variable w_i
6 If configuration j satisfies $\sum_i w_i * x_i \geq 0$ (pricing inequality) then configuration j enters as j^{th} column of A_{ij}
7 Dual costs are re-computed and the process terminates when no more configurations satisfy the pricing inequality.

4 Implementation

One of our goals is to provide a software system which can 1) extract association rules from given configurations and is 2) able to use new configuration rules (upgraded product) to validate attribute association rules. Figure 3 shows the implementation flow of arriving customer driven attribute associations for predicting future configurations. We use *Apriori* algorithm through R-arules package to find a list of interesting (by support and confidence threshold) association among product attributes [11]. All derived association rules are then used in the optimization model which is implemented using IBM ILOG Cplex 12.5 and $c\#$ .net environment. At the end, a list of all consistent association rules is available with the corresponding set of configurations.

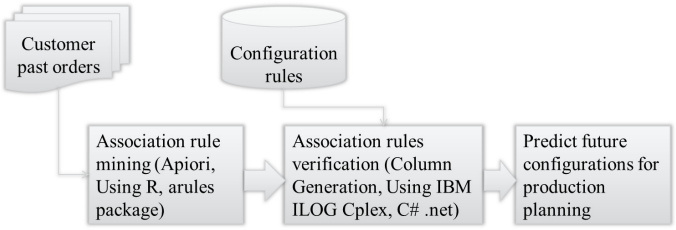


Figure 3. Implementation flow of customer drive attributes association rule mining

4.1 Association rule mining

We use Apriori algorithm implemented in "arules" package of R to derive a list of association rules. Number of attributes present in all customer orders are in the range of 500- 1,000. Total number of order is in the range of 10K to 1 Million. Even for very high cut off of support and confidence value (80%) number of generated association rules ranges in tens of thousands. One way to reduce the number

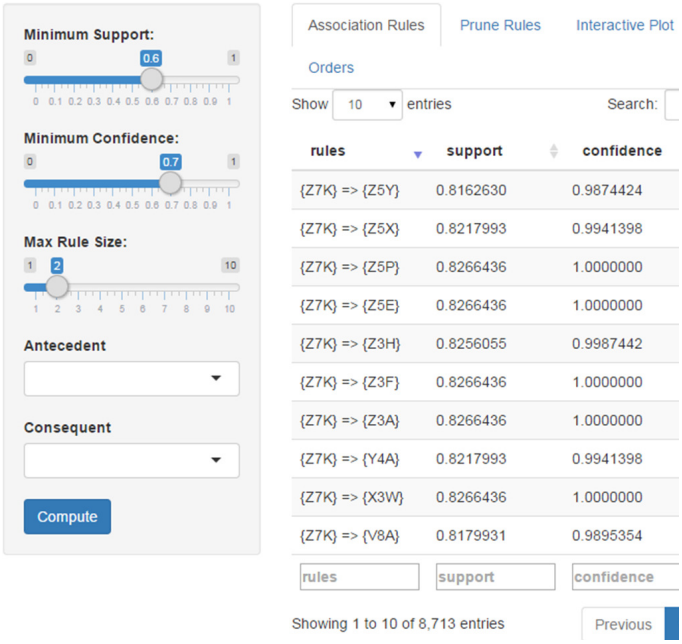


Figure 4. System overview: Deriving association rules from prior configurations using R-arules

of association rules is by limiting the attribute association level. We usually limit the association between 2 or 3 three attributes. Figure 4 shows a screen shot of actual system for association rule mining. The user can select different computational parameters such as minimum support and confidence, rule size, limiting antecedent (left-hand-side) and consequent attributes to build the desired association rules. Association rule which are according to the user specified parameters are computed from given configuration set. These association rules are used as the target to predict future product configurations.

4.2 Predicting configuration set

As a next step after computing attribute associations (support and confidence), we build the optimization model as per section 3 to compute configurations as per target input characteristics. Two optimization models are used:

1 *Master model* as per section 3.1 (OPT_1) which models selection of configuration such that sum of absolute deviation from target association rules can be minimized.
2 *Sub model* as per section 3.4 which models all configuration rules as linear inequalities.

Both the optimization model receives input from each other in every iteration of column generation procedure described in section 3.5. Optimization models run iteratively until stopping criteria (no improvement to the current solution) is met. At any iteration of column generation procedure, sub model gives a single configuration which is used as new decision variable to the master problem. We have implemented both the optimization model using cplex 12.5. As a result of the optimization procedure, a configuration set is build adhering given association rule targets. In the next section, we will discuss our first computation result with developed models.

5 Computational Results

In this section, we discuss typical computational parameters and associated numbers with input data and decision variables. We have tested our methods and models mainly on automotive data. The historical configurations are analysed at specific granularity (product-line/body style/market/engine type) to reflect current sales planning. Generally, about 500-1000 unique attributes are to be specified in a configuration set. However, not all attributes are available for customers choice as some of them are related to production. In our analysis, we have considered between 100 and 200 attributes which are available to customers. Three vehicle segments are used to test the methodology that has been developed. Table 4 shows various parameters of the data segments that we have selected. The number of orders is the number of prior configurations used to find the attribute associations (support and confidence). For this experiment, the maximum number of attributes in an association rule is limited to two i.e. association among two arbitrary attributes are computed.

Experiment #	# of attributes	# of orders	# association rules	# Pruned association rules (after applying configuration rules)
$Segment_1$	200	30,000	2,000	1,200
$Segment_2$	120	25,000	1,800	1,300
$Segment_3$	100	10,000	1,500	1,100

Table 4. Computational experiments with three different vehicle segments

Starting from a set of configurations and attributes, we use additional parameters such as minimum support and minimum confidence to compute attribute association rules. In this experiment, our aim is to find association rules which are significant (e.g. above minimum support and confidence support). After applying data mining methods, we get a large number of attribute association rules which are then filtered as per their direct dependencies/conflict with configuration rules as explain in section 2.2. By doing so, we see a significant reduction in the number of association rules. These rules are now ready to be validated with respect to the upgraded product.

Now we apply optimization model discussed in section 3.1 to build a set of configurations so that association and configuration rules are met together. In most of the cases, not all computed association rules are applicable to the upgraded product. Therefore, it is important to use only consistent rules to predict the set of future configurations. We applied the optimization model discussed in section 3.1 to build such a configuration set.

An important use of the set of predicted future configurations is to find part demand estimates for future production. In order to see the influence of consistent customer buying behaviour in parts demand, we computed parts frequency associated with order sets, where 1) only consistent attribute associations are used to predict the order set and 2) all attribute associations available after pruning w.r.t. configuration rules are used to predict the order set. The above association rules are also supplemented with a few sales forecasts (at single attribute level) to include estimates of new attributes which are not present in past orders.

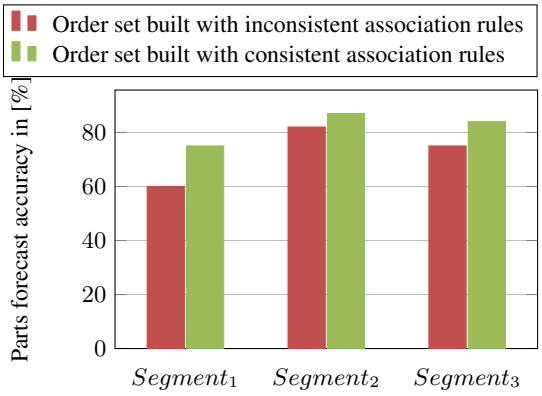


Figure 5. Comparing part demand forecast accuracy between configuration set built with consistent and inconsistent attribute association rules

Figure 5 shows the part forecast accuracy of the two scenarios discussed above. The order sets are compared with real customer orders to find the match with respect to estimated part number. About 10,000 part forecasts are compared and we used part demand matching parameter $\pm 10\%$ i.e. if the estimated value of part demand is within $\pm 10\%$ of actual demand then this forecast is considered good. With this measurement, we have compared two order set computed with consistent and non-consistent attribute association. In figure 5, for all the 3 cases we see significant improvement in forecast accuracy when consistent sets of input information are used.

6 Related work

Data mining techniques in manufacturing system are widely used to provide detailed insights regarding processes and products, such as customer segmentation, production control and quality controls [7]. In mass customization, the uncovering of aggregated level of customer buying information becomes crucial due to high product variety [14]. Data mining techniques such as association rule mining have been used in many applications such as predicting a sub-

assembly selection, and lead to significant improvement in order fulfilment process [13]. The main challenge in association rule mining technique is how to find useful association from a large set of possible attribute choices [3]. As association rules are derived from historical demands, another challenge is to validate association rules with respect to engineering changes to the product. Configuration models which capture configuration restrictions can be used to validate the list of association rules. It turns out that validating product attribute association rules against configuration rules can be formulated as Probabilistic Satisfiability Problem (PSAT) [2]. Optimization techniques such as column generation can be used to find a solution of PSAT problem [9].

Another way of building reasoning between product attributes from known configurations is through feature models [4]. The basic idea is to deduce rules/ constraints from existing product variants to support reverse engineering [12]. Feature models, combined with configuration rules, can represent hierarchical relations among different product attributes, modelling a complete set of configurations implicitly [6]. However, our aim is to build a small set of explicit configurations which can be used for production planning. The product comparison matrix is another intuitive way to highlight the differences between two products [5]. However, building such a matrix for different customer buying behaviour is a challenging task.

In our work, we have formulated configuration problem as an optimization model to give an integrated solution within the column generation framework of validating set of association rules. Our model takes support and confidence value to attribute associations to build a set of configurations adhering given input targets. In case of conflicting association rules, the model that has been developed attempts to find the maximum number of association rules which can be satisfied after considering product configuration changes.

7 Future work

In this paper, we have discussed a framework for learning customer buying behaviour through data mining and optimization-based techniques. In mass customization, due to frequent changes in products, we are required to validate product attribute associations learnt from customer prior demand. The association rule mining technique when combined with the configuration problem gives the required framework for calculating consistent and feasible attribute associations. These associations among attributes can be used as inputs for predicting configurations for future production planning. The proposed framework uses data mining libraries from R to find association rules. The integrated framework with optimization models provides the ability to perform tests on many scenarios before using any association discovered from past data to future product planning.

One application of discovering attribute associations is to use them for predicting the set of future configurations. As per our initial computational results, such a configuration set results in considerable improvement in part demand forecasts. Currently, we only consider attribute associations which are frequent (e.g. above minimum support or confidence). In the next step, the association rule mining algorithm can be enhanced to look for other relations. Also, in current implementation we simply remove attribute association which are having the conflict with each other or with product configuration rules. As a next step, we will try to develop approaches which can readjust attribute association in case of conflicts. For example, we can have the same selection rate for attributes if they are selected together. Further computational tests are required to validate and improve our assumptions on attribute associations which can improve the selection of future configurations.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami, 'Mining association rules between sets of items in large databases', in *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD-93), Washington DC*, pp. 207–216, (1993).

[2] KimAllan Andersen and Daniele Pretolani, 'Easy cases of probabilistic satisfiability', *Annals of Mathematics and Artificial Intelligence*, **33**(1), 69–91, (2001).

[3] Yves Bastide, Nicolas Pasquier, Rafik Taouil, Gerd Stumme, and Lotfi Lakhal, 'Mining minimal non-redundant association rules using frequent closed itemsets', in *Computational Logic CL 2000*, eds., John Lloyd, Veronica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, LusMoniz Pereira, Yehoshua Sagiv, and PeterJ. Stuckey, volume 1861 of *Lecture Notes in Computer Science*, 972–986, Springer Berlin Heidelberg, (2000).

[4] Guillaume Bécan, Razieh Behjati, Arnaud Gotlieb, and Mathieu Acher, 'Synthesis of attributed feature models from product descriptions: Foundations', Rapport de Recherche RR-8680, Inria Rennes, (feb 2015).

[5] Guillaume Bécan, Nicolas Sannier, Mathieu Acher, Olivier Barais, Arnaud Blouin, and Benoit Baudry, 'Automating the formalization of product comparison matrices', in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pp. 433–444, New York, NY, USA, (2014). ACM.

[6] Guillaume Bcan, Mathieu Acher, Benoit Baudry, and SanaBen Nasr, 'Breathing ontological knowledge into feature model synthesis: an empirical study', *Empirical Software Engineering*, 1–48, (2015).

[7] A.K. Choudhary, J.A. Harding, and M.K. Tiwari, 'Data mining in manufacturing: a review based on the kind of knowledge', *Journal of Intelligent Manufacturing*, **20**(5), 501–521, (2009).

[8] Gerard Cornuejols, Milind Dawande, Michel Gamache, Francois Soumis, Gerald Marquis, and Jacques Desrosiers, 'A column generation approach for large-scale aircrew rostering problems', *Oper. Res.*, **47**, 247–262, (February 1999).

[9] Fabio G. Cozman and Lucas Fargoni di Ianni, 'Probabilistic satisfiability and coherence checking through integer programming', *International Journal of Approximate Reasoning*, **58**(0), 57 – 70, (2015). Special Issue of the Twelfth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2013).

[10] T. Hadzic, S. Sathiamoorthy, R. M. Jensen, H. R. Andersen, J. Møller, and H. Hulgaard, 'Fast backtrack free product configuration using precompiled solution space representations', in *Proceedings of the International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems*, (2004).

[11] Michael Hahsler, Bettina Grün, and Kurt Hornik, 'arules - a computational environment for mining association rules and frequent item sets', *Journal of Statistical Software*, **14**(15), 1–25, (9 2005).

[12] Roberto E. Lopez-Herrejon, Lukas Linsbauer, Jos A. Galindo, Jos A. Parejo, David Benavides, Sergio Segura, and Alexander Egyed, 'An assessment of search-based techniques for reverse engineering feature models', *Journal of Systems and Software*, **103**(0), 353 – 369, (2015).

[13] Efthimia Mavridou, DionisisD. Kehagias, Dimitrios Tzovaras, and George Hassapis, 'Mining affective needs of automotive industry customers for building a mass-customization recommender system', *Journal of Intelligent Manufacturing*, **24**(2), 251–265, (2013).

[14] Rainer Paffrath, 'Mining product configurator data', in *Modern Concepts of the Theory of the Firm*, eds., Gnter Fandel, Uschi Backes-Gellner, Manfred Schlter, and JoergE. Staufenbiel, 110–121, Springer Berlin Heidelberg, (2004).

[15] Tilakraj Singh and Narayan Rangaraj, 'Generation of predictive configurations for production planning', in *Proceedings of the 15th International Configuration Workshop*, eds., Michel Aldanondo and Andreas Falkner, pp. 79–86. CEUR Workshop Proceedings, (2013).

[16] C. Sinz, A. Kaiser, and W. Küchlin, 'Formal methods for the validation of automotive product configuration data', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **17**(1), 75–97, (JAN 2003). Special issue on configuration.

[17] Peter Walley, Renato Pelessoni, and Paolo Vicig, 'Direct algorithms for checking consistency and making inferences from conditional probability assessments', *Journal of Statistical Planning and Inference*, **126**(1), 119 – 151, (2004).

Intelligent Supporting Techniques for the Maintenance of Constraint-based Configuration Systems[1][2]

Florian Reinfrank, Gerald Ninaus, Franz Wotawa, Alexander Felfernig
Institute for Software Technology
Graz University of Technology
Inffeldgasse 16b/II, 8010 Graz, Austria
{**firstname.lastname**}**@ist.tugraz.at**

Abstract. Constraint-based systems like knowledge-based recommendation and configuration are well established technologies in many different product areas like cars, computers, notebooks, and financial services. Such systems reduce the number of valid products and configurations regarding the customers' preferences. The relationship between product variables and customer questions is represented by constraints. Nowadays, constraint-based configuration systems represent volatile product assortments: Variables must be adapted, new product features lead to new questions for the customer, and / or the constraints must be updated. We call such scenarios maintenance tasks.

In complex constraint-based configuration systems the maintenance task is time consuming and error prone. Previous research focused on the detection of conflicts, repair actions for the conflicts, and redundant constraints. In this paper we give an overview about these techniques and present new approaches like recommendation, well-formedness violation, simulation, and knowledge base verification for the support of knowledge engineers.

1 Introduction

In e-Commerce applications constraint-based configuration systems are used to show which combinations of product variable assignments can be combined and offered to potential customers. Due to complex restrictions to adapt the product assortment regarding the customers' preferences, intelligent techniques can be used if the customers' preferences can not be fulfilled.

A knowledge engineer develops and maintains such knowledge bases. Based on knowledge (for example, knowledge of bikes) from domain experts, the engineer defines product variables and variable domains (e.g., the domain of the product variable $BikeType$' is $MountainBike$, $CityBike$, and $RacerBike$), prepares additional questions presented to potential customers (e.g., 'What is the main usage of the bike?'), and develops relationships (constraints) between questions and products (e.g., if the main usage of the bike is $everyday\_life$ then the bike should be of the type $CityBike$).

Such knowledge bases must be updated over time. For example, in the last years bikes with an electric engine became popular. The knowledge engineer has to extend the current knowledge base with new product attributes (e.g., introducing a new product feature $eBike$) and questions (e.g., 'Do you want an electric engine assistance?'). In complex constraint-based configuration system updates are time consuming and error prone because unexperienced knowledge engineers have to adapt the knowledge base and unexpected dependencies between constraints exist.

In this paper we show how we can support knowledge engineers when they maintain a constraint-based configuration system. The approaches can be used in many scenarios like knowledge-based recommendation, knowledge-based configuration, or feature models. Due to complex restrictions to adapt the product assortment regarding the customers' preferences, intelligent techniques can be used if the preferences can not be fulfilled.

This paper is organized as follows. Section 2 gives an overview about constraint-based configuration systems. It introduces a running example and relevant definitions for this paper. Our new approaches to support knowledge engineers in maintaining constraint-based configuration systems are described in Section 3. A summary in Section 4 concludes this paper.

2 Related Work

In this Section we give an overview about constraint-based configuration systems, introduce a running example for this paper and define relevant terms which are necessary to explain the intelligent supporting techniques from Section 3.

For our constraint-based configuration system we use the constraint satisfaction problem (CSP) modeling technique [14]. A CSP is a triple (V, D, C) and consists of a set of variables V and a set of domains D where each domain $dom(v_i)$ represents all valid assignments for a variable v_i, e.g., $dom(v_i) = \{val_1, \ldots, val_n\}$. The set C contains all constraints which restrict the number of valid instances of a constraint-based configuration system. Basically, a constraint consists of a set of assignments a for variables and relations between them. The set $A(c_i)$ is the set of assignments a constraint c_i has. If a constraint contains only one assignment, we denote such constraints *unary constraint* or *assignment* [13]. Furthermore, the constraints can be divided into two different types. First, the set C_{KB} contains all constraints which describe the domain. For example, it is not allowed to use mountain bike tires ($TireWidth > 50mm$) for racing bikes ($BikeType = RacerBike$), s.t. $c = \neg(BikeType = RacerBike \wedge TireWidth > 50mm)$. Second, the committed cus-

[1] We thank the anonymous reviewers for their helpful comments.
[2] The work presented in this paper has been conducted within the scope of the research project ICONE (Intelligent Assistance for Configuration Knowledge Base Development and Maintenance) funded by the Austrian Research Promotion Agency (827587).

tomers' preferences are represented as constraints in the set C_R.

The following example is denoted as a CSP and shows a bike knowledge base. It contains variables which represent product variables as well as customer requirements. The set C_R contains an example for customer requirements.

$V = \{BikeType, FrameSize, eBike, TireWidth, UniCycle, Usage\}$
$D = \{$
$dom(BikeType) = \{MountainBike, CityBike, RacerBike\},$
$dom(FrameSize) = \{40cm, 50cm, 60cm\},$
$dom(eBike) = \{true, false\},$
$dom(TireWidth) = \{23mm, 37mm, 57mm\},$
$dom(UniCycle) = \{true, false\},$
$dom(Usage) = \{Competition, EverydayLife, HillClimbing\}$
$\}$
$C_{KB} = \{$
$c_0 := BikeType = MountainBike \rightarrow TireWidth > 37mm \wedge FrameSize \geq 50cm;$
$c_1 := BikeType = RacerBike \rightarrow TireWidth = 23mm \wedge FrameSize = 60cm;$
$c_2 := BikeType = CityBike \rightarrow TireWidth = 37mm \wedge FrameSize \geq 50cm;$
$c_3 := \neg(BikeType \neq CityBike \wedge eBike = true);$
$c_4 := Usage = EverydayLife \rightarrow BikeType = CityBike;$
$c_5 := Usage = HillClimbing \rightarrow BikeType = MountainBike;$
$c_6 := Usage = Competition \rightarrow BikeType = RacerBike \wedge FrameSize = 60cm;$
$c_7 := eBike = true \rightarrow TireWidth = 37mm;$
$c_8 := UniCycle = false;$
$\}$
$C_R = \{$
$c_9 : FrameSize = 50cm;$
$c_{10} : Usage = Competition;$
$c_{11} : eBike = true;$
$\}$
$C = C_{KB} \cup C_R$

The example contains some anomalies in terms of conflicts, redundancies and well-formedness violations. Figure 1 gives an overview of different types of anomalies. In the following, we list definitions to define the anomalies.

The constraint set C restricts the set of valid instances. While C_{KB} remains stable during a user session she can add her preferences in the set C_R. An instance is given, if at least one customer preference is added to C_R. Definition 1 introduces the term 'instance'.

Definition 1 ***'Instance'****: An instance is given if at least one constraint in the set C_R, s.t. $C_R \neq \emptyset$.*

In a complete instance all variables in the knowledge base have at least one assignment. Definition 2 introduces the definition for a complete instance.

Definition 2 ***'Complete Instance'****: An instance is complete iff all product variables have an assignment, such that $\forall_{v \in V} v \neq \emptyset$.*

Instances can either fulfill all constraints in a constraint set C (consistent) or not (inconsistent). Definition 3 defines the term 'consistent instance'.

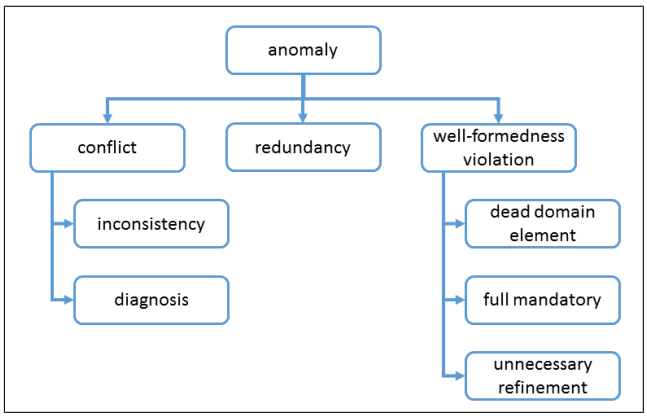


Figure 1. Different types of anomalies.

Definition 3 ***'Consistent Instance'****: An instance (complete or incomplete) is consistent, if no constraint in C is violated.*

In constraint-based configuration systems it can happen, that the system can not offer consistent instances to a user (**anomaly**) because it is not possible to satisfy all constraints (see Definition 3). Such a 'no solution could be found' dilemma is caused by at least one conflict between a) the constraints in the knowledge base C_{KB} and the customer requirements C_R or b) within the set C_{KB} itself. Definition 4 introduces a formal representation of a conflict.

Definition 4 ***'Conflict'****: A conflict is a set of constraints $CS \subseteq \{C_{KB} \cup C_R\}$ which can not be fulfilled by the CSP, s.t. CS is inconsistent.*

If we have an inconsistency in our knowledge base, we can say that $C_{KB} \cup C_R$ is always a conflict set. To have a more detailed information about the inconsistency, we introduce the term 'minimal conflict' which is described in Definition 5.

Definition 5 ***'Minimal Conflict'****: A minimal conflict CS is a conflict (see Definition 4) and the set CS only contains constraints which are responsible for the conflict, s.t. $\nexists_{c \in CS} CS \setminus \{c\}$ is inconsistent.*

When we focus on the set C_R and say, that C_{KB} is consistent, our example contains two minimal conflict sets. $CS_1 = \{c_9, c_{10}\}$ because it is not possible to have a bike for *competition* with a frame size of $50cm$ and $CS_2 = \{c_{10}, c_{11}\}$ because bikes used for *competition* do not support $eBikes$. The example shows that a knowledge base can have more than one conflict. In such cases we can help users to resolve the conflicts with diagnosis. A diagnosis Δ is a set of constraints. The removal of the set Δ from C_R leads to a consistent knowledge base, formally described in Definition 6.

Definition 6 ***'Diagnosis'****: A diagnosis Δ is a set of constraints $\Delta \subseteq C_R \cup C_{KB}$. When removing the set Δ from $C_R \cup C_{KB}$, the knowledge base will be consistent, s.t. $C_R \cup C_{KB} \setminus \Delta$ is consistent.*

Assuming that C_{KB} is consistent (see Definition 3), we can say that the knowledge base always will be consistent if we remove C_R. In Definition 7 we introduce the term 'minimal diagnosis' which helps to reduce the number of constraints within a diagnosis.

Definition 7 ***'Minimal Diagnosis'****: A minimal diagnosis Δ is a diagnosis (see Definition 6) and there doesn't exist a subset $\Delta' \subset \Delta$ which has the same property of being a diagnosis.*

The example configuration knowledge base contains two minimal diagnoses. The removal of the set $\Delta_1 = \{c_9, c_{11}\}$ or $\Delta_2 = \{c_{10}\}$ leads to a consistent configuration knowledge base. After having calculated diagnoses and removed the constraints which are in one diagnosis, we can ensure a consistent knowledge base which is necessary for calculating redundancies and well-formedness violations.
A redundancy is a set of redundant constraints in the knowledge base. A constraint c is redundant, if a knowledge base KB' without the constraint c has the same semantics[3] as the knowledge base KB which contains the constraint. Redundant constraints are formally described in Definition 8.

Definition 8 ***'Redundant constraint'****: A constraint c is redundant iff the removal of the constraint from C_{KB} leads to the same semantics, s.t. $C_{KB} \setminus \{c\} \models c$.*

In our example, the constraint c_7 is redundant, since only $CityBikes$ can be $eBikes$ (c_3) and have tires with a width of $37mm$ (c_2).
While conflicts, diagnoses, and redundancies focus on constraints, well-formedness violations identify anomalies based on variables and domain elements [2]. We now introduce well-formedness violations in constraint-based configuration systems.
The first well-formedness violation focuses on dead domain elements. A dead domain element is an element which can never be assigned to its variable in a consistent instance (see Definition 3). Definition 9 introduces a formal description of dead elements.

Definition 9 ***'Dead domain elements'****: A domain element $val \in dom(v)$ is dead iff it is never in a consistent instance, s.t. $C_{KB} \cup \{v = val;\}$ is inconsistent.*

The assignments $FrimeSize = 40cm$ and $UniCycle = true$; can never be part of a consistent instance because $MountainBikes$ and $CityBikes$ require at least $50cm$ and $RacerBikes$ require a $FrameSize$ of $60cm$ and our current knowledge base does not support $UniCycles$.
On the other hand, we can have domain elements which are assigned to each consistent instance. We denote such domain elements *full mandatory* and introduce definition 10.

Definition 10 ***'Full mandatory'****: A domain element $val_1 \in dom(v_i)$ is full mandatory iff there is no consistent (complete or incomplete) instance where the variable v_i does not have the assignment val_1, s.t. $C_{KB} \cup \{v_i \neq val_1\}$ is inconsistent.*

The knowledge base can never be consistent if $UniCycle \neq false$. In that case, we can say that the domain element $false$ of the domain $dom(UniCycle)$ is full mandatory and $UniCycle = true$ can never be in a consistent knowledge base (dead domain element).
Another well-formedness violation is called unnecessary refinement. Such an unnecessary refinement consists of two variables. If the first variable has an assignment, it is possible to predict the assignment of the second variable because the second variable can only have exactly one consistent assignment. A formal definition is given in Definition 11.

Definition 11 ***'Unnecessary refinement'****: A knowledge base contains a variable pair v_i, v_j. For each domain element val_1 of variable v_i, we can say that variable v_j always has the same assignment $v_j = val_2$, s.t. $\forall_{val_1 \in dom(v_i)} \exists_{val_2 \in dom(v_j)} v_i = val_1 \wedge v_j \neq val_2$ is inconsistent.*

[3] We use the term '*semantics*' to describe a knowledge base KB' with the same solution set as KB.

In our example the variable pair $Usage$ and $BikeType$ is unnecessary refined because whenever $Usage = EverydayLife$ the $BikeType = CityBike$, $Usage = HillClimbing$ always leads to $BikeType = MountainBike$, and $Usage = Competition$ is always combined with the assignment $BikeType = RacerBike$. If such a violation occurs, we can recommend the knowledge engineer to remove the variable $Usage$ and replace it with the variable $BikeType$ in the constraints.

3 Intelligent Support for the Maintenance of Constraint-based configuration systems

In this Section we describe existing (conflict and redundancy management) and new (recommendation, well-formedness, simulation, metrics) intelligent techniques to support knowledge engineers in their maintenance tasks.

3.1 Intelligent Recommendation

Constraint-based knowledge bases can have hundreds or thousands of variables, domain elements, and constraints. If there is a maintenance task (e.g., inserting new tire sizes), recommendation techniques help to differentiate between relevant and not relevant information within the knowledge base. For example, the tires of a bike probably have an influence on the frame of a bike but does not influence the bell of a bike. In such cases, recommendation techniques detect items (variables, domain elements, constraints, test cases) which are influenced by the tires and the knowledge engineer can focus on these items. We describe four different types of recommendation to support knowledge engineers in their maintenance tasks [4].
The first recommendation approach is the *most viewed* recommendation which is user-independent. It can be useful for new engineers of a product domain.
Second, *recently added* lists new items (products, product variables, questions, and constraints) in the knowledge base. It is user-dependent since it considers the last log in of the knowledge engineer and helps to get a fast understanding of the previous changes in the knowledge base.
The next type of recommendation is *collaborative filtering*. This type of recommendation takes the ratings for items into account and looks for knowledge engineers with similar ratings. In our case, we don't have ratings but use the interaction with items as ratings. If a knowledge engineer looks at products, she 'rates' the item with 1. 2 will be added by the knowledge engineer if she is editing an item. Table 1 shows an example for a collaborative filtering recommendation for knowledge engineer u_0 based on our example in Section 2.

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
u_0		1		1	2	1		?	
u_1		1	1		1	1		1	
u_2		1	2	1			2	2	
u_3	1		1				2		

Table 1. An example for collaborative filtering. 1 means that the item c_i is viewed by the user u_j, 2 means that the item is edited and ' ' means that the item is neither viewed nor edited by the user.

In table 1 we try to find out if we should recommend item c_7 to knowledge engineer u_0. The common process to find recommendable items is twofold. First, we try to find knowledge engineers with similar interests. In our example, u_1 and u_2 have nearly the same items viewed or edited. Second, we have to evaluate if the similar

knowledge engineers are interested in the item. Therefore, we use the Pearson correlation [3, 6]. In our example, u_1, and u_2 have viewed / edited item c_7 and we can recommend c_7 to knowledge engineer c_0. Another recommendation approach is the usage of *content-based filtering*. The basic idea is to find similar items compared to a reference item. We take the variable names and domain values from a constraint and evaluate the similarities between the reference item and all other items. The similarities are measured by the TF-IDF (term frequency and inverse document frequency) algorithm [6, 8] where the item is the document and the terms are the variables and domain elements. Table 2 shows the similarity values between constraint c_7 as reference constraint with the other constraints.

constraint	similarity
c_0	0.50
c_1	0.17
c_2	0.50
c_3	0.00
c_4	0.00
c_5	0.00
c_6	0.33
c_8	0.00

Table 2. Similarities between constraint c_7 with the other constraints based on content based recommendation

With this approach, we can say, that there is a high relationship between constraint c_7 with the constraints c_0 and c_2 and a weak relationship with the constraints c_6 and c_1.

3.2 Intelligent Anomaly Management

As mentioned in Section 2, there are many anomalies in our example knowledge base. In the following, we describe algorithms for detecting conflicts, diagnoses, redundancies, and well-formedness violations as well as explanations for those anomalies. These algorithms reduce the time to detect the anomalies and explain the anomaly to get a higher understanding of the knowledge base.

Junker [7] described a divide-and-conquer approach to detect conflicts in knowledge bases. The algorithm takes two sets of constraints as input. C_{KB} is a set of constraints which can not be part of a diagnosis. The constraints in the set C_R will be taken into account for calculating a diagnosis. If the set C_R is not empty and $C_R \cup C_{KB}$ is not consistent, the algorithm returns a set of constraints which is a minimal conflict (see Definition 5).

Algorithm 1 QuickXPlain (C_{KB}, C_R):Δ

▷ C_{KB}: set of not diagnosable constraints
▷ C_R: set of diagnosed constraints
if $isEmpty(C_R)$ *or* $consistent(C_{KB} \cup C_R)$ **then**
  $return\ \emptyset$;
else
  $return\ QuickXPlain'(C_{KB}, \Delta, C_R)$;
end if

The algorithm *QuickXPlain'* takes three sets as input. While Δ is initially empty, the set C_{KB} contains the constraints which can not be part of a conflict and the constraints which are part of a conflict are in the set C_R. Note that the set C_{KB} can also be empty. The algorithm is a recursive divide-and-conquer algorithm. It splits the set C_R into two parts (C_1 and C_2) and adds the part C_1 to C_{KB}. C_2 will be evaluated by doing a recursive call with C_2 as the set which has to be evaluated.

Algorithm 2 QuickXPlain' (C_{KB}, Δ, C_R):Δ

▷ C_{KB}: Set of constraints which can't be part of a conflict
▷ Δ: Set of constraints which can be part of a conflict
▷ C_R: Set of constraints which will be evaluated
if $\Delta \neq \emptyset$ *and* $inconsistent(C_{KB})$ **then**
  $return\ \emptyset$;
end if
if $singleton(C_R)$ **then**
  $return\ C_R$;
end if
$k \leftarrow \lceil \frac{r}{2} \rceil$;
$C_1 \leftarrow \{c_1, ..., c_k\} \in C_R$;
$C_2 \leftarrow \{c_{k+1}, ..., c_r\} \in C_R$;
$\Delta_1 \leftarrow QuickXPlain'(C_{KB} \cup C_1, C_1, C_2)$;
$\Delta_2 \leftarrow QuickXPlain'(C_{KB} \cup \Delta_1, \Delta_1, C_1)$;
$return(\Delta_1 \cup \Delta_2)$;

Contrary to QuickXPlain, FastDiag is an algorithm to calculate a minimal diagnosis and has C and C_R as input. C contains all constraints, s.t. $C = C_{KB} \cup C_R$. If C is an empty set, FastDiag has no diagnosable set and the algorithm stops. It also stops if the set $C \setminus C_R$ is inconsistent, because this set contains inconsistencies, but will not be diagnosed. If both preconditions are fulfilled, Algorithm 4 will calculate one diagnosis.

Algorithm 3 FASTDIAG(C_R, C):Δ

▷ C_R: Set of constraints which will be diagnosed
▷ C: inconsistent knowledge base including all constraints
if $C = \emptyset \vee inconsistent(CKB - C)$ **then**
  $return\ \emptyset$;
else
  $return\ DIAG(\emptyset, C_R, C)$
end if

First of all, DIAG checks whether C is consistent. If it is consistent, each subset of C is also consistent and no constraint in C can be a part of the diagnosis. Otherwise, C_R will be divided into two subsets C_1 and C_2. Each subset will be removed from C separately and checked again in a recursive manner. If $C \setminus C_1$ is consistent, we can say that C_2 is consistent and an empty set will be returned. If it is inconsistent, at least one constraint in C_1 must be part of the diagnosis and therefore C_1 will be divided and tested again unless $|C| = 1$. The algorithm returns $\Delta_1 \cup \Delta_2$ which is a minimal diagnosis.

With the previous algorithms we can a.) support customers when they do not get any products for their preferences ($C_{KB} \cup C_R$ is inconsistent) and b.) support knowledge engineers when they maintain a constraint-based configuration system with conflicts in C_{KB}. For a detailed description of the visualization of conflicts we refer the reader to [16].

When we can assume that C_{KB} is consistent, we can continue with redundancy and well-formedness checks. Please note that the following algorithms are applied to the constraint set C_{KB} and ignore C_R, s.t. $C = C_{KB}$.

The first approach for detecting redundancies has been proposed by Piette [9]. The approach is the following: a knowledge base aggregated with its negotiation must be inconsistent, formally described as $C \cup \overline{C}$ is inconsistent and $\overline{C} = \{\neg c_0 \vee \neg c_1 \vee ... \vee \neg c_n\}$. By removing a constraint c_i separately from C, the algorithm checks whether the result of $C - \{c_i\} \cup \overline{C}$ is still inconsistent. If this is the case, then the constraint c_i is redundant and can be removed. Finally, the algorithm

Algorithm 4 DIAG(Δ, C_R, C):Δ

▷ Δ: Set of diagnosed constraints
▷ C_R: Set of constraints which will be diagnosed
▷ C: $C_R \cup C_{KB}$

if $\Delta \neq \emptyset$ *and* $consistent(C)$ **then**
  $return\ \emptyset$;
end if
if $singleton(C_R)$ **then**
  $return\ C_R$;
end if
$k \leftarrow \lceil \frac{|C_R|}{2} \rceil$;
$C_1 \leftarrow \{c_0, ..., c_k\} \in C_R$;
$C_2 \leftarrow \{c_{k+1}, ..., c_n\} \in C_R$;
$\Delta_1 \leftarrow DIAG(C_1, C_2, C - C_1)$;
$\Delta_2 \leftarrow DIAG(\Delta_1, C_1, C - \Delta_1)$;
$return(\Delta_1 \cup \Delta_2)$;

returns the set C without redundant constraints.

Algorithm 5 SEQUENTIAL(C): Δ

▷ C: knowledge base
▷ $\overline{C}$: the complement of C
▷ Δ: set of redundant constraints

$C_t \leftarrow C$;
for all c_i *in* C_t **do**
  if $isInconsistent(C_t - c_i \cup \overline{C})$ **then**
    $C_t \leftarrow C_t - \{c_i\}$;
  end if
end for
$\Delta \leftarrow C - C_t$;
$return\ \Delta$;

Another approach (CoreDiag) has been proposed by Felfernig et al. [5]. Instead of a linear approach, they adapt the QuickXPlain algorithm. The divide-and-conquer approach of this algorithm checks whether removing a set of constraints C_1 leads to an inconsistency formally described as $C - C_1 \cup \overline{C}$ is inconsistent. If it is not inconsistent, C_1 must be further divided and tested again.

Algorithm 6 CoreDiag (CKB): Δ

▷ C : set with all constraints
▷ $\overline{C}$: the complement of C
▷ Δ: set of redundant constraints

$\overline{C} \leftarrow \{\neg c_1 \vee \neg c_2 \vee ... \vee \neg c_n\}$;
$return(C - \text{CoreD}(\overline{C}, \overline{C}, C))$;

CoreD (Algorithm 7) checks, if $B \subseteq C$ is inconsistent. An inconsistency of $B \cup \overline{C}$ means that the subset is not redundant and no constraint of B will be a part of Δ. $singleton(C) = true$ means that $|C|$ is redundant and will be returned. Otherwise the constraint set C will be further divided and the subsets will be checked recursively.
With the presented approaches we can calculate one conflict, diagnosis, or constraint set without redundancies. In complex knowledge bases we can assume that many anomalies are in the knowledge base. For calculating all conflicts / diagnoses / redundant constraint sets, we use Reiter's HSDAG approach [12]. This approach takes the result of one of the algorithms above and expands branches for each constraint in the result set. The constraint will be inserted into the set which can not be part of the result, e.g. a constraint c_i will be removed from C_R and added to C_{KB} in the QuickXPlain algorithm.

Algorithm 7 CoreD(B, Δ, C): Δ

▷ B: Consideration set
▷ Δ: Constraints added to B
▷ C: set of constraints to be checked for redundancy

if $\Delta \neq \emptyset$ *and* $inconsistent(B)$ **then**
  $return\ \emptyset$;
end if
if $singleton(C)$ **then**
  $return(C)$;
end if
$k \leftarrow \lceil \frac{|C|}{2} \rceil$;
$C_1 \leftarrow \{c_1, c_2, ..., c_k\} \in C$;
$C_2 \leftarrow \{c_{k+1}, c_{k+2}, ..., c_n\} \in C$;
$\Delta_1 \leftarrow \text{CoreD}(B \cup C_2, C_2, C_1)$;
$\Delta_2 \leftarrow \text{CoreD}(B \cup \Delta_1, \Delta_1, C_2)$;
$return(\Delta_1 \cup \Delta_2)$;

Hence the shifted constraint can not be part of an anomaly and further anomalies can be detected.
Both algorithms (SEQUENTIAL and CoreDiag) can be used to detect redundant constraints. As mentioned in Section 2 a constraint consists of a set of variable assignments $A(c_i)$. When we want to test if an assignment of a constraint is redundant, we have to remove the assignment from the constraint and check, if the knowledge base is still redundant. For a detailed description of assignment-based redundancy detection we refer the reader to [11].
We also have to discuss the usefulness of redundant constraints. On the one hand, desired redundancies can help to increase the understanding of a knowledge base. For example, if many implications (e.g. $A \rightarrow B; B \rightarrow C; C \rightarrow D$) are in the knowledge base, a constraint $A \rightarrow D$ may helps to understand the knowledge base. On the other hand, redundant constraints can increase the effort for updates. If the redundant constraint are not identified by the knowledge engineer, the knowledge base does not have a correct behavior anymore.
Next, we describe the algorithms to detect well-formedness violations. First, Algorithm 8 takes sets of constraints (C) and variables (V) as input parameters and returns a set of variable assignments. Each of the assignments can never be consistent with C. The suggestion for the knowledge engineer is, that the domain elements which will be returned by the algorithm can be deleted.

Algorithm 8 DeadDomainElement (C, V): Δ

▷ C: knowledge base constraints
▷ V: knowledge base variables
▷ Δ set with inconsistent variable assignments

for all $v_i \in V$ **do**
  for all $dom_j \in dom(v_i)$ **do**
    $C' = C \cup \{v_i = dom_j\}$
    if $inconsistent(C')$ **then**
      $\Delta \leftarrow \{v_i = dom_j\}$
    end if
  end for
end forreturn Δ

While we can evaluate if a domain element can never be in a consistent instance, we can also check if a domain element must be in a consistent instance of a knowledge base. We denote such domain elements as *full mandatory*. Algorithm 9 checks whether the knowledge base will be inconsistent, if the domain element dom_j is not selected.

Algorithm 9 FullMandatory (C, V): Δ

```
                                  ▷ C: knowledge base constraints
                                  ▷ V: knowledge base variables
                          ▷ Δ set with inconsistent variable assignments
for all v_i ∈ V do
    for all dom_j ∈ dom(v_i) do
        C' = C ∪ {v_i ≠ dom_j}
        if inconsistent(C') then
            Δ ← {v_i ≠ dom_j}
        end if
    end for
end forreturn Δ
```

If variable v_i contains a full mandatory domain element, we can say, that each other domain element of v_i is a dead element. If a domain element is full mandatory, we suggest the knowledge engineer to delete all other domain elements or to remove the variable itself.
Finally, we introduce an algorithm to detect unnecessary refinements between variables (see Definition 11). Algorithm 10 returns a set of constraints. Each of these constraints describe one unnecessary refinement between two variables and each domain element between both variables. The assignments between the variables are conjunctive and each domain element of variable v_i is in a disjunctive order, e.g. $(v_i = val_{i1} \wedge v_j = val_{j1}) \vee (v_i = val_{i2} \wedge v_j = val_{j2}) \vee (v_i = val_{i3} \wedge v_j = val_{j3})$.

Algorithm 10 UnnecessaryRefinement (C, V): Δ

```
                                  ▷ C: knowledge base constraints
                                  ▷ V: knowledge base variables
                                  ▷ Δ set with constraints
for all v_i ∈ V do
    for all v_j ∈ V|v_i ≠ v_j do
        A = ∅;                                  ▷ set with assignments
        for all dom_k ∈ dom(v_i) do
            dompair = false;
            C' ← C ∪ {v_i = dom_k}
            for all dom_l ∈ dom(v_j) do
                C'' ← C' ∪ {v_j ≠ dom_l}
                if inconsistent(C'') ∧ dompair = false then
                    dompair = true;
                    A ← A ∪ {v_i = dom_k ∧ v_j = dom_l}
                end if
            end for
        end for
        if |A| = |dom(v_i)| then
            Δ ← Δ ∪ disjunctive(A)
        end if
    end for
end forreturn Δ
```

The performance of this algorithm depends on the number of variables, their domain size, the number of unnecessary refinements, and the performance of the solver. In our short study with 14 knowledge bases (up to 34 variables and domain sizes from two to 47) the detection of unnecessary refinements requires up to 375 ms (with 42 unnecessary refinements) for the detection of all possible unnecessary refinements (Intel Xeon @ 2.4Ghz * 6 cores, 24GB RAM).
To get a deep understanding of the anomalies we need to explain them to the knowledge engineer [2]. For the calculation of an explanation we use the QuickXPlain algorithm (see Algorithm 2) for each type of anomaly. We take the set of constraints (e.g. set of dead elements) and add this set to C_{KB} in the algorithm. Now we have ensured, that the constraint which describes the anomaly, can't be part of Δ in the algorithm. Next, we have to negate the constraint set which describes the anomaly. Since the negation of the anomaly can never be consistent, QuickXPlain will return the set of constraints which is responsible for the anomaly. For example, the dead domain element $UniCycle = true$ will be negated and added to C. In that case, QuickXPlain will return the set $\{c_8\}$ as an explanation for the dead domain element.

3.3 Simulation

Due to the huge complexity of calculating all possible instances for all possible assignments (see Section 2) in constraint-based configuration systems we use Gibbs' simulation to estimate the consistency rate cr for a specific set of assignments A [11]. With this approximation, we can ...

> ... estimate the restriction rate (number of consistent instances compared to all instances) and evaluate the knowledge base (see Section 3.4).
> ... generate test cases for boundary value analysis [11].
> ... rank diagnoses and conflicts (assuming that a knowledge base with nearly the same restriction rate compared to the current knowledge base is preferred).
> ... generate reports for variety management (e.g. 'How many bikes can be used for $Competition$, $EverydayLife$, and $HillClimbing$?').

An assignment is a constraint which contains one variable a_v, one domain element a_d, and a relationship between variable and domain element a_r (see Section 2). Examples for assignments are $eBike = true$; and $BikeType = MountainBike$. Algorithm 11 is divided into three functions and shows the basic algorithm for estimating the consistency rate for a set of assignments.
The function $Gibbs(KB, A)$ is the main function of this algorithm. It has a knowledge base KB and a set of assignments A as input. The knowledge base contains sets of variables $V \in KB$ and constraints $C \in KB$. The set CC (checks) contains all results from consistency checks. A consistency check is either consistent (1) or inconsistent (0). The number of minimum calls is constant and given in variable $mincalls$. The total number of consistent checks is given in variable $consistent$. $threshold$ is a constant and required to test, if the current set of consistency checks has a high accuracy. If the variable $verify$ is greater than the $threshold$, we can not guarantee, that the current result is accurate. Therefore, we have to execute the loop again. In the while-loop we first have to generate a set of new random assignments. Since assignments are also special types of constraints, we add them to the set $C \in KB$ and do a consistency check again. If $randA \cup C \in KB$ is consistent, we add 1 to the set CC and increment the variable $consistent$. Otherwise, we add 0 to the set CC. Finally, we verify all previous consistency checks. If the variable $verify$ is lower than the variable $threshold$ and we have more consistency checks than $mincalls$, we can return the number of consistent checks divided by the total number of checks.
The function $generateRandAssign(KB)$ is responsible for the generation of new assignments. $Random(C)$ returns the number of assignments which has to be generated randomly. $Random(V)$ takes a variable from the knowledge base. If the variable is already part of another assignment, the variable won't be used again. $Random(R)$ selects a relation between the variable and the domain

Algorithm 11 GibbsSampling

function GIBBS(KB, A): Δ
  $CC = \emptyset$ ▷ set of consistency check results $\{0, 1\}$
  $mincalls = 200$ ▷ constant
  $threshold = 0.01$ ▷ constant
  $consistent = 0$
  $verify = Double.Max\_Value$
  while $n < mincalls \vee verify > threshold$ **do**
    $randA = A \cup$ GENERATERANDASSIGN(KB)
    $C.addAll(randA)$ ▷ $C \in KB$
    if $isConsistent(KB)$ **then**
      $consistent++$
      $CC.add(1)$
    else
      $CC.add(0)$
    end if
    $C.removeAll(randA)$
    $verify =$ VERIFYCHECKS(CC)
    $n++$
  end while
  return $consistent/n$
end function
function GENERATERANDASSIGN(KB):A
  $A = \emptyset$ ▷ A: set of assignments
  $n = Random(C) > 0$: ▷ generate n assignments
  for $i = 0; i < n; i++$ **do**
    $a_v = Random(V)$ ▷ $V \in KB$
    $a_r = Random(R)$
    $a_d = Random(dom(a_v))$
    $A.add(a)$
  end for
  return A
end function
function VERIFYCHECKS(CC):Δ
  $CC_1 = CC.split(0, |CC|/2)$
  $CC_2 = CC.split((|CC|/2) + 1, |CC|)$
  $mean1 = mean(CC_1)$
  $mean2 = mean(CC_2)$
  if $mean1 \geq mean2$ **then**
    return $mean1 - mean2$
  else
    return $mean2 - mean1$
  end if
end function

elements. In our case, variables can have textual domain elements (e.g. the brand of a bike) or numeric domain elements (e.g. the price of a bike). While the set of relations for textual domain elements is $R = \{=, \neq\}$, the set is extended to $R = \{=, \neq, <, \leq, >, \geq\}$ for numerical domain elements. Finally, $Random(dom(a_v))$ selects a domain element from $dom(a_v)$ randomly.
The function $verifyChecks(CC)$ tests if the number of consistent and inconsistent checks are normally distributed. Therefore, we first divide the set with the consistency check results CC into two parts. We evaluate the mean of both sets CC_1 and CC_2 and test if both mean values are near to each other. If they have nearly the same value, we can say that the consistent checks are normally distributed in both sets and return the difference between $mean1$ and $mean2$.

3.4 Knowledge base Evaluation

As mentioned in the previous Sections, we can analyze a knowledge base in different ways and collect a lot of information about the knowledge base. Finally, we can also evaluate the knowledge base in terms of metrics. Those metrics a) help to get information about the quality of the knowledge base and b) get information about the quality of previous changes. Next, we will describe some metrics, use them to answer five questions, and measure three goals for constraint-based configuration systems (goal-question-metrics). The metrics are based on a literature review focusing on knowledge engineering. An overview of the literature review is given in [10]. In the following list we describe several metrics.

- Number of variables $|V| \in KB$.
- Average domain size $domsize$: $\frac{\sum_{v_i \in V} |dom(v_i)|}{|V|}$
- Number of constraints $|C_{KB}| \in KB$
- Number of minimal conflicts $|CS|$: see Definition 3
- Minimal cardinality CS $MCCS$: the lowest number of constraints in a conflict set
- Number of minimal diagnoses $|\Delta|$: see Definition 5
- Minimal cardinality diagnosis $MC\Delta$: the lowest number of constraints in a diagnosis
- Number of redundancy sets $|R|$
- Maximal cardinality redundancy set MCR: the largest number of constraints in a redundancy set
- dead elements DE: number of dead elements compared to the total number of all domain elements

$$DE = \frac{\sum_{v_i \in V} \sum_{d_j \in dom(v_i)} \begin{cases} 0 & C \cup \{v_i = d_j\} \neq \emptyset \\ 1 & else \end{cases}}{|V| \times domsize}$$

- full mandatory FM: number of full mandatory domain elements compared to the total number of all domain elements

$$FM = \frac{\sum_{v_i \in V} \sum_{d_j \in dom(v_i)} \begin{cases} 0 & C \cup \{v_i \neq d_j\} = \emptyset \\ 1 & else \end{cases}}{|V| \times domsize}$$

- unnecessary refinement UR: whenever a variable v_i has an assignment, we can predict the assignment of variable v_j, s.t. $dom(v_i) \rightarrow dom(v_j)$
- restriction rate RR: $\frac{|C|}{|V|}$
- restriction rate RR_2: $\frac{\sum_{c_i \in C} \frac{\#vars(c_i)}{\#vars(C)} |C|}{|V|}$ where $\#vars(c_i)$ is the number of variables in c_i.
- variable influence factor $VIF(v_i)$: number of constraints in which a variable v_i appears related to the number of constraints, e.g., $VIF(v_i) = \frac{\sum_{c_i \in C} \begin{cases} 1 & v_i \in c_i \\ 0 & else \end{cases}}{|C|}$.
- variable influence factor VIF_{all}: average influence of all variables $\sum_{v_i \in V} \frac{\sqrt{(VIF(v_i) - \frac{\sum_{v_j \in V} VIF(v_j)}{|V|})^2}}{|V|}$
- coverage $coverage$: GIBBSSAMPLING($KB, \emptyset$) (see Section 3.3)

With the metrics we collected a lot of information about the knowledge base. To evaluate the knowledge base, we aggregate the metrics and use the goal-question-metrics approach [1] to quantify the quality of the knowledge base.

The aggregation of metrics will be used to answer questions relating to one or more goals. Next, we are listing the questions and the corresponding metrics for each question:

Q1: *Is the configuration knowledge base complete?*:
$|V|, domsize, |C|, coverage, |CS|, |\Delta|$
Q2: *Does the knowledge base contain anomalies?*:
$|CS|, |\Delta||R|, DE, FM, UR$
Q3: *Does the configuration knowledge base have an admissible performance?*:
$|V|, domsize, |C|, |R|DE, FM, UR$
Q4: *Is the configuration knowledge base modifiable?*:
$MCCS, MC\Delta, MCR, DE, FM, UR, RR, RR_2$,
$VIF_{all}, Coverage$
Q5: *Is the configuration knowledge base understandable?*:
$MCCS, MC\Delta, MCR, DE, FM, UR, RR, RR_2$,
$VIF_{all}, coverage$

Based on the answers for these questions we can evaluate the quality of a knowledge base. The quality will be measured in terms of three goals which we will list in the following:

G1: A configuration knowledge base must be **maintainable**, such that it is easy to change the semantics of the knowledge base in a desired manner (corresponding questions: Q2 anomalies, Q4 modifiability)
G2: A configuration knowledge base must be **understandable**, such that the effort for a maintainability task for a knowledge engineer can be evaluated (corresponding questions: Q2 anomalies, Q5 understandability)
G3: A configuration knowledge base must be **functional**, such that it represents a part of the real world (e.g. a bike configuration knowledge base; corresponding questions: Q1 completeness, Q2 anomalies, Q3 performance).

The results of the GQM-approach can be explained by a comparison with the measurements of previous versions of the knowledge base. The comparison can show, if maintainability, understandability, and functionality increases or decreases over time and explain the changes (based on a comparison of the answers for the questions and metrics). For a detailed description of the GQM-approach for constraint-based configuration systems we refer the reader to [10].

4 Summary

In this paper we presented approaches to improve the maintenance for constraint-based configuration systems. We described the state-of-the-art in conflict and redundancy management and introduced recommendation for the support of knowledge engineers. New anomaly detection algorithms can be used to detect well-formedness violations. Simulation techniques in the context of constraint-based configuration systems allow us to approximate metrics for the goal-question-metrics approach. We implemented these approaches in our web-based system called 'iCone' (Intelligent environment for the development and maintenance of configuration knowledge bases) [15].[4]

While we presented novel approaches to support knowledge engineers, further research has to be done in the *verification* of the new recommendation, simulation, and metrics evaluation techniques. Furthermore, *micro tasks* can be used to collect and verify assumptions of knowledge engineers about the knowledge base. Further research should also be done in the context of *stakeholder integration*. For example, in the software engineering process it is common that several stakeholders (e.g. customers and users) can participate in the engineering process. For the integration of different stakeholders and to optimize the knowledge engineering, further research should also be done in the context of *knowledge engineering processes* and *knowledge base development*.

[4] http://ase-projects-studies.ist.tugraz.at:8080/iCone/

REFERENCES

[1] Victor R. Basili. Software modeling and measurement: The goal/question/metric paradigm. Technical report, University of Maryland at College Park College Park, MD, USA, College Park, MD, USA, 1992.
[2] Alexander Felfernig, David Benavides, Jos A. Galindo, and Florian Reinfrank. Towards anomaly explanation in feature models. *Workshop on Configuration*, pages 117 – 124, 2013.
[3] Alexander Felfernig, Paul Blazek, Florian Reinfrank, and Gerald Ninaus. *Intelligent User Interfaces for Configuration Environments*, volume 1, pages 89 – 106. Morgan Kaufmann, 2014.
[4] Alexander Felfernig, Stefan Reiterer, Martin Stettinger, Florian Reinfrank, Michael Jeran, and Gerald Ninaus. Recommender systems for configuration knowledge engineering. *Workshop on Configuration*, pages 51 – 54, 2013.
[5] Alexander Felfernig, Christoph Zehentner, and Paul Blazek. Corediag: Eliminating redundancy in constraint sets. In Martin Sachenbacher, Oskar Dressler, and Michael Hofbaur, editors, *DX 2011. 22nd International Workshop on Principles of Diagnosis*, pages 219 – 224, Murnau, GER, 2010.
[6] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems: An Introduction*, volume 1. University Press, Cambridge, 2010.
[7] Ulrich Junker. Quickxplain: preferred explanations and relaxations for over-constrained problems. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, pages 167–172. AAAI Press, 2004.
[8] Michael Pazzani and Daniel Billsus. Content-based recommendation systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 325–341. Springer Berlin / Heidelberg, 2007.
[9] Cédric Piette. Let the solver deal with redundancy. In *Proceedings of the 2008 20th IEEE International Conference on Tools with Artificial Intelligence - Volume 01*, pages 67–73, Washington, DC, USA, 2008. IEEE Computer Society.
[10] Florian Reinfrank, Gerald Ninaus, Bernhard Peischl, and Franz Wotawa. A goal-question-metrics model for configuration knowledge bases. *Configuration Workshop*, 2015.
[11] Florian Reinfrank, Gerald Ninaus, Franz Wotawa, and Alexander Felfernig. Maintaining constraint-based configuration systems: Challenges ahead. *Configuration Workshop*, 2015.
[12] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.
[13] Stuart Russell and Peter Norvig. *Artificial Intelligence. A modern approach*, volume 2. Prentice Hall, New Jersey, 2003.
[14] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.
[15] Franz Wotawa, Florian Reinfrank, Gerald Ninaus, and Alexander Felfernig. icone: intelligent environment for the development and maintenance of configuration knowledge bases. *IJCAI 2015 Joint Workshop on Constraints and Preferences for Configuration and Recommendation*, 2015.
[16] Franz Wotawa, Martin Stettinger, Florian Reinfrank, Gerald Ninaus, and Alexander Felfernig. Conflict management for constraint-based recommendation. *IJCAI 2015 Joint Workshop on Constraints and Preferences for Configuration and Recommendation*, 2015.

Maintaining Constraint-based Configuration Systems: Challenges ahead

Florian Reinfrank, Gerald Ninaus, Franz Wotawa, Alexander Felfernig
Institute for Software Technology
Graz University of Technology
8020 Graz, Austria
firstname.lastname@ist.tugraz.at

Abstract. Constraint-based configuration systems like knowledge-based recommendation and configuration are used in many different product areas such as cars, bikes, mobile phones, and computers. The development and maintenance of such systems is a time-consuming and error prone task because the content of such systems and the responsible knowledge engineers are changing over time.

Much research has been done to support knowledge engineers in their maintenance task. In this paper we give a short overview of previous research in the context of intelligent techniques to support the maintenance task and give an overview of future research aspects in this area. This paper focuses on intelligent simulation techniques for generating metrics, predicting boundary values for automated test case generation, assignment-based (instead of constraint-based) anomaly management, and processes for the development of constraint-based configuration systems.

1 Introduction

The number of e-commerce web sites and the quantity of offered products and services is increasing enormously [2]. This triggered the demand of intelligent techniques that improve the accessibility of complex item assortments for users. Such techniques can be divided into configuration-based systems and recommendation systems. When the e-commerce system has highly configurable products (e.g. cars), configuration-based systems can help users to configure the product based on their needs. If, on the other hand, the e-commerce system contains many different products, intelligent recommendation techniques can help to find the product, which fits best to the user's needs [16]. We can differentiate between collaborative systems (e.g., www.amazon.com [16]), content-based systems (e.g., www.youtube.com [16]), critiquing-based systems (e.g., www.movielens.org [4]), and constraint-based recommendation systems (e.g., www.myproductadvisor.com [6]). The favored type of recommendation systems depends on the domain in which the recommendation system will be used. For example, in highly structured domains where almost all information about a product is available in a structured form, critiquing and constraint-based recommendation systems are often the most valuable recommendation approach.

Such systems are used, for example in the notebook domain. Such product domains - like the notebook domain - change over time because new product characteristics can fit to new customer needs. For example, ten years ago the number of cpu cores for notebooks was not a configurable variable. Nowadays, users can choose between one, two, or four cpu cores. This example shows that constraint-based recommendation systems have to be updated over time. While adding a variable to the product might be easy to handle, adding and editing constraints can be a time consuming and error prone task. This problem occurs in complex constraint-based recommendation systems with many existing constraints.

A lot of research has been done in the last years to tackle this challenge. For example, recommendation techniques can help to support knowledge engineers in their maintenance tasks, via reducing the sets of constraints so that the engineer can focus on the relevant constraints. Other examples for the support of the maintenance tasks are anomaly detection, dependency detection, and metrics measurement. An example application for the maintenance of constraint-based configuration systems is iCone (Intelligent Environment for the Development and Maintenance of configuration knowledge bases) [21, 26].[1]

Based on intelligent techniques to support knowledge engineers in their maintenance tasks, this paper focuses on further aspects in the maintenance of constraint-based configuration systems and picks up four research aspects (see below) for improving existing development and maintenance environments for constraint-based configuration systems like knowledge-based configuration and recommendation systems.

The paper is organized as follows: Section 2 (preliminaries) gives an overview of constraint-based configuration systems and a running example. Section 3 contains four aspects of the context of constraint-based configuration system development and maintenance. Section 3.1 is dealing with simulation techniques for constraint-based configuration systems. Section 3.2 shows principles of test case generation based on software engineering for constraint-based systems. An introduction for assignment-based anomaly management is given in Section 3.3. Section 3.4 goes beyond maintaining constraint-based configuration systems and takes a look into development pro-

[1] `http://ase-projects-studies.ist.tugraz.at:8080/iCone`

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

cesses for configuration systems. Section 4 summarizes this paper.

2 Preliminaries

In this Section we will describe constraint-based configuration systems, the terms assignment, consistency, and redundancy, and introduce a short knowledge-based recommendation system for notebooks as an example for a constraint-based configuration system.

We use the terminology of constraint satisfaction problems (CSP) [25] to represent configuration systems. Constraint-based configuration systems are defined as a triple $KB = \{V, D, F\}$. V is a set of product and customer variables. All variables have a selection strategy v_{sel} which describes, if a variable can have more than one value. $v_{sel} = singleselect$ shows that the variable v can have zero or one assignment, e.g. each product has one *price*, e.g. $price = 399$. If a variable can have more than one value, we differ between $multipleAND$ and $multipleOR$. $v_{sel} = multipleAND$ points out that a variable can have more than one assignment. For example, a notebook can have two wireless connections like *bluetooth* AND $WLAN$, s.t. $wireless\_connection_{sel} = multipleAND$. On the other hand, a customer wants to have a notebook with two OR four *cpu_cores*. We denote such a selection strategy as $multipleOR$, s.t. $cpu\_cores_{sel} = multipleOR$.

Each variable $v_i \in V$ has a domain $dom(v_i) \in D$ that contains the set of all possible values (not only the assigned values). Each variable can have *zero to n* finite assignments. Products F_P, customer requirements F_R, and constraints which are defining the relationship between product variables and customer variables F_C are in the filter set F.

Customer requirements represent the preferences of customers in the recommendation / configuration process. The set of customer preferences is denoted as F_R. For example, a customer can have the preference that a notebook should be cheaper than 599 EUR, s.t. $\{price < 599\} \in F_R$. Furthermore, customers can be asked for their usage scenarios, which might can be $multimedia, office, gaming$. If a user has more than one usage scenario, we duplicate the variable *usage_scenario* for this user, s.t. $usage\_scenario1 = multimedia \wedge usage\_scenario2 = office$.

Constraints which can also be denoted as filters in F_C define the relationship between customer preferences and product variables and are defined in the set $F_C \in F$. For example, the relationship between the customer's *usage_scenario* and the product attributes is $f_1 := usage\_scenario = gaming \rightarrow cpu\_cores > 2$. Additionally, constraint-based recommendation systems have a set of products. This set is denoted as $F_P \in F$ and contains one disjunctive query with all products, s.t. $F_P = \{product_0 \vee product_1 \vee ... \vee product_n\}$ and each product is a conjunctive query of the product variables, s.t. $product_0 = \{price = 399 \wedge cpu\_cores = 2\}$. The aggregation of customer requirements, constraints, and products represent the filters, s.t. $F_R \cup F_C \cup F_P = F$.

Each filter can be divided into **assignments**. An assignment consists of a variable v, a relationship, and a value d which is an element of the domain $dom(v)$. The different types of relationships depend on the values in the corresponding domain. If, for example, the domain consists only of numbers, we can say that the types of relationships are $<, \leq, =, \neq, \geq, >$ whereas for domains with strings we reduce the different types of relations to $=, \neq$.

To consider the selection strategy of variables within the filters, we have to duplicate the variables. For example, if a customer wants to have a notebook for *gaming* and $office$, we replace the variable $usage\_scenario \in V$ with $usage\_scenario1 \in V$ and $usage\_scenario2 \in V$ with the same domain in the knowledge base KB. We also have to extend the affected filters in F, such that we have to replace the affected assignments in the example constraint $usage\_scenario = gaming \rightarrow cpu\_cores > 2 \in F_C$ with the assignments $(usage\_scenario1 = Gaming \vee usage\_scenario2 = Gaming) \rightarrow cpu\_cores > 2 \in F_C$.

To check if at least one product fits to the customer's preferences, we do consistency checks, s.t. $V \cup D \cup F \neq \emptyset$. If at least one product in the constraint-based configuration system is presented to the customer, we can say, that the knowledge base is **consistent**. Otherwise, the knowledge base contains inconsistencies. For dealing with inconsistencies, we refer the reader to [3, 5, 10, 11, 12, 15].

If the knowledge base is consistent, we can further evaluate whether the knowledge base contains **redundancies**. A redundancy is given, if the removal of a constraint from F_C leads to the same semantics [15, 20].

In the following we describe a notebook domain. The simplified domain is represented as a knowledge-based recommendation system.

$$V = \{price, cpu\_cores, usage\_scenario\}$$
$$price_{sel} = singleselect$$
$$cpu\_coressel = singleselect$$
$$usage\_scenariossel = multipleAND$$

$$D = \{$$
$$dom(price) = \{399, 599, 799, 999\},$$
$$dom(cpu\_cores) = \{2, 4\},$$
$$dom(usage\_scenario) = \{office, multimedia, gaming\}$$
$$\}$$

$$F_C = \{$$
$$f_1 := usage\_scenario = office \rightarrow (price < 599 \wedge cpu\_cores = 2)$$
$$f_2 := usage\_scenario = multimedia \rightarrow ((price < 999 \wedge cpu\_cores = 4) \vee price < 799)$$
$$f_3 := usage\_scenario = gaming \rightarrow cpu\_cores = 4$$
$$\}$$

$$F_R = \emptyset$$

$$F_P = \{$$
$$(price = 399 \wedge cpu\_cores = 2) \vee \quad (p_0)$$
$$(price = 599 \wedge cpu\_cores = 4) \vee \quad (p_1)$$
$$(price = 799 \wedge cpu\_cores = 2) \vee \quad (p_2)$$
$$(price = 999 \wedge cpu\_cores = 4) \quad (p_3)$$
$$\}$$

$$F = F_C \cup F_R \cup F_P$$

$$KB = V \cup D \cup F$$

3 Challenges in the Development and Maintenance of Constraint-based Configuration Systems

In the following, we describe basic approaches to increase the maintainability, understandability, and functionality of constraint-based configuration systems. Therefore we use simulation techniques (Section 3.1), automated test case generation (Section 3.2), assignment-based anomaly management (Section 3.3), and the consideration of the development process (Section 3.4).

3.1 Simulation

In the context of constraint-based configuration systems we use simulation to approximate the number of consistent constraint sets compared to the number of all possible constraint sets. This technique can be used to calculate metrics - like the number of valid configurations - or to approximate the dependency between variables [21]. On the one hand we loose minimal accuracy when calculating the possible number of consistent constraint sets whereas, on the other hand, it is possible to approximate metrics which can not be calculated in an efficient manner. In the following, we describe the basic functionality of simulation for constraint-based configuration systems and give an example simulation in Figure 1.

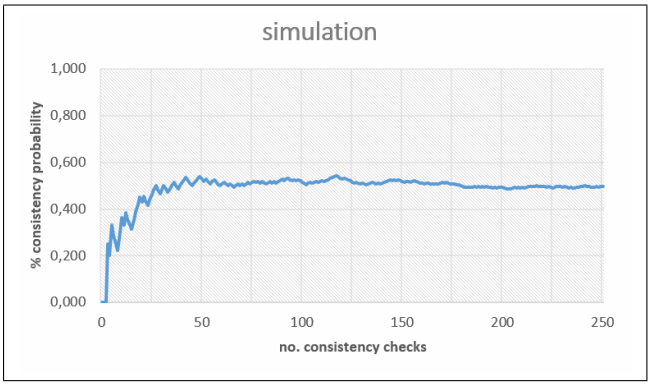


Figure 1. Example simulation for approximated consistency. We assume that a high number of consistency checks leads to a representative sample of the configuration knowledge base (Law of large numbers). In this example the average number of consistent configurations is approx. 50%.

Due to the huge complexity for calculating all possible instances for all possible assignments in constraint-based configuration systems, we use Gibbs' simulation to estimate the consistency rate *coverage* for a specific set of assignments A [22]. An assignment is a filter constraint which contains one variable a_v, one domain element a_d, and a relationship between variable and domain element a_r (see Section 2). Algorithm 1 is divided into three functions and shows the basic algorithm for estimating the consistency rate for a set of assignments.

The function $Gibbs(KB, A)$ is the main function of this algorithm. It has a knowledge base KB and a set of assignments A as input. The knowledge base contains sets of variables $V \in KB$ and filters $C \in KB$ (see Section 2). The set CC (checks) contains all results from consistency checks.

Algorithm 1 GibbsSampling

```
function GIBBS(KB, A): Δ
    CC = ∅                          ▷ set of consistency check results {0,1}
    mincalls = 200                                              ▷ constant
    threshold = 0.01                                            ▷ constant
    consistent = 0
    verify = Double.Max_Value
    while n < mincalls ∨ verify > threshold do
        randA = A ∪ GENERATERANDASSIGN(KB)
        F.addAll(randA)                                         ▷ F ∈ KB
        if isConsistent(KB) then
            consistent++
            CC.add(1)
        else
            CC.add(0)
        end if
        F.removeAll(randA)
        verify = VERIFYCHECKS(CC)
        n++
    end while
    return consistent/n
end function
function GENERATERANDASSIGN(KB):A
    A = ∅                                   ▷ A: set of assignments
    n = Random(F ∈ KB):                     ▷ generate n assignments
    for i = 0; i < n; i++ do
        a_v = Random(V ∈ KB)                                    ▷ V ∈ KB
        a_r = Random(Rel)
        a_d = Random(dom(a_v) ∈ D ∈ KB)
        A.add(a)
    end for
    return A
end function
function VERIFYCHECKS(CC):Δ
    CC_1 = CC.split(0, |CC|/2)
    CC_2 = CC.split((|CC|/2) + 1, |CC|)
    mean1 = mean(CC_1)
    mean2 = mean(CC_2)
    if mean1 ≥ mean2 then
        return mean1 − mean2
    else
        return mean2 − mean1
    end if
end function
```

A consistency check is either consistent (1) or inconsistent (0). The number of minimum calls is constant and given in variable *mincalls*. The total number of consistent checks is given in the programming variable *consistent*. *threshold* is a constant and required to test if the current set of consistency checks has a high accuracy. The variable $verify$ contains the result of the last verification returned by the function $VERIFYCHECKS$. If the variable $verify$ is greater than the *threshold*, we can not guarantee that the current result is accurate. In that case we have to execute the loop again. In the while-loop we first have to generate a new set of random assignments. Since assignments are also special types of constraints, we add the set $randA$ to the set $F_C \in KB$ and do a consistency check. If KB with the randomly generated assignments is consistent, we add 1 to the set CC and incre-

ment the variable *consistent*. Otherwise, we add 0 to the set CC. Finally, we verify all previous consistency checks. If the variable $verify$ is lower than the variable *threshold* and we have more consistency checks than *mincalls*, we can return the number of consistent checks divided by the total number of checks.

The function $generateRandAssign(KB)$ is responsible for the generation of new assignments. $Random(F)$ returns the number of assignments which have to be generated randomly. This number depends on the number of filters in the knowledge base, the number of available variables and domain elements, since in small knowledge bases it can happen that we can not generate more than *mincalls* assignments. $Random(V)$ takes a variable from the knowledge base. If the variable is already part of another assignment, the variable won't be used again except the selection strategy v_{sel} is either $multipleAND$ or $multipleOR$. $Random(R)$ selects a relation between the variable and the domain elements. In our case, variables can have textual domain elements (e.g. the brand of a notebook) or numeric domain elements (e.g. the price of a notebook). While the set of relations for textual domain elements is $Rel = \{=, \neq\}$, the set is extended to $Rel = \{=, \neq, <, \leq, >, \geq\}$ for numerical domain elements (see Section 2). Finally, $Random(dom(a_v))$ selects a domain element from $dom(a_v)$ randomly.

The function $verifyChecks(CC)$ tests if the numbers of consistent and inconsistent checks are normally distributed. Therefore, we first divide the set with the consistency check results CC into two parts. We evaluate the mean of both sets CC_1 and CC_2 and test, if both mean values $mean(CC_1)$ and $mean(CC_2)$ are close to each other. If they have nearly the same values, $(\sqrt{(mean(CC_1) - mean(CC_2))^2} \leq threshold)$, we can say that the consistent checks are normally distributed and return the difference between $mean(CC_1)$ and $mean(CC_2)$.

In our iCone implementation we use the simulation technique in three different ways. First, we evaluate the *coverage* metric which defines the number of consistent configurations compared to the number of all possible configurations [22]. Second, we use this technique to generate random assignments for test cases (see Section 3.2). Finally, we use this technique to approximate the consistency rate *coverage* for at least two variables and their domain elements. Figure 2 shows the probability that the combination of two assignments is consistent. For example, approximately 100% of the notebook configurations are consistent, if the $usage\_scenario = multimedia \wedge cpu\_cores = 2$.

3.2 Test Case generation

In this Section we want to describe a basic approach to generate test cases for constraint-based configuration systems.

In software engineering, boundary value analysis are *those situations directly on, above, and beneath the edges of input equivalence classes* [19]. To use this type of software testing in the context of configuration systems, we can say that the *edges* are within variable assignments. For example, if $price = 399$ is consistent, $price = 599$ is consistent too, and $price = 799$ is inconsistent, the boundary would be between the domain elements 599 and 799. In Figure 2 we can see that, under circumstances, some combinations are inconsistent (e.g.

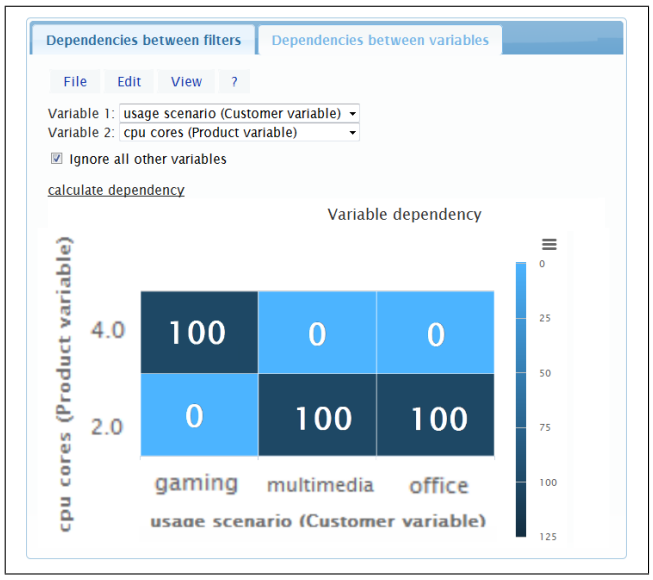


Figure 2. Example simulation for approximated consistency

$usage\_scenario = multimedia \wedge cpu\_cores = 2$) and some are consistent (e.g. $usage\_scenario = gaming \wedge cpu\_cores = 2 = inconsistent$). We can use the simulation technology (see Section 3.1) to generate various sets of filter constraints to get some boundaries. Table 1 shows a list of randomly generated test cases. Note that the number of assignments in the test case can be different (see Algorithm 1).

tc	$filterconstraint$	$coverage$
t_0	$cpu\_cores = 2 \wedge$ $usage\_scenario = office$	0.50
t_1	$cpu\_cores = 2 \wedge$ $usage\_scenario = multimedia$	0.50
t_2	$price = 799 \wedge$ $usage\_scenario = gaming$	0.00
t_3	$price = 599 \wedge$ $usage\_scenario = gaming$	0.50
t_4	$cpu\_cores = 4 \wedge$ $usage\_scenario = multimedia$	0.50
t_5	$cpu\_cores = 4$	~ 0.54

Table 1. An example for randomly generated test cases.

The next step is to evaluate these randomly generated boundary test cases according to the domain experts' knowledge. Our example test cases show, that between the test cases t_2 and t_3 is a boundary because the coverage is different.

After the randomly detected boundaries via simulation we have to evaluate the boundary. Such evaluations have to be done by stakeholders of the knowledge base and can be done via micro tasks [7]. In this context, several stakeholders can be asked if the results of randomly generated test cases are valid or not. Such answers can be collected within a case base. Table 2 gives an example case base.

87.5% of the stakeholders agree that t_2 is correct, which means that the test case should be inconsistent and the test case currently leads to an inconsistency. On the other hand, 62.5% of the stakeholders think that t_3 should not be consistent. This example represents a conflict between the knowledge engineers' opinions of the knowledge base. For such scenarios we have to offer relevant information to the stakehold-

stakeholder	*testcase*	*correct?*
s_0	t_2	*yes*
s_1	t_2	*yes*
s_2	t_2	*yes*
s_3	t_2	*yes*
s_4	t_2	*yes*
s_5	t_2	*no*
s_6	t_2	*yes*
s_7	t_2	*yes*
s_0	t_3	*no*
s_1	t_3	*no*
s_2	t_3	*yes*
s_3	t_3	*yes*
s_4	t_3	*no*
s_5	t_3	*no*
s_6	t_3	*no*
s_7	t_3	*yes*

Table 2. An example case base for evaluating randomly generated test cases.

ers such as mails, forum, and content-based recommendation [16].

Finally, a result of the discussion leads to a consistent knowledge base (filter constraints $f \in F_C$ have to be updated or removed) which represents the real product domain. If the knowledge base has to be maintained, intelligent techniques like the detection of minimal conflicts and diagnoses [21] help to detect the causes for the difference between the knowledge base and the real world.

3.3 Assignment-based anomaly management

The anomaly management research describes different approaches to detect and explain anomalies [21, 26]. For example, QuickXplain can detect conflicts [17], FastDiag finds minimal diagnoses for these conflicts [13], Sequential [20] and CoreDiag [15] can remove maximal sets of filter constraints without changing the semantics of the knowledge base (redundancy detection). Well-formedness violations can detect domain elements which can never be selected (*deadelements*) or have to be selected ($fullmandatories$) or can only exit if specific domain elements of other variables are selected as well ($unnecessaryrefinements$) [21].

While all of these algorithms focus on filters, little attention has been paid to the context of assignment-based anomaly detection. Compared to constraint-based perspectives, an assignment-based view can a) find out which assignments within a filter lead to the anomaly and b) detect more redundancies when one assignment within a filter constraint with more than one assignment can not be detected with common algorithms.

Alternatively, we can check the assignments within a filter instead of the filter itself for anomalies. Algorithm 2 gives an example for an assignment-based algorithm. This algorithm extends the *Sequential* algorithm introduced by Piette [20]. First of all, we have to generate the negation of all filter constraints in the knowledge base. We denote the negation $\bar{F_C}$ and define a disjunctive query of the original knowledge base $\neg f_1 \vee \ /f_2 \vee \ /f_3$. If the negation of the knowledge base in combination with the original knowledge base is inconsistent, s.t. $F_C \cup \bar{F_C}$ is inconsistent, the knowledge base has not changed its semantics. If a filter will be removed from the knowledge base (but not from the negation of the knowledge base) and the combination is still inconsistent, we can say that the knowledge base has kept its semantics and the removed filter constraint is redundant.

Algorithm 2 AssignmentSequential

```
function AssignmentSequential(KB): R
                                   ▷ KB: knowledge base
  F̄_C = ¬f_1 ∨ ¬f_2 ∨ ¬f_3
  R = ∅
  for all f ∈ F_C do
    for all a ∈ A(f) do
      A.remove(a)
      if (F_C ∪ F̄_C) isinconsistent then
        R.add(a)
      else
        A.add(a)
      end if
    end for
  end for
  return R
end function
```

While the Sequential algorithm removes filter constraint by filter constraint from F_C, we divide the filter constraint into its assignments and remove assignment by assignment. Therefore, we introduce the set $A(f)$ which describes the set of assignments of filter constraint $f \in F_C$. When we remove an assignment from $A(f)$, we next have to consider the relations between the assignments. Figure 3 shows the graphical representation of all filter constraints and their assignments in our example knowledge base in a conjunctive order. When we remove an assignment a from $A(f)$ we will further replace the upper relation. For example, the removal of the assignment $usage\_scenario = office$ of filter f_1 replaces the upper implication $\rightarrow$ with the top node of those elements which will not be connected to the conjunctive constraint. In our case, this is relation '$\wedge$'.

Algorithm 2 introduces an approach to detect redundant assignments within a knowledge base. The approach is straight forward: First, we have to generate the negation of $\bar{F_C}$. Then we select filter by filter. For each filter we remove assignment by assignment a. Finally, we check if the knowledge base with the changed filter f is still inconsistent with $\bar{F_C}$. If it is inconsistent, we can say that the removed assignment a is redundant.

Figure 4 shows the redundant assignments of our example knowledge base. In the first row we see the original filters and the result for the $usage\_scenario$ variable (green box). Then we remove assignment by assignment and see the result of the filter constraints in the column *result*. The yellow boxes suggest that the adapted filter constraints lead to the same semantics as the original knowledge base. We can remove $cpu\_cores = 2$ from filter constraint f_1 and the assignments $price < 999$ and $cpu\_cores = 4$ from filter constraint f_2 without changing the semantics of the knowledge base.

Similar adaptations can also be done e.g., for QuickXPlain [17], FastDiag [14], and CoreDiag [15]. While these algorithms use a divide and conquer approach based on filters, future research can also consider assignments instead of filters to calculate the anomalies.

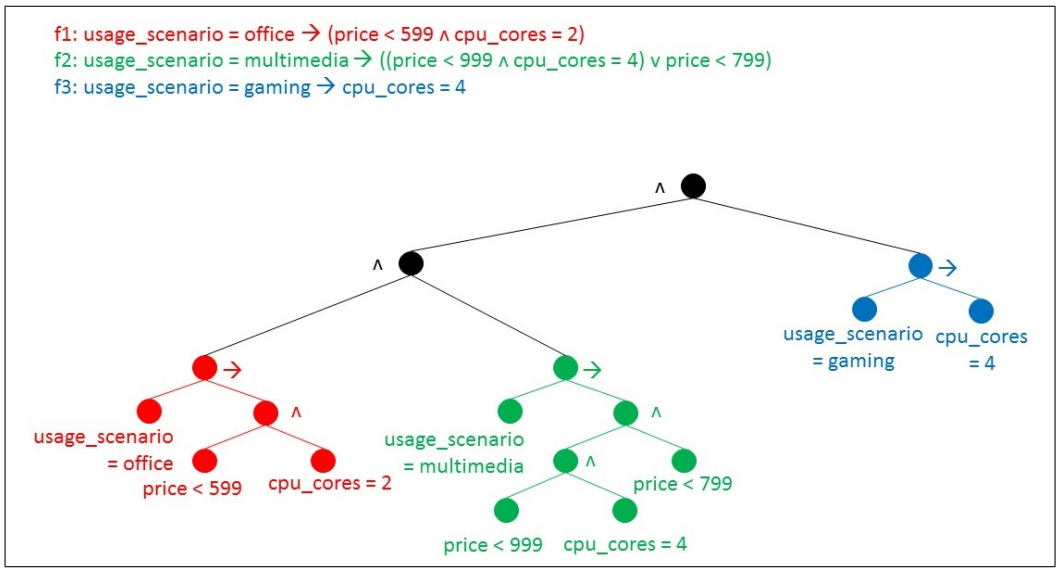


Figure 3. Conjunctive query of all filters $f \in F_C$ in our example knowledge base. In Algorithm 2 we remove assignment by assignment from the knowledge base and check the consistency instead of the whole filter (f_1, f_2, f_3).

f1	f2	f3	result		redundant
usage_scenario = office → (price < 599 ∧ cpu_cores = 2)	usage_scenario = multimedia → ((price < 999 ∧ cpu_cores = 4) v price < 799)	usage_scenario = gaming → cpu_cores = 4	usage_scenario = office usage_scenario = multimedia usage_scenario = gaming	p0 p0, p1 p1, p3	
price < 599 ∧ cpu_cores = 2	usage_scenario = multimedia → ((price < 999 ∧ cpu_cores = 4) v price < 799)	usage_scenario = gaming → cpu_cores = 4	usage_scenario = office usage_scenario = multimedia usage_scenario = gaming	p0 ∅ ∅	
usage_scenario = office → cpu_cores = 2	usage_scenario = multimedia → ((price < 999 ∧ cpu_cores = 4) v price < 799)	usage_scenario = gaming → cpu_cores = 4	usage_scenario = office usage_scenario = multimedia usage_scenario = gaming	p0,p2 p0, p1 p1, p3	
usage_scenario = office → price < 599	usage_scenario = multimedia → ((price < 999 ∧ cpu_cores = 4) v price < 799)	usage_scenario = gaming → cpu_cores = 4	usage_scenario = office usage_scenario = multimedia usage_scenario = gaming	p0 p0, p1 p1, p3	cpu_cores = 2
usage_scenario = office → price < 599	((price < 999 ∧ cpu_cores = 4) v price < 799)	usage_scenario = gaming → cpu_cores = 4	usage_scenario = office usage_scenario = multimedia usage_scenario = gaming	p0 p0, p1 p1	
usage_scenario = office → price < 599	usage_scenario = multimedia → (cup_cores = 4 v price < 799)	usage_scenario = gaming → cpu_cores = 4	usage_scenario = office usage_scenario = multimedia usage_scenario = gaming	p0 p0, p1 p1, p3	price < 999
usage_scenario = office → price < 599	usage_scenario = multimedia → price < 799	usage_scenario = gaming → cpu_cores = 4	usage_scenario = office usage_scenario = multimedia usage_scenario = gaming	p0 p0, p1 p1, p3	cpu_cores = 4
usage_scenario = office → price < 599	usage_scenario = multimedia	usage_scenario = gaming → cpu_cores = 4	usage_scenario = office usage_scenario = multimedia usage_scenario = gaming	∅ p0, p1, p2, p3 ∅	
usage_scenario = office → price < 599	usage_scenario = multimedia → price < 799	cpu_cores = 4	usage_scenario = office usage_scenario = multimedia usage_scenario = gaming	∅ p1 p1, p3	
usage_scenario = office → price < 599	usage_scenario = multimedia → price < 799	usage_scenario = gaming	usage_scenario = office usage_scenario = multimedia usage_scenario = gaming	∅ ∅ p0, p1, p2, p3	

Figure 4. Results for consistency checks. The columns f_1, f_2, and f_3 show the filter constraints when one assignment will be removed. The first row shows the results (green background) of the original filter constraints. The yellow background suggests, that the removal of the assignments leads to the same results.

3.4 Constraint-based configuration system development

A lot of research has been done in the maintenance of constraint-based systems. For example, we can evaluate the quality of knowledge bases [22] and check if the knowledge base has anomalies [5, 21]. Therefore, we can evaluate if we are doing the knowledge base maintenance efficiently.

Less work has been done in the context of knowledge base development, a task which is crucial for an effective constraint-based configuration system. Next, we want to summarize previous work in the context of knowledge base development processes and try to give hints for transferring research results from the software engineering discipline into the knowledge base development research area.

Development processes for constraint-based configuration systems

An overview of knowledge base engineering processes is given in [9, 24].

Common-KADS focuses on different models (organization, task, agent, communication, and expertise) of the knowledge base. For example, the expertise model tries to describe knowledge from a static, functional, and a dynamic view. While this system tries to consider all stakeholders, it does not prioritize the knowledge and does not try to solve conflicts in the knowledge before it will be transferred into a constraint-based configuration system [23].

The **MIKE** engineering process can be seen as an iterative process and is divided into the activities *elicitation*, *interpretation*, *formalization / operationalization*, *design*, and *implementation*. The entire development process, i.e. the sequence of knowledge acquisition, design, and implementation, is performed in a cycle inspired by a spiral model as process model. Every cycle produces a prototype as output which can be evaluated by tests in the real target environment. The evaluation of each activity will be done by domain experts. While the result of the implementation activity can be evaluated by domain experts, a deep understanding of modelling techniques is required to evaluate the results of elicitation, interpretation, and formalization activities [1].

Protege-II is used to model method and domain ontologies. A method ontology defines the concepts and relationships that are used by a problem solving method for providing its functionality. Domain ontologies define a shared conceptualization of a domain. Both ontologies can be reused in other domains which may reduce the effort to build-up a new knowledge base with similar elements [18].

Development in the Software Engineering Discipline

Compared to development processes for constraint-based configuration systems we give an overview of actual trends in the engineering of such systems and create a link to the currently existing development processes for constraint-based configuration systems.

A relevant task in software engineering is **requirements engineering**. Transferring this aspect into the context of developing constraint-based configuration systems we can say that products, product variables, questions to customers, variable domains, and filters can be functional requirements whereas interface development (e.g. to an ERP-system), performance, and collaborative development are non-functional requirements. When knowledge base engineering processes have to be finalized with a given budget and time, we also have to prioritize such requirements. Therefore, we have to rank the requirements based on their necessity and effort (time and budget) for a functional knowledge base. The prioritization should be done by different stakeholders to include as many knowledge as possible into the prioritization process.

While many different constraint-based configuration systems will be developed, each of them is developed from scratch. Similar to requirements engineering, most of the aspects of a new knowledge base are new and reuse is not possible. On the other hand, several requirements are domain independent. For such requirements, the implementation in a software could be done with **design patterns**. Such patterns can help to reduce the time effort for the realization of a requirement in a knowledge engineering process. For example, a notebook recommendation system contains products, questions to customers, and relationships between products and customers (filter constraints). In this domain, products have different prices and customers will be asked for their maximum price. While the product variable *product_price* may have hundreds of different prices (domain elements), the customer will not choose e.g. between $product\_price = 799.90$ or $product\_price = 799.99$ but wants to have for example ten different prices (e.g. $customer\_price \leq 400$ or $product\_price \leq 600$ or ... or $product\_price \leq 2200$). The relationship between those variables can be denoted as *mapping* which could be a design pattern.

4 Conclusion

In this paper we gave an overview of future research in the context of developing and maintaining constraint-based configuration systems. Such systems can be constraint-based configuration, knowledge-based recommendation systems, or feature models. We introduced a simulation technique in the context of constraint-based configuration systems, show some hints for automatic test case generation and gave an overview of assignment-based anomaly detection instead of constraint-based conflicts, redundancies, and well-formedness detection. Finally, we showed how requirements engineering and design patterns can be used for knowledge base engineering processes.

Acknowledgements

The work presented in this paper has been conducted within the scope of the research project ICONE (Intelligent Assistance for Configuration Knowledge Base Development and Maintenance) funded by the Austrian Research Promotion Agency (827587).

REFERENCES

[1] J. Angele, D. Fensel, D. Landes, and R. Studer. Developing knowledge-based systems with mike. *Automated Software Engineering*, 5(4):389–418, 1998.

[2] Ivan Arribas, Francisco Perez, and Emili Tortosa-Ausina. Measuring international economic integration: Theory and evidence of globalization. *World Development*, 37(1):127 – 145, 2009.

[3] David Benavides, Alexander Felfernig, Jos A. Galindo, and Florian Reinfrank. Automated analysis in feature modelling and product configuration. *ICSR*, pages 160 – 175, 2013.

[4] Li Chen and Pearl Pu. Evaluating critiquing-based recommender agents. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, AAAI'06, pages 157–162. AAAI Press, 2006.

[5] Alexander Felfernig, David Benavides, Jos A. Galindo, and Florian Reinfrank. Towards anomaly explanation in feature models. *Workshop on Configuration*, pages 117 – 124, 2013.

[6] Alexander Felfernig and Robin Burke. Constraint-based recommender systems: technologies and research issues. In *Proceedings of the 10th international conference on Electronic commerce*, ICEC '08, pages 3:1–3:10, New York, NY, USA, 2008. ACM.

[7] Alexander Felfernig, Sarah Haas, Gerald Ninaus, Michael Schwarz, Thomas Ulz, and Martin Stettinger. Recturk: Constraint-based recommendation based on human computation. *CrowdRec*, June 2014.

[8] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, editors. *Knowledge-based configuration. From research to business cases*, volume 1. Morgan Kaufmann, 2014.

[9] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, editors. *Knowledge Engineering for configuration systems*, pages 139 – 155. Volume 1 of Felfernig et al. [8], 2014.

[10] Alexander Felfernig, Florian Reinfrank, and Gerald Ninaus. Resolving anomalies in configuration knowledge bases. *IS-MIS*, 1(1):1 – 10, 2012.

[11] Alexander Felfernig, Florian Reinfrank, Gerald Ninaus, and Paul Blazek. *Conflict Detection and Diagnosis Techniques for Anomaly Management*, pages 73 – 87. Volume 1 of Felfernig et al. [8], 2014.

[12] Alexander Felfernig, Florian Reinfrank, Gerald Ninaus, and Paul Blazek. *Redundancy Detection in Configuration Knowledge*, pages 157 – 166. Volume 1 of Felfernig et al. [8], 2014.

[13] Alexander Felfernig and Monika Schubert. Personalized diagnoses for inconsistent user requirements. *AI EDAM*, 25(2):175–183, 2011.

[14] Alexander Felfernig, Monika Schubert, and Christoph Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *AI EDAM*, 26(1):53–62, 2012.

[15] Alexander Felfernig, Christoph Zehentner, and Paul Blazek. Corediag: Eliminating redundancy in constraint sets. In Martin Sachenbacher, Oskar Dressler, and Michael Hofbaur, editors, *DX 2011. 22nd International Workshop on Principles of Diagnosis*, pages 219 – 224, Murnau, GER, 2010.

[16] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems: An Introduction*, volume 1. University Press, Cambridge, 2010.

[17] Ulrich Junker. Quickxplain: preferred explanations and relaxations for over-constrained problems. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, pages 167–172. AAAI Press, 2004.

[18] Mark A Musen, Henrik Eriksson, John H Gennari, and Samson W and Tu. Protg-ii: A suite of tools for development of intelligent systems from reusable components. *Proc Annu Symp Comput Appl Med Care*, 1994.

[19] Glenford J. Myers, Tom Badgett, and Corey Sandler. *The art of software testing*. John Wiley & Sons, 3 edition, 2012.

[20] Cédric Piette. Let the solver deal with redundancy. In *Proceedings of the 2008 20th IEEE International Conference on Tools with Artificial Intelligence - Volume 01*, pages 67–73, Washington, DC, USA, 2008. IEEE Computer Society.

[21] Florian Reinfrank, Gerald Ninaus, and Alexander Felfernig. Intelligent techniques for the maintenance of constraint-based systems. *Configuration Workshop*, 2015.

[22] Florian Reinfrank, Gerald Ninaus, Bernhard Peischl, and Franz Wotawa. A goal-question-metrics model for configuration knowledge bases. *Configuration Workshop*, 2015.

[23] G. Schreiber, B. Wielinga, R. de Hoog, H. Akkermans, and W. Van de Velde. Commonkads: a comprehensive methodology for kbs development. *IEEE Expert*, 9(6):28–37, Dec 1994.

[24] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data & Knowlege Engineering*, 25:161 – 197, 1998.

[25] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[26] Franz Wotawa, Florian Reinfrank, Gerald Ninaus, and Alexander Felfernig. icone: intelligent environment for the development and maintenance of configuration knowledge bases. *IJCAI 2015 Joint Workshop on Constraints and Preferences for Configuration and Recommendation*, 2015.

Coupling Two Constraint-Based Systems Into an On-line Façade-layout Configurator

Andrés F. Barco[1] and **Élise Vareilles**[1] and **Paul Gaborit**[1] and **Jean-Guillaume Fages**[2] and **Michel Aldanondo**[1]

Abstract.

We present the coupling of two constraint-based environments into an on-line support system for façade-layout configuration in the context of building renovation. The configuration consist on the definition and allocation of a set of rectangular parameterizable panels over a façade surface. The coupling allows to solve two configuration tasks while gaining efficiency and modularity. First, it allows to configure a set of questions relating the renovation model needed to determine limits for panels' size and panels' weight. Second, it allows to configure a constraint satisfaction model for each of the façades to renovate. Two constraint-based systems handle the filtering of incompatible values and the generation of layout plans in a web-service setup. The first service performs initial filtering to set panels' limits, based on the questionnaire, using a constraint filtering engine called `CoFiADe`. The second service uses several façade-layout configuration algorithms, using as underlying engine the constraint solver `Choco`, to generate compliant layout-plan solutions. We show that by dividing filtering and search, and by coupling the two constraint-based systems, we gain modularity and efficiently as each service focuses on their own strengths. Services executing tasks may be hosted in different network-nodes and thus may be seen as independent communicating agents.

1 Preliminaries

Constraint-base façade-layout configuration. Product configuration refers to the task of building a target product using predefined components, respecting requirements from customers and following some rules that shape a correct configuration [26, 29]. This task have been increasingly supported by intelligent systems given the complexity and size of relations within a single product. For instance, configuring a computer from memories, buses, cards and so on, involves a large number of possibilities and solutions for the user. The possible numbers of outputs for a configuration is in relation to the number of components and relations within the product, and is inversely proportional to the number of rules that restrict combinations. We call these kind of problems *combinatorial problems*.

A particular scenario on product configuration arises from the context of building thermal renovation as an effort to reduce current energetic consumption levels [5, 6, 19]. Here, the problem lies on the configuration of rectangular parameterizable panels, and their attaching devices called fasteners, that must be allocated over the façade surface in order to provide an insulation envelope [8, 14, 28]. The problem is also known as layout synthesis or space planning. A configuration solution is a plan which satisfies optimization criteria and a set of constraints (such as geometrical, weight or resources constraints) provided by users and the façade itself. As part of the product configuration family problems, an instance of façade-layout configuration problem has a huge search space that depends on the size of the panels and the elements on the façade, such as windows, doors and supporting areas (see Figure 1). In consequence, to solve this configuration problem, we choose to rely on a technique from artificial intelligence and operation research called constraint satisfaction problems [15, 18].

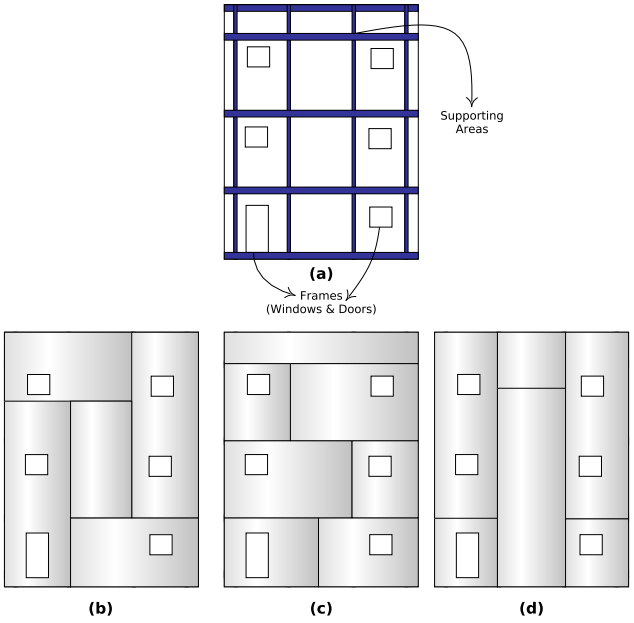


Figure 1: A façade is a rectangular surface with supporting areas, windows and doors. Layout-plans solutions made out configured rectangular panels.

Constraint satisfaction problems (CSPs) are conceived to allow the end-user to state the logic of the computation rather than its flow. For example, in the context of scheduling, instead stating a set of steps to avoid tasks overlapping, the user declares *"for any pair of tasks they must not overlap"*. The user may do so by stating a) variables representing elements of the problem, b) a set of potential values associated to each variable and c) relations over the stated variables also known as constraints [12, 18]. Variables may have different do-

[1] Université de Toulouse, Mines d'Albi, Route de Teillet Campus Jarlard, 81013 Albi Cedex 09, France, email: abarcosa@mines-albi.fr
[2] COSLING S.A.S., 2 Rue Alfred Kastler, 44307 Nantes Cedex 03, France

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

main representation such as integer or boolean, and relation among variables may be expressed in different ways such as compatibility tables and first order formulas. Solving a CSP means finding an assignment of values for each variable in such a way that all constraints are satisfied.

It turns out that constraint technology fits neatly in the constrained nature of product configuration and layout synthesis. On one hand, the knowledge (constraints) that restrict possible configuration of elements (variables) is easily modeled under the declarative framework of constraint satisfaction problems. On the other hand, constraint-based configurators are able to presents different solutions to users, often optimal, even when they do not provide all configuration parameters leaving their preferences unknown.

Now, for constraint-based implementations we differentiate between two related concepts: Solving a problem and filtering (narrow) possibilities. Whereas solving a problem involves a robust inference engine and search strategies, filtering algorithms work, in essence, on how to efficiently remove variable values that are restricted by the established relations, i.e., the constraints [2, 4]. In the problem at hand both the filtering and the search take part; filtering to set panel's size and weight limits and, search in order to generate compliant layout plans.

Related work. Layout configuration techniques have been used within different contexts and scenarios. For instance, finding solutions for room configurations [30], apartment layouts [16] and activities within a business office [13]. Also, some tools have been implemented using different approaches, here we name a few of them. For example, in [25] Shikder et al. present a prototype for the interactive layout configuration of apartment buildings including design information and an iterative design process. In [3] is introduced *Wright*, a constraint-based layout generation system that exploits disjunctions of constraints to manage the possibilities on positioning two-dimensional objects in a two-dimensional space. Another system, *Loos* [9], is able to configure spaces using rectangles that can not be overlapped but that may have holes. It uses test rules applied by steps to the rectangles in order to reach a good configuration based on its orientation and relation with other rectangles. The same authors have developed *Seed* [10, 11]: A system based on *Loos* used for early stages on architectural design. The system *Hegel* [1] (for Heuristic Generation of Layouts) is yet another space planning tool that simulates human design based on experimental cases. Finally, Medjdoub et al. presents in [17] the system *Archiplan* which integrates geometrical and topological constraints to apartment layout planning.

Regardless the considerable number of applications on layout configuration, our problem include three characteristics never considered simultaneously: Its deals with the allocation of an *unfixed number of rectangular* panels that must not overlap, frames (existing windows and doors) must be *overlapped* by one and only one panel, and façades have specific areas providing certain *load-bearing capabilities* that allow to attach panels. Thus, as far as we know, no support system nor design system is well-suited for addressing such particularities. Also, most systems are desktop-oriented and not web-oriented, making difficult to adapt new requirements and functionalities as they need new versions to be released.

Contribution and structure. Traditional general purpose constraint solvers, such as `Gecode` [24], `Choco` [20] and the finite domain module of `Oz` [23], make clear distinction between filtering and search [21]. These environments provide methods for invoking constraint propagation, i.e., executing the filtering algorithm of constraints, and methods for invoking search for one solution, several solutions or optimal ones [22]. Nonetheless, to the best of our knowledge, studies focusing on constraint-based configuration involving filtering and search does not make clear distinction between these concepts and their implementation focuses on the search. This means that solutions exploits the capabilities of constraint solvers as black-box environments, typically creating a dedicated search heuristic for the problem.

Our goal is two-fold. First, we propose an architecture that divides initial filtering and consequent search for constraint-based product configuration. The architecture allow us to solve two configuration tasks; configure a set of questions relating the renovation model needed in the renovation process and needed to determine limits for panels' size and panels' weight and; configure a constraint satisfaction problem for each of the façades to renovate. In a second time, we present an on-line support system, and formalize its behavior, for the problem of façade-layout configuration. The architecture, implemented in the on-line system, couples two different constraint-based technological tools to gain efficiency and modularity. We use façade-layout configuration to illustrate our method as it is the goal of the project we are into, but our results can be adapted to deal with different kind of products.

The paper is divided as follows. A brief description of the industrial scenario and elements is presented in Section 2. In section 3, we introduce details of the two configuration tasks performed by the support system. The service oriented architecture, along with details of the constraint-services' behavior, is presented in Section 4. In Section 5, we discuss the benefits of the tasks division and coupling of the constraint systems. Some conclusions are drawn in Section 6. A bibliography is provided at the end of the document.

2 Renovation Modus Operandi

The problem at hand deals with the configuration of rectangular panels and their arrangement over a façade surface. In order to start the configuration process, specific information have to be extracted from a set of spatial entities. The renovation is carried out on *façades* that are part of a given building, *buildings* that are part of a given block, and *blocks* that are part of a given *working site*. Each of these spatial entities have geometrical and structural properties and may have different environmental conditions that must be taken into account for the layout-plan definition.

Information about spatial entities is acquired by the support system by means of an (JSON) input file describing all geometrical and structural properties, and by means of a web-based questionnaire for each spatial entity in the input file. Thus, if the input file contains information for one working site with two blocks, each block with one building and three façades in each building, the user answers to eleven questionnaires (one for the working site, two for the blocks, two for the buildings and six for the façades). After questionnaire completion, the lower bound and upper bound for panels' size and panel's weight have been deduced. Also, given that several instances for façades are need to be solve, at the end of the questioning stage the systems creates a constraint satisfaction model for each façade using the inputed information and the deduced limits for panels' size and weight. Here, each constraint satisfaction model instance is parameterized according with the façade information (e.g. environmental conditions) and the particular deduced panels' limits.

Lets consider the information flow from the user perspective. Figure 2 presents the representative actions made by the user and the

responses by the on-line support system. It also illustrates the fact that to the end-user should be transparent all the configuration process. The complete sequence of the configuration goes as follows.

Step 1:- The user uploads a file containing the geometry and structural specification of spatial entities. The support system stores information in a data base.

Step 2:- The filtering service presents a questionnaire for each of the spatial entities in the input file.

Step 3:- The user answers the questions (leaving in blank the questions he does not know the answer).

Step 4:- Using the information about spatial entities (database) and their environmental/user conditions (user answers) the system deduce lower and upper bounds for panels' size and panels' weight by using the filtering service.

Step 5:- If a manual configuration is desired, the user draws each panel on the clients GUI. Each panel is assured to be consistent with the problem requirements by sending its information to validate into the solving service (the validator module).

Step 6:- If a semi-automatic configuration is desired, the user draws some panels and then asks the solving service to finish the configuration.

Step 7:- If an automatic configuration is desired, the user asks the solving service to provide compliant layout solutions.

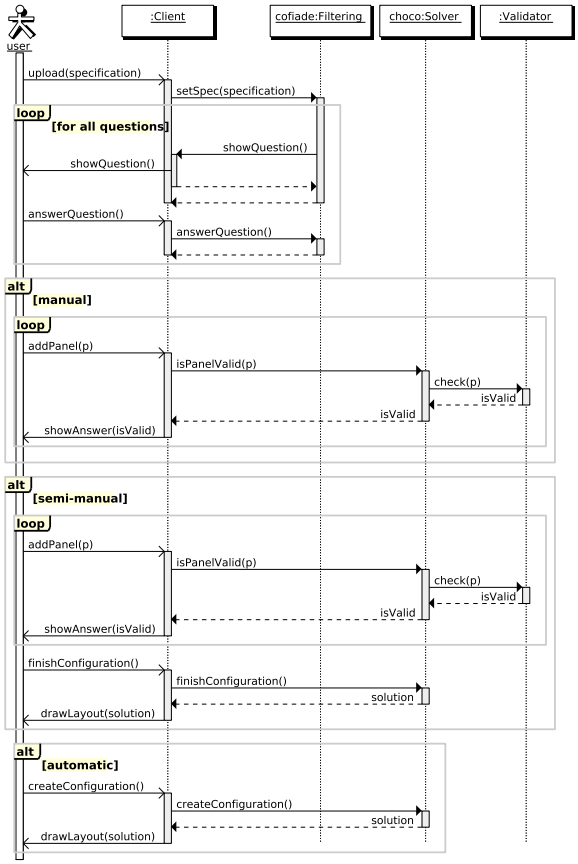


Figure 2: Sequence diagram for on-line support system.

It is worth mentioning the consistency among the different steps: Information at each level is propagated downwards and is never propagated upwards (we will further study this along the document). Also, of major importance is the fact that each façade may have its own panels' lower and upper bounds and thus solving a given façade is done with a particular set of arguments.

3 Support System Configuration Tasks

In this section we present the two configuration tasks within the support system: The configuration of a questionnaire to be filled by the end-user and, the configuration of a constraint satisfaction model for each façade to renovate used as input for layout-plans generation.

3.1 The Questionnaire

The renovation includes four spatial entities, namely, working site, block, building and façade, and some configurable components, namely, panels and fasteners (fasteners are devices to attach panels onto the façades). A hierarchical view is presented in Figure 3. Once the input file has been read by the support system, it can proceed by configuring a set of questions for each spatial entity in the file. Then, after the user answer the questionnaires, the system configures, i.e., deduces, the limits for panels' size and panels' weight for *each façade*. The questionnaires ask the following information.

Working site. This is the bigger spatial division in the renovation. It is commonly referred by a name and is well-know by the community. Values provided by the user are:

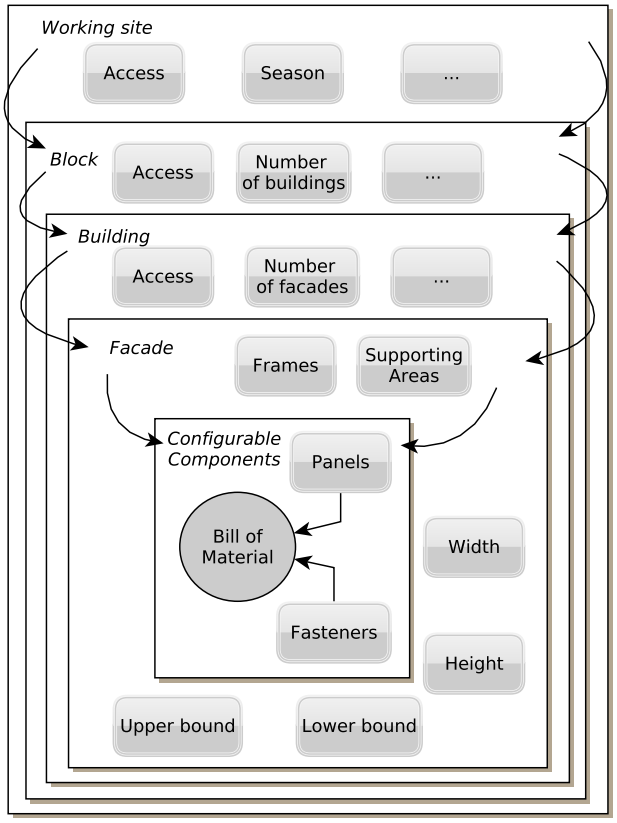


Figure 3: Spatial entities and arguments for renovation.

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

- Number of blocks in the working site? Number.
- Working site is in a windy region? {yes, no}
- Season in which the on-site work will take place? {summer, fall, winter, spring}
- Target for cost? Euros.
- Target for performance? $W.m^{-2}.K^{-1}$
- Obstacles presence? {yes, no}
- Accessibility to the working site? {easy, medium, hard}
- Panel's width ($\underline{w_{ws}}$) and height ($\underline{h_{ws}}$) lower bound? $[0, \infty]$
- Panel's width ($\overline{w_{ws}}$) and height ($\overline{h_{ws}}$) upper bound? $[0, \infty]$
- Panel's maximum weight ($\overline{we_{ws}}$)? $[0, \infty]$

Block. Is a set of buildings which are usually attached by a common wall. Values provided by the user are:

- Number of buildings in the block? Number.
- Obstacles presence? {yes, no}
- Accessibility to the block? {easy, medium, hard}
- Panel's width ($\underline{w_{bl}}$) and height ($\underline{h_{bl}}$) lower bound? $\underline{w_{bl}} \in [\underline{w_{ws}}, \overline{w_{ws}}]$ and $\underline{h_{bl}} \in [\underline{h_{ws}}, \overline{h_{ws}}]$
- Panel's width ($\overline{w_{bl}}$) and height ($\overline{h_{bl}}$) upper bound? $\underline{w_{bl}} \in [\underline{w_{ws}}, \overline{w_{ws}}]$ and $\underline{h_{bl}} \in [\underline{h_{ws}}, \overline{h_{ws}}]$
- Panel's maximum weight ($\overline{we_{bl}}$)? $[0, \overline{we_{ws}}]$

Building. Is the actual place where apartment are arranged and is the host of several façades. Values provided by the user are:

- Number of façades in the building? Number.
- Obstacles presence? {yes, no}
- Accessibility to the block? {easy, medium, hard}
- Panel's width ($\underline{w_{bg}}$) and height ($\underline{h_{bg}}$) lower bound? $\underline{w_{bg}} \in [\underline{w_{bl}}, \overline{w_{bl}}]$ and $\underline{h_{bg}} \in [\underline{h_{bl}}, \overline{h_{bl}}]$
- Panel's width ($\overline{w_{bl}}$) and height ($\overline{h_{bl}}$) upper bound? $\underline{w_{bg}} \in [\underline{w_{bl}}, \overline{w_{bl}}]$ and $\underline{h_{bg}} \in [\underline{h_{bl}}, \overline{h_{bl}}]$
- Panel's maximum weight ($\overline{we_{bg}}$)? $[0, \overline{we_{bl}}]$

Façade. Maybe seen as a big wall, but is in fact a composition of apartment along with its doors, windows and so on. Values provided by the user are:

- Obstacles presence? {yes, no}
- Accessibility to the block? {easy, medium, hard}
- Type of attaching device: {bottom, top, lateral}
- Panel's width ($\underline{w_{fc}}$) and height ($\underline{h_{fc}}$) lower bound? $\underline{w_{fc}} \in [\underline{w_{bg}}, \overline{w_{bg}}]$ and $\underline{h_{fc}} \in [\underline{h_{bg}}, \overline{h_{bg}}]$
- Panel's width ($\overline{w_{bl}}$) and height ($\overline{h_{bl}}$) upper bound? $\underline{w_{fc}} \in [\underline{w_{bg}}, \overline{w_{bg}}]$ and $\underline{h_{fc}} \in [\underline{h_{bg}}, \overline{h_{bg}}]$
- Panel's maximum weight ($\overline{we_{fc}}$)? $[0, \overline{we_{bg}}]$

This information collection has two specific goals. On the one hand, it will provide details about renovation aspects, such as the targeted performance, that are needed in the configuration process. On the second hand, it provides upper bound for panel's size and panel's weight. Indeed, given the manufacturing, environmental, accessibility and even weather conditions, the size of panels composing the layout plan are limited as well as their weight. Thus, the aforementioned inputs have direct impact over configurable components and are defined by their compatibility relations.

3.2 One FaçAde One CSP

One critical aspect of the support system is the ability to configure a constraint satisfaction model for each façade to renovate. This is important because, first, each façade has (potentially) different size, number of windows, supporting areas etc. Possible positions for panels, for instance, must lie between zero and the façade width and height. Simply put, each façade has its own configuration parameters used in the constrain satisfaction model and in the layout generation process. And second, each façade may have different accessibility conditions, obstacles or even user preferences. Thus, panels' size limits, as well as their weight, are constrained by the specific conditions of the façade and not only by the conditions of the working site, block or building.

When configuring these CSP instances it is important to conserve downwards consistency. Downwards consistency refers to the fact that information on higher level of the renovation are is propagated to the inferior levels, i.e., working site → blocks → buildings → façades, but it can not propagate upwards. As an example consider only accessibility conditions, obstacles presence and panels' size limits, for the specification in Figure 4.

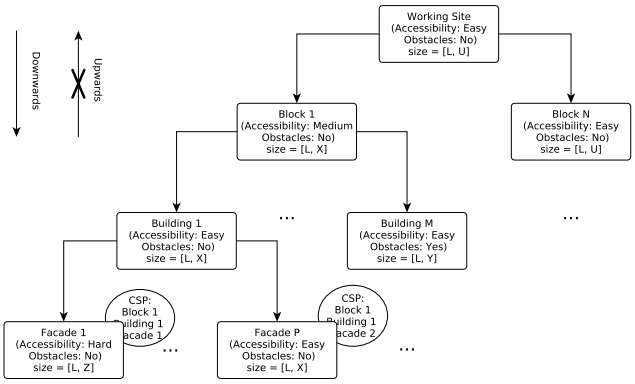


Figure 4: Downwards consistency among entities.

Note that inferior entities on the hierarchy inherit values from superior levels. But, it is not the case that information on superior levels should be consistent with information on inferior levels. In Figure 4, for instance, façade 1 has a hard accessibility condition and thus the upper bound for panels' size is reduced to a given Z. This upper bound is not propagated upwards to the building 1; it conserves its inherited value X. Consequently, façade 2 will inherit the value of X as no further reduction is needed for their panels configuration. Naturally, it is the case that $Z \subset X \subset U$. Using this information a CSP is configured for each façade to renovate.

4 Support System Constraint Services

In order to divide configuration tasks we divide the support system in two services that may be implemented in different servers. In essence, information about the renovation, entered by means of the input file and the questionnaire, is filtered by means of the first

service called *Filtering Service*. Then, the second service called *Solving Service*, upon user request, uses its configuration algorithms to provide complaint layout-plans solutions. Figure 5 presents the architecture of the on-line support system.

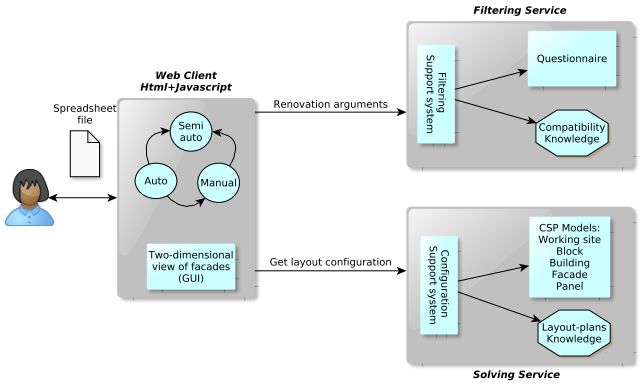


Figure 5: Service-based architecture for on-line configurator.

In order to give a clear understanding on how the services works, let us describe the input and output in a formal way. For each of the services the input is a tuple of the form $\langle \mathcal{SPEC}, \mathcal{V}, \mathcal{D}(\mathcal{V}), \mathcal{C}(V) \rangle$ with $|\mathcal{V}| = |\mathcal{D}(\mathcal{V})|$ and

- $\mathcal{SPEC} = \langle \mathcal{WS}, \mathcal{BK}, \mathcal{BG}, \mathcal{FAC} \rangle$; $\mathcal{WS}$ variables describing the working site, $\mathcal{BK}$ variables describing blocks, $\mathcal{BG}$ variables describing buildings and $\mathcal{FAC}$ variables describing façades.
- $\mathcal{V} = \langle \mathcal{P}, \mathcal{FA} \rangle$; $\mathcal{P}$ variables describing a single panel and $\mathcal{FA}$ variables describing a single fastener.
- $\mathcal{D}(\mathcal{V}) = \langle \mathcal{D}(\mathcal{P}), \mathcal{D}(\mathcal{FA}), \rangle$; domain for each one of the variables in $\mathcal{V}$.
- $\mathcal{C}(\mathcal{V})$ a set of constraints over variables in $\mathcal{V}$.

Information in $\mathcal{SPEC}$ describes only properties of the spatial entities such as the number, sizes, positions, etc. Variables in $\mathcal{V}$ and $\mathcal{D}(\mathcal{V})$, on the other hand, are manufacturer depended and includes size and position of panels and fasteners, and the initial domains which depends on the manufacturing process. Constraint in $\mathcal{C}(\mathcal{V})$ are extracted from the problem domain by an expert and are different in each service.

4.1 Filtering Service

4.1.1 Mapping

The filtering service is in charge of removing domain values from elements in $\mathcal{D}(\mathcal{V})$ that are not allowed by the established constraints. Here, constraints $\mathcal{C}(\mathcal{V})$ describe valid combination among different parameters in $\mathcal{SPEC}$ and variables in $\mathcal{D}(\mathcal{V})$. We denote this set of constraints $\mathcal{C}_f(\mathcal{V})$ to distinguish them from the ones used on the solving service. These constraints are formalized as compatibility tables (presented in next section). Formally, the filtering is a mapping $\mathcal{M}$ from variables and domains to domains

$$\mathcal{M}(\mathcal{SPEC}, \mathcal{V}, \mathcal{D}(\mathcal{V}), \mathcal{C}_f(\mathcal{V})) \rightarrow \mathcal{D}'(\mathcal{V}) \tag{1}$$

The result $\mathcal{D}'(\mathcal{V})$ contains the new domain for panels and fasteners, where $\mathcal{D}'(\mathcal{V}) \subseteq \mathcal{D}(\mathcal{V})$.

As stated previously, the initial filtering has as goal setting domains for configurable components and takes spatial entities information and constraints to do so. In our on-line support system we use the `CoFiADe` [27] system to perform this filtering. Several reasons support our choice. First, the system is already on-line, making it usable in no time. Second, it is well conceived for supporting decision-making processes. And third, it uses efficient compatibility tables for domain pruning; applying a given compatibility table is made in constant time $\mathcal{O}(1)$.

4.1.2 Compatibility Knowledge

Configurable components of the renovation are panels and fasteners to attach panels.

Panels. Configurable by fixing their width, height, weight and position over the façade.

Fasteners. Configurable by fixing its length and setting its type {bottom, top, lateral}.

The following compatibility tables, presented from Table 1 to Table 8, show the allowed combination between user's input values and configurable components values.

4.2 Solving Services

4.2.1 Search

The second service in the support system is in charge of layout-plans generation. The system uses several algorithms to generate layout plans but, although their behavior are quite different, their semantic remains the same.

Now, while information of $\mathcal{SPEC}$ and $\mathcal{V}$ are the same as the filtering services, it is not the case for domains and constraints. To differentiate them lest call the input domains $\mathcal{D}_s(\mathcal{V})$ and the constraints $\mathcal{C}_s(\mathcal{V})$. Intuitively, variable domains $\mathcal{D}_s(\mathcal{V})$ are provided by the mapping of the filtering service, i.e.,

$$\mathcal{M}(\mathcal{SPEC}, \mathcal{V}, \mathcal{D}(\mathcal{V}), \mathcal{C}_f(\mathcal{V})) = \mathcal{D}'(\mathcal{V}) = \mathcal{D}_s(\mathcal{V}) \tag{2}$$

where $\mathcal{D}(\mathcal{V})$ is the initial variable domain of the problem. Constraints in $\mathcal{C}_s(\mathcal{V})$ are expressed as first order formulas and express, not compatibility among elements but, requirements for valid layout plans (see next section for a description of these constraints). Both resolution approaches implemented at the solving service respect all constraints; the first approach resolving conflicts at positioning each panel (greedy fashion) whereas the second approach uses the open constraint programming environment `Choco` [20] to explore the solution space.

The output of the server's process is a set of layout-plan solutions. Formally, the server's process is a function of the form

$$\mathcal{F}(\mathcal{SPEC}, \mathcal{V}, \mathcal{D}_s(\mathcal{V}), \mathcal{C}_s(\mathcal{V}), \mathcal{H}) = \langle \mathcal{X}, \mathcal{Y}, \mathcal{DX}, \mathcal{DY} \rangle \tag{3}$$

where $\mathcal{X}$ and $\mathcal{Y}$ represent the origin of coordinates for each panel in the solution, and $\mathcal{DX}$ and $\mathcal{DY}$ the width and height, respectively, for each panel in the solution. Additionally, the function is parameterized by an heuristic $\mathcal{H}$ stating the way the solution space is explored. Available strategies are greedy and depth-first search.

Table 1: Relation C_1 between environmental conditions of spatial entities and panel's size, where α and β are upper-bounds for panel's width and height, respectively, when constrained by environmental conditions.

C_1	
Wind	Panel's size
yes	$(w_p \leq \alpha) \wedge (h_p \leq \beta)$
no	$\varnothing$

Table 2: Relation C_2 season that on-site work will take place and and panel's size, where θ and τ are upper-bounds for panel's width and height, respectively, when constrained by the season.

C_2	
Season	Panel's size
Summer $\vee$ Spring	$\varnothing$
Fall $\vee$ Winter	$(w_p \leq \theta) \wedge (h_p \leq \tau)$

Table 3: Relation C_3 between obstacles in spatial entities and panel's size, where ϕ and σ are upper-bounds for panel's width and height, respectively, when constrained by the presence of obstacles.

C_3	
Obstacles	Panel's size
yes	$(w_p \leq \phi) \wedge (h_p \leq \sigma)$
no	$\varnothing$

Table 4: Relation C_4 between accessibility of spatial entities and panel's size, where λ and π are upper-bounds for panel's width and height, respectively, when constrained by medium level accessibility conditions, and Λ and Π are upper-bounds for panel's width and height, respectively, when constrained by hard level accessibility conditions.

C_4	
Accessibility	Panel's dimensions
easy	$\varnothing$
medium	$(w_p \leq \lambda) \wedge (h_p \leq \pi)$
hard	$(w_p \leq \Lambda) \wedge (h_p \leq \Pi)$

Table 5: Relation C_5 between renovation cost and panel's insulation: It illustrates the fact that the quality of the insulation depends on the user budget.

C_5	
Cost	Panel's insulation
$<$ 50000	low
[50000,100000]	medium
$\geq$ 100000	high

Table 6: Relation C_6 between desired performance and panel's insulation: It illustrates the fact that the quality of the insulation depends on the desired final energetic performance.

C_6	
Performance	Panel's insulation
$<$ 25	high
[25,50]	medium
$\geq$ 50	low

Table 7: Relation C_7 between panel's weight and fasteners' positions.

C_7	
Weight	Position Fasteners
$<$ 500	$\varnothing$
[500,1000]	{top,bottom}
$>$1000	bottom

Table 8: Relation C_8 between fasteners' position and number of fasteners.

C_8	
Position fasteners	# fasteners
{top,bottom}	[2,4,6]
laterals	[4,6]

4.2.2 *Layout knowledge*

Let F denote the set of frames and S the set of supporting areas. Let $o_{e.d}$ and $l_{e.d}$ denote the origin and length, respectively, of a given entity e in the dimension d, with $d \in [1, 2]$. For instance, $o_{fr.1}$ denotes the origin in the horizontal axis and $l_{fr.1}$ denotes the width of frame fr. Additionally, lb_d and ub_d denote the length lower bound and length upper bound, respectively, in dimension d for all panels.

Each panel is described by its origin point w.r.t. the façade origin and its size. For convenience, lets assume that $\mathcal{P}$ is the set of panels composing the layout-plan solution. Then, each $p \in \mathcal{P}$ is defined by $\langle o, l \rangle$ where

- $o_{p.d} \in [0, o_{fac.d}]$ is the origin of panel p in dimension d.
- $l_{p.d} \in [lb_{p.d}, ub_{p.d}]$ is the length of panel p in dimension d.

The following six constraints express the relations among panels, and panels and façade that must respect a layout solution.

(a) *Manufacturing and transportation limitations constrain panel's size with a give upper bound* ub *in one or both dimensions.*

$$\forall_p \in P \; l_{p.d} \leq ub_d$$

(b) *(diffN) For two given panels p and q there is at least one dimension where their projections do not overlap.*

$$\forall_p \in P, \forall_q \in P, p \neq q, \exists d \in [1, 2] \mid \\ o_{p.d} \geq o_{q.d} + l_{q.d} \vee o_{q.d} \geq o_{p.d} + l_{p.d}$$

(c) *A given panel p must either be at the façade edge or ensure that enough space is left to fix another panel.*

$$\forall_p \in P \; o_{p.d} + l_{p.d} \leq l_{fac.d} - lb_k \vee o_{p.d} + l_{p.d} = l_{fac.d}$$

(d) *Each frame over the façade must be completely overlapped by one and only one panel. Additionally, frames' borders and panels' borders must be separated by a minimum distance denoted by* Δ.

$$\forall_f \in F, \exists p \in P \mid \\ o_{p.d} + \Delta \leq o_{f.d} \wedge o_{f.d} + l_{f.d} \leq o_{p.d} + l_{p.d} + \Delta$$

(e) *The entire façade surface must be covered with panels.*

$$\textstyle\sum_{i \in P} \prod_{d \in [1,2]} (o_{i.2} + l_{i.2}) = \prod_{d \in [1,2]} l_{fac.d}$$

(f) *Panels' corners must be matched with supporting areas in order to be properly attached onto the façade.*

$$\forall_p \in P, \exists s \in S \mid o_{s.d} \leq o_{p.d} \vee o_{p.d} + l_{p.d} \leq o_{s.d} + l_{s.d}$$

5 (De)Coupling Benefits

In order to efficiently couple two constraint-based systems it is necessary to assign disjoint tasks for them. In our approach, we divide initial filtering and solving in two different services. Benefits for this tasks division are rather simple.

On the one hand we apply the well-known principle *Divide and conquer*. In our on-line system this principle allow us to add or remove variables, domains and questions in the filtering service, i.e., by means of adding or removing compatibility tables. In addition, as we use `CoFiADe`, we may mix different variable representation as integer domains, continuous domains and symbolic domains whereas in most constraint systems mixing variable domains is not allowed or is not efficient enough. For instance, given the reduced number of constraints for continuous domains in `Choco`, the representation have to be changed to integer domains which in consequence involve additional and time consuming efforts.

On the other hand, as a benefit of tasks division, we improve performance by avoiding the use of binary equalities and binary inequalities constraints whose computational time is $\mathcal{O}(n * m)$, where n and m are the number of values in the domain of the two variables involved in the constraint. Thus, at the moment of finding solutions, the underlying constraint solver, in our case `Choco`, propagates and applies search using only those constraints defining a layout plan.

Regarding the performance the two configuration tasks must be studied separately. As commented before, applying a given compatibility table in the filtering service is made in constant time. Thus, the time involved in the filtering service depends on the questionnaire that depends on the number of buildings, facades and so on. On the solving service, by contrast, the performance depends on the underlying filtering and search provided by `Choco`. Execution over façades with size 40×10 meters, 50×12 meters and 60×15 meters takes between one and two seconds. The use of a dedicated heuristic that exploits the problem structure allows to reach such good performance.

The decoupling of tasks, and coupling of two constraint-based systems, is supported by the underlying declarative model. Indeed, the monotonic properties of constraint-based systems make it possible to add information (constraints) on one system without loosing any solution on the other system. Thus, the declarative view of constraint satisfaction make it possible to handle services as independent communication agents.

6 Conclusions

The aim of this paper has been to introduce an architecture of constraint-based product configuration that couples two constraint-based systems. We have presented an architecture that divided initial variable domain filtering and search space exploration. The method divide and conquer allow us to make straightforward adaptations to each service separately. Our approach have been applied to façade-layout configuration and implemented in an on-line support system. For this particular scenario we have

1. Formalize each service behavior and the relation among them.
2. Presented the constraints, expressed in compatibility tables and carried out by the `CoFiADe` system, for initial filtering.
3. Presented the constraints, expressed as first order formulae and carried out by `Choco` constraint programming environment, that are used to generate compliant layout solutions.
4. Show how to solve the configuration tasks by coupling the two constraint-based systems.
5. Show that consistency and integrity of solutions are straightforward modeled and implemented thanks to the monotonic properties of constraint satisfaction problems.
6. Show that the underlying coupling and communication methods are transparent for the user (it only interacts with a friendly web-based interface).

This work is part of a project on buildings thermal renovation assisted by intelligent systems. A configuration problem arise in this context: Configuring a specific set of rectangular panels with respect to create a building envelope. As the industrial scenario evolves the support system must be able to adapt to new requirements. Thus, strategic directions for our work are three-fold. On the one hand, improve each service by adding new constraints. On the second hand, definition of an API that allow us to replace the underlying processes in each service without loosing solutions. For instance, the solving service may be replaced by the same implementation of the model but using a different constraint solver (e.g., `Gecode` [24], `ILOG CPLEX CP` [7]). We consider that a good support system must be robust enough to allow such adaptations. Finally, a consistent benchmark must be carried on in order to compare the performance when dividing configuration tasks and when they are executed by the same service/constraint system.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the TBC Générateur d'Innovation company, the Millet and SyBois companies and all partners in the CRIBA project, for their contributions on recollecting buildings renovation information. Special thanks to the referees for their comments and to Philippe Chantry from École des Mines d'Albi for his contribution to the on-line system graphical interface and additional abstractions.

REFERENCES

[1] Ö. Akin, B. Dave, and S. Pithavadian, 'Heuristic generation of layouts (hegel): based on a paradigm for problem structuring', *Environment and Planning B: Planning and Design*, **19**(1), pp. 33 – 59, (1992).
[2] R. Barták, 'Constraint Programming: In Pursuit of the Holy Grail', in *Proceedings of the Week of Doctoral Students (WDS)*, (June 1999).
[3] Can A. Baykan and Mark S. Fox, 'Artificial intelligence in engineering design (volume i)', chapter WRIGHT: A Constraint Based Spatial Layout System, 395–432, Academic Press Professional, Inc., San Diego, CA, USA, (1992).
[4] S. C. Brailsford, C. N. Potts, and B. M. Smith, 'Constraint satisfaction problems: Algorithms and applications', *European Journal of Operational Research*, **119**(3), pp. 557 – 581, (1999).
[5] The Energy Conservation Center, *Energy Conservation Handbook*, The Energy Conservation Center, Japan, 2011.
[6] U.S. Green Building Council, *New Construction Reference Guide*, 2013.
[7] IBM IBM ILOG CPLEX. Ibm ilog cplex optimization studio cp optimizer users manual, 2014.
[8] M. Falcon and F. Fontanili, 'Process modelling of industrialized thermal renovation of apartment buildings', *eWork and eBusiness in Architecture, Engineering and Construction*, 363–368, (2010).
[9] U. Flemming, 'Knowledge representation and acquisition in the LOOS system', *Building and Environment*, **25**(3), 209 – 219, (1990).
[10] U. Flemming, C.A. Baykan, R.F. Coyne, and M.S. Fox, 'Hierarchical generate-and-test vs constraint-directed search', in *Artificial Intelligence in Design '92*, eds., J.S. Gero and Fay Sudweeks, 817–838, Springer Netherlands, (1992).
[11] U. Flemming and R. Woodbury, 'Software environment to support early phases in building design (seed): Overview', *Journal of Architectural Engineering*, **1**(4), 147–152, (1995).

[12] Esther Gelle and Rainer Weigel, 'Interactive configuration based on incremental constraint satisfaction', in *IFIP TC5/WG 5.2 Workshop Series Knowledge-Intensive CAD*, pp. 117–126, Helsinki, Finland, (1995).
[13] M. M. D. Hassan, G. L. Hogg, and D. R. Smith, 'Shape: A construction algorithm for area placement evaluation', *International Journal of Production Research*, **24**(5), pp. 1283–1295, (1986).
[14] Bjrn Petter Jelle, 'Traditional, state-of-the-art and future thermal building insulation materials and solutions - properties, requirements and possibilities', *Energy and Buildings*, **43**(10), 2549 – 2563, (2011).
[15] U. Junker, *Configuration.*, Chapter 24 of Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, NY, USA, 2006.
[16] K.J. Lee, H.W. Kim, J.K. Lee, and T.H. Kim, 'Case-and constraint-based project planning for apartment construction.', *AI Magazine*, **19**(1), pp. 13–24, (1998).
[17] B. Medjdoub and B. Yannou, 'Separating topology and geometry in space planning', *Computer-Aided Design*, **32**(1), 39 – 61, (2000).
[18] U. Montanari, 'Networks of constraints: Fundamental properties and applications to picture processing', *Information Sciences*, **7**(0), 95 – 132, (1974).
[19] L. Pérez-Lombard, J. Ortiz, and C. Pout, 'A review on buildings energy consumption information', *Energy and Buildings*, **40**(3), 394 – 398, (2008).
[20] C. Prud'homme and JG. Fages, 'An introduction to choco 3.0 an open source java constraint programming library', in *CP Solvers: Modeling, Applications, Integration, and Standardization. International workshop.*, Uppsala Sweden, (2013).
[21] Christian Schulte, *Programming Constraint Services*, volume 2302 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2002.
[22] Christian Schulte and Mats Carlsson, 'Finite domain constraint programming systems', *Handbook of constraint programming*, 495–526, (2006).
[23] Christian Schulte and Gert Smolka. Finite domain constraint programming in oz. a tutorial., 2000.
[24] Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. Modeling and programming with gecode, 2010.
[25] S Shikder, A Price, and M Mourshed, 'Interactive constraint-based space layout planning', *W070-Special Track 18th CIB World Building Congress May 2010 Salford, United Kingdom*, 112, (2010).
[26] Timo Soininen, Juha Tiihonen, Tomi Männistö, and Reijo Sulonen, 'Towards a general ontology of configuration', *Artif. Intell. Eng. Des. Anal. Manuf.*, **12**(4), 357–372, (September 1998).
[27] Élise Vareilles. Paul Gaborit. Michel Aldanondo. Sabinne Carbonnel. Laurent Steffan, 'Cofiade constraints filtering for aiding design', in *Actes des neuviemes Journées Francophones de Programmation par Contraintes*, Toulouse France, (2012).
[28] E. Vareilles, A. F. Barco, M. Falcon, M. Aldanondo, and P. Gaborit, 'Configuration of high performance apartment buildings renovation: a constraint based approach', in *Conference of Industrial Engineering and Engineering Management (IEEM)*. IEEE., (2013).
[29] Dong Yang, Ming Dong, and Rui Miao, 'Development of a product configuration system with an ontology-based approach', *Computer-Aided Design*, **40**(8), 863 – 878, (2008).
[30] M. Zawidzki, K. Tateyama, and I. Nishikawa, 'The constraints satisfaction problem approach in the design of an architectural functional layout', *Engineering Optimization*, **43**(9), pp. 943–966, (2011).

Solving Combined Configuration Problems: A Heuristic Approach[1]

Martin Gebser[2] and **Anna Ryabokon**[3] and **Gottfried Schenner**[4]

Abstract. This paper describes an abstract problem derived from a combination of Siemens product configuration problems encountered in practice. Often isolated parts of configuration problems can be solved by mapping them to well-studied problems for which efficient heuristics exist (graph coloring, bin-packing, etc.). Unfortunately, these heuristics may fail to work when applied to a problem that combines two or more subproblems. In the paper we show how to formulate a combined configuration problem in Answer Set Programming (ASP) and to solve it using heuristics à la hclasp. The latter stands for heuristic clasp that is nowadays integrated in clasp and enables the declaration of domain-specific heuristics in ASP. In addition, we present a novel method for heuristic generation based on a combination of greedy search with ASP that allows to improve the performance of clasp.

1 Introduction

Researchers in academia and industry have tried different approaches to configuration knowledge representation and reasoning, including production rules, constraints languages, heuristic search, description logics, etc.; see [16, 14, 9] for surveys. Although constraint-based methods remain de facto standard, Answer Set Programming (ASP) has gained much attention over the last years because of its expressive high-level representation abilities.

As evaluation shows ASP is a compact and expressive method to capture configuration problems [15, 18, 9], i.e. it can represent configuration knowledge consisting of component types, associations, attributes, and additional constraints. The declarative semantics of ASP programs allows a knowledge engineer to choose the order in which rules are written in a program, i.e. the knowledge about types, attributes, etc. can be easily grouped in one place and modularized. Sound and complete solving algorithms allow to check a configuration model and support evolution tasks such as reconfiguration. Generally, the results prove that ASP has limitations when applied to large-scale product (re)configuration instances [1, 5]. The best results in terms of runtime and solution quality were achieved when domain-specific heuristics were applied [17, 12].

In this paper we introduce a combined configuration problem that reflects typical requirements frequently occurring in practice at Siemens. The parts of this problem correspond (to some extent) to classical computer science problems for which there already exist some well-known heuristics and algorithms that can be applied to speed up computations and/or improve the quality of solutions.

As the main contribution, we present a novel approach on how heuristics generated by a greedy solver can be incorporated in an ASP program to improve computation time (and obtain better solutions). The application of domain-specific knowledge formulated succinctly in an ASP heuristic language [8] allows for better solutions within a shorter solving time, but it strongly deteriorates the search process when some additional requirements (conflicting with the formulated heuristics) are included. On the other hand, the formulation of complex heuristics might be cumbersome using greedy methods. Therefore, we exploit a combination of greedy methods with ASP for the generation of heuristics and integrate them to accelerate an ASP solver. We evaluate the method on a set of instances derived from configuration scenarios encountered by us in practice and in general. Our evaluation shows that for three different sets of instances solutions can be computed an order of magnitude faster than compared to a plain ASP encoding.

The remainder of this paper is structured as follows. Section 2 introduces a combined configuration problem (CCP) which is exemplified in Section 3. Section 4 discusses heuristics for solving the CCP. We present our evaluation results in Section 5. Finally, in Section 6 we conclude and discuss future work.

2 Combined Configuration Problem

The Combined Configuration Problem (CCP) is an abstract problem derived from a combination of several problems encountered in Siemens practice (railway interlocking systems, automation systems, etc.). A CCP instance is defined by a directed acyclic graph, called just graph later on in this paper for simplicity. Each vertex of the graph has a type and each type of the vertices has a particular size. In addition, each instance comprises two sets of vertices specifying two vertex-disjoint paths in the graph. Furthermore, an instance contains a set of areas, sets of vertices defining possible border elements of each area and a maximal number of border elements per area. Finally, a number of available colors as well as a number of available bins and their capacity are given.

Given a CCP instance, the goal is to find a solution that satisfies a set of requirements. All system requirements are separated into the corresponding subproblems which must be solved together or in combinations:

- **P1 Coloring** *Every vertex must have exactly one color.*
- **P2 Bin-Packing** *For every color a Bin-Packing problem must be solved. For every color the same number of bins are available. Every vertex must be assigned to exactly one bin of its color and*

[1] This work was funded by COIN and AoF under grant 251170 as well as by FFG under grant 840242. An extended version of this paper is to appear in the proceedings of the 13th International Conference on Logic Programming and Non-monotonic Reasoning.

[2] Aalto University, HIIT, Finland and University of Potsdam, Germany, email: martin.gebser@aalto.fi

[3] Alpen-Adria-Universität Klagenfurt, Austria, email: anna.ryabokon@aau.at

[4] Siemens AG Österreich, Austria, email: gottfried.schenner@siemens.com

for every bin it holds that the sum of sizes must be smaller or equal to the bin capacity.

- **P3 Disjoint Paths** *Vertices of different paths cannot be colored in the same color.*
- **P4 Matching** *Each border element must be assigned to exactly one area such that the number of selected border elements of an area does not exceed the maximal number of border elements and all selected border elements of an area have the same color.*
- **P5 Connectedness** *Two vertices with the same color must be connected via a path that contains only vertices of that color.*

Origin In the railway domain the given graph represents a track layout of a railway line. A coloring **P1** can then be thought as an assignment of resources (e.g. computers) to the elements of the railway line. In real-world scenarios different infrastructure elements may require different amounts of a resource that is summarized in **P2**. This may be hardware requirements (e.g. a signal requiring a certain number of hardware parts) or software requirements (e.g. an infrastructural element requiring a specific processing time). The requirements of **P1** and **P2** are frequently used in configuration problems during an assignment of entities of one type to entities of another type [11, 5].

The constraint of **P3** increases availability, i.e. in case one resource fails it should still be possible to get from a source vertex (no incoming edges) of the graph to a target vertex (no outgoing edges) of the graph. In the general version of this problem one has to find n paths that maximize availability. The CCP uses the simplified problem where 2 vertex-disjoint paths are given.

P4 stems from detecting which elements of the graph are occupied. The border elements function as detectors for an object leaving or entering an area. The Partner Units Problem [2, 1] is a more elaborate version of this problem. **P5** arises in different scenarios, e.g. if communication between elements controlled by different resources is more costly, then neighboring elements should be assigned to the same resource whenever possible.

3 Example

Figure 1 shows a sample input CCP graph. In this section we illustrate how particular requirements can influence a solution. Namely, we add the constraints of each subproblem one by one. If only **P1** is active, any graph corresponds to a trivial solution of **P1** where all vertices are colored white.

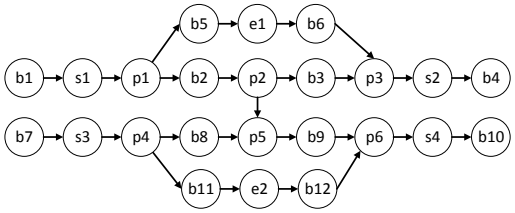


Figure 1: Input CCP graph and a trivial solution of Coloring (**P1**)

Let us consider the input graph as a Bin-Packing problem instance with four colors and three bins per color of a capacity equal to five. The vertices of type b, e, s and p have the sizes 1, 2, 3 and 4, respectively. A solution of Coloring and Bin-Packing (**P1-P2**) is presented in Figures 2 and 3.

Moreover, two vertex-disjoint paths are declared by $path1 = \{b1, s1, p1, b2, p2, b3, p3, s2, b4\}$ as well as $path2 = \{b7, s3, p4, b8, p5, b9, p6, s4, b10\}$, and the Disjoint Paths constraint (**P3**) is active. Consequently, in this case the solution shown in Figure 2 violates this constraint and must be modified as given in Figure 4 where the vertices of different paths are colored with different colors ($path1$ with dark grey and grey, and $path2$ with white and light grey).

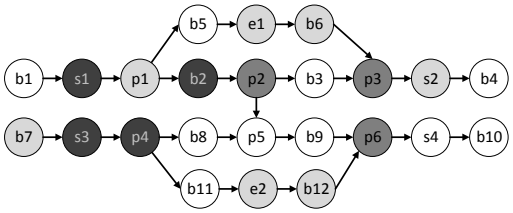


Figure 2: Used colors in a solution of the Coloring and Bin-Packing problems (**P1**-**P2**)

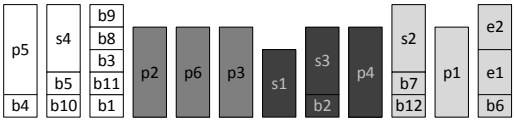


Figure 3: Used bins in a solution of the Coloring and Bin-Packing problems (**P1**-**P2**)

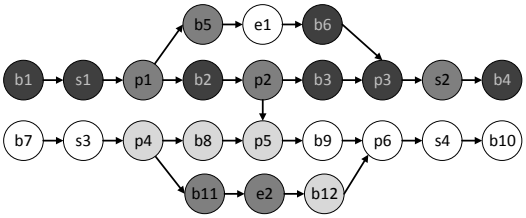


Figure 4: Solution of the Coloring, Bin-Packing and Disjoint Paths problems (**P1**-**P3**)

Figure 5 shows an example of Matching (**P4**). In this example there are seven areas in the matching input graph, each corresponding to a subgraph surrounded with border elements (Figure 1). For instance, area $a1$ represents the subgraph $\{b1, s1, p1, b2, b5\}$ and area $a2$ the subgraph $\{b5, e1, b6\}$. The corresponding border elements, $\{b1, b2, b5\}$ and $\{b5, b6\}$, are displayed in Figure 5.

Assume that an area can have at most 2 border elements assigned to it. In the resulting matching (Figure 5) $b1$, $b2$ are assigned to $a1$, whereas $b5$, $b6$ are assigned to $a2$. Note that the sample selected matching shown in Figure 5 is not valid with the coloring presented previously, because, for example, $b5$ and $b6$ are assigned to the same area $a2$ although they are colored differently.

In addition, the coloring solution shown in Figure 4 violates the Connectedness constraint (**P5**). Therefore, the previous solutions must be updated to take the new requirements into account. Figure 6 shows a valid coloring of the given graph that satisfies all problem requirements (**P1**-**P5**).

4 Combining Heuristics for Configuration Problems

We formulated the CCP using ASP and the corresponding encoding can be found at `http://isbi.aau.at/hint/problems`.

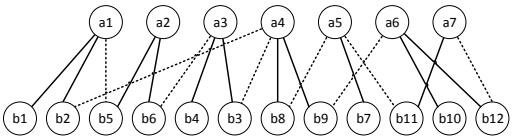


Figure 5: A sample input and solution graphs for **P4**. The selected edges of the input graph are highlighted with solid lines.

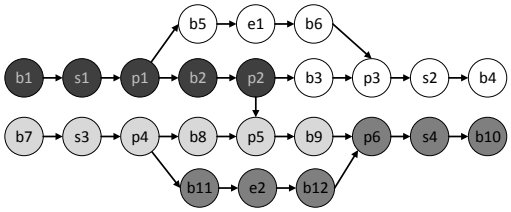


Figure 6: A valid solution for **P1**-**P5**

To formulate a heuristic within ASP we use the declarative heuristic framework developed by Gebser et al. [8]. In this formalism the heuristics are expressed using atoms $\_heuristic(a, m, v, p)$, where a denotes an atom for which a heuristic value is defined, m is one of four modifiers (init, factor, level and sign), and v, p are integers denoting a value and a priority, respectively, of the definition. A number of shortcuts are available, e.g. $\_heuristic(a, v, l)$, where a is an atom, v is its truth value and l is a level. The heuristic atoms modify the behavior of the VSIDS heuristic [10]. Thus, if a $\_heuristic$ atom is true in some interpretation, then the corresponding atom a might be preferred by the ASP solver at the next decision point.

There are different ways to incorporate heuristic atoms in a program. The standard approach [8] requires an implementation of a heuristic at hand using a pure ASP encoding, whereas the idea of our method is to delegate the (expensive) generation of a heuristic to an external tool and then to extend the program with generated heuristic atoms to accelerate the ASP search. Below we will exemplify how both approaches can be applied.

4.1 Standard generation of heuristics in ASP

Several heuristics can be used for the problems that compose the CCP. For instance, for the coloring of vertices (**P1**) we seek to use as few colors as possible by the following rule:

```
1 _heuristic(vertex_color(V,C),true,MC-C) :-
      vertex(V), color(C), nrofcolors(MC).
```

Listing 1: Heuristic for an assignment of colors to vertices

Roughly speaking, this rule means that the assignment of colors to vertices must be done in an ascending order of colors. Given a $vertex(b1)$ and two colors $nrofcolors(2)$ encoded as $color(1)$ and $color(2)$, the solver can derive two heuristic atoms $\_heuristic(vertex\_color(b1, 1), true, 1)$ and $\_heuristic(vertex\_color(b1, 2), true, 0)$. These atoms indicate the solver that the atom $vertex\_color(b1, 1)$ must be assigned the truth value $true$ first since the atom with the higher level is preferred.

Additionally, we can apply the well-known Bin-Packing heuristics for the placement of colored vertices into the bins of specified capacity (**P2**). The Bin-Packing problem is known to be an NP-hard combinatorial problem. However, there are a number of approximation algorithms (construction heuristics) that allow efficient computation of good approximations of a solution [6], e.g. Best/First/Next-Fit heuristics. They can, of course, be used as heuristics for the CCP.

Let the Bin-Packing problem instance be encoded using a set of predicates among which $nrofbins/1$ and $order/2$ denote a number of bins in the instance and an ordered set of input vertices, respectively. The predicate $vertex\_bin/2$ is used to encode a solution and denotes an assignment of a vertex to a bin. As shown in Listing 2, given a (decreasing) order of vertices, we can force the solver to place vertex V_i into the lowest-indexed bin for which the size of already placed vertices does not exceed the capacity, i.e. in a first-fit bin. The heuristic never uses a new bin until all the non-empty bins are full and it can be expressed by rules that generate always a higher level for the bins with smaller number:

```
1 binDomain(1..NB) :- nrofbins(NB).
2 offset(NB+1) :- nrofbins(NB).
3 _heuristic(vertex_bin(V,B),true,M+O*NB-B) :-
      binDomain(B), nrofbins(NB), order(V,O),
      offset(M).
```

Listing 2: First-Fit heuristic for an assignment of vertices to bins

It is also possible (with an intense effort) to express other heuristics for **P1**-**P5** that guide the search appropriately and allow to speed up the computation of solutions if we solve these problems separately. However, as our experiments show, the inclusion of heuristics for different problems at the same time might drastically deteriorate the performance for real-world CCP instances.

4.2 Greedy Search

From our observations in the context of product configuration, it is relatively easy to devise a greedy algorithm to solve a part of a configuration problem. This is often the case in practice, because products are typically designed to be easily configurable. The hard configuration instances usually occur when new constraints arise due to the combination of existing products and technologies.

The same can be said for the CCP. Whereas it is easy to develop greedy search algorithms for the individual subproblems, it becomes increasingly difficult to come up with an algorithm that solves the combined problem. For instance, a greedy algorithm for the Matching problem of the CCP (**P4**) can be formulated as follows: For every vertex v find a related area a with the fewest assigned vertices so far and match v with a. The algorithm assumes that all border elements are colored with one color, as it trivially satisfies the coloring requirement of the matching problem. A greedy algorithm for solving the CCP wrt. Coloring, Bin-Packing and Connectedness (**P1**, **P2** and **P5**) can be described as follows:

1. Select the first available color c and add the first vertex not assigned to any bin to a queue Q;
2. Get and remove from Q the first element v, label it with c and try to assign it to a bin using some Bin-Packing heuristic, e.g. First-Fit or Best-Fit [6];
3. If v is assigned to some bin, add neighbors of v to Q;
4. If $Q \neq \emptyset$, then goto 2;
5. Otherwise, if there are unassigned vertices, then make the color c unavailable and goto 1.

Suppose one wants to combine these two algorithms. One strategy would be to run greedy Matching and then solve the Bin-Packing

Algorithm 1: Greedy & ASP

Input: A problem P, an ASP program Π solving the problem P
Output: A solution S

```
1 GreedySolution ← solveGreedy(P);
2 H ← generateHeuristic(GreedySolution);
3 return solveWithASP(Π, H);
```

problem taking matchings into account. Namely, the combined algorithm preforms the following steps:

1. Call the matching greedy algorithm and get a set of matchings $M = \{(v_1, a_1), \ldots, (v_n, a_m)\}$;
2. For each vertex v_i of the input graph G do:
   (a) Assign a new color to v_i, if v_i has no assigned color;
   (b) Put v_i into a bin, as in the greedy Bin-Packing (steps 2-3);
   (c) If v_i is a border element, then retrieve an area a_j that matches v_i in M and color all vertices of this area in the same color as v_i.

The combined algorithm might violate Connectedness, because it colors all border vertices assigned to an area with the same color. However, these vertices are not necessarily connected. That is, there might be a solution with a different matching, but the greedy algorithm tests only one of all possible matchings. Moreover, there is no obvious way how to create an algorithm solving all three problems efficiently. This is a clear disadvantage of using ad-hoc algorithms in contrast to the usage of a logic-based formalism like ASP, where the addition of constraints is just a matter of adding some rules to an encoding. On the other hand, domain-specific algorithms are typically faster and scale better than ASP-based or SAT-based approaches that cannot be used for large instances. For instance, the memory demand of the greedy Bin-Packing algorithm is polynomial in graph size.

4.3 Combining Greedy Search and ASP

One way to let a complete ASP solver and a greedy search algorithm benefit from each other is to use the greedy algorithm to compute upper bounds for the problem to solve. The tighter upper bound usually means smaller grounding size and shorter solving time, because the greedy solver being domain-specific usually outperforms ASP for the relaxed version of the problem. For instance, running the greedy algorithm for the Bin-Packing problem and Matching problem gives upper bounds for the maximal number of colors, i.e. number of different Bin-Packing problems to solve. The same applies to the Matching problem. This kind of application of greedy algorithms has a long tradition in branch and bound search algorithms, where greedy algorithms are used to compute the upper bound of a problem. For an example see [19], where a greedy coloring algorithm is used to find an upper bound for the clique size in a graph for the computation of maximum cliques. In this paper we investigate a novel way to combine greedy algorithms and ASP (Algortihm 1). Consequently, in our approach we, first, use a greedy algorithm to find a solution of a relaxed version of the problem. Next, this solution is converted into a heuristic for an ASP solver which assigns the atoms of the greedy solution a higher heuristic value.

As an example for solving the complete CCP problem, we can, first, find an unconnected solution for the combination of Coloring, Bin-Packing, Disjoint paths and Matching problems (**P1**-**P4**), and then, use the ASP solver to fix the Connectedness property (**P5**). The idea of combining local search with a complete solver is also found in large neighborhood search [4].

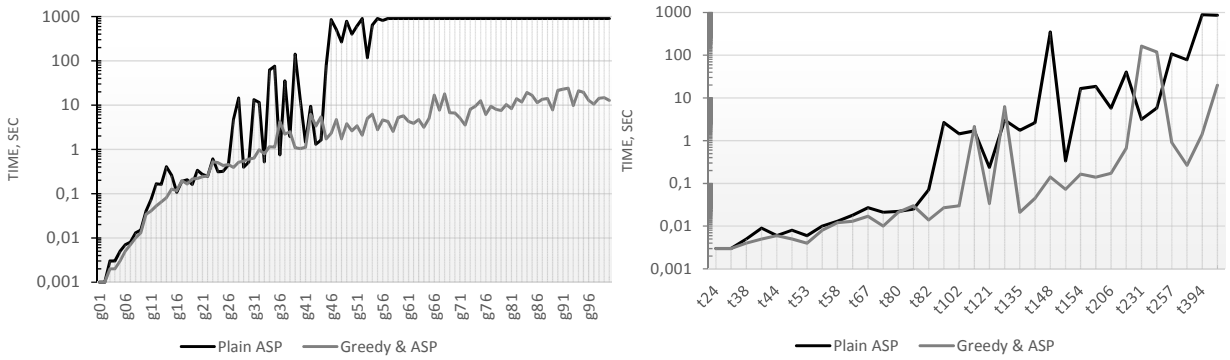
5 Experimental results

Experiment1 In our evaluation we compared a plain ASP encoding of the CCP with an ASP encoding extended with domain-specific knowledge. The Bin-Packing problem (**P2**) of the CCP corresponds to the classic Bin-Packing problem and the same heuristics can be applied. We implemented several Bin-Packing heuristics such as First/Best/Next-Fit (Decreasing) heuristics using ASP as shown in Section 4.1. For the evaluation we took 37 publicly available Bin-Packing problem instances[5], for which the optimal number of bins $optnrofbins$ is known, and translated them to CCP instances. The biggest instance of the set includes 500 vertices and 736 bins of the capacity 100. In the experiment, the maximal number of colors was set to 1 and the maximal number of bins was set to $2 \cdot optnrofbins$. All instances were solved by both approaches[6]. For a plain ASP encoding the solver required at most 27 seconds to find a solution whereas for the heuristic ASP program solving took at most 6 seconds, which is 4.5 times faster. The best results for the heuristic approach were obtained using the First-Fit heuristic with the decreasing order of vertices. Corresponding solutions utilized less bins then the ones obtained with the plain ASP program. Moreover, using First-Fit heuristic, for 23 from 37 instances a solution with optimal number of bins was found and for 13 other instances at most 4 bins more were required. The plain ASP encoding resulted in solutions that used on average 4 bins more than corresponding solutions of the heuristic approach.

Experiment2 In the next experiment we tested the same Bin-Packing heuristics implemented in ASP for the combined CCP, i.e. when all subproblems **P1**-**P5** are active, on 100 real-world test instances of moderate size (maximally 500 vertices in an input). The instances in this experiment were derived from a number of industrial configuration tasks. Neither the plain program nor the heuristic program were able to improve runtime/quality of solutions. Moreover, our greedy method described in Section 4.2 also failed to find a connected solution, i.e. when **P5** is active. For this reason, we investigated the combined approach (Greedy & ASP) described in Section 4.3. This approach uses the greedy method to generate a partial solution ignoring the Connectedness constraint and provides this solution as $\_heuristic$ atoms to the ASP solver. Our experiments show (see Figure 7a) that the combined approach can solve all 100 benchmarks from the mentioned set, whereas the plain encoding solves only 54 instances (the time frame was set to 900 seconds in this and the next experiment). Moreover, for those instances which were solved using both approaches, the quality of solutions measured in terms of used bins and colors was the same. However, the runtime of the combined approach was 18 times faster on average and required at most 24 seconds instead of 848 seconds needed for the plain ASP encoding.

Experiment3 In addition, we tested more complex real-world instances (maximally 1004 vertices in an input)[7] which we have also submitted to the ASP competition 2015. Similarly to *Experiment2*

[5] http://www.wiwi.uni-jena.de/Entscheidung/binpp/index.htm
[6] The evaluation was performed using clingo version 4.3.0 from the Potassco ASP collection [7] on a system with Intel i7-3030K CPU (3.20 GHz) and 64 GB of RAM, running Ubuntu 11.10.
[7] The instances are available at: http://isbi.aau.at/hint/problems



(a) *Experiment2* (b) *Experiment3*

Figure 7: Evaluation results using Plain ASP and Greedy & ASP

we compared the plain ASP encoding to the combined approach from Section 4.3. Again, regarding the quality of solutions, both approaches are comparable, i.e. they use on average the same number of colors and bins, with the combined approach having a slight edge. Generally, from 48 instances considered in this experiment, 36/38 instances were solved using the plain/combined encoding, respectively. On average/maximally the plain encoding needed 69/887 seconds to find a solution whereas the combined method took 14/196 seconds, respectively, which is about 5 times faster. Figure 7b shows the influence of heuristics on the performance for the instances from *Experiment3* that were solved by both approaches within 900 seconds. Although the grounding time is not presented for both experiments, we note that it requires about 10 seconds using both approaches for the biggest instance when all subproblems **P1**-**P5** are active.

6 Discussion

Choosing the right domain-specific heuristics for simple backtrack-based solvers is essential for finding a solution at all, especially for large and/or complex problems. The role of domain-specific heuristics in a conflict-driven nogood learning ASP solver seems to be less important when it comes to solving time. Here the size of the grounding and finding the right encoding is often the limiting factor. Nevertheless, *domain-specific heuristics are very important* to control the order in which answer sets are found and are an alternative to optimization statements. As we have shown, domain-specific heuristics also provide a mechanism to combine greedy algorithms with ASP solvers, which opens up the possibility to use ASP in a meta-heuristic setting. However, the possible applications go beyond this. The same approach could be used to repair an infeasible assignment using an ASP solver. This is currently a field of active research for us and has applications in the context of product reconfiguration. Reconfiguration occurs when a configuration problem is not solved from scratch, but some parts of an existing configuration have to be taken into account.

An open question is how to combine heuristics for different subproblems in a modular manner without the adaptation of every domain-specific heuristic. Here approaches like search combinators [13] from the constraint programming community might be useful. Another interesting topic for future research would be how to learn heuristics from an ASP solver, i.e. to investigate the variable/value order chosen by an ASP solver for medium size problem instances and use heuristics in a backtrack solver for larger instances that are out of scope of an ASP solver due to the grounding size. Some aspects of this topic were discussed in [3].

REFERENCES

[1] M. Aschinger, C. Drescher, G. Friedrich, G. Gottlob, P. Jeavons, A. Ryabokon, and E. Thorstensen, 'Optimization Methods for the Partner Units Problem', in *Proceedings of CPAIOR*, pp. 4–19, (2011).

[2] M. Aschinger, C. Drescher, G. Gottlob, P. Jeavons, and E. Thorstensen, 'Tackling the Partner Units Configuration Problem', in *Proceedings of IJCAI*, pp. 497–503, (2011).

[3] M. Balduccini, 'Learning and using domain-specific heuristics in ASP solvers', *AI Communications*, **24**(2), 147–164, (2011).

[4] Raffaele Cipriano, Luca Di Gaspero, and Agostino Dovier, 'A hybrid solver for large neighborhood search: Mixing gecode and easylocal++', in *Hybrid metaheuristics*, 141–155, Springer, (2009).

[5] G. Friedrich, A. Ryabokon, A. A. Falkner, A. Haselböck, G. Schenner, and H. Schreiner, '(Re) configuration based on model generation', in *Proceedings of LoCoCo*, pp. 26–35, (2011).

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[7] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, *Answer Set Solving in Practice*, Morgan & Claypool Publishers, 2012.

[8] M. Gebser, B. Kaufmann, J. Romero, R. Otero, T. Schaub, and P. Wanko, 'Domain-Specific Heuristics in Answer Set Programming', in *Proceedings of AAAI*, (2013).

[9] L. Hotz, A. Felfernig, M. Stumptner, A. Ryabokon, C. Bagley, and K. Wolter, 'Configuration Knowledge Representation and Reasoning', *Knowledge-Based Configuration: From Research to Business Cases*, 41–72, (2014).

[10] C.F. Madigan, S. Malik, M.W. Moskewicz, L. Zhang, and Y. Zhao, 'Chaff: Engineering an efficient SAT solver', in *Proceedings of DAC*, (2001).

[11] W. Mayer, M. Bettex, M. Stumptner, and A. Falkner, 'On solving complex rack configuration problems using CSP methods', in *Proceedings of the IJCAI Workshop on Configuration*, (2009).

[12] A. Ryabokon, G. Friedrich, and A. A. Falkner, 'Conflict-Based Program Rewriting for Solving Configuration Problems', in *Proceedings of LPNMR*, pp. 465–478, (2013).

[13] T. Schrijvers, G. Tack, P. Wuille, H. Samulowitz, and P. J. Stuckey, 'Search combinators', *Constraints*, **18**(2), 269–305, (2013).

[14] C. Sinz and A. Haag, 'Configuration', *IEEE Intelligent Systems*, **22**(1), 78–90, (2007).

[15] T. Soininen, I. Niemelä, J. Tiihonen, and R. Sulonen, 'Representing configuration knowledge with weight constraint rules', in *Proceedings of the Workshop on ASP*, pp. 195–201, (2001).

[16] M. Stumptner, 'An overview of knowledge-based configuration', *AI Communications*, **10**(2), 111–125, (1997).

[17] E. C. Teppan, G. Friedrich, and A. A. Falkner, 'QuickPup: A Heuristic Backtracking Algorithm for the Partner Units Configuration Problem', in *Proceedings of IAAI*, pp. 2329–2334, (2012).

[18] J. Tiihonen, M. Heiskala, A. Anderson, and T. Soininen, 'WeCoTin - A practical logic-based sales configurator', *AI Communications*, **26**(1), 99–131, (2013).

[19] Etsuji Tomita and Toshikatsu Kameda, 'An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments', *Journal of Global Optimization*, **37**(1), 95–111, (2007).

Towards a benchmark for configuration and planning optimization problems

Luis Garcés Monge[1], Paul Pitiot[1,2], Michel Aldanondo[1], Elise Vareilles[1]
[1]University Toulouse – Mines Albi, France
[2] 3IL-CCI Rodez, France

Abstract. Computer science community is always interested in « benchmarks », e.g. standard problems, by which performance of optimization approaches can be measured and characterized. This article aims at present our research perspectives to achieve a benchmark for concurrent configuration and planning optimization problems. A benchmark is a set of reference models that represents a particular kind of problem. Product configuration and project planning are classic problems abundantly handled in the literature. Their coupling in an integrated model is a more and more handled complex problem; but there is a lack of benchmark in spite of the need expressed by the community during last configuration workshops [config, 2013/2014]]. Two approaches may be combined to obtain a benchmark: (i) generalization of existing real applications (for example, automotive, telecommunication or computer industry), (ii) or using a structural analysis of theoretical model of the problem. In this article, we propose a meta-model of concurrent configuration and planning problem using these two approaches. It shall allow us to supply a representative and complete benchmark, in order to accurately estimate the contribution of existing optimization methods.

1 Introduction

Benchmarking of optimization approaches is crucial to assess performance quantitatively and to understand their weaknesses and strengths. There are numerous academic benchmarks associate with various classes of optimization problem (linear / nonlinear problems, constrained problems, integer or mixed integer programming, etc.). Studies, reports and websites of [Shcherbina, 2009] [Domes et al., 2014] [Mittelmann , 2009] [Gilbert and Jonsson, 2009] are particularly accomplished examples of existing optimization benchmark with a multitude of articles and algorithms benchmarked on great variety of test functions (see for example [Shcherbina et al., 2003], [Pál et al., 2012] or [Auger and Ros, 2009]).

More than an academic tool, a benchmark should also be representative of real-world problems. For a specific domain, a benchmark represents a reference which should be used by company's decision-makers to select an approach or an algorithm. But it is not always easy for them to know of which theoretical cases cover their practical cases. Benchmark on configuration field could illustrate this aspect with various industrial cases: automotive [Amilhastre et al., 2002], [Kaiser et al., 2003], [Sinz et al., 2003], power supply [Jensen, 2005], train design [Han and Lee, 2011], etc. A data-base of industrials cases was started on [Subbarayan, 2006] but it is not any more maintained.

Our previous research projects [Pitiot et al., 2013] aim at producing decision aiding tools for a specific problem subject to a growing interest in mass customization community: the coupling between product and project environments. Numerous authors [Baxter, D. et al., 2007] [Zhang et al. 2013] [Hong et al., 2010] or [Li et al., 2006], [Huang and Gu, 2006] showed the interest to take into account simultaneously the product and project dimensions in a decision aiding tool. This concurrent process has two main interests: i) Allowing to model, and thus to take into account, interactions between product and project (for example, a specific product configuration forbids using certain resources for project tasks), ii) Avoid the traditional sequence: configure product then plan its production which is the source of multiple iterations when selected product can't be obtained in satisfying conditions (mainly in terms of cost and cycle time).
In spite of the growing interest of the community and industrialists, there is no standard (benchmark) for this concurrent problem.

In this article, we propose a meta-model of the whole problem (configuration, planning and coupling) which will be used for a theoretical investigation. We also propose to generate representative instances of the problem. By representative, we mean both:

- Representative of the diversity that could be obtained by theoretical investigation of the meta-model
- Representative of the diversity of industrial existing cases (models and decision aiding process); especially for the configuration part due to its diversity.

Therefore, the paper is organized as follow. The next section details the problem and its combinatorial aspect. The third section proposes first elements relevant to a meta-model of the benchmark tool. Some elements associated with cases diversity are discussed.

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

2 Addressed problem

For our benchmark, the addressed problem is limited to the coupling between product configuration and project planning. We will describe both environments and the coupling of them in next sub-sections.

2.1 Concurrent configuration and planning

Product configuration problem is a multi-domain, multidisciplinary, multiobjective problem [Viswanathan and Linsey, 2014], [Tumer and Lewis, 2014]. That generates a wide diversity of possible models to represent. We will try to define a classification of existing product models and modelize it in the proposed meta-model. Planning problems are generally more framed (e.g. temporal precedence, resources consumption, cycle time or delay, etc.). To generate various problem instances we can act on the shape of the project graph and on the dispersal of the values assigned for the resources of tasks (cost, cycle time, etc.). Thus, we need to define in our meta-model of the product / project a kind of generic model for each part and for the coupling. The aim of the next step of our study will be to analyze industrial cases and to define this generic model.

Many authors, since [Mittal and Frayman, 1989], [Soininen et al., 1998], [Aldanondo et al., 2008] or [Hofstedt and Schneeweiss, 2011] have defined configuration as the task of deriving the definition of a specific or customized product (through a set of properties, sub-assemblies or bill of materials, etc…) from a generic product or a product family, while taking into account specific customer requirements. Some authors, like [Schierholt 2001], [Bartak et al., 2010] or [Zhang et al. 2013] have shown that the same kind of reasoning process can be considered for production process planning. They therefore consider that deriving a specific production plan (operations, resources to be used, etc...) from some kind of generic process plan while respecting product characteristics and customer requirements, can define production planning. More and more studies tackle the coupling of both environment [Baxter, D. et al., 2007] [Zhang et al. 2013] [Hong et al., 2010] or [Li et al., 2006], [Huang and Gu, 2006]. Many configuration and planning studies (see for example [Junker, 2006] or [Laborie, 2003]) have shown that each problem could be successfully considered as a constraint satisfaction problem (CSP). CSP's are also widely used by industrials [Kaiser et al., 2000]. Considering that using a CSP representation, we could both represent constrained and unconstrained problems, we will use it to represent each environment and the coupling.

2.2 Combinatorial optimization problem

In previous concurrent model, some variables represent decisions of the user (customer or decision-maker on product or project environment). We assume that those decision variables are all discrete variables, so that an instantiation of all these decisions variables corresponds to a particular product / project. Indeed in reality and regardless of the environment, decisions correspond to choices between various combinations. In product environment, decisions correspond to architectural choices between various combinations of sub-systems, or to a choice among various variants for every sub-system. In project environment, decisions correspond to resources choices between various variants.
Combinatorial constrained optimization problems consist in a search of a combination of all decision variables that respects constraints of the problem [Mezura-Montes and Coello Coello, 2011]. Instantiation of every decision variable in CSP model corresponds to a specific product/project which could be analyzed and scored according user's multiple preferences or objectives (cost, delay, etc.). As those objectives could be antagonist, algorithm has to find in a short time a set of approximately efficient solutions that will allow the decision maker to choose a good compromise solution. Using Pareto dominance concept, the optimal set of solutions searched is called the optimal Pareto front.
This allows us to define a multiobjective combinatorial constrained optimization problem: a search between various combinations to find a selection of solutions which are the closest possible of the optimal Pareto front.

3 Meta-model description

This part aims at present the first elements relevant to a meta-model of a concurrent configuration and planning problem which will be used to generate data on benchmark.

3.1 Constrained optimization problem

The constrained optimization problem (O-CSP) is defined by the quadruplet <V, D, C, f > where V is the set of decision variables, D the set of domains linked to the variables of V, C the set of constraints on variables of V and f the multi-valued fitness function. The set V gathers: the product variables and the process variables (we assume that duration process variables are deduced from product and resource). In our meta-model, we define two kind of variable: description variables and decision variables. The first ones could be discrete or continuous and allow description of the problem in each environment. On other hand, the decision variables are all discrete, that thus define the combinatorial optimization problem to solve. Those variables, linked by various constraints, describe product and project. In product side, we consider that a generic product can be described by a set of properties or a set of components or a mix of both as proposed in [Aldanondo et al., 2008]. Product description variables can be associated with product properties or component type. The definition domains of these variables are either symbols (for example: type of finish…) or discrete numbers (for example: flight range…). The configuration constraints that link these variables show the allowed com-

binations of variable values. On figure 1, we represent various groups of variables. It illustrates both the fact that a system is composed of multiple sub-systems, and also that the system and its components are analyzed according to several points of view from various disciplines. Finally, each description variable can have an influence on the product cost and can be therefore associated with a cost variable defined on a real domain.

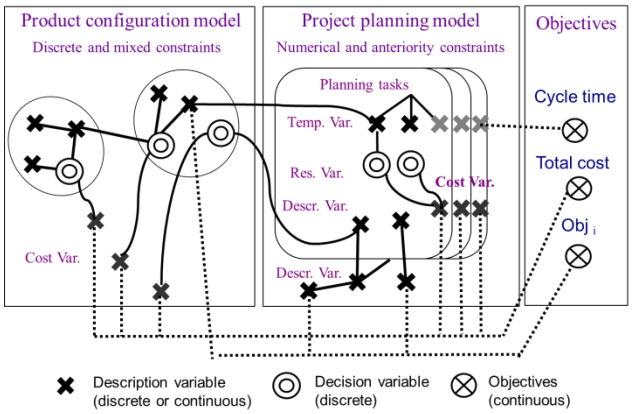


Figure 1 – Meta-model of the Constrained optimization problem

On project side, we consider that a generic production process can be described with a set of planning operations (supplying, manufacturing, assembling…) linked with anteriority constraints. Each operation is defined with:

- Three operation temporal variables: possible starting time, possible finish time, possible duration, defined on a real domain,
- Two operation resource variables: required resource, defined on a symbolic domain, quantity of resource, defined on integer domain.

Planning constraints link temporal variables in order to represent temporal precedence. Resources description variables can influence the production process cost and thus are linked to cost variable.

The coupling materializes by some coupling constraints that link at least one variable of the configuration model with at least one variable of the planning model. In terms of objective variable, the global cost can be defined as the sum of all product cost and operation cost variables. The global cycle time corresponds with the earliest possible finishing time of the last operation of the production process. The definition of these coupling constraints completes the model and allows the representation in figure 1 of the global constraint model associating configuration and planning.

3.2 Structural analysis

To be able to generate various problems, we analyze the meta-model structure, e.g. relations between variables. It is necessary to describe the types of relations ("pattern") existing between variables in every environment (product / project / coupling). Each of these environments corresponds to a subset of continuous or discrete variables connected by constraints. To generate various models, we can act on the number of variables, on theirs domains or on their relations (constraints). Every variable possesses a domain gathering the set of the values or the possible intervals for this variable. Combinatorial problems stem from cartesian product of every domain of decision variables. A first variation would be obviously the number of variables and the average number of states by variable. For a given complexity, we could also evaluate impact of a few number of variables with large domains or the opposite.

We can also generate diversity by acting on constraints: constraints density, number and kind of constraints. These variations will allow generating models more or less difficult to solve, especially because they define the ratio between feasible and unfeasible solutions and thus the difficulty of the search.

Finally, we can act on distribution of the values affected to each state for each variable involved in evaluation of objectives. For example, it concerns acting on the costs and the performances of components or on the costs and durations of project tasks. This will allow us to act on the density of solutions in the search space.

3.3 Problem specific analysis

3.3.1 Product environment

Product environment is a multi-domain, multidisciplinary and thus multiobjective context. In meta-model, product configuration model corresponds to a description of relation between architectural or components choices represented by decision variables. Each domain or discipline describes its own point of view of the product and its decomposition using constraints. Their analysis could take into account some context description variables. The result is a fragmented model stemming from the aggregation of these analyses all connected with the decision variables.

For the objective aspect, every configuration model takes into account cost dimension. Other objective could also appear like technical performance, environmental impact, etc. For cost aspect, we expect that at least a cost variable is linked (directly or not) to each component choice.

Concerning the distribution of values that allows to calculate objective satisfaction, we assume that the model has to be balanced in order: (i) to be an interesting optimization problem to solve and (ii) to be representative of real problems. For the optimization aspect, if an option is systematically better than others, the optimization problem will not be very hard to solve. Furthermore, it corresponds to a better description of the reality where that kind of option will not be conserved in the catalog.

Relations between variables and distribution of values are generally consistent at elemental level, e.g. considering and

analyzing only few variables using a specific point of view. Indeed in realty, option choices are generally coherent; in the sense that existence of each option is justified by differences with other options and those differences generally correspond to an application of some basic relations or behaviors. We identify four kind of basic behavior between two variables:

- Positively correlated: the increase of the one leads to the increase of other one. For example, performing components will be more expansive.
- Negatively correlated: the increase of the one leads to the decrease of other one. For example, components with low environmental impact will be more expansive.
- Aggregation: values of a variable are summation of values of some others variable. For example, global product cost is summation of every component costs.
- Compatibility/incompatibility of some combinations of values: some values of different variables will be incompatible.

Effects of a positive or a negative correlation aren't necessarily linear but this study will be limited to linear interactions. Figure 2 shows possible linear correlations between two variables. Of course, extension dealing with three, four or five variables will be considerate as for example flight range, flying speed, seat capacity and cost.

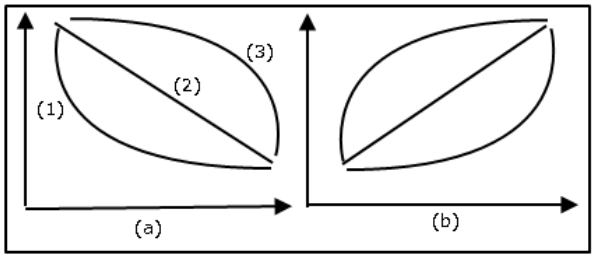


Figure 2 – Negative (a) and positive (b) correlations with three possible case: (1) reducer, (2) linear, (3) amplifier.

It is the accumulation of a large number of simple and sometimes conflicting elementary behaviors that gives its complexity to the problem. Furthermore, real problems also show some additional singularities on elementary level (for example, a high performing solution for a component) or at system level (for example, the choice of a standard configuration, e.g. a selection of standard components, could lead to an important discount).

3.3.2 Project environment

On project side, meta-model is more framed on its diversity:

- project is a set of task to achieve,
- tasks are linked by chronological and precedence constraints,
- tasks are described by some temporal description variable (duration, beginning, end) and some variables that represent resource choices.

On this side, decision variable are the resource choices (make, buy or make by subcontract decision). In this same way as in product side, the different options for each resource choice are going to differ with regard to the objectives. For example considering cost and duration objective, a cost and duration could be assigned to each resource choice, then total cost is obtained by a summation and project cycle time by a constraint propagation on temporal constraints.

As in product side, values distribution between various resource choices has to be balanced and consistent in order to represent real problems. We must unsure there is no useless or dominant option and value distributions must represent accumulation of some basic behavior. Here for example, we expect that there is a positive correlation between cost and quantity/quality of resources or a negative correlation between duration and quantity/quality of resources. Except these particular aspects, project environment can contain other description variables and other objectives connected with decision variables.

4 Conclusion

The goal of this paper was to present our research perspectives for a benchmark on concurrent configuration and planning. This problem is more and more studied. Although there are a lot of cases of Knowledge-based configuration systems applied on the industrial practice and project planning, there is a real lack of real-word inspired benchmark. In this study, we propose the first elements of a meta-model that can represent this diversity and that will allow to generate various test models for our benchmark goal.

References

[Auger and Ros, 2009] Anne Auger and Raymond Ros. Benchmarking the pure random search on the BBOB-2009 testbed. In Franz Rothlauf, editor, GECCO, pp. 2479–2484. ACM, (2009)

[Aldanondo et al., 2008] M. Aldanondo, E. Vareilles. Configuration for mass customization: how to extend product configuration towards requirements and process configuration, Journal of Intelligent Manufacturing, vol. 19 n° 5, pp. 521-535A (2008)

[Amilhastre et al, 2002] J. Amilhastre, H. Fargier, P. Marquis, Consistency restoration and explanations in dynamic csps - application to configuration, in: Artificial Intelligence vol.135, pp. 199-234, (2002)

[Bartak et al., 2010] R. Barták, M. Salido, F. Rossi. Constraint satisfaction techniques in planning and scheduling, in: Journal of Intelligent Manufacturing, vol. 21, n°1, pp. 5-15 (2010)

[Baxter, D., 2007] Baxter, D. An engineering design knowledge reuse methodology using process modelling.

Research in Engineering Design, 18 (1) pp. 37-48, (2007)

[Domes et al., 2014] F. Domes, M. Fuchs, H. Schichl and A. Neumaier. The Optimization Test Environment, Optimization and Engineering, vol. 15, pp. 443–468, (2014)

[Gilbert and Jonsson, 2009] J.C. Gilbert and X. Jonsson. LIBOPT - An environment for testing solvers on heterogeneous collections of problems - The manual, version 2.1. Technical Report RT-331, INRIA, (2009)

[config, 2013/2014] workshops on configuration : 2013 : http://ws-config-2013.mines-albi.fr/, 2014 : http://confws.ist.tugraz.at/ConfigurationWorkshop2014/

[Han and Lee, 2011] S.Han , J. Lee. Knowledge-based configuration design of a train bogie. Journal of Mechanical Science and Technology. Volume 24, Issue 12, pp 2503-2510, (2011)

[Hofstedt and Schneeweiss, 2011]. P. Hofstedt, D. Schneeweiss. FdConfig: A Constraint-Based Interactive Product Configurator. 19th International Conference on Applications of Declarative Programming and Knowledge Management, (2011)

[Huang and Gu, 2006] Huang, H.-Z. and Gu, Y.-K., Development mode based on integration of product models and process models. Concurrent Engineering: Research and Applications. 14, 1. 27-34, (2006)

[Hong et al., 2010] G. Hong, D. Xue, Y. Tu,, Rapid identification of the optimal product configuration and its parameters based on customer-centric product modeling for one-of-a-kind production, in: Computers in Industry Vol.61 n°3, pp. 270–279, (2010)

[Jensen, 2005] Jensen, R., Lars, S.: Power Supply Restoration, Master's thesis, IT University of Copenhagen, (2005)

[Junker, 2006] U. Junker. Handbook of Constraint Programming, Elsevier, chap. 24, Configuration, pp. 835-875 (2006)

[Kaiser et al., 2003] A. Kaiser, K. Wolfgang, S. Carsten. Formal methods for the validation of automotive product configuration data. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 17(2), April Special Issue on configuration. (2003)

[Laborie, 2003] P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results, in: Artificial Intelligence vol 143, 2003, pp 151-188.

[Li et al., 2006] L. Li, L. Chen, Z. Huang, Y. Zhong, Product configuration optimization using a multiobjective GA, in: I.J. of Adv. Manufacturing Technology vol. 30, 2006, pp. 20-29.

[Mezura-Montes and Coello Coello 2011] E. Mezura-Montes, C. Coello Coello, Constraint-Handling in Nature-Inspired Numerical Optimization: Past, Present and Future, in: Swarm and Evolutionary Computation, Vol. 1 n°4, 2011, pp. 173-194

[Mittal and Frayman, 1989] S. Mittal, F. Frayman. Towards a generic model of configuration tasks, proc of IJCAI, p. 1395-1401(1989)

[Mittelmann , 2009] H. Mittelmann, Benchmarks, http://plato.asu.edu/sub/benchm.html, 2009.

[Pàl et al., 2012] László Pál, Tibor Csendes, Mihály Csaba Markót, and Arnold Neumaier Black Box Optimization Benchmarking of the GLOBAL Method, , Evolutionary Computation, Vol. 20, No. 4 , pp. 609-639, (2012)

[Pitiot et al., 2013] P. Pitiot, M. Aldanondo, E. Vareilles, P. Gaborit, M. Djefel, S. Carbonnel, Concurrent product configuration and process planning, towards an approach combining interactivity and optimality, in: I.J. of Production Research Vol. 51 n°2, 2013 , pp. 524-541.

[Schierholt 2001] K. Schierholt. Process configuration: combining the principles of product configuration and process planning AI EDAM / Volume 15 / Issue 05 / novembre 2001 , pp 411-424

[Shcherbina, 2009] O. Shcherbina, COCONUT benchmark, http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html, 2009.

[Shcherbina et al., 2003] O. Shcherbina, A. Neumaier, Djamila Sam-Haroud, Xuan-Ha Vu and Tuan-Viet Nguyen, Benchmarking global optimization and constraint satisfaction codes, pp.211--222 in: Ch. Bliek, Ch. Jermann and A. Neumaier (eds.), Global Optimization and Constraint Satisfaction, Springer, Berlin 2003.

[Sinz et al., 2003] Sinz, C., Kaiser, A., Küchlin, W.: Formal methods for the validation of automotive product configuration data. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 17, 2003, pp.75-97.

[Soininen et al., 1998] T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen, Towards a General Ontology of Configuration., in: Artificial Intelligence for Engineering Design, Analysis and Manufacturing vol 12 n°4, 1998, pp. 357–372.

[Subbarayan, 2006] http://www.itu.dk/research/cla/externals/clib/, 2006.

[Tumer and Lewis, 2014] I. Tumer and K. Lewis. Design of complex engineered systems. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 28, pp 307-309. 2014.

[Viswanathan and Linsey, 2014] Vimal Viswanathan and Julie Linsey, Spanning the complexity chasm: A research approach to move from simple to complex engineering systems. AI EDAM 28(4): pp. 369-384, 2014.

[Kaiser et al., 2000] A. Kaiser, K. Wolfgang, S. Carsten. Proving consistency assertions for automotive product data management. J. Automated Reasoning, 24(1-2):145-163, February 2000.

[Zhang et al., 2013] L. Zhang, E. Vareilles, M. Aldanondo. Generic bill of functions, materials, and operations for SAP2 configuration, in: I.J. of Production Research Vol. 51 n°2, pp. 465-478, (2013).

Different Solving Strategies on PBO Problems from Automotive Industry

Thore Kübart [1] and **Rouven Walter** [2] and **Wolfgang Küchlin**[2]

Abstract. SAT solvers have proved to be very efficient in verifying the correctness of automotive product documentations. However, in many applications a car configuration has to be optimized with respect to a given objective function prioritizing the selectable product components. Typical applications include the generation of predictive configurations for production planning and the reconfiguration of non-constructible customer orders. So far, the successful application of core guided MaxSAT solvers and ILP-based solvers like CPLEX have been described in literature. In this paper, we consider the linear search performed by DPLL-based PBO solvers as a third solution approach. The aim is to understand the capabilities of each of the three approaches and to identify the most suitable approach for different application cases. Therefore we investigate real-world benchmarks which we derived from the product description of a major German premium car manufacturer. Results show that under certain circumstances DPLL-based PBO solvers are clearly the better alternative to the two other approaches.

1 Introduction

An already well-established approach in the automotive industry is to describe the set M of technically feasible vehicle configurations by a propositional formula φ such that $M = \{\tau \mid \tau(\varphi) = 1\}$ holds, where τ is a satisfying assignment of formula φ [11, 19], i.e. every model of φ is a feasible configuration. SAT solvers are the method of choice for calculating configurations that comprise certain options $o_1, \ldots, o_m$: A model has to be determined that satisfies the formula $\varphi \wedge \bigwedge_{i=1}^{m} o_i$. If there is no such model of the desired configuration, the user is often interested in an alternative model of optimal configured options o_i with respect to given priorities w_i. To reach a best possible configuration, a model of the formula φ has to be calculated that optimizes the target function $\sum_{i=1}^{m} w_i o_i$.

In the literature of the last few years different applications of this optimization problem are described as well as several approaches to solve it. Similar optimization problems arise for example from the task to minimize or maximize product properties such as price or weight [21]. Another example is the task of optimal reconfiguration, e.g., the selected options for a car are not feasible with the constraint set [21]. Furthermore valid configurations that optimize linear objective functions play an important role in demand forecasts [18]. We compare the underlying solving approaches by analyzing real-world instances of a major German premium car manufacturer.

This work is organized as follows: Section 2 introduces the basics of *Propositional Logic*, *Maximum Satisfiability* (MaxSAT), *Pseudo-Boolean Optimization* (PBO) and *Integer Linear Programming* (ILP) and their respective algorithmic solving techniques. Section 3 points out related work. Section 4 describes different optimization problems in automotive configuration. Section 5 presents a detailed evaluation of the different introduced optimization approaches for these problems including a discussion of the results. Finally, Section 6 concludes this work.

[1] Steinbeis-Transferzentrum Objekt- und Internet-Technologien, Sand 13, 72076 Tübingen, Germany

[2] Symbolic Computation Group, WSI Informatics, Universität Tübingen, Germany, www-sr.informatik.uni-tuebingen.de

2 Preliminaries

In this work, we focus on propositional logic with the standard logical operators $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ over the set of Boolean variables X and with the constants $\bot$ and $\top$, representing false and true, respectively. The set of variables of a formula φ is denoted by $\mathrm{var}(\varphi)$. A formula φ is called *satisfiable*, if and only if there is an *assignment* τ, a mapping from the set of Boolean variables $\mathrm{var}(\varphi)$ to $\{0, 1\}$, under which the formula φ evaluates to 1. The evaluation of a formula under an assignment τ is the standard evaluation procedure for propositional logic, denoted by $\tau(\varphi)$. The values 0 and 1 are also referred to as *false* and *true*. The well-known NP-complete *SAT* problem asks the question whether a formula is satisfiable or not [6].

The established input format of a SAT solver nowadays is a *conjunctive normal form* (CNF) of a formula φ, where φ is transformed into a conjunction of *clauses* and each clause is a disjunction of *literals* (variables or negated variables). The variable of a literal l is denoted by $\mathrm{var}(l)$. For a formula $\varphi = \bigwedge_{i=1}^{k} \bigvee_{j=1}^{m_i} l_{i,j}$ in CNF we also make use of the notation of φ as a set of clauses where each clause is a set of literals: $\varphi = \{\{l_{1,1}, \ldots, l_{1,m_1}\}, \ldots, \{l_{k,1}, \ldots, l_{k,m_k}\}\}$. The transformation of an arbitrary formula φ into a CNF is done by a Tseitin- or Plaisted-Greenbaum-Transformation [16, 20] (denoted as $\mathrm{Tseitin}(\varphi)$). The resulting formula is not semantically equivalent, but equisatisfiable. Also, the models of φ and $\mathrm{Tseitin}(\varphi)$ are the same when restricted to the original variables $\mathrm{var}(\varphi)$.

2.1 MaxSAT

For a given clause set $\varphi = \{c_1, \ldots, c_m\}$, $m \in \mathbb{N}$, the *MaxSAT* problem [13] asks for the maximum number of clauses which can be simultaneously satisfied:

$$\mathrm{MaxSAT}(\varphi) = \max \left\{ \sum_{i=1}^{m} \tau(c_i) \;\middle|\; \tau \in \{0,1\}^{|\mathrm{var}(\varphi)|} \right\} \quad (1)$$

The corresponding problem of finding the minimum number of clauses which can be simultaneously unsatisfied is called MinUNSAT.

Juha Tiihonen, Andreas Falkner, and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

The *partial weighted MaxSAT* problem is an extended version where: (i) An additional clause set φ_{hard} of *hard* clauses is taken into account, which has to be satisfied, and (ii) weights $w_i \in \mathbb{N}$ are assigned to the *soft* clauses of $\varphi = \{c_1, \ldots, c_m\}$. The resulting problem, $\text{PWMaxSAT}(\varphi_{\text{hard}}, \varphi)$, consists of finding the maximal sum of weights of satisfied clauses of φ while satisfying φ_{hard}. To simplify reading we refer to partial weighted MaxSAT just as MaxSAT in the rest of this work.

MaxSAT can be solved by using a SAT solver as a black box, e.g. by linear search or binary search. Firstly, a fresh variable b_i, called *blocking variable*, is added to each soft clause, which serves to enable or disable the clause. Linear search iteratively checks the SAT instance φ_{hard} and φ with an additional constraint

$$\text{CNF}\left(\sum_{i=1}^{m} w_i \cdot \neg b_i > k\right), \tag{2}$$

where initially $k = 0$. With this check we search for a model with a sum of weights of at least 1. The constraint $\sum_{i=1}^{m} w_i \cdot \neg b_i > k$ is a Pseudo-Boolean constraint (see Subsection 2.2 for details), which can be transformed to a CNF, see for example [4, 8]. The degree k is increased to the sum of weights of the last model plus one in order to check if we can find a better model. Binary search, in contrast, follows the same scheme but restricts the search space with a lower and an upper bound simultaneously. Linear search requires m SAT calls in the worst case, whereas binary search requires only $\log_2(m)$ SAT calls in the worst case.

Another approach is the usage of unsatisfiable cores delivered by a SAT solver for the unsatisfiable case, which was introduced in [3, 9]. The idea is to iteratively call the SAT solver and relax the soft clauses contained in the unsatisfiable core by introducing blocking variables until the formula becomes satisfiable. Solvers using an unsatisfiable core approach performed well on industrial instances in recent MaxSAT competitions[3].

The OPEN-WBO framework [15] is based on using MiniSat-like solvers [7] and was one of the best performing MaxSAT algorithms on industrial instances in the recent MaxSAT competition. Both linear search and unsatisfiable-core guided solvers are included in different variations within the OPEN-WBO framework. The default solver, called WBO, is an unsatisfiable-core guided modification of [3] which partitions the soft clauses [2, 14]. Therefore, only a subset of the soft clauses are given to the SAT solver to make the SAT solver focus on relevant clauses. On the other hand, this can lead to additional SAT calls in the case where a model is found but not all soft clauses were considered. We used this solver for our evaluations, see Section 5.

2.2 DPLL-based PBO

In addition to clauses we consider linear pseudo-Boolean (LPB) constraints, which are linear inequalities of the form

$$\sum_{i=1}^{k} a_i l_i \triangleright b, \qquad \triangleright \in \{<, \leq, >, \geq, =\}, \tag{3}$$

where $a_i \in \mathbb{Z}$ and l_i are literals of Boolean variables. Under some assignment τ, the left side is the sum over the coefficients of the satisfied literals and the LPB constraint is satisfied iff the respective inequality holds. For example, clauses $\bigvee_{i=1}^{m} l_i$ can be generalized as LPB constraints $\sum_{i=1}^{m} l_i \geq 1$.

[3] http://www.maxsat.udl.cat/

Pseudo-Boolean solving (PBS) is the decision problem whether a set of LPB constraints can be satisfied by an assignment τ. Hence, PBS is a generalization of SAT. Like SAT solvers, most PBS solvers which prove satisfiability are able to provide a satisfying assignment τ to the user.

Given a satisfiable set of LPB constraints another problem is to identify a best possible assignment τ with respect to a linear objective function:

$$\begin{array}{ll} \min & \sum_i c_i l_i \\ \text{s.t.} & \bigwedge_j \left[\sum_i a_{j,i} l_{j,i} \triangleright b_j\right] \end{array} \tag{4}$$

This problem is called pseudo-Boolean optimization (PBO) [17].

In order to solve the satisifiability problem PBS, DPLL-style algorithms can be used to benefit from recent progress of modern SAT solvers. One approach is to transform the LPB-constraints into CNF and to apply SAT solvers to the resulting formula. Another approach is based on generalized constraint propagation and conflict-based learning, i. e. DPLL-based SAT solvers are enabled to handle LPB constraints directly. Generally learning methods analyze the conflict and learn a new constraint which is falsified on the conflict level and which propagates a new assignment on a higher decision level.

Given a PBS solver, the PBO problem of Formula (4) itself can be solved by iteratively applying the solver to perform a linear search or a binary search. Both approaches proceed analogously to the linear and binary search approaches for MaxSAT using a SAT solver.

In the linear search approach, models of the formula φ are calculated in order to gradually approach the optimal objective value. Through an extra LPB constraint, a model providing a better objective value is enforced. If the extra LPB constraint leads to an unsatisfiable PBS instance, the model last calculated is the optimal one.

In the binary search approach the search space is bisected by every single PBS-instance. If the optimal objective value lies inside the interval $[L, U]$, a model with the objective value $\leq M = (L+U)/2$ is searched for. If such a model exists, the minimal objective value lies inside $[L, M]$. If no such model exists, the minimal objective value lies inside $[M, U]$.

For the calculations in Section 5 we used the Sat4j library. The library is based on a Java version of MiniSat, which was expanded by generalized constraint propagation and conflict-based learning. The PBO solver contained therein realizes a simple linear search. The PBS solver called for this linear search is able to perform the following two learning methods.

The *clause-based learning method* calculates so-called UIPs (unique implications points) by means of propositional resolution just like in modern SAT solvers. SAT solvers such as MiniSat derive a UIP on the basis of the conflict clause and the reason clauses which were propagating the assignments of the conflict clause. If a conflict constraint occurs in the form of a LPB constraint, Sat4j first reduces this constraint to a conflict clause

$$K = \bigvee_{l \in \omega_C,\ \beta(l)=0} l, \tag{5}$$

where β is the partial assignment, that leads to the conflict in the LPB constraint ω_C. Analogously, reasons given by LPB constraints are reduced to reason clauses: If ω is a LPB constraint, that propagates $\tilde{l}$ under the partial assignment β, the clause

$$R(\tilde{l}) = \bigvee_{l \in \omega,\ \beta(l)=0} l \vee \tilde{l},\ \omega \models R(\tilde{l}), \tag{6}$$

is an implication of ω and also propagates the literal $\tilde{l}$ under β.

In a second learning method that is implemented in Sat4j more expressive LPB constraints instead of clauses are learned. For this purpose the principle of propositional resolutions in forms of Hookers cutting planes [10] is directly applied to the LBP constraints [5].

2.3 Integer Linear Programming

Like linear programming, integer linear programming deals with the optimization of linear objective functions over a set which is limited by linear equations and inequations. The difference is that while in linear optimizations any real values can be taken on, in integer optimization some or all variables are restricted to whole-number values.

$$\begin{array}{ll} \min & \sum_i c_i x_i \\ \text{s.t.} & \\ & \bigwedge_j \left[\sum_i a_{j,i} x_{j,i} \triangleright b_j\right] \\ & x_{j,i} \in \mathbb{Z} \end{array} \tag{7}$$

Pseudo-Boolean Optimization, see Formula (4), can be easily transformed into 0-1 integer linear programming (ILP): Negative literals $\neg x$ are replaced by $(1 - x)$ and Boolean variables become decision variables $x \in \{0, 1\}$. Consequently, commercially available state-of-the-art ILP solvers such as CPLEX [1] can be used to solve PBO. Usually, they are based on the branch and cut strategy.

3 Related Work

Sinz et al. suggest a SAT-based procedure to check consistency of the product documentation of a German car manufacturer [19]. Therefore, they consider vehicle configurations as assignments to propositional variables and define a propositional formula, called product overview formula (POF), which evaluates to true iff the vehicle configuration is technically feasible. Their work also lays the foundation for formulating the restrictions of the product documentation as a constraint in mathematical models.

Tilak Singh et al. make use of such a product overview formula in [18]. They describe a mathematical model to calculate car configurations aimed at providing the production planning with test variants before actual sales orders are received. Their configurations are calculated on the basis of PBO problems that are solved by CPLEX. This approach is described in greater detail in Section 4.

Walter et al. [21] describe the possible usage of MaxSAT in automotive configuration by pointing out various use cases. Whenever faced with an over-constrained configuration, MaxSAT can be used to reconfigure the invalid configuration by providing an optimal solution, e.g. giving a repair suggestion of the (possibly prioritized) over-constrained selections of a user by a minimal number of changes.

4 Optimization Problems from Automotive Configuration

The product overview defines the set of valid product configurations and is usually describable as a set of propositional constraints. This means a configuration is only valid if it satisfies the conjunction of all constraints, which is also called the product overview formula (POF). The physical demand of building components for a valid configuration is determined by the bill of materials (BOM). The combination of the product overview and the bill of material is referred to as the product documentation.

Reconfiguration of invalid configurations, as described in [21], is an important issue in automotive configuration. Several practical relevant use cases exist, such as the reconfiguration of customer orders, the reconfiguration of constraints for a given fixed order or the computation of a maximal/minimal car w.r.t. to an assigned value of the options like weights (kg).

Another major task is the prognosis of future part demands in production planning. However, historical demands cannot easily be extrapolated because planning situations in the automotive industry are constantly changing. A typical planning state used for making a forecast comprises

- the product documentation (option A is only available, if option B is selected; part x is needed exactly iff the order satisfies the part rule φ_x; etc.),
- estimated customer buying behavior (option C is selected by 30% of the customers; etc.),
- capacity restrictions (only 5000 units of part X are available; etc.),
- production plans that fix the total number of planned vehicles.

From a mathematical point of view a planning state consists of two parts:

1. The Boolean formula POF, whose models describe the technically feasible configurations,
2. the statistical frequency of certain atoms of the product formula.

A common approach to evaluate the planning state is to calculate an amount of N technically feasible variations that approximates the statistical guidelines as good as possible. Eventually, for indicating the future unit demand, the calculated variations are analyzed by means of the BOM.

Singh et al. [18] propose a linear optimization model for finding such a set of N technically feasible variations. A solution of their model is calculated by *column generation*: In every iteration a technically feasible variation is determined which further improves the solution set. For this, a PBO problem is solved. Its objective function is given by the dual variables of the current approximation and its constraints are given by the restrictions of the product overview.

In column generation, calculated configurations of previous iterations are partially replaced by new configurations. Thus, solved PBO instances do not necessarily result in a configuration also contained in the final solution. Therefore, it is particularly important to ensure an efficient calculation of the individual PBO instances.

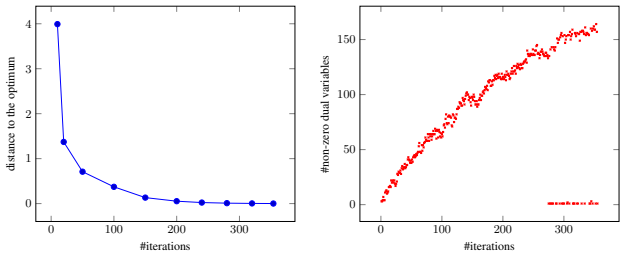


Figure 1. Typical approximation to the optimum using column generation

The typical approximation to the planning state when using column generation is shown in Figure 1. As one can see on the left side, the majority of iterations is used to overcome the last small step towards the optimum. Yet, illustrated by the graph on the right side,

approximation is accompanied by increasingly complex target functions of the PBO instances: the number of non-zero dual variables is increasing. It must be noted that the maximum number of non-zero dual variables is usually proportional to the number of statistical requirements of the planning state.

Concerning implementations of column generation described in the literature, integer linear programming is the method of choice for producing columns. This approach is also taken in [18] using CPLEX. An important objective of this paper is to investigate whether DPLL-based methods are suitable alternatives to CPLEX for calculating predictive configurations.

5 Experimental Evaluation

In this section we present our main contribution by evaluating the different previously described methods on optimization problems from automotive configuration.

As test environment for the experimental evaluations we used Windows 7 Professional 64 Bit on an Intel(R) Core(TM) i7-4800MQ CPU with 2.70 GHz and 2 GB main memory.

5.1 Benchmark Statistics

We evaluate several test series, each of which is composed of a real-world product overview and of randomly generated linear objective functions. We looked at product overviews of two different types, see A and B in Table 1. Both types of product overviews differ in the extent to which attributes are fixed. Additionally, Table 1 shows the minimum and maximum number of clauses for the product overviews of one type. For generating objective functions, n variables were randomly selected (by using Java's Random class with the default constructor) and each was assigned to a random integer coefficient from a range between $-10,000,000$ and $10,000,000$. That way, 10 objective functions were generated for different numbers of variables ($n = 10, 20, \ldots, 200$) so that one test series contains a total of 200 PBO instances. Corresponding test series were generated for 13 different product overviews of type A and for 6 different product overviews of type B.

Type	Fixed Attributes	#Clauses
A	market, model, body type, engine, steering type	1200-5900
B	market, model, body type	19100-137700

Table 1. Characteristics of type A and B

5.2 Results

The test series from the previous section were solved by 3 solvers:

1. OpenWBO as a core guided MaxSat solver [15].
2. Sat4j as an implementation of the DPLL-based linear search [12].
3. CPLEX as an ILP solver [1].

For solving the PBO instances a timeout of 60 seconds was set. In order to set up the linear search correctly, two different learning methods of the underlying PBS solver of Sat4j were tested. Concerning the test series of type A, Figure 2 shows the average running time for learning of

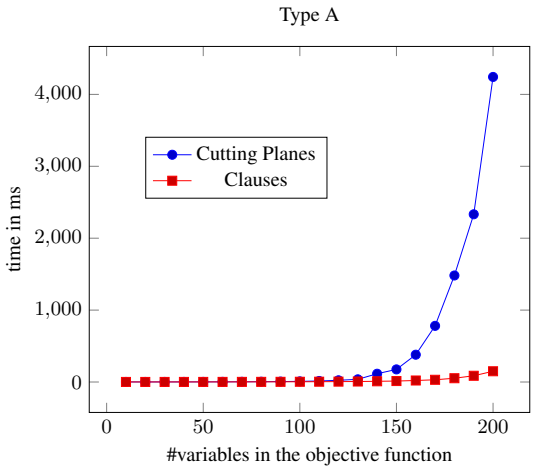


Figure 2. Learning Cutting Planes vs. Clauses in Sat4j

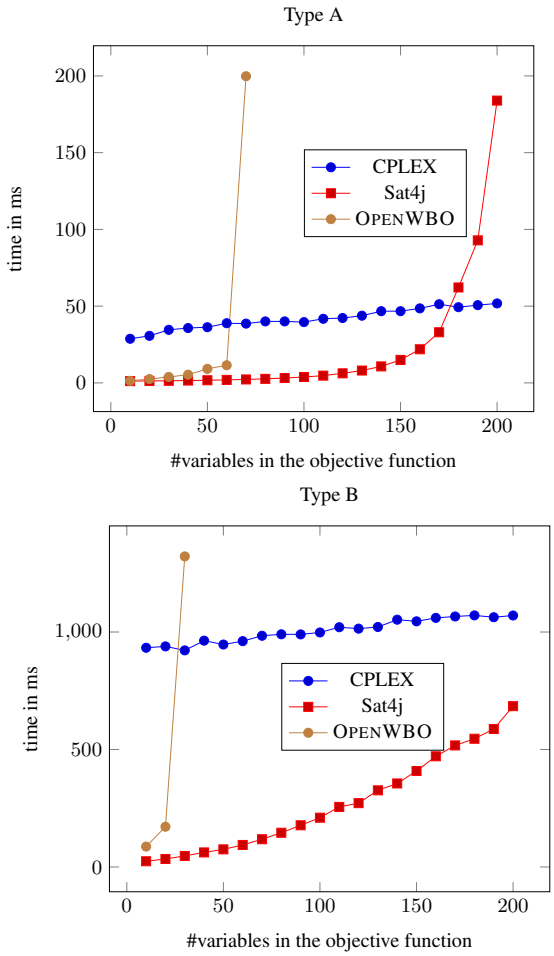


Figure 3. Running times for type A and B

1. clauses
2. cutting planes

depending on the complexity of the objective function. Based on these results, Sat4j was limited to learning clauses when comparing the 3 solvers.

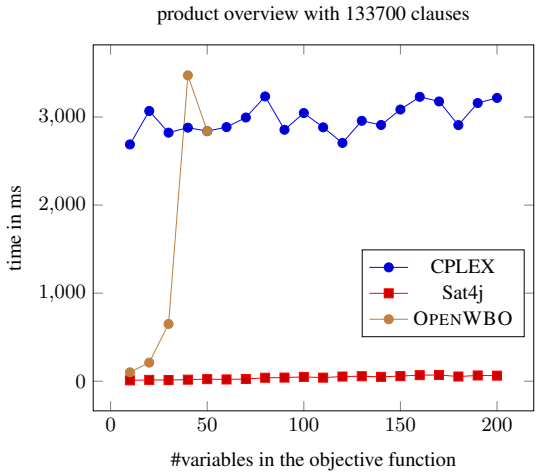


Figure 4. Running times for the largest product overview of type B

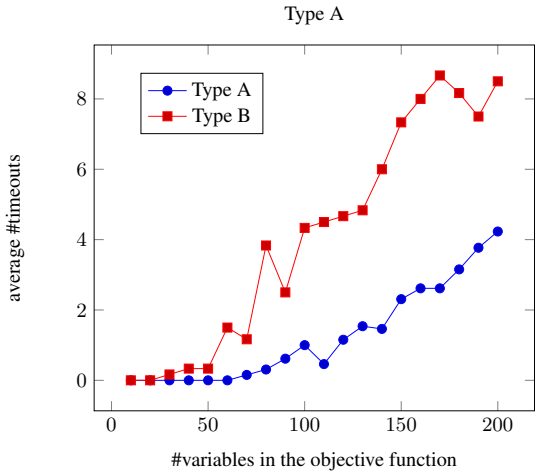


Figure 5. Average number of timeouts for OpenWBO

In Figure 3 the running times of OpenWBO, Sat4j and CPLEX are compared. The upper graph shows the average running times of the test series of type A concerning objective functions of different complexity. Accordingly the bottom graph displays the results of the test series of type B. Time was limited in both subfigures (200ms, 1400ms). In average, for all test series of type A and B, the DPLL-based solvers OpenWBO and Sat4j perform significantly better than CPLEX with regard to objective functions of low complexity. In test series of type A the solver Sat4j performs better than CPLEX concerning objective functions of up to $N = 170$ variables.

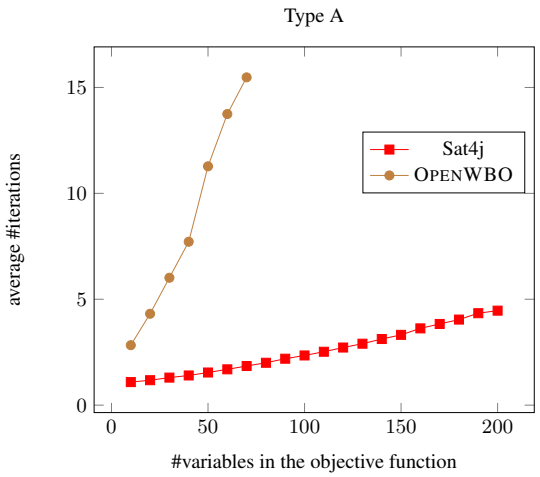


Figure 6. Average number of DPLL-calls for OpenWBO and Sat4j

The MaxSAT solver OpenWBO, however, produces reliable running times for up to $N = 70$ variables.
In contrast to the DPLL-based solvers, there was only a slight increase in the running times for CPLEX with a growing number of variables in the objective function. Hence, once a certain length of the target function has been reached, CPLEX leads to better results.

Across all test series of type B, Sat4j performed on average better than CPLEX (Figure 3, bottom graph). Compared to CPLEX however, OpenWBO leads to slower running times for objective functions of only $N = 30$ variables. This observation is especially illustrated by the most extensive product overview of type B (137700 clauses) as demonstrated in Figure 4, where the time-domain is restricted to 3400 ms.

Timeouts were observed exclusively for the MaxSAT solver OpenWBO. Complementary to the results of running times, Figure 5 shows the average number of timeouts when solving 10 instances.

For a better understanding of the observed difference in running times of both DPLL-based approaches, Figure 6 shows the average number of DPLL-calls. The graphs refer exclusively to the test series of type A and represent the numbers of SAT/UNSAT-calls for OpenWBO or, in case of Sat4j, the numbers of PBS-calls.

5.3 Discussion

The test instances described in Section 5.1 differ with respect to the following aspects:

- the complexity of the objective function (number of variables $N = 10, 20, \ldots, 200$)
- the size of the product overview (number of models and number of clauses; type A and B)

The findings of the previous section lead us to the following conclusions:

- Using DPLL allows the quick calculation of a model of a formula - in that way DPLL-based solvers are superior to ILP solvers.
- Yet, when objective functions become more and more complex, the particular suitability of CPLEX in solving 0-1-optimization problems dominates.

The latter point is evident in the strong increase of running times for the DPLL-based methods once a certain length of objective functions has been reached.

In comparison with OPENWBO, Sat4j leads to significantly better running times. This is also due to the special suitability of the examined instances for a linear search (see Figure 6). The number of iterations for a linear search increases only linearly with the numbers of variables in the objective function - this is a surprising result. Hence, the nonlinear increase of running times can only be explained by the growing complexity of PBS instances in linear search.

Concerning product overviews of type A, the critical length of the objective function for Sat4j is between $N = 160$ and $N = 200$. With increasing scale of the product overview, this critical value shifts upwards. In some cases of the product overviews of type B, the critical value is greater than $N = 200$, such as seen in Figure 4.

In order to optimally configure customer orders, an optimal subset of about 20 options has to be determined. For this purpose the linear search implemented in Sat4j and the core guided MaxSAT solver OPENWBO are more than sufficient.

For calculating predictive configurations for the product planning (see Figure 1) Sat4j can be far more effective than the other tested methods, at least for the first iterations of a column generation based process. Yet it is possible that the critical area of linear search is reached depending on the given number of statistical requirements in the planning state. In such a case configurations should be calculated by CPLEX just like in [18].

6 Conclusion

We compared current state-of-the-art solvers to calculate optimal product configurations of a major German car manufacturer. So far, the use of core guided MaxSAT solvers and ILP solvers like CPLEX for PBO-instances of automotive industry was described in the literature. For the purpose of comparison we additionally applied a DPLL-based linear search.

Results were analyzed with respect to the granularity of the product overview and with respect to complexity of the objective functions. The results show that the investigated approaches have different suitability for different application cases. For reconfiguration the linear search performed by a PBS-Solver is a stronger alternative compared to core guided MaxSAT. For calculating predictive configurations – up to a certain amount of given frequency restrictions – the DPLL-based linear search is even more suitable than CPLEX.

An important result is the small number of iterations observed in linear search. The approach of linear search thus appears to be especially suitable for PBO-instances whose constraints are given by a product overview. For reliable usage of the DPLL-based linear search, also for instances of long objective functions, a customized PBS solver needs to be developed. Such a solver must be able to more efficiently solve PBS instances that are characterized by a set of product overview clauses and one extensive LPB constraint.

REFERENCES

[1] IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html, June 2015.

[2] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy, 'Improving SAT-based weighted MaxSAT solvers', in *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012*, ed., Michela Milano, volume 7514 of *Lecture Notes in Computer Science*, pp. 86–101. Springer, (2012).

[3] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy, 'Solving (weighted) partial MaxSAT through satisfiability testing', in *Theory and Applications of Satisfiability Testing - SAT 2009*, ed., Oliver Kullmann, volume 5584 of *Lecture Notes in Computer Science*, 427–440, Springer Berlin Heidelberg, (2009).

[4] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel, 'A translation of pseudo boolean constraints to SAT', *Journal on Satisfiability, Boolean Modeling and Computation*, **2**(1–4), 191–200, (2006).

[5] Donald Chai and Andreas Kuehlmann, 'A fast pseudo-boolean constraint solver', *IEEE Transactions on CAD of Integrated Circuits and Systems*, **24**(3), 305–317, (2005).

[6] Stephen A. Cook, 'The complexity of theorem-proving procedures', in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pp. 151–158, New York, NY, USA, (1971). ACM.

[7] Niklas Eén and Niklas Sörensson, 'An extensible SAT-solver', in *Theory and Applications of Satisfiability Testing—SAT 2003*, eds., Enrico Giunchiglia and Armando Tacchella, volume 2919 of *Lecture Notes in Computer Science*, 502–518, Springer Berlin Heidelberg, (2004).

[8] Niklas Eén and Niklas Sörensson, 'Translating pseudo-boolean constraints into SAT', *Journal on Satisfiability, Boolean Modeling and Computation*, **2**, 1–26, (2006).

[9] Zhaohui Fu and Sharad Malik, 'On solving the partial MAX-SAT problem', in *Theory and Applications of Satisfiability Testing—SAT 2006*, eds., Armin Biere and Carla P. Gomes, volume 4121 of *Lecture Notes in Computer Science*, 252–265, Springer Berlin Heidelberg, (2006).

[10] John N. Hooker, 'Generalized resolution and cutting planes', *Annals of Operations Research*, **12**(1), 217–239, (1988).

[11] Wolfgang Küchlin and Carsten Sinz, 'Proving consistency assertions for automotive product data management', *Journal of Automated Reasoning*, **24**(1–2), 145–163, (2000).

[12] Daniel Le Berre and Anne Parrain, 'The Sat4j library, release 2.2', *Journal on Satisfiability, Boolean Modeling and Computation*, **7**(2–3), 59–6, (2010).

[13] Chu Min Li and Felip Manyà, 'MaxSAT, hard and soft constraints', in *Handbook of Satisfiability*, eds., Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 19, 613–631, IOS Press, (2009).

[14] Ruben Martins, Vasco M. Manquinho, and Inês Lynce, 'On partitioning for maximum satisfiability', in *ECAI 2012 - 20th European Conference on Artificial Intelligence*, eds., Luc De Raedt, Christian Bessière, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter J. F. Lucas, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pp. 913–914. IOS Press, (2012).

[15] Ruben Martins, Vasco M. Manquinho, and Inês Lynce, 'Open-WBO: A modular MaxSAT solver', in *Theory and Applications of Satisfiability Testing – SAT 2014*, eds., Carsten Sinz and Uwe Egly, volume 8561 of *Lecture Notes in Computer Science*, pp. 438–445. Springer International Publishing, (2014).

[16] David A. Plaisted and Steven Greenbaum, 'A structure-preserving clause form translation', *Journal of Symbolic Computation*, **2**(3), 293–304, (September 1986).

[17] Olivier Roussel and Vasco M. Manquinho, 'Pseudo-boolean and cardinality constraints', in *Handbook of Satisfiability*, eds., Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 22, 695–733, IOS Press, (2009).

[18] Tilak Raj Singh and Narayan Rangaraj, 'Generation of predictive configurations for production planning', in *Proceedings of the 15th International Configuration Workshop*, eds., Michel Aldanondo and Andreas Falkner, pp. 79–86, Vienna, Austria, (August 2013).

[19] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin, 'Formal methods for the validation of automotive product configuration data', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **17**(1), 75–97, (January 2003). Special issue on configuration.

[20] G. S. Tseitin, 'On the complexity of derivations in the propositional calculus', *Studies in Constructive Mathematics and Mathematical Logic*, **Part II**, 115–125, (1968).

[21] Rouven Walter, Christoph Zengler, and Wolfgang Küchlin, 'Applications of MaxSAT in automotive configuration', in *Proceedings of the 15th International Configuration Workshop*, eds., Michel Aldanondo and Andreas Falkner, pp. 21–28, Vienna, Austria, (August 2013).

A Heuristic, Replay-based Approach for Reconfiguration

Alois Haselböck and **Gottfried Schenner** [1]

Abstract. Reconfiguration is an important aspect of industrial product configuration. Once an industrial artefact has been built according to an initial configuration, constant reconfigurations are necessary during its lifetime due to changed requirements or a changed product specification. This reconfigurations should affect as few parts of the running system as possible. Due to the large number of involved components, approaches based on optimization are often not usable in practice. This paper introduces a novel approach for reconfiguration based on a replay heuristic (the product is rebuilt from scratch while trying to use as many decisions from the legacy configuration as possible) and describes its realisation using the standard solving technologies Constraint Satisfaction and Answer Set Programming.

1 INTRODUCTION

Configuration is the task of deriving a valid, complete and purposeful system structure assembled from a set of components [14]. For non-trivial configuration problems, like product configuration, we distinguish the following levels of models (cf. Table 1): the language used to represent and solve the configuration problem (M4), the problem domain model (M3), the problem instance model (M2), and the configuration model (M1).

Table 1. Different levels of models in a configuration application.

M4	Modelling Language	e.g. UML class diagrams and OCL, CSP, ASP, ...
M3	Problem Domain Model	Generic specification of the component catalogue
M2	Problem Instance Model	Requirements specification of a concrete configuration problem
M1	Configuration Model	Solution to M2: a configuration object network

M3 is a generic specification of the problem domain. In an object-oriented environment, this would be the class model. Constraints are usually used to describe the different dependencies and restrictions between the different objects. Such a model M3 defines the space of all the technically feasible configuration solutions. Model M2 is a concrete problem instance, containing specific requirement definitions which are usually formalized in terms of constraints, initial configuration objects and requirement and resource parameters. M2 is based on M3 and uses its language and concepts (M4). Finally, M1 - a configuration - consists of all the instances, their properties and relationships realizing a final system. If this configuration is complete and consistent w.r.t. M3 and M2, M1 is said to be a solution of the given configuration problem.

[1] Siemens AG Österreich, Corporated Technology, Vienna, Austria, {alois.haselboeck, gottfried.schenner}@siemens.com

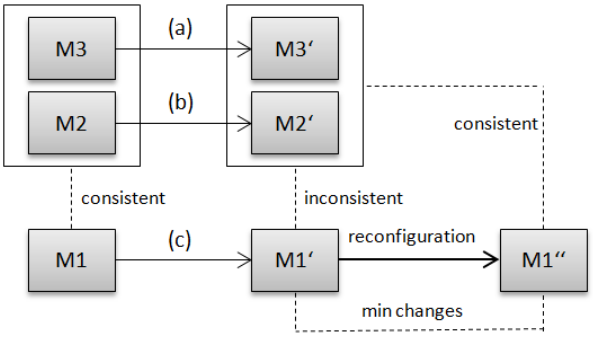


Figure 1. Reconfiguration scenarios.

Reconfiguration is the task of changing a given configuration. Various scenarios are possible why a (partial) configuration becomes inconsistent and reconfiguration is necessary (cf. Figure 1):

(a) The problem domain M3 has been changed. Reasons could be changes in the product catalogue, changes in the structure of the product line, regulation changes, etc. A legacy configuration already installed in the field may be inconsistent now to the new problem domain description and must be reconfigured.
(b) The requirements in M2 has been changed or extended. Again, a legacy configuration which is inconsistent now w.r.t. the changed requirements must be adapted.
(c) A configuration (M1) has been changed by an external process (e.g. by a manual user input or by reading a configuration from an external repository) and is now inconsistent. Again, reconfiguration must find a consistent modification of the configuration.

In all these cases, a crucial demand is that the reconfigured solution is as close as possible to the original configuration. The definition of the quality of a reconfiguration (How close is the new configuration to the legacy configuration?) could get quite subtle. [Friedrich et al., 2011], e.g., use cost functions for the different change operators. Reconfiguration is then the problem of minimizing the overall modification costs. In this paper, we are using a more light-weight approach: We don’t define cost functions but use a rather heuristic and simple definition of minimality: the number of differences between the original and the reconfigured solution should be as small as possible. This corresponds to equal cost values for all types of modifications.

Juha Tiihonen, Andreas Falkner, and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

We present methods how such reconfiguration problems can be modelled and solved by variations of standard, complete solving techniques (like SAT solving or backtracking). A challenge here is that reconfiguration starts with an inconsistent configuration fragment and standard solving (e.g. backtracking) would immediately return a `failure` result. Our idea is to start solving from scratch, but trying to re-build the search tree following the decisions of a given (inconsistent) legacy configuration. That's why we call it *replay-based* reconfiguration. The composition of a reconfiguration will deviate from the legacy configuration in cases where inconsistencies are to be avoided.

Our main contributions in this work are: (1) An Answer Set Programming (ASP) encoding of the reconfiguration problem using the special predicate `_heuristic` of `clingo` (Potassco [12]). (2) A CSP encoding of the reconfiguration problem using a novel value ordering heuristic which prefers value assignments from a legacy configuration. (3) Experimental evaluation and indications of up to which problem sizes these methods are applicable.

The rest of the paper is organized as follows: Section 2 sketches a small hardware configuration problem which will serve as example for the subsequent sections. Section 3 describes how the task of reconfiguration can be modelled and solved in 2 different frameworks: ASP and standard CSP. We compare and evaluated these techniques in Section 4 and conclude the work with a discussion of related works and a conclusion.

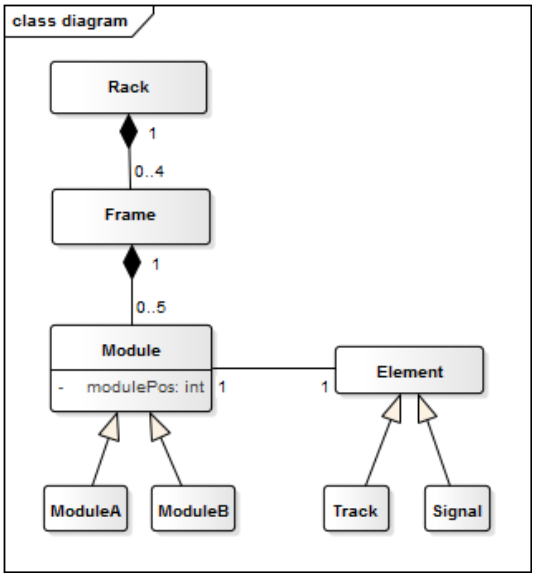


Figure 2. Problem domain model (M3) of a hardware configuration problem example.

2 EXAMPLE

A small example from the domain of railway interlocking hardware should demonstrate the dynamics of the configuration problems we want to model and solve. Of course, real-world problems are much larger and the object network and dependencies between the objects are more complex and varied.

Figure 2 shows the UML diagram and represents the problem domain M3 (cf. Table 1). A part of configuring an interlocking system is to create appropriate control modules for each outdoor element (e.g., a signal or a switch point) and to place them into the right slots of a frame, which in turn must be inserted into a rack. At the beginning, only the outdoor elements are known. Racks, frames and modules must be created during solving. In our example tracks require modules of type 'ModuleA' and signals require modules of type 'ModuleB'.

A concrete problem instance is defined by a set of outdoor elements of different kinds (model level M2). The goal is to find the right set and constellation of racks, frames, and modules, such that each element is connected to exactly one module. Of course, we aim for a minimal set of hardware. Additionally, various types of constraints restrict the allowed constellations. Typical examples of such constraints are: Some types of models should not be mixed within a frame. Certain types of modules must not be mounted on neighbouring places in the frame.

It shall be noted that for a concrete problem instance on model level M2, it is not known beforehand how many racks, frames, and modules are needed for a valid solution on model level M1. This is why such kinds of problems are called *dynamic* problems in contrast to *static* problems.

3 Approach

According to Fig. 1, inputs to the reconfiguration solver are the problem descriptions M3' and M2', and the legacy configuration M1'. Output is M1" which should be consistent w.r.t. M3' and M2' and as close as possible to M1'.

The basic of idea of our reconfiguration approach is to influence a solver to search in the neighbourhood of the legacy configuration. This is achieved by defining a heuristic for the solver to choose for every decision the same value that has been used in the legacy configuration, whenever possible. In a way this reconstruct the search tree which was built creating the legacy configuration. Deviations from that search tree should only happen when previously consistent choices are inconsistent now.

Our approach will perform poorly, if there is no consistent configuration close to an inconsistent legacy configuration. But the approach will perform well if only a small percentage of the overall configuration is modified during reconfiguarion, which is the case in most reconfiguration scenarios in practice especially for large configurations. E.g., from our experience in the rail automation domain, most system modifications are only very local changes in the outdoor installation.

To show that our approach is applicable to different solving paradigms we implemented it in two standard AI technologies, answer set programming (ASP) and constraint satisfaction (CSP).

For the ASP implementation we used Potassco's `clingo` which allows to define domain specific heuristics within ASP with a special predicate `_heuristic`. With this predicate the solver can be influenced to prefer certain atoms during solving. By giving the legacy configuration as preferred heuristic facts we achieve a kind of replay of the original configuration. Details are described in Chap. 3.1.

CSP systems often are more open to adaptations of the search procedure than ASP solvers. For our experiments, we used Choco which allows to plug in your own variable and value ordering heuristics used during backtrack search. We wrote variable and value ordering heuristics which prefer the decisions of the legacy configuration. Details are described in Chap. 3.2.

We chose Potassco and Choco for our experimental implementations because they are well recognized implementations of ASP

and CSP technology. Potassco is very fast and regularly wins competitions. Choco is a standard constraint solver in Java. Most likely there are CSP systems with better performance, but we belief that all solvers based on a backtracking scheme suffer from the same fundamental behaviour of sometimes running into heavy backtracking as shown in our evaluation (cf. Sec. 4).

3.1 Answer set programming

ASP is a declarative problem solving approach, which originated in the area of knowledge representation and reasoning [10]. To implement our approach we used the Potassco ASP implementation [12] and our OOASP framework [16]. OOASP provides special predicates for describing object-oriented knowledge bases in ASP. Our running example can be declared in OOASP as follows:

```
ooasp_class("hw","Rack").
ooasp_class("hw","Frame").
ooasp_class("hw","Module").
ooasp_class("hw","ModuleA").
ooasp_class("hw","ModuleB").
ooasp_class("hw","Element").
ooasp_class("hw","Track").
ooasp_class("hw","Signal").

ooasp_subclass("hw","ModuleA","Module").
ooasp_subclass("hw","ModuleB","Module").
ooasp_subclass("hw","Track","Element").
ooasp_subclass("hw","Signal","Element").

ooasp_assoc("hw","Frame_modules",
    "Frame",1,1,
    "Module",0,5).

ooasp_attribute("hw","Module",
  "position","integer").
ooasp_attribute_minInclusive("hw","Module",
  "position",1).
ooasp_attribute_maxInclusive("hw","Module",
  "position",5).

ooasp_assoc("hw","Module_element",
  "Module",1,1,
  "Element",1,1).
ooasp_assoc("hw","Rack_frames",
  "Rack",1,1,
  "Frame",0,4).
```

In a similar manner the predicates $ooasp\_isa$, $ooasp\_attribute\_value$ and $ooasp\_associated$ are used to define (partial) configurations i.e. instantiations of the object-model.

One standard reasoning task of OOASP is to complete a partial configuration i.e. to add components to the partial configuration until all constraints of the knowledge base are satisfied. For example given a partial configuration consisting only of one track (represented by the fact $ooasp\_isa("c","Track","A1")$), completing a configuration will return all configurations containing one track, with at least one module of type A, one frame and one rack.

3.1.1 Heuristic reconfiguration

Given an inconsistent legacy configuration the default reconfiguration reasoning task in OOASP finds a valid configuration that is cheapest in terms of some user defined cost function. The costs can be either domain-specific or cardinality based. The cardinality based cost function simply counts the difference (in number of facts) of the legacy configuration and the reconfigured configuration. This default reconfiguration task in OOASP is implemented using ASP optimization statements. Unfortunately it has a bad performance for large problem sizes due to the large number of possible configurations. This was one of the motivations for developing a novel approach to reconfiguration with OOASP based on heuristics.

Heuristic reconfiguration for OOASP described in this paper uses the special predicates $\_heuristic(ATOM, TRUTHVALUE, LEVEL)$ from [11] to express heuristics in ASP . If during search $\_heuristic(ATOM, TRUTHVALUE, LEVEL)$ can be derived and LEVEL > 0 then the ASP solver will prefer setting atom ATOM to TRUTHVALUE, if given a choice. With the heuristic predicates the order in which solutions (answer sets) are found can be influenced. It does not affect the set of found solutions.

To implement our heuristic approach we add the facts describing the legacy configuration as heuristics facts $\_heuristic(FACTFROMLEGACY, true, 1)$ to the ASP program and run the default OOASP configuration task with this heuristic information. This way the ASP solver is expected to find configurations that are close (cardinality based) to the legacy configuration first.

For example given the heuristic information below the ASP solver will try to assign track A1 first to the module M1 and set its position to 4.

```
% user supplied fact
ooasp_isa("c","Track","A1")
% legacy configuration converted to heuristic
_heuristic(
  ooasp_isa("c","ModuleA","M1"),
  true,1).
_heuristic(
  ooasp_attribute_value(
    "c",
    "position",
    "M1",4),
  true,1).
_heuristic(
  ooasp_associated(
    "c",
    "Module_element",
    "M1","A1"),
  true,1).
...
```

3.2 Constraint satisfaction

The encoding of a dynamic problem with a standard constraint formalism (like MiniZinc[2] or Choco[3]) makes it necessary to define a maximum number of instances of each object type. If, e.g., we allow at most 5 racks for our example problem in Section 2, the maximum numbers of instances for the other types can be computed by cardinality propagation: 20 frames, 320 modules and 320 elements. For complex networks of classes, this cardinality propagation is not trivial [4].

We briefly sketch here the CSP encoding of configuration problems we used in our implementation. We use a pseudo code notation,

[2] http://www.minizinc.org/
[3] http://choco-solver.org/

which could directly be translated to a concrete CSP notation (e.g. Choco). To represent the instances of a class, we use an array of boolean variables representing which element is actually used in a solution and which not. E.g., let r be that variable array for the class `Rack`. nr be the maximum number of racks.

$$r_i \in \{0,1\}, \quad \forall_i \in \{1,...,nr\}$$

The following symmetry breaking constraints states that unused instances are always in the rear part of the array:

$$r_i = 0 \rightarrow r_j = 0, \quad \forall_{i,j} \in \{1,...,nr\}, i < j$$

Attribute encoding is straight-forward. For each possible instance of a class, a CSP attribute variable is used. E.g., attribute `modulePos` of modules is represented by the variable array mp (let nm be the maximum number of modules):

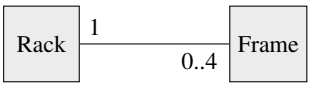
$$mp_i \in \{-1,1,...,16\}, \quad \forall_i \in \{1,...,nm\}$$

We provide a special attribute value (-1 for mp) for attributes of unused components. The following constraint states that unused components must have this special value, and used components must have a value from the regular domain of the attribute.

$$m_i = 0 \leftrightarrow mp_i = -1, \quad \forall_i \in \{1,...,nm\}$$

The interesting part of the model is the encoding of associations. A very general approach is to represent each association by a matrix of boolean variables on the instances of the two involved classes. A matrix entry $m_{ij} = 1$ means that object with index i is associated to object with index j. This representation needs a lot of CSP variables and makes the formulation of consistency constraints on associations quite intricate, causing low solving performance. Another approach [7] models association links as ports; an object has n association variables, where n is the maximum cardinality of the association.

We use a simpler representation: For a 1:n-association, we use a variable array on the n side of the association. Each such variable points to the associated object, or has value -1, if not connected. Example: The association



is represented as an integer variable fr for each frame:

$$fr_i \in \{-1,1,...,nr\}, \quad \forall_i \in \{1,...,nf\}$$

The special value -1 is used if a frame is not associated to a rack at all. This special value is also used for unused frames.

$$f_i = 0 \rightarrow fr_i = -1, \quad \forall_i \in \{1,...,nf\}$$

Additional consistency constraints are needed to rule out invalid association constellations. Each used frame must be connected to a rack:

$$f_i = 1 \rightarrow fr_i \in \{1,...,nr\}, \quad \forall_i \in \{1,...,nf\}$$

Frames can only be connected to used racks:

$$fr_i \geq 1 \rightarrow r_{fr_i} = 1, \quad \forall_i \in \{1,...,nf\}$$

Only up to 4 frames are allowed in a rack:

$$|\,\{fr_j \mid j \in \{1,...,nf\}, fr_j = i\}\,| \leq 4, \quad \forall_i \in \{1,...,nr\}$$

Example for a constraint on attributes and associations: All modules in a frame must have different module positions:

$$mf_i = mf_j \wedge mf_i \geq 1 \rightarrow mp_i \neq mp_j, \quad \forall_{i,j} \in \{1,...,nm\}, i < j$$

It should be noted that such object-constraint mapping on dynamic problems (where the number of instances in a solution is not known beforehand) has many disadvantages: (1) The representation of objects as a flat set of constraint variables is very unnatural and hard to read, debug, and maintain. This can be mitigated by an automatic translation from objects to constraints. (2) A maximal set of possible object instances must already be provided at problem formulation. Decisions on maximum values are in general not easy; too few objects could rule out possible solutions; too many objects blows up the problem size. (3) Current constraint solvers (mainly based on backtracking) are very sensitive to changes. Small changes in the variable structure or of the constraints could hugely influence solving performance, which makes a repeated tuning of the variable and value ordering heuristics necessary. (4) The representation of associations is crucial. A simple representation, as described above, does not directly support n:m associations and ordered associations. More elaborate encodings are difficult to handle in terms of constraint formulations, and often impair performance. (5) Modelling of inheritance additionally increases representation complexity. (6) Attributes of more complex types, like reals, multi-valued variables, or strings, are often not supported at all in constraint systems.

To formalize *reconfiguration* in terms of standard CSP, we first need to define a metric to have a notion of the distance between a legacy configuration and a reconfigured solution. Let (V, D, C) be a CSP with variables V, their domains D, and constraints C. An assignment is a tuple (v, d), $v \in V$, $d \in D_v$, representing the assignment of value d to variable v. Let A be a set of assignments. $vars(A)$ is the set of variables in A: $vars(A) = \{v \mid (v,d) \in A\}$. $vars(A1, A2)$ is the set of variables both in $A1$ and $A2$: $vars(A1, A2) = \{v \mid v \in vars(A1), v \in vars(A2)\}$. $\mathrm{diff}(A1, A2)$ is the number of assignments with different values on the common variables in $A1$ and $A2$:

$$\mathrm{diff}(A1, A2) = |\,\{v \mid v \in vars(A1, A2), \\ (v, d1) \in A1, (v, d2) \in A2, d1 \neq d2\}\,|$$

diff defines a simple metric on the space of all assignment sets of a CSP. The CSP reconfiguration problem can now be simply defined as follows: Let (V, D, C) be a CSP. Let A be an assignment on V or a subset on V. A is potentially inconsistent w.r.t. the constraints C. A *reconfiguration* A' is a consistent assignment on the variables V (i.e., A' is a solution of the corresponding CSP) which minimizes $\mathrm{diff}(A, A')$.

Reconfiguration can be simply implemented by slightly changing the backtracking search procedure. We can't start with the legacy configuration (a variable assignment), because it is potentially inconsistent and standard backtracking would stop with output `failure` immediately. But we could solve the problem from scratch and use the legacy configuration to guide expanding the search tree. Each time when a value for the current variable is selected, the value of the legacy configuration - if an assignment tuple for that variable is

part of the legacy configuration - is preferably chosen. Of course, if that value is inconsistent with the current constellation, an alternative value is taken. But basically, the legacy configuration is replayed, and changes are made only because of inconsistent states. The result is a consistent configuration which is very similar to the legacy configuration, which is exactly what we want - we want minimal reconstruction of the system in the field.

A brief note on our implementation: We used the constraint solver Choco V3.2.2 to represent and solve configuration problems (as described in the first part of this subsection) and replay-based reconfiguration. Choco allows to plug in your own value selector by implementing the interface `IntValueSelector`. With only a few lines of code, we extended the standard value selector of Choco by preferring the values stored in a given legacy configuration. If a legacy value for the current variable is not available or inconsistent, standard Choco value selection is used.

Advantages: (1) This method is light-weight, i.e., no additional modelling concepts (like cost functions) are needed. Changes in the existing backtracking search procedures are minimal. (2) Most of the existing backtracking algorithms (intelligent backtracking, etc.) and variable and value ordering heuristics can still be used with only minimal adaptations. (3) Replay-based reconfiguration can be applied to inconsistent configuration fragments, which is not the case for standard backtracking and consistency algorithms. (4) The method is complete (a solution is found, if one exists).

Disadvantages: (1) It is not guaranteed, that a solution with minimal changes is found. The quality of the solution depends on the variable/value ordering heuristics used. Nevertheless, results of our prototypical implementation have shown, that the reconfiguration solutions are often the optimum or very close to the optimum. (2) Replaying an inconsistent configuration may lead search into inconsistent branches which may heavily impair performance. (3) The approach is suited only for domains where a cardinality based cost function is applicable, e.g. homogeneous hardware configuration problems.

4 EVALUATION

We did experimental evaluations on our ASP (cf. Section 3.1) and CSP (cf. Section 3.2) implementations using randomly generated problem instances for the example problem domain sketched in Fig. 2. We ran the tests on a standard Windows 7 machine with a Intel dual-core i5 CPU and 8 GB RAM. We used clingo V4.4.0 from the Potassco suite for the ASP implementation, and Choco V3.2.2 for the CSP implementation.

It should be mentioned that we intentionally didn't use a high-performance hardware setting and we did not do any coding optimizations. Of course, ASP and CSP experts could find encodings which would perform better, but we wanted to test if AI techniques like ASP and CSP could be used by engineers with just standard skills in these techniques.

The input values for our HW configuration problem are the number and types of `Elements`. We generated randomly a set of input problem instances, solved them, made random changes to the solutions and applied the heuristic replay-based solvers (both ASP and CSP) to solve the reconfiguration problem. We measured solving time, memory consumption, and quality of the reconfiguration solution (i.e., how close is the result to the original configuration).

It should be mentioned that integration of reasoning functionality into our configuration infrastructure – an object-oriented data model and environment implemented in Java – was easier with Choco than with clingo, because Choco provides a Java API.

4.1 Performance: Time

Figures 3 and 4 show the solving time results for our ASP and CSP implementations. The x-axis shows the different problem sizes in terms of number of `Elements`. Note that the number of objects in a configuration solution is far higher than these values, because all the HW elements (racks, frames, modules) and internal variables are created during solving. The y-axis shows the solving execution time in seconds. For ASP, this is the sum of grounding and SAT solving time.

We incremented the problem size, i.e. the number of `Elements`, in steps of 10. We generated 40 different configurations per problem size, modified them randomly and solved the reconfiguration problem. Black circles in the plots are the measured times for each run. Blue squares are the mean runtime values for that problem size. Red triangles indicate timeouts (time $>$ 2 minutes).

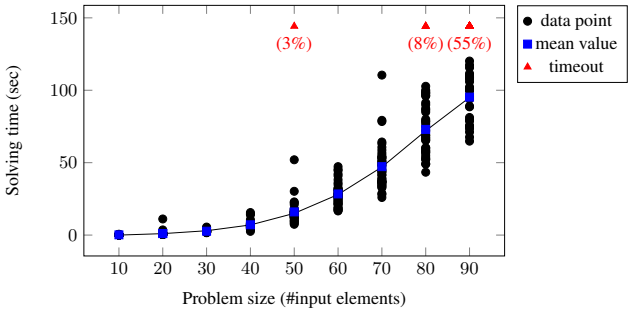


Figure 3. ASP solving times.

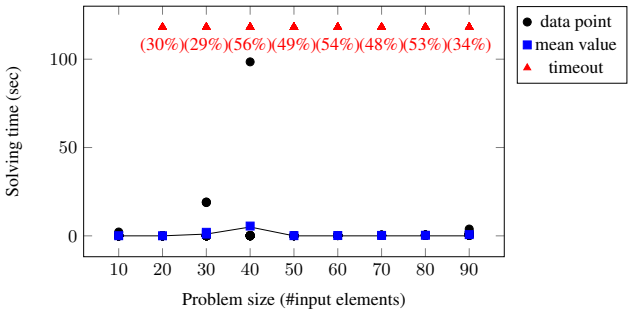


Figure 4. CSP solving times.

We made the following interesting observations:

- ASP is much more robust than CSP. For problems up to a size of ca. 90 `Elements` ASP finds a solution in a well predictable time. In contrast, CSP often needs a lot of time even for small problems and very often runs into timeout.
- Consequently, the sizes of problems where ASP finds a solution in acceptable time is also well predictable. In our test environment, ca. 100 `Elements` are the upper limit for ASP.
- CSP is much more sensitive to the input problem constellation. If the backtracking procedure makes invalid choices in the first part

of the search tree, backtracking gets out of hand. This is why CSP runs very often into timeout even for small problem sizes. In cases without or with little backtracking, CSP is very fast, even for large problems.
If one is willing to tune the variable and value ordering heuristics for her/his specific problem instance, CSP can solve much bigger problems than ASP very efficiently. The key is to avoid backtracking.

- The variance of runtime continuously grows with problem size for ASP. This is not the case for CSP. If CSP manages to solve the problem, it can do it most of the time very quickly. For the solvable problem sizes, there is rarely a difference in the CSP runtimes depending on the size of the input variables.

4.2 Performance: Memory

For all the test cases, we also measured indicators for memory consumption (cf. Fig. 5). For ASP, we used grounding size in MBytes. For CSP, we counted the number of CSP variables used. Note that, aside from user-defined variables (cf. Section 3.2), Choco creates a lot of additional, internal variables. Of course, grounding memory size for ASP and numbers of variables in CSP cannot be compared directly, but they give good indicators about the memory growth rate depending on the input configuration size.

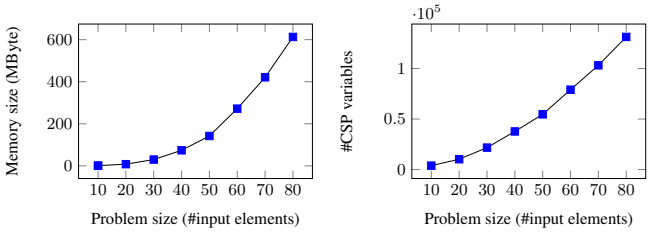


Figure 5. (a) ASP grounding memory size. (b) CSP number of variables.

Not surprisingly, memory of both ASP and CSP grows with accelerated speed depending on the problem size. ASP grows with a slightly higher rate. Not only in the context of reconfiguration, ASP often shows its limits at grounding. Most of the execution time and a big amount of memory is used for grounding.

To give estimations of consumed memory for CSP is a bit more subtle. As shown in Fig. 5(b), we used the number of variables as memory indicator. A rough memory profile using Choco's statistics functionality has shown, that for 100,000 variables ca. 20 MByte RAM is consumed (for the cases without heavy backtracking). This means that CSP's footprint is roughly 20 times smaller than ASP's footprint.

4.3 Performance: Quality

To evaluate the quality of a reconfiguration we measured the distance of the original, legacy configuration to the reconfigured solution. We used a graph-based difference metric counting the differences in the rack/frame/module constellation of the legacy configuration to the reconfiguration.

The first and simplest case is to provide a legacy configuration which is already consistent. This means that a reconfiguration should reproduce the legacy configuration without any changes. Both ASP and CSP did this in many test cases of various sizes.

The more interesting case is a legacy configuration which is inconsistent to the problem description. For each problem size (starting from 10 `Elements` up to 80 `Elements` in steps of 10), we randomly modified up to 20% of a valid configuration. For each problem size, we did 40 different tests.

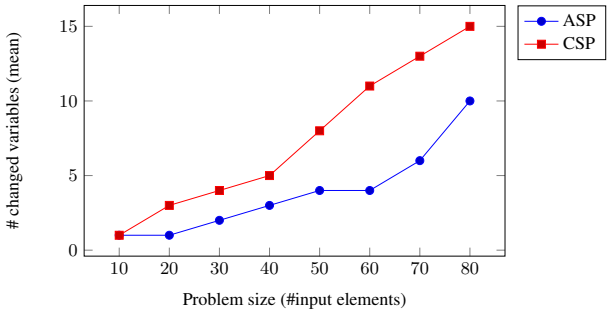


Figure 6. ASP and CSP reconfiguration quality.

The results are shown in Fig. 6. ASP most of the time finds solutions of high quality. In fact, for smaller problem sizes we could manually verify that ASP nearly all the times finds the optimal solution. With CSP, the mean distance to the legacy configuration is a bit higher than with ASP, but has an acceptable quality on most of the cases.

4.4 Evaluation Summary

Table 2 gives a summarized comparison of our ASP and CSP reconfiguration encodings and the results of our experimental evaluations. For solving placement problems like our hardware example, there is no clear winner. If the problem is of moderate size, ASP provides a sound, predictable and easy-to-use reasoning functionality. For larger problem, CSP may be better, but probably additional coding is needed for tuning search.

5 RELATED WORKS

Related to our work presented in this paper are all techniques for finding a valid reconfiguration for a given, possibly inconsistent configuration (fragment). The main research approaches are:

Repair-based approaches. Repair-based approaches aim for finding diagnoses and repairs to conflicting requirements or constraints [5]. Usually, methods from model-based diagnosis are used (e.g., minimal hitting sets). Repair-based approaches are mainly studied in the context of recommender systems [6]. These approaches are complete, they are based on a clean, formal theory, and they typically take user needs into account. Those repairs are in favour which may be of most usefulness for the user. When applied in a configuration context based on consistency and search algorithms, repair-based methods introduce additional reasoning techniques which must be integrated into the configurator framework. Our heuristic, replay-based approach uses conventional solving techniques with slight modifications for reconfiguration.

Minimization of modification costs. The basis of these approaches is the definition of cost functions for the different modification operations [9]. Reconfiguration is then finding a valid modification of the configuration which minimizes the sum of all modification costs. Such techniques have been, e.g., intensively studied in the research project RECONCILE[4]. The possibility of defining elaborate cost functions for configuration modifications along with a complete optimization search procedure (in [9], based on ASP) is a great advantage for applications where modifications in the field are expensive. But this comes at the price of considerable additional modelling concepts for cost functions and often declined solving performance. Compared to that, our approach is light-weight in the sense that no additional modelling is necessary, and most of the advanced backtracking algorithms with only minimal adaptations are applicable.

Table 2. Comparison of ASP and CSP reconfiguration modelling and behaviour.

	ASP	CSP
Robustness	High Predictable	Low Very sensitive to input constellation and problem formulation
Performance	Good for small problem sizes Does not scale for larger problems	Very good, if no or little backtracking, else timeout Probability of timeout grows with problem size
Memory footprint	High (grounding!)	Low
Solution quality	Very good Most of the time the optimum or close to the optimum	Never better than ASP, but most of the time acceptable
Integrability	Typically, ASP systems are not as easy to integrate into a Java environment as CSP systems. The ASP solvers Potassco[5] and Smodels[6] provide C++ libraries, DLV[7] recently provided a Java interface (JDLV[8]).	Many CSP solvers support APIs to programming languages like Java (e.g. Choco[9], JaCoP[10]) or C++ (e.g. Gecode[11], Minion[12]).
Problem encoding	Favoured is an automatic transformation from the object-oriented problem description to ASP. Direct ASP encoding is also possible, because ASP has a compact syntax and is readable.	Direct encoding of an object-oriented data model with a CSP system is quite intricate and error-prone. Automatic transformation is highly recommended.

Local search methods. The main alternatives to backtracking-like search are so-called heuristic (or local) search strategies which try to find solutions in a hill-climbing manner (e.g. greedy search algorithms, genetic algorithms). In the context of product configuration, Generative CSP (GCSP) is an extension of standard CSP and has been introduced in [8] for large, dynamic configuration problems. In GCSP, mainly local search techniques - *repair-based local search* - are used because no fast complete search methods are available yet for dynamic systems. Repair-based local search tries to find local modifications of an inconsistent or incomplete configuration. Thus, this technique intrinsically can deal with inconsistent configuration (fragments). Complex, dynamic problems can be modelled in a very natural way using object-oriented concepts. The main disadvantage of local search methods is that they are incomplete – they may get stuck in a local optimum during search and may not find a solution, even if one exists. Compared to that, our approach is complete, because it is based on an exhaustive tree search (backtracking).

Rule-based approaches. Especially in model-driven engineering, a lot of research in model synchronization has been done and is still on-going. Correspondences between two models are defined as transformation rules, describing how values from one model are mapped to another model. Examples of such systems are triple graph grammars [13] or JTL [2]. Model synchronization (corresponding to reconfiguration in our definition) is done by triggering the transformation rules. Applying such methods to a reconfiguration problem in product configuration means that all necessary types of modification operations for transforming an invalid configuration to a valid one must be specified explicitly. Our heuristic, replay-based approach does not need any additional knowledge like transformation rules. Reconfiguration is guided by a legacy configuration and a declarative problem specification (models M3 and M2 in Tab. 1).

Common to all these approaches – at least to a certain degree – is that reconfiguration actions are modelled on a declarative level. The specification of potential modification operations and reconfiguration reasoning are separated. Another approach used in industry (e.g. in factory facilities, steel plants) is to offer upgrade or modernization packages. There the focus lies on finding and recommending modernization packages which are appropriate to add functionalities to a system in the field.

6 CONCLUSION AND FUTURE WORK

There is no standard way of doing reconfiguration for product configuration especially for large problem sizes. In this paper we showed the implementation of a heuristic approach to reconfiguration using standard solving techniques and the applicability up to moderate problem sizes. The main challenge for using these techniques in an industrial setting are grounding size and solving time. Grounding size typically can be influenced by finding a better encoding or by problem decomposition. Solving time is also influences by the encoding of the problem and by finding the right heuristic for the domain.

Because of the heterogeneous nature of the constraints in product configuration coming up with a good encoding and heuristics for a problem is currently as much an art as a science and requires an experienced knowledge engineer. Also it requires experiments with different solving paradigms as SAT, ASP, CSP, MIP etc. Therefore we welcome the further integration of AI and OR solving techniques that have taken place in the last years as we believe there will not be THE solving technique for product (re-)configuration in the foreseeable future.

For the future we plan to further study and improve heuristic reconfiguration solving techniques and to apply them to fields beyond

[4] http://isbi.aau.at/reconcile/
[5] http://potassco.sourceforge.net/
[6] http://www.tcs.hut.fi/Software/smodels/
[7] http://www.dlvsystem.com/
[8] http://www.dlvsystem.com/jdlv/
[9] http://choco-solver.org/
[10] http://www.jacop.eu/
[11] http://www.gecode.org/
[12] http://minion.sourceforge.net/

product configuration. As we have seen in our experiments, CSP solving – though it is very fast and produces good results if it doesn't fall into a heavy backtracking trap – currently isn't robust enough to be applied in an industrial environment. We believe that the integration of additional heuristics which are automatically derived from the problem domain or techniques like lazy clause generation [17] will fix this problem of poor robustness. Also the integration of CSP and ASP (CASP [1]) looks promising.

Interesting fields beyond product configuration, where reconfiguration methods could be applied, are:

- *Production configuration:* With the increasing demand for individualized products, the need for flexible production processes, modular factories and intelligent production infrastructures is also increasing. Factories of the future are generic production facilities, that can be easily adapted to the needs of the product to be manufactured [3]. This means, before the factory can manufacture products of a product line, it has to be physically reconfigured for the specific production setting. This includes reconfiguration of the cyber-physical components of the factory [15], and therefore the need for fast, flexible and robust reconfiguration technologies.
- *Model synchronization:* In model-driven engineering, model synchronization is the task of finding a mapping between overlapping concepts of two different models. Typically, the overlaps of the two models are described as a correspondence model, including constraints which define the dependencies and interactions between the models. This situation can be seen as a reconfiguration problem: Given are two models (e.g., configuration instances of two different configuators) which have been changed in the course of a new system version, and a correspondence model. The reconfiguration problem is now to find changes in the two evolved models which are (a) consistent to their domain model, (b) consistent to the correspondence model, and (c) as close as possible to the original models.
- *Case-based reasoning:* In case-based reasoning, a database of solutions from previous problems are used to find a solution to a new problem [18]. Usually, no perfectly fitting solution could be found, but one which solved a similar problem. We think that our heuristic, replay-based reconfiguration procedures could be applied to the reuse/revise phase of case-based reasoning: To solve a new problem try to rebuild the configuration of a solved problem.

ACKNOWLEDGEMENTS

This work has been funded by the Vienna Business Agency in the programme ZIT13-plus within the project COSIMO (Collaborative Configuration Systems Integration and Modeling) under grant number 967327, and by FFG FIT-IT within the project HINT (Heutistic Intelligence) under grant number 840242.

REFERENCES

[1] Marcello Balduccini and Yuliya Lierler, 'Integration schemas for constraint answer set programming: a case study', *TPLP*, **13**(4-5-Online-Supplement), (2013).

[2] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio, 'Jtl: A bidirectional and change propagating transformation language', in *Software Language Engineering*, eds., Brian Malloy, Steffen Staab, and Mark van den Brand, volume 6563 of *Lecture Notes in Computer Science*, 183–202, Springer Berlin Heidelberg, (2011).

[3] D. Dhungana, A. Falkner, A. Haselböck, and H. Schreiner, 'Smart factory product lines: A configuration perspective on smart production ecosystems', in *Manuscript submitted for publication*, (2015).

[4] Ingo Feinerer and Gernot Salzer, 'Numeric semantics of class diagrams with multiplicity and uniqueness constraints', *Software and System Modeling*, **13**(3), 1167–1187, (2014).

[5] Alexander Felfernig, Gerhard Friedrich, Monika Schubert, Monika Mandl, Markus Mairitsch, and Erich Teppan, 'Plausible repairs for inconsistent requirements', in *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pp. 791–796, (2009).

[6] Alexander Felfernig, Erich Teppan, Gerhard Friedrich, and Klaus Isak, 'Intelligent debugging and repair of utility constraint sets in knowledge-based recommender applications', in *Proceedings of the 2008 International Conference on Intelligent User Interfaces, January 13-16, 2008, Gran Canaria, Canary Islands, Spain*, pp. 217–226, (2008).

[7] Adel Ferdjoukh, Anne-Elisabeth Baert, Annie Chateau, Remi Coletta, and Clémentine Nebut, 'A CSP approach for metamodel instantiation', in *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013*, pp. 1044–1051, (2013).

[8] Gerhard Fleischanderl, Gerhard Friedrich, Alois Haselböck, Herwig Schreiner, and Markus Stumptner, 'Configuring large systems using generative constraint satisfaction', *IEEE Intelligent Systems*, **13**(4), 59–68, (1998).

[9] Gerhard Friedrich, Anna Ryabokon, Andreas A. Falkner, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner, '(re)configuration based on model generation', in *LoCoCo*, pp. 26–35, (2011).

[10] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, *Answer Set Solving in Practice*, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Publishers, 2012.

[11] M. Gebser, B. Kaufmann, J. Romero, R. Otero, T. Schaub, and P. Wanko, 'Domain-Specific Heuristics in Answer Set Programming', in *Proceedings of the AAAI*, (2013).

[12] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Thomas Schneider, 'Potassco: The potsdam answer set solving collection', *AI Commun.*, **24**(2), 107–124, (2011).

[13] Frank Hermann, Hartmut Ehrig, Claudia Ermel, and Fernando Orejas, 'Concurrent model synchronization with conflict resolution based on triple graph grammars', in *Fundamental Approaches to Software Engineering*, eds., Juan de Lara and Andrea Zisman, volume 7212 of *Lecture Notes in Computer Science*, 178–193, Springer Berlin Heidelberg, (2012).

[14] U. Junker, 'Configuration', in *Handbook of Constraint Programming*, eds., F. Rossi, P. vanBeek, and T. Walsh, pp. 837–873. Elsevier, (2006).

[15] Kunming Nie, Tao Yue, Shaukat Ali, Li Zhang, and Zhiqiang Fan, 'Constraints: The core of supporting automated product configuration of cyber-physical systems', in *Model-Driven Engineering Languages and Systems*, eds., Ana Moreira, Bernhard Schätz, Jeff Gray, Antonio Vallecillo, and Peter Clarke, volume 8107 of *Lecture Notes in Computer Science*, 370–387, Springer Berlin Heidelberg, (2013).

[16] Gottfried Schenner, Andreas Falkner, Anna Ryabokon, and Gerhard Friedrich, 'Solving object-oriented configuration scenarios with asp.', *Proceedings of the 15th International Configuration Workshop*, 55–62, (2013).

[17] Peter J. Stuckey, 'Lazy clause generation: Combining the power of SAT and CP (and MIP?) solving', in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings*, pp. 5–9, (2010).

[18] Hwai-En Tseng, Chien-Chen Chang, and Shu-Hsuan Chang, 'Applying case-based reasoning for product configuration in mass customization environments', *Expert Syst. Appl.*, **29**(4), 913–925, (2005).

Arc Consistency with Negative Variant Tables

Albert Haag[1]

Abstract. In this paper I discuss a fundamental difference between positive and negative variant tables (tables listing excluded combinations) from the viewpoint of a modeler. I provide an approach to achieving arc consistency with negative tables that can be integrated into an existing configurator that already implements methods for arc consistency. I also provide a simple necessary condition to test whether a further restriction of given domains is possible using a negative table. As a positive table is equivalent to a negative table representing the complement, this condition test also applies for positive tables that have a small complement. I refer to this process as *double negation*. A prototypical implementation in Java exists that covers the work described both here and in [6]. I present aggregated results of applying *double negation* to variant tables from real product test data in [6]. This also validates the overall functional correctness of the entire approach.

1 Introduction

Tabular data are an important element of product configuration models. In the context of the *SAP Variant Configurator - SAP VC* [3] a table that lists valid combinations of product properties is referred to as a *variant table*. The number of columns of a variant table is called its *arity*. Each column of the table is mapped to a product property, e.g., *Color*.

The simplest form of a product model is just as one single variant table. The term *variant table* derives from this. Table 1, below is an example of a variant table that completely describes the model of a configurable t-shirt by listing all variants. The t-shirt has three properties *Color*, *Size*, and *Print*[2]. Given the values that appear in the table[3] it is evident that 11 of possible 24 combinations have been selected as valid for configuration. I expand on this model in Section 3.

Now, SAP customers have long requested the ability to also maintain tables of disallowed (excluded) combinations of product properties. These would be called *negative variant tables*.

Whereas, a positive table implicitly defines a bound on the overall (global) domains of the associated properties[4], this is different for a negative variant table. There are at least two interpretations of the motivation for maintaining a negative table:

1. The persons maintaining a negative table are aware of overall (global) domains for the affected product properties. The negative form of the table is merely chosen as shorthand for maintaining the complement of an otherwise very large positive table.

Table 1. Variant table for a simple configurable t-shirt

Color	Size	Print
Black	Small	MIB
Black	Medium	MIB
Black	Large	MIB
Black	Medium	STW
Black	Large	STW
Red	Medium	STW
Red	Large	STW
White	Medium	STW
White	Large	STW
Blue	Medium	STW
Blue	Large	STW

2. The persons maintaining the table are expressing exclusions completely independently of any thought of what the affected property domains might be.

One obvious way of dealing with negative variant tables that pertains to the first case is to provide support for complementing the table with respect to global domains of the product properties at maintenance time[5]. Note that it is not possible to calculate the complement to the global domains if these are unconstrained[6].

I shall focus on the second case as requests by SAP customers clearly indicate this setting. Then, it is not legal to complement a negative table at maintenance time with regard to the global domains even if these are finite, because they may change after the table was maintained. The table may have its own update cycle and should, therefore, not need to be touched each time a global domain changes.

In knowledge-based configuration variant tables function as table constraints. The extensional form of a table constraint (explicitly listing all tuples of values implied by the table) closely corresponds to the relational form of a variant table (directly storing the variant table transparently in a relational database[7]). Each value a_{ij} (where i is the index of the row, and j is the index of the column) that occurs in a table cell states a Boolean proposition p_{ij} that the corresponding mapped property v_j is assigned to that value ($p_{ij} \models (v_j = a_{ij})$). In this case, a row r_i in the variant table directly maps to a tuple of propositions τ_i in the associated table constraint representing the

[1] SAP SE, Germany, email: albert.haag@t-online.de

[2] The example is taken from [1]. I use and extend it both here and in [6]

[3] I have kept the shorthand codes of MIB (for "Men in Black") and STW (for "Save the Whales") used in [1]

[4] That is to say, for a positive table a value that does not occur in a particular table column can be removed from the domain of the property associated with that column

[5] If the challenge lies in the large size of the positive variant table, offering maintenance of the table directly in a compressed format is another option. Compressions of Table 1 are depicted in [6]. For the SAP VC some support is provided for complementing tables at maintenance time and maintaining tables in non-extensional form, i.e., with cells containing non-atomic entities such as real-valued intervals or sets of values

[6] Unconstrained domains are not that common in traditional business settings, but they do occur. For real-valued numeric properties the global domains may often be (bounded/unbounded) continuous intervals

[7] In this case, the table cells will contain only atomic values (Strings or numbers)

conjunction of its elements, i.e., for fixed row index i and arity k $r_i = (a_{i1}, \ldots, a_{ik})$ and $\tau_i = (p_{i1}, \ldots, p_{ik}) \models p_{i1} \wedge \ldots \wedge p_{ik}$.

A central form of constraint evaluation is propagation to achieve arc consistency, which removes any values from the current domains of the properties constrained by the variant table that are not supported in the intersection of the table with these domains. This is a proven way in practice to limit the choices available to the user in interactive configuration. I refer to this simply as constraint propagation in the sequel[8].

In this paper, I do not propose an own algorithm for constraint propagation with negative tables[9]. Rather, I assume that a configurator will already implement constraint propagation for positive tables, but it may not implement a corresponding algorithm (such as [9]) for negative table constraints[10].

A tuple representing a disallowed combination of propositions $(p_1, \ldots, p_k)$ in a negative variant table of arity k allows k obvious inferences:

$$p_{j_1} \wedge \ldots \wedge p_{j_{k-1}} \rightarrow \neg p_{j_k} \quad (1)$$

These could be applied at run-time if and where the configurator supports it, e.g., where it is possible to remove a value from a domain in a way that can be represented in the configurator[11].

In this paper, I characterize what can be achieved using pre-existing means of constraint propagation beyond (1) with negative tables that are maintained independently from the global domains for the mapped properties. I give a condition that can be tested during evaluation to determine if further restrictions via constraint propagation are possible (Section 4). The condition states that at least all but one of the domains must be sufficiently restricted in order to achieve further filtering with the negative variant table. In a way, this generalizes (1), which states that an inference is possible if all but one value assignments are known.

The condition test rests on the fact that the solution set of a negative variant table with arity k can be easily decomposed at run-time into $k + 1$ disjoint parts, of which k are in the form of Cartesian products (referred to as *c-tuples* in Section 2.2). This decomposition is one of the results presented in this paper and applies independently and on top of the methods implemented in the configurator for arc consistency.

A positive variant table can be transformed into a negative one by negating (complementing) it. When processing this negative table the table is logically negated again, yielding an identical solution set, but not an identical structure to the original table. I refer to this as *double negation*, and discuss it as one possible approach to compression in Section 5. Otherwise, compression of variant tables is a topic I deal with in [6].

I distinguish between information known to the modeler at the time the variant table is maintained (maintenance-time) from information known when configuring the product (run-time). Some calculations can already be performed at maintenance-time, others best at run-time. It is important that run-time operations do not impede the overall performance of the configurator. Maintenance-time operations should be performant as well, but this is less critical.

In Section 2, I introduce the notation and some formalisms. Section 3 uses the product model of a t-shirt taken from [1] (Table 1) to illustrate the concepts. As already mentioned, Sections 4 and 5 deal with deriving the necessary condition test and with double negation, respectively. In Section 6, I comment on the status of the implementation. I conclude with some further observations in Section 7.

Finally, a disclaimer: While the motivation for this work lies in my past at SAP and is based on insights and experiences with the product configurators there [3, 5], all work on this paper was performed privately during the last two years after transition into partial retirement. The implementation is neither endorsed by SAP nor does it reflect ongoing SAP development.

2 Framework

My scope here is restricted to the problem of constraint propagation with negative tables. I construct a framework for this that is based on operations with sets. After an overview of the basic framework, I define the relevant sets in Section 2.1, and present the actual framework I use for negative variant tables in Section 2.2.

A variant table $\mathcal{T}$ is characterized as follows: its *arity* expresses how many columns it has. The columns are indexed with $j : 1 \leq j \leq k$, where k is the arity. Each column is mapped to a *product property*, v_j. I assume $\mathcal{T}$ to be in relational form, i.e., each table cell contains an atomic value (a string or number). Variant tables that allow intervals or wild cards in cells are not directly representable in relational form. Many of the results here could be extended to cover this case, but I consider it out of scope here.

I view a variant table $\mathcal{T}$ in its role as a constraint only[12]. In the language of constraints the product properties mapped to the columns of the variant table are the constraint variables. I continue to refer to them as *product properties*. The assumed relational form of the variant table simplifies the correspondence between a table cell a_{ij} (where i is the index of the row, and j is the index of the column) and a value assignment $v_j = a_{ij}$. Thus each row directly corresponds to a value assignment tuple $(v_1 = a_1, \ldots, v_k = a_k)$. In its role as a constraint, each row in $\mathcal{T}$ expresses a k-tuple of valid value assignments.

Each product property, v_j, has a domain. The domain known at maintenance-time for v_j is called the *global domain*, Ω_j, which may be *unconstrained* (infinite) if unknown (or otherwise infinite as would be the case for a continuous interval). For negative variant tables, I treat global domains as *unconstrained*, as discussed in Section 1. I refer to a domain known at run-time for v_j as a *run-time restriction*, R_j, even if it also unconstrained or infinite.

I assume that the configurator already implements constraint propagation methods for arc consistency, e.g., implements some form of a GAC algorithm [2, 8]. As a consequence, I do not delve into the details of any particular GAC algorithms here[13].

The following example illustrates the concepts introduced so far, albeit for a positive variant table. Examples for negative variant tables are constructed in Section 3.

[8] I am only concerned with the propagation on each table constraint individually. The question of how to achieve overall arc consistency and how to resolve inconsistencies is up to the methods pre-implemented in the configurator. The SAP VC, for example, does not backtrack, but performs constraint propagation via forward filtering

[9] The implementation is part of a prototype for exploring the compression approach in [6]. This also handles negative variant tables, and thus an own approach is implicit in that context

[10] If it does implement such an algorithm, then the algorithm itself implicitly involves calculating the complement to the domains known at run-time

[11] This will not be the case when the run-time domain is *unconstrained*, but then all methods of dealing with negative tables referred to cannot be meaningfully applied

[12] In the SAP VC configurator variant tables are also used in procedural fashion, e.g., in "rules"

[13] If the configurator also already implements some form of arc consistency with negative table constraints such as [9], the basic approach will still apply. I shall indicate what differences must then be observed in the appropriate places, below

Example 1 *Table 1 is a positive variant table of t-shirt variants. Looking at this table, the global domains are*

- $\{Black, Blue, Red, White\}$ *for the product property "*$Color$*", denoted by* v_1
- $\{Large, Medium, Small\}$ *for the product property "*$Size$*", denoted by* v_2
- $\{MIB, STW\}$ *for the product property "*$Print$*", denoted by* v_3

If the customer wants a red t-shirt, but specifies nothing else, then the run-time restrictions are

- $R_1 = \{Red\}$ *for the product property "*$Color$*"*
- $R_2 = \{Large, Medium, Small\}$ *for the product property "*$Size$*"*
- $R_3 = \{MIB, STW\}$ *for the product property "*$Print$*"*

A GAC algorithm can then further restrict the domains of the property "$Size$*" to* $\{Large, Medium\}$ *and of the property "*$Print$*" to* $\{STW\}$*. (There are only two rows in the table involving the color "*Red*".)*

2.1 Definitions and Notation of Relevant Sets

Let $\mathcal{T}$ denote a variant table of arity k for product properties $v_1 \dots v_k$. These are the product properties that are related via $\mathcal{T}$ as a constraint, and the only properties to consider when looking at $\mathcal{T}$ in isolation, but their existence is not directly tied to $\mathcal{T}$. Let $\Omega_1 \dots \Omega_k$ be known global domains of $v_1 \dots v_k$. I define the *solution space*, $\boldsymbol{\Omega}$, as

$$\boldsymbol{\Omega} := \Omega_1 \times \ldots \times \Omega_k \tag{2}$$

Any subset of the solutions space consisting of valid tuples defines a constraint relation on $v_1 \dots v_k$. In particular, $\mathcal{S}^{\mathcal{T}}$, the set of valid tuples defined by $\mathcal{T}$, is a subset of $\boldsymbol{\Omega}$. If $\mathcal{T}$ is in relational (extensive) form, then $\mathcal{S}^{\mathcal{T}} = \mathcal{T}$ for a positive table, and $\mathcal{S}^{\mathcal{T}} = \boldsymbol{\Omega} \setminus \mathcal{T}$ for a negative table. I refer to $\mathcal{S}^{\mathcal{T}}$ as the *solution set* of $\mathcal{T}$. Generally, a solution set as defined above may not be finite.

Let $\tau = (\tau_1, \dots, \tau_k) \in \boldsymbol{\Omega}$ denote a tuple in the solution space. Given any $\boldsymbol{X} \subseteq \boldsymbol{\Omega}$, I define the j-th *column domain* of $\boldsymbol{X}$ as

$$\pi_j(\boldsymbol{X}) := \bigcup_{\tau \in \boldsymbol{X}} \{\tau_j\} \tag{3}$$

I call $\pi_j(\boldsymbol{X})$ the *projection* of $\boldsymbol{X}$ onto the j-th component, and define the *projection* or *constraint propagation* operator π as:

$$\pi(\boldsymbol{X}) := \pi_1(\boldsymbol{X}) \times \ldots \times \pi_k(\boldsymbol{X}) \tag{4}$$

The complement of a column domain $\pi_j(\boldsymbol{X})$ with respect to a run-time restriction R_j plays a central role in the decomposition in Section 2.2. Hence, I find it convenient to abbreviate its notation as:

$$\overline{\pi_j(\boldsymbol{X})}^R := R_j \setminus \pi_j(\boldsymbol{X}) \tag{5}$$

For notational convenience I refer to any set $\boldsymbol{C} \subseteq \boldsymbol{\Omega}$ that is a Cartesian product $\boldsymbol{C} = C_1 \times \ldots \times C_k$ as a *c-tuple*. All of the following are c-tuples

- $\boldsymbol{\Omega}$ itself
- $\pi(\boldsymbol{X})$ the tuple of column domains for in (4)
- the tuple of run-time restrictions, denoted by

$$R = R_1 \times \ldots \times R_k$$

I refer to R as a whole as as a *run-time restriction tuple*

In Example 1 it can be seen that it is possible to eliminate $\{Small\}$ and $\{MIB\}$ after deciding on a red t-shirt. This is the basic inference of arc consistency, which I refer to as *constraint propagation*: filtering out values that are no-longer part of any valid tuple.

Given a run-time restriction tuple R and a variant table $\mathcal{T}$ with solution set $\mathcal{S}^{\mathcal{T}}$, then the set of remaining valid tuples is $R \cap \mathcal{S}^{\mathcal{T}}$. I abbreviate the solution set $\mathcal{S}^{R \cap \mathcal{S}^{\mathcal{T}}}$ by $\mathcal{S}^{\mathcal{T},R}$. If $\mathcal{T}$ is positive, then $\mathcal{S}^{\mathcal{T},R} = \mathcal{T} \cap R$. If $\mathcal{T}$ is negative, then $\mathcal{S}^{\mathcal{T},R} = R \setminus \mathcal{T}$.

Constraint propagation restricts the run-time restrictions R_j to $\pi_j(R \cap \mathcal{S}^{\mathcal{T}})$.

2.2 Negative Variant Tables

For clarity, I denote a *negative* variant table by $\mathcal{U}$. I take $\mathcal{U}$ to be in relational form and have arity k. The complement of $\mathcal{U}$ with respect to $\pi(U)$ is a positive table constraint that can be calculated at maintenance-time and depends only on $\mathcal{U}$ itself. I denote this complement by $\overline{\mathcal{U}}$

$$\overline{\mathcal{U}} = \pi(U) \setminus \mathcal{U} \tag{6}$$

Note that $\overline{\mathcal{U}} = \emptyset$ by construction if $\mathcal{U}$ has only a single line, because then $\pi(U) = \mathcal{U}$.

Given a run-time restriction tuple R, the solution set of $\mathcal{U} \wedge R$ is $\mathcal{S}^{\mathcal{U},R} = R \setminus \mathcal{U}$. This can be decomposed into two disjoint parts (either of which may be empty, see Section 3 for examples):

$$\mathcal{S}^{\mathcal{U},R} = (R \setminus \pi(U)) \cup (\overline{\mathcal{U}} \cap R) \tag{7}$$

The first part $(R \setminus \pi(U))$ is just R with the c-tuple spanning all values that occur in $\mathcal{U}$ removed. The second part then re-adds all tuples in $\pi(\mathcal{U})$ that occur both in $\overline{\mathcal{U}}$ and R.

The second part $(\overline{\mathcal{U}} \cap R)$ is just the solution set of $\overline{\mathcal{U}} \wedge R$. As $\overline{\mathcal{U}}$ is a positive variant table, this can be processed by the means available to the configurator[14].

The first part $(R \setminus \pi(U))$ can be decomposed into k disjoint c-tuples $\boldsymbol{C}_j^{\mathcal{U},R}$ as follows (see Proposition 2):

$$\begin{aligned}
\boldsymbol{C}_1^{\mathcal{U},R} &= \overline{\pi_1(\mathcal{U})}^R \times R_2 \times R_3 \times \ldots \times R_k \\
\boldsymbol{C}_2^{\mathcal{U},R} &= \pi_1(\mathcal{U}) \times \overline{\pi_2(\mathcal{U})}^R \times R_3 \times \ldots \times R_k \\
&\cdots \\
\boldsymbol{C}_k^{\mathcal{U},R} &= \pi_1(\mathcal{U}) \times \pi_2(\mathcal{U}) \times \pi_3(\mathcal{U}) \times \ldots \times \overline{\pi_k(\mathcal{U})}^R
\end{aligned} \tag{8}$$

or more explicitly for the omitted rows $2 < j < k$:

$$\boldsymbol{C}_j^{\mathcal{U},R} = (\pi_1(\mathcal{U}) \times \ldots \times \pi_{j-1}(\mathcal{U})) \times \overline{\pi_j(\mathcal{U})}^R \times (R_{j+1} \times \ldots \times R_k)$$

Proposition 2 $(R \setminus \pi(\mathcal{U}))$ *in* (7) *can be decomposed into the disjoint union*[15] (8) *for an arbitrary ordering of the columns.*

Proof
Each $\boldsymbol{C}_j^{\mathcal{U},R}$ is a subset of R by construction.
The j-th component of $\boldsymbol{C}_j^{\mathcal{U},R}$ is $\overline{\pi_j(\mathcal{U})}^R$. This is disjoint to $\pi_j(\mathcal{U})$. So $\boldsymbol{C}_j^{\mathcal{U},R}$ is disjoint to all $\boldsymbol{C}_p^{\mathcal{U},R} \quad \forall j < p \leq k$.
No tuple $x \in \pi(\mathcal{U})$ is in any $\boldsymbol{C}_j^{\mathcal{U},R}$, because $x \in \boldsymbol{C}_j^{\mathcal{U},R} \Rightarrow x_j \in \overline{\pi_j(\mathcal{U})}^R \Rightarrow x_j \notin \pi_j(\mathcal{U}) \Rightarrow x \notin \pi(\mathcal{U})$.

[14] If the configurator implements an algorithm for negative GAC such as [9], then $\overline{\mathcal{U}}$ need not be explicitly calculated. Processing would not be affected for this part

[15] I exclusively use the term *disjoint union* to refer to a union of disjoint sets [4]. I denote the disjoint union of two sets A and B by $A \cup B$, which implies that $A \cap B = \emptyset$

For all other tuples $y \in R$, there is at least one component with $y_j \notin \pi_j(\mathcal{U})$. Let j^* denote the smallest such index. Then, it follows by construction that $y \in \boldsymbol{C}_{j^*}^{\mathcal{U},R}$, because $\forall p < j^* : y_p \in \pi_p(\mathcal{U})$, $y_{j^*} \in \overline{\pi_{j^*}(\mathcal{U})}^R$, and $\forall p > j^* : y_p \in R_p$. ∎

The decomposition (8) is meaningful, because if $(R \setminus \pi(\mathcal{U}))$, the first part in (7), does not allow further constraint propagation, then the GAC algorithm need not be applied for $\overline{\mathcal{U}} \cap R$ in the second part at all. (All values in R are allowed by the first part.) I give a simple criteria for when this is the case in Section 4. This is a necessary precondition for a further reduction. Generally, it is easy to perform constraint propagation on a constraint that is a c-tuple[16].

3 Examples

I base my examples on the t-shirt model I introduced in Table 1. The global domains are listed in Example 1. In a real application setting, Table 1 would be used as the product model as is. For purposes of exposition here, I assume that the model evolves over time and further colors, sizes, and prints might be added in later model updates along with associated constraints. The restriction to the initial 11 valid tuples in Table 1 implements constraints to the effect that MIB implies $Black$ and STW implies $\neg Small$.

In this section, I use $\mathcal{U}$ to denote a negative variant table, and $\mathcal{T}$ to denote a positive one.

3.1 T-Shirt Example: STW

The constraint $STW \rightarrow \neg Small$ in the t-shirt example can be formulated as a single exclusion, yielding a negative variant table $\mathcal{U}$ with one row, $k = 2$, $v_1 = Print$, and $v_2 = Size$

$$\mathcal{U} = \begin{pmatrix} STW & small \end{pmatrix}$$

As noted, $\overline{\mathcal{U}} = \emptyset$.

Assume that the restricted domains at run-time are just the global domains, i.e., $R_1 = \Omega_1$ (property $Print$) and $R_2 = \Omega_2$ (Property $Size$). Then, the solution set $\mathcal{S}^{\mathcal{U},R}$ is directly given by the decomposition (8) of the first term in (7)[17]. This means that $\pi_1(\mathcal{U}) = \{STW\}$, $\pi_2(\mathcal{U}) = \{Small\}$ and:

$$\begin{aligned} \boldsymbol{C}_1^{\mathcal{U},R} r &= \{MIB\} \times \{Large, Medium, Small\} \\ \boldsymbol{C}_2^{\mathcal{U},R} &= \{STW\} \times \{Large, Medium\} \end{aligned} \tag{9}$$

The solution set is the five tuples in (9):

$$(MIB, Large), (MIB, Medium), (MIB, Small),$$
$$(STW, Large), (STW, Medium)$$

Constraint propagation does not produce a domain reduction (verifiable by inspection).

If, instead, the domain restriction for v_2 at run-time is $R_2 = \{Small\}$ (but still $R_1 = \Omega_1$), then

$$\begin{aligned} \boldsymbol{C}_1^{\mathcal{U},R} &= \{MIB\} \times \{Small\} \\ \boldsymbol{C}_2^{\mathcal{U},R} &= \emptyset \quad (= \{STW\} \times (\{Small\} \setminus \{Small\})) \end{aligned}$$

The solution set is now the tuple $(MIB, Small)$. Constraint propagation produces a domain reduction of R_1 to $\{MIB\}$.

[16] Constraint propagation on a solution set in the form of a c-tuple means intersecting the c-tuple with the given run-time restriction tuple R. In a product configuration context the values will most often be ordered. Hence, set operations can use binary search and are relatively efficient

[17] The unaffected property *Color* could take any value. I take this up in Section 3.5

3.2 T-Shirt Example: MIB

The constraint $MIB \rightarrow Black$ in the t-shirt example could be formulated as three exclusions against the original global domain of the property *Color* yielding a negative variant table with three rows, $k = 2$, $v_1 = Print$, and $v_2 = Color$:

$$\mathcal{U} = \begin{pmatrix} MIB & Red \\ MIB & White \\ MIB & Blue \end{pmatrix}$$

Using the global domains given in Example 1, this means that $\pi_1(\mathcal{U}) = \{MIB\}$ and $\pi_2(\mathcal{U}) = \{Red, White, Blue\}$:

$$\begin{aligned} \boldsymbol{C}_1^{\mathcal{U},R} &= \{STW\} \times \{Black, Red, White, Blue\} \\ \boldsymbol{C}_2^{\mathcal{U},R} &= \{MIB\} \times \{Black\} \end{aligned} \tag{10}$$

It still holds that $\overline{\mathcal{U}} = \emptyset$. So the solution set is still directly given by the first component in (7) and its decomposition in (8), and it follows from (10) that there are five tuples in the solution set. Again, constraint propagation does not produce a domain reduction.

If, instead, the domain restriction for v_2 at run-time is $R_2 = \{Red, Blue\}$ (but still $R_1 = \Omega_1$), then

$$\begin{aligned} \boldsymbol{C}_1^{\mathcal{U},R} &= \{STW\} \times \{Red, Blue\} \\ \boldsymbol{C}_2^{\mathcal{U},R} &= \emptyset \quad (= \{MIB\} \times (\{Red, Blue\} \setminus \pi_2(\mathcal{U}))) \end{aligned}$$

The solution set $\mathcal{S}^{\mathcal{U},R}$ is now two tuples (STW, Red) and $(STW, Blue)$. Constraint propagation produces a domain reduction of R_1 to $\{STW\}$

3.3 Original T-Shirt Example in a Single Negative Table $\mathcal{U}$

In the sequel, the order of the complete set of constraint variables is $v_1 = Color$, $v_2 = Size$, $v_3 = Print$.

Let T be the variant table of the t-shirt in its original positive form given in Table 1. $\mathcal{T}$ has 11 solutions out of a possible 24. Thus complementing the table with respect to the global domains in Example 1 yields a negative table $\mathcal{U}$ with thirteen rows, $k = 3$, and

$$\mathcal{U} = \begin{pmatrix} Black & Small & STW \\ Red & Small & MIB \\ Red & Medium & MIB \\ Red & Large & MIB \\ Red & Small & STW \\ White & Small & MIB \\ White & Medium & MIB \\ White & Large & MIB \\ White & Small & STW \\ Blue & Small & MIB \\ Blue & Medium & MIB \\ Blue & Large & MIB \\ Blue & Small & STW \end{pmatrix}$$

Let $R_1 = \Omega_1$ ($Color$), $R_2 = \Omega_2$ ($Size$), and $R_3 = \Omega_3$ ($Print$). $\pi_j(\mathcal{U}) = R_j$ for every j, therefore all $\overline{\pi_j(\mathcal{U})}^R = \emptyset$, and all $\boldsymbol{C}_j^{\mathcal{U},R} = \emptyset$. In this case, $\mathcal{T} = \overline{\mathcal{U}}$, and the tuples in Table 1 are just the solution set.

3.4 Extending the T-Shirt Model

3.4.1 Extending the Global Domains

First, assume that after $\mathcal{U}$ in Section 3.3 has been maintained, the global domains of the properties are extended – without adding any exclusions ($\mathcal{U}$ remains unchanged)[18] – as follows:

- $Yellow$ and $DarkPurple$ are added to Ω_1 ($Color$)

$$\Omega_1 = \{Black, Red, White, Blue, Yellow, DarkPurple\}$$

- XL and XXL are added to Ω_2 ($Size$)

$$\Omega_2 = \{Large, Medium, Small, XL, XXL\}$$

- $none$ is added to Ω_3 ($Print$)

$$\Omega_3 = \{MIB, STW, none\}$$

Let the run-time domain restrictions reflect the changed global domains $R_1 = \Omega_1$, $R_2 = \Omega_2$, and $R_3 = \Omega_3$. Since $\mathcal{U}$ is not changed, $\pi(U)$ does not change either. It still holds that

$$\begin{aligned}\pi_1(\mathcal{U}) &= \{Black, Red, White, Blue\}\\ \pi_2(\mathcal{U}) &= \{Large, Medium, Small\}\\ \pi_3(\mathcal{U}) &= \{MIB, STW\}\end{aligned}$$

Hence, $\overline{\mathcal{U}}$ is not changed either (still equal to Table 1), and with

$$\begin{aligned}\overline{\pi_1(\mathcal{U})}^R &= \{Yellow, DarkPurple\}\\ \overline{\pi_2(\mathcal{U})}^R &= \{XL, XXL\}\\ \overline{\pi_3(\mathcal{U})}^R &= \{none\}\end{aligned}$$

(8) now yields

$$\begin{aligned}\boldsymbol{C}_1^{\mathcal{U},R} &= \{Yellow, DarkPurple\} \times R_2 \times R_3\\ \boldsymbol{C}_2^{\mathcal{U},R} &= \pi_1(\mathcal{U}) \times \{XXL, XL\} \times R_3\\ \boldsymbol{C}_3^{\mathcal{U},R} &= \pi_1(\mathcal{U}) \times \pi_2(\mathcal{U}) \times \{none\}\end{aligned} \tag{11}$$

In this example, both components $R \setminus \pi(\mathcal{U})$ and $\overline{\mathcal{U}} \cap R$ in (7) are non-empty and contribute to the solution set $\mathcal{S}^{\mathcal{U},R}$.

Note that $|R| = 6 \times 5 \times 3 = 90$, and (looking at (11)) the total number of solutions s for $\mathcal{U}$ is

$$s = |\boldsymbol{C}_1^{\mathcal{U},R}| + |\boldsymbol{C}_2^{\mathcal{U},R}| + |\boldsymbol{C}_3^{\mathcal{U},R}| + |\overline{\mathcal{U}}| = 30 + 24 + 12 + 11 = 77$$

3.4.2 Extending the Constraints

If product management notices that $Yellow$ does not go with MIB, then corresponding exclusions must be added to $\mathcal{U}$. This can be done in several ways. In this example, I assume that $Yellow$ and the corresponding exclusions are added before any of the other changes to the domains are made[19]. Then, the global domain given in Example 1 for the product property $Color$, denoted by Ω_1, is augmented by the value $Yellow$, and Ω_2 ($Size$) and Ω_3 ($Print$) remain unchanged. The following exclusions must be added to $\mathcal{U}$ in 3.3:

$$\begin{aligned}&\neg(Yellow, Small, MIB)\\ &\neg(Yellow, Medium, MIB)\\ &\neg(Yellow, Large, MIB)\\ &\neg(Yellow, Small, STW)\end{aligned}$$

In this situation

$$\pi_1(\mathcal{U}) = \{Yellow, Black, Red, White, Blue\}$$

changes, and $\overline{\mathcal{U}}$ changes accordingly. Two values are added to allow the color of $Yellow$ in sizes $Large$ and $Medium$ with the print STW. So $|\overline{\mathcal{U}}| = 13$. (11) holds with the modified version of $\pi_1(\mathcal{U})$. Again, both components in (7) contribute to the solution set $\mathcal{S}^{\mathcal{U},R}$, and after adding all remaining new values

$$\begin{aligned}\boldsymbol{C}_1^{\mathcal{U},R} &= \{DarkPurple\} \times R_2 \times R_3\\ \boldsymbol{C}_2^{\mathcal{U},R} &= \pi_1(\mathcal{U}) \times \{XXL, XL\} \times R_3\\ \boldsymbol{C}_3^{\mathcal{U},R} &= \pi_1(\mathcal{U}) \times \pi_2(\mathcal{U}) \times \{none\}\end{aligned} \tag{12}$$

As above, $|R| = 6 \times 5 \times 3 = 90$. Looking at (12), the total number of solutions s for $\mathcal{U}$ is now

$$s = 15 + 30 + 15 + 13 = 73$$

(The four new exclusions are subtracted from the solutions in Section 3.4.1)

3.5 Excursion on Table Compression

From Sections 3.1 and 3.2 it is clear that the t-shirt table can be expressed in much more compact form by looking at the two constraints in two individual negative variant tables than at the single table in Section 3.3. In both Sections 3.1 and 3.2, the solution set is defined only by the decomposition of $(R \setminus \pi(U))$ into c-tuples. An overall solution set can be obtained by expanding these solution sets to account for the respective unconstrained property, and then intersecting the resulting two expanded solution sets.

Let the properties be ordered as $v_1 = Color$, $v_2 = Size$, and $v_3 = Print$, and the solution space $\boldsymbol{\Omega}$ given by the global domains in Example 1. In Section 3.1 the property $v_1 = Color$ is unconstrained. Set $\pi_1(\mathcal{U}) = \Omega_1$ (which implies $\overline{\pi_1(\mathcal{U})}^R = \emptyset$ for all $R \subset \boldsymbol{\Omega}$). Then, the decomposition of $R \setminus \pi(U)$ in (9) now results in three c-tuples $C'_{j(STW)}$ (one of them empty by construction):

$$\begin{aligned}C'_{1(STW)} &:= \emptyset \quad (= \overline{\pi_1(\mathcal{U})}^R \times \Omega_2 \times \Omega_3)\\ C'_{2(STW)} &:= \Omega_1 \times \{Large, Medium\} \times \Omega_3\\ C'_{3(STW)} &:= \Omega_1 \times \{Small\} \times \{MIB\}\end{aligned} \tag{13}$$

Similarly, for Section 3.2 (10) can be replaced by:

$$\begin{aligned}C'_{1(MIB)} &:= \{Black\} \times \Omega_2 \times \Omega_3)\\ C'_{2(MIB)} &:= \emptyset \quad (= \{Black\} \times \overline{\pi_2(\mathcal{U})}^R \times \Omega_3)\\ C'_{3(MIB)} &:= \{Red, White, Blue\} \times \Omega_2 \times \{STW\}\end{aligned} \tag{14}$$

The solution set $\mathcal{S}^{\mathcal{T}}$ of the original positive $\mathcal{T}$ given in Table 1 is

[18] Leaving $\mathcal{U}$ unchanged implicitly changes the underlying positive constraints. It now no-longer holds that $MIB \rightarrow black$, This is taken to be an intended consequence of using a negative variant table in a product model

[19] This assumption is made to keep the example simple. The general problem of achieving a good compression directly from a positive table is addressed in [6]

the intersection of the solution sets given by (13) and (14):

$$\begin{aligned}\mathcal{S}^{\mathcal{T}} = \quad & C'_{2(STW)} \cap C'_{1(MIB)} \quad \cup \\ & C'_{2(STW)} \cap C'_{3(MIB)} \quad \cup \\ & C'_{3(STW)} \cap C'_{1(MIB)} \quad \cup \\ & C'_{3(STW)} \cap C'_{3(MIB)}\end{aligned}$$

$$\begin{aligned}= \quad & \{Black\} \times \{Large, Medium\} \times \Omega_3 \quad \cup \\ & \{Red, White, Blue\} \times \{Large, Medium\} \times \{STW\} \quad \cup \\ & \{Black\} \times \{Small\} \times \{MIB\} \quad \cup \\ & \emptyset \quad (= \{Red, White, Blue\} \times \{Small\} \times \emptyset)\end{aligned}$$

The 11 tuples of $\mathcal{T}$ are represented as 3 c-tuples. The complexity of this compares favorably with the MDD representation in [1] for the same example. The representation also compares favorably with the compression of the table to *c-nodes* introduced in [7] if a suitable heuristic is applied. Thus, it should be possible to recover this compact representation from the full table. This is a topic of [6].

4 Arc Consistency for Negative Variant Tables

Let a negative table $\mathcal{U}$ of arity k and a finite run-time restriction tuple R be given. (I assume in the sequel that $\overline{\pi_j(\mathcal{U})}^R$ is finite for all columns.)

One approach at arc consistency with $\mathcal{U}$ is to use (7) directly at run-time to discard any tuples in $\mathcal{U}$ from R constructing $\pi(\mathcal{S}^{\mathcal{U},R})$ at the same time. The STR-Negative algorithm [9] is such an algorithm.

Here, I propose an alternative approach based on (7) and the decomposition (8). As already noted, a further restriction of R is only possible if the c-tuples in (8) allow a restriction. The lemma and its corollary below provide a simple necessary condition for this.

Lemma 3 *If $\overline{\pi_p(\mathcal{U})}^R \neq \emptyset$ for some column with index p, then $\forall j \neq p : \pi_j(R \cap \mathcal{S}^{\mathcal{U}}) = R_j$, i.e., no further reduction of any of the other domains is possible by constraint propagation using $\mathcal{U}$.*

Proof Suppose that the premise of the lemma holds. Without loss of generality, sort the columns such that $p = 1$. Then,

$$\boldsymbol{C}_1^{\mathcal{U},R} = \overline{\pi_1(\mathcal{U})}^R \times R_2 \times R_3 \times \ldots \times R_k$$

Any value in $\overline{\pi_1(\mathcal{U})}^R \neq \emptyset$ supports all values in R_j for $j \geq 2$. ∎

This has a trivial but important consequence:

Corollary 4 *If $\overline{\pi_j(\mathcal{U})}^R \neq \emptyset$ for more than one column, then no reduction of domains is possible by constraint propagation using $\mathcal{U}$.*

Lemma 3 and Corollary 4 generalize (1) to state that a reduction via constraint propagation is possible, if at least all but one of the domains are sufficiently restricted at run-time so that they lie within the range of $\pi(\mathcal{U})$, i.e., $\overline{\pi_j(\mathcal{U})}^R = \emptyset$.

Recall that $\pi(\mathcal{U})$ is determined when maintaining the variant table, whereas R is only known at run-time.

The examples in Section 3 illustrate that either both or only one of the components in (7) need to be considered at run-time. Let a negative table $\mathcal{U}$ of arity k and a run-time restriction tuple R be given. Then, three cases can happen:

1. $\overline{\mathcal{U}} = \emptyset$. In this case, arc consistency is obtained solely by computing the decomposition (8) of k c-tuples at run-time. Constraint propagation is achieved through directly intersecting these with R in a way that quickly tests the precondition of Lemma 3 and Corollary 4.
2. $R \setminus \pi(U) = \emptyset$. In this case, the GAC algorithm implemented in the configurator is applied to $\overline{\mathcal{U}}$ (or a GAC-negative algorithm is applied to $\mathcal{U}$ if this seems better).
3. Both components $(R \setminus \pi(U))$ and $\overline{\mathcal{U}}$ are non-empty. In this case, the component $(R \setminus \pi(U))$ is decomposed and processed as in the first case, testing the pre-conditions in Lemma 3 and Corollary 4 at the same time. The constraint propagation methods of the configurator need to be applied to $\overline{\mathcal{U}}$, if this is still indicated after processing the first part.

5 Double Negation of a Positive Variant Table $\mathcal{T}$

In the sequel, I take $\mathcal{T}$ to be a positive table. I denote the negation of $\mathcal{T}$ as the negative table $\neg\mathcal{T} := \pi(\mathcal{T}) \setminus \mathcal{T}$. Trimming any run-time restrictions to $\pi(\mathcal{T})$, $\neg\mathcal{T}$ (as a negative table) and $\mathcal{T}$ have the same solution space, and constraint propagation on $\mathcal{T}$ can be replaced by constraint propagation on $\neg\mathcal{T}$. The approach seems advantageous, if $\neg\mathcal{T}$ has a decomposition of the solution set (7) with a non-empty first part (8), i.e., if $\overline{\neg\mathcal{T}}$ is smaller than $\mathcal{T}$. I refer to this approach as *double negation*.

As an example, consider the extended model of the t-shirt as specified in Section 3.4.2. A representation as a positive table $\mathcal{T}$ has 73 tuples (rows) as inidicated there. Negating $\mathcal{T}$ against the extended global domains yields a table $\neg\mathcal{T}$ with 17 rows. $\neg\mathcal{T}$ is just the table $\mathcal{U}$ of Section 3.4.2 (and thus $\overline{\mathcal{U}}$ there corresponds to $\overline{\neg\mathcal{T}}$ here). Thus, as outlined there, $\overline{\neg\mathcal{T}}$ has a decomposition as in (7). It has 13 rows and the c-tuples indicated in (12).

Double negation is not only an approach at compression for a table with a small complement, but also allows applying Lemma 3 and Corollary 4 as a simple test, before actually evaluating the remaining doubly negated table $\overline{\neg\mathcal{T}}$. The process could be iterated, i.e., $\overline{\neg\mathcal{T}}$ could be doubly negated in turn. A fixed point occurs if $\mathcal{T} = \overline{\neg\mathcal{T}}$.

6 Implementation/Work in Progress

I have implemented the approach for treating negative tables I describe here, including *double negation*, within a Java prototype addressing the greater context of compressing and compiling variant tables [6].

As customers so far have not had the opportunity of maintaining negative tables directly in product configuration models, I have no real data to evaluate this approach. Experiences are currently limited to testing for functional correctness. Besides testing with exemplary tables such as variations of the t-shirt, I have applied the *double negation* approach to 238 SAP VC variant tables. These tables are also the basis for the evaluation of the approach at compression in [6].

Complementing a sparse table is not feasible if the solution space is large. The implementation performs complementation on a representation I term a *Variant Decision Diagram − VDD*, and produces a result in a compressed form (see [6]). I have so far limited attempts at double negation to those of the 238 tables where the number of tuples of the complement is smaller than the number of rows (tuples) in the original (relational) table. The result is given in Table 2. All tables where this criterium does not apply are counted as *Skipped*. Of the remaining tables where double negation was attempted, those where a reduction was realized (i.e., the VDD of $\mathcal{T}$ was larger than

that of $\overline{\neg\mathcal{T}}$) are counted as *Reduced*. The total number of tables in the model and the number of those neither skipped nor reduced are given for completeness.

I give more detailed results of tests with the implementation in [6]. As pointed out there, it is not yet clear whether double negation yields a gain over the general compression approach.

Table 2. Tables amenable to double negation

Number of tables			
Total	Skipped	Reduced	Not reduced
238	181	39	18

7 Conclusion

I presented an approach to handling configuration constraints defined by negative variant tables that can be integrated as an add-on to an existing configurator, such as the SAP VC, with little risk. The original mechanisms and architecture of the configurator are not touched.

SAP customers specifically request that a constraint for a negative table not be sensitive to subsequent changes to the global domains of the affected product properties. This approach meets that requirement. As the footnote to the example in Section 3.4.1 points out, this may mean that some implicit positive constraints, valid before a change to the global domains, may no-longer be valid afterwards.

This emphasizes a modeling aspect of the problem: Is it feasible and beneficial to offer the option of negative variant tables as part of the modeling environment in this fashion? This may be a tricky question. The implemented prototype offers a means for experimenting with the functionality, and thus a basis for discussing the merits and demerits of negative variant tables.

One main idea, implicit in the approach, is to make use of the distinction between information known at maintenance-time to the modeler (the table content) and information known at run-time to the configurator (the current domain restrictions). For a negative table $\mathcal{U}$ the finite column domains $\pi_j(\mathcal{U})$ can be determined at maintenance time. For a given run-time restriction of the domains R the complements of the column domains to R (denoted by $\overline{\pi_j(\mathcal{U})}^R$) must be calculated at run-time. This calculation is efficient, as the required set-operations can make use of the natural order imposed on the domain values by the business context.

If more than one $\overline{\pi_j(\mathcal{U})}^R$ is non-empty at run-time, constraint propagation does not yield a further restriction of the run-time domains. If one $\overline{\pi_j(\mathcal{U})}^R$ is non-empty, then only the run-time restriction for that column can be further restricted. When a restriction is possible by constraint propagation, the GAC algorithm only needs to be applied to the positive table $\overline{\mathcal{U}}$, the complement of $\mathcal{U}$ to $\pi(\mathcal{U})$. The remaining part of the solution set of $\mathcal{U}$ is the set-difference between two c-tuples (Cartesian products). I showed in Proposition 2 that such a set-difference can be decomposed into k disjoint c-tuples, where k is the arity of $\mathcal{U}$. Constraint propagation on c-tuples is again based on set operations that are assumed to be efficient, given the value ordering on the domains.

Of course, the approach also applies to the general case that negative tables are merely meant as a short-cut to maintaining an otherwise lengthy positive variant table.

The approach can also be applied to a positive variant table $\mathcal{T}$ by negating it, and treating its complement $\neg\mathcal{T}$ as a negative table. In this process $\neg\mathcal{T}$ is negated again ($\overline{\neg\mathcal{T}}$), which I refer to as *double negation*. The purpose of double negation is that it may yield a beneficial (partial) compression of the table, i.e., the table $\overline{\neg\mathcal{T}}$ may be smaller than $\mathcal{T}$, with the remaining part of the solution set being a set of k c-tuples that additionally allows testing whether constraint propagation is possible at all for a given run-time restriction R.

ACKNOWLEDGEMENTS

I would like to thank all that took the time to comment on previous versions of this paper, and have thus contributed to its current form. This includes the anonymous reviewers, but also colleagues at SAP, particularly Conrad Drescher and Andreas Krämer. I tired to incorporate all their suggestions. Any remaining flaws and dis-improvements are solely my responsibility.

REFERENCES

[1] H.R. Andersen, T. Hadzic, and D. Pisinger, 'Interactive cost configuration over decision diagrams', *J. Artif. Intell. Res. (JAIR)*, **37**, 99–139, (2010).
[2] C. Bessiere, 'Constraint propagation', in *Handbook of Constraint Programming*, eds., F. Rossi, P. van Beek, and T. Walsh, chapter 3, Elsevier, (2006).
[3] U. Blumöhr, M. Münch, and M. Ukalovic, *Variant Configuration with SAP, second edition*, SAP Press, Galileo Press, 2012.
[4] K. Ferland, *Discrete Mathematics*, Cengage Learning, 2008.
[5] A. Haag, 'Chapter 27 - product configuration in sap: A retrospective', in *Knowledge-Based Configuration*, eds., Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, 319 – 337, Morgan Kaufmann, Boston, (2014).
[6] A. Haag, 'Column oriented compilation of variant tables', in *Proceedings of the 17th International Configuration Workshop, Vienna, Austria, September 10-11, 2015.*, pp. 89–96, (2015).
[7] G. Katsirelos and T. Walsh, 'A compression algorithm for large arity extensional constraints', in *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, ed., Christian Bessiere, volume 4741 of *Lecture Notes in Computer Science*, pp. 379–393. Springer, (2007).
[8] C. Lecoutre, 'STR2: optimized simple tabular reduction for table constraints', *Constraints*, **16**(4), 341–371, (2011).
[9] Honbo Li, Yanchun Liang, Jinsong Guo, and Zhanshan Li, 'Making simple tabular reductionworks on negative table constraints', in *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.*, eds., Marie desJardins and Michael L. Littman. AAAI Press, (2013).

Column Oriented Compilation of Variant Tables

Albert Haag[1]

Abstract. The objective of this work is to improve variant table evaluation in a configurator by compiling/compressing the tables individually to reduce both processing time and space. The main result is a proposed simple heuristic for decomposing the variant table into subtables and a representation linking these subtables in a directed acyclic graph (*DAG*). The size of the compression obtained by this heuristic for examples used in [2, 10] is comparable to that achieved there. However, a formal analysis of complexity has not yet been completed. A prototype implemented in Java exists. Objectives in designing it were to keep it completely decoupled from any particular configurator, while using little machinery in order to keep software maintenance costs low. Testing both on abstract examples and on tables that resemble real customer data is ongoing and looks promising. Non-atomic table cells (such as real intervals, or value sets) are supported. My approach to negative variant tables [8] has been incorporated into the implementation.

1 Preamble

Following the usage of the *SAP Variant Configurator - SAP VC* [4] I term a table that lists all valid combinations of properties of a product as a *variant table*. One use of a variant table in configuration is as a table constraint. However, variant tables and their maintenance by modelers entail some special considerations that have implications beyond what is formally captured by that concept:

- The product properties and the values referred to have a business meaning outside of configuration. Value domains for a product property are naturally maintained in a defined sort order
- Variant tables may be stored in a database table outside the model. A data definition in a database management system (*DBMS*) may exist that defines database keys etc.
- Tables will often be *relational*, i.e., table cells will be atomic values (*strings* or *numbers*), but non-atomic entries may occur [2]
- Customers have the tendency to maintain large wide tables, i.e., normalization is often sacrificed in favor of fewer tables. In such cases, compression techniques seem particularly advantageous
- A variant tables can have its own individual update cycle. The overall model should not need to be compiled or otherwise processed each time a change is made to a table

The relevant functionality a configurator has to provide regarding variant tables is to

- restrict domains via contraint propagation (general arc consistency *GAC* (see [3])), treating the variant table as a constraint relation.
- query the table using a key defined in the DBMS
- iterate over the current solution set of the table given domain restrictions for the associated product properties

I assume that an existing (legacy) configurator has already implemented this functionality. This will include means for efficiently testing membership in a domain (*memberp*), testing if domains intersect (*intersectsp*), and for calculating the intersection of domains (*intersection*).

2 Introduction and Notation

In [8] I look at tables that list excluded combinations of product properties. I term these *negative variant tables*. The techniques and the associated prototypical implementation I present here cover that approach as well. Here, except in Section 7, I limit the exposition to positive tables.

For simplicity, I take a positive variant table $\mathcal{T}$ to be given as a relational table of values (atoms). I relax the assumption about $\mathcal{T}$ being relational later in Section 3.3. If $\mathcal{T}$ has k columns and r rows it is an $r \times k$ array of values: $\mathcal{T} = \left(a_{ij}\right); i = 1 \ldots r; j = 1 \ldots k$. k is the *arity* of $\mathcal{T}$. Each column is mapped to a product property such as *Color*, *Size*, ... The product properties are denoted by $v_j : j = 1 \ldots k$.

Following notation in [8], I define the column domains $\pi_j(\mathcal{T})$ as the set of all values occurring in the j-th column of T:[3]

$$\pi_j(\mathcal{T}) := \bigcup_{i=1}^{r} \{a_{ij} \in \mathcal{T}\} \tag{1}$$

I define s_j as the number of elements in $\pi_j(\mathcal{T})$:

$$s_j := |\pi_j(\mathcal{T})| \tag{2}$$

I call

$$\pi(\mathcal{T}) := \pi_1(\mathcal{T}) \times \ldots \times \pi_k(\mathcal{T})$$

the global domain tuple for $v_1, \ldots, v_k$ and I denote a run-time domain restriction for the product property v_j as $R_j \subseteq \pi_j(\mathcal{T})$.

For ease of notation, I refer to any subset of $\pi(\mathcal{T})$ that is a Cartesian product as a *c-tuple*. Both $\pi(\mathcal{T})$ itself and the tuple of run-time restrictions $R := R_1 \times \ldots \times R_k$ are c-tuples.

$\mathcal{T}$ can be seen as a set of value tuples and, as such, $\mathcal{T} \subseteq \pi(\mathcal{T})$, but $\mathcal{T}$ is not necessarily a c-tuple. In the special case that $\mathcal{T}$ itself is a c-tuple[4], i.e., $\mathcal{T} = \pi_1(\mathcal{T}) \times \ldots \times \pi_j(\mathcal{T})$, then

$$s := \sum_{j=1}^{k} s_j \tag{3}$$

[1] SAP SE, Germany, email: albert.haag@t-online.de

[2] The SAP VC allows real-valued intervals and and sets of values in table cells. Such a table cannot be transparently stored as a relation table in a DBMS

[3] $\pi_j(\mathcal{T})$ can be seen as the *projection* of $\mathcal{T}$ for the j-th column

[4] In this case, it holds that for any given c- tuple (run-time restriction) R

$$\pi(\mathcal{T} \cap R) = \mathcal{T} \cap R$$

Juha Tiihonen, Andreas Falkner, and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

values suffice to represent the $N := (\prod_{j=1}^{k} s_j)$ tuples in $\mathcal{T}$. In this case, the c-tuple $\pi(\mathcal{T})$ is a compressed way of representing the array a_{ij}. This observation is central to attempts to compress a given table into a disjoint union[5] of as few as possible c-tuples. It has been utilized in various other work. Notably, I cite [10, 6] in this context.

When $\mathcal{T}$ is viewed as a constraint relation, $v_1 \dots v_k$ are just the constraint variables, and each value $x \in \pi_j(\mathcal{T})$ maps to a proposition $p(v_j, x)$ that states that v_j can be consistently assigned to x: $p(v_j, x) \models (v_j = x)$. In this case, a row (tuple) r_i in $\mathcal{T}$ directly maps to a conjunction of such propositions: $r_i = (a_{i1}, \dots, a_{ik}) \models \tau_i := p(v_1, a_{i1}) \wedge \dots \wedge p(v_k, a_{ik})$, and $\mathcal{T}$ itself represents the disjunction: $\mathcal{T} \models \bigvee_{i=1}^{r} \tau_i$.

Given the definition of s in (3), there are s distinct propositions associated with $\mathcal{T}$, one for each value in each $\pi_j(\mathcal{T})$. Hence, given any value assignment to these s propositions, $\mathcal{T}$, seen as a logical expression implementing the constraint relation, will evaluate to 1 ($\top$/true) or 0 ($\bot$/false), and $\mathcal{T}$ defines a Boolean function:

$$\mathcal{F} : 2^{\pi_1(\mathcal{T}) \times \dots \pi_k(\mathcal{T})} \rightarrow \{0, 1\} \tag{4}$$

$\mathcal{F}$ can be represented by a BDD (*Ordered Binary Decision Diagram*) or one of its cousins [12]. BDDs have the enticing property that finding the right ordering of their Boolean variables (the propositions $p(v_j, x)$) can lead to a very compact representation. Furthermore, this representation can potentially be found by existing agnostic optimization algorithms. The complexity of this optimization is high, and heuristics are employed in practice. The construction of an optimal BDD is not suitable for configuration run-time, but must be performed in advance. Hence, this approach is referred to as a *compilation approach*. For configuration, this has been pursued in [9]. The approach using *multi-valued decision diagrams* (*MDD*) [2] is related to the BDD approach. *Zero-Suppressed decision diagrams* (*ZDDs*) [12] are another flavor of *BDD*, which I refer to again below.

From a database point of view, approaches based on compression that allow fast reading but slow writing have been developed, among them column-oriented databases [14]. The SAP HANA database supports column orientation as well. My work, here, is not directly based on database techniques, although thinking about column-orientation was the trigger for the heuristics detailed in Section 4[6].

Both the BDD approaches and the database approaches entail a maintenance-time transformation into a compact representation that facilitates run-time evaluation, and both strive for a compact representation (*"hard to write, easy to read"*). This would also apply to various approaches at identifying c-tuple subsets of $\mathcal{T}$ whether with the explicit notion of achieving compression [10] or of simply speeding up constraint propagation (GAC) algorithms [6].

Thus, all these approaches could be termed as compression or compilation or as both. Indeed, I conjecture that the ultimately achievable results to be more or less identical, up to differences forced by the respective formalism, which may be unfavorable in some circumstances. For example, my experiences suggest that a BDD may be less suitable than a ZDD for compiling a single table constraint, because in the latter propositions need only be represented where they occur in positive form (see [12]). The approach I follow here is motivated by looking at ways of decomposing a table into disjoint subtables based on a particular heuristic.

In Section 3, I introduce the basic approach to decomposing a table. In Section 4, I discuss the heuristic. My running example is a (single) variant table listing all variants of a t-shirt. The example is taken and adapted further from [2]. The t-shirt has three properties *Color* (v_1), *Size* (v_2), and *Print* (v_3) with global domains:

- $\pi_1(\mathcal{T}) := \{Black, Blue, Red, White\}$
- $\pi_2(\mathcal{T}) := \{Large, Medium, Small\}$
- $\pi_3(\mathcal{T}) := \{MIB(\text{Men in Black}), STW(\text{Save the Whales})\}$

Of the 24 possible t-shirts only 11 are valid due to constraints that state that MIB implies $black$ and STW implies $\neg small$ as depicted in table (1). In Section 5, I extend this example to be slightly more complex.

I have implemented a prototype in Java that meets the functionality requirements listed above. Here, I refer to it as the *VDD prototype*. It functions standalone, independent of any particular configurator. A feature of this implementation is that it can be selectively applied to some tables, while processing others with the existing means of the configurator. Results obtained using this prototype validate the approach (see Section 6). In Section 7, I comment on results for *double negation* of variant tables, an approach I develop in [8]. Real runtime performance evaluations have not been done, but in Section 8 I discuss what results I have. I close this paper with an outlook and conclusions (Section 9).

Finally, a disclaimer: While the motivation for this work lies in my past at SAP and is based on insights and experiences with the product configurators there [4, 7], all work on this paper was performed privately during the last two years after transition into partial retirement. The implementation is neither endorsed by SAP nor does it reflect ongoing SAP development.

3 Decomposition

3.1 Column Oriented Decomposition

Let s be defined as in (3). Given a table $\mathcal{T}$ of arity k with r rows $r_i = (a_{i1}, \dots, a_{ik})$ and selecting one of the s propositions $p(v_j, x)$ associated with $\mathcal{T}$, then $\mathcal{T}$ can be decomposed into those rows in which x occurs in column j and those where it doesn't. Define $\mathcal{L}(\mathcal{T}, j, x)$ as the sub-table of $\mathcal{T}$ consisting of those rows that do not reference $p(v_j, x)$:

$$\mathcal{L}(\mathcal{T}, j, x) := \{r_i = (a_{i1}, \dots, a_{ik}) \in \mathcal{T} \quad | \quad a_{ij} \neq x\} \tag{5}$$

$\mathcal{L}(\mathcal{T}, j, x)$ has the same arity k as $\mathcal{T}$. In the complementary subtable $\mathcal{T} \setminus \mathcal{L}(\mathcal{T}, j, x)$ all values in the j-th column are equal to x by construction. Define $\mathcal{R}(\mathcal{T}, j, x)$ as the sub-table of arity $(k-1)$ obtained by removing the j-th column from $\mathcal{T} \setminus \mathcal{L}(\mathcal{T}, j, x)$:

$$\mathcal{R}(\mathcal{T}, j, x) := \{(a_{ih}) \subset (\mathcal{T} \setminus \mathcal{L}(\mathcal{T}, j, x)) \quad | \quad h \neq j\} \tag{6}$$

Given a table $\mathcal{T}$ and a proposition $p(v_j, x)$ associated with $\mathcal{T}$, then I call $\mathcal{L}(\mathcal{T}, j, x)$ defined in (5) the *left sub-table* of $\mathcal{T}$ and $\mathcal{R}(\mathcal{T}, j, x)$ defined in (6) the *right sub-table* of $\mathcal{T}$. Either $\mathcal{L}(\mathcal{T}, j, x)$ and/or $\mathcal{R}(\mathcal{T}, j, x)$ may be empty.

The variant table of the 11 variants of the t-shirt example is shown in Table 1. It illustrates a decomposition of this table table based on the proposition $p(v_2, Medium)$. $\mathcal{L}(\mathcal{T}, 2, Medium)$ is in boldface, and $\mathcal{R}(\mathcal{T}, 2, Medium)$ is underlined. (The *value block* of cells $\{a_{i2} \in \mathcal{T} \quad | \quad a_{i2} = Medium\}$ is in italics.)

[5] I exclusively use the term *disjoint union* to refer to a union of disjoint sets [5]. I denote the disjoint union of two sets A and B by $A \cup B$ which implies that $A \cap B = \emptyset$

[6] It would be interesting to investigate whether a column-oriented database could in itself be beneficially employed in this context. This was proposed by colleague at SAP some time ago, but has not been followed up on

Table 1. Basic decomposition of a table

Color	Size	Print
Black	**Small**	**MIB**
Black	*Medium*	MIB
Black	**Large**	**MIB**
Black	*Medium*	STW
Black	**Large**	**STW**
White	*Medium*	STW
White	**Large**	**STW**
Red	*Medium*	STW
Red	**Large**	**STW**
Blue	*Medium*	STW
Blue	**Large**	**STW**
Blue	**Small**	**STW**

The decomposition process can be continued until only empty subtables remain. The question, which propostion (value block) to decompose on next at each non-empty subtable will depend a suitable heuristic (see Section 4).

3.2 Variant Decision Diagram - *VDD*

A variant table can be represented as a decomposition tree. The root represents the entire table. It is labeled with a first chosen proposition $p(v_{j_1}, x_1)$. It has two children. One, termed the *left child*, represents the left sub-table $\mathcal{L}(\mathcal{T}, j_1, x_1)$. The other, termed the *right child*, represents the right sub-table $\mathcal{R}(\mathcal{T}, j_1, x_1)$. Each of these children can in turn have children if it can be decomposed further. An empty left child is labeled by a special symbol $\bot$. An empty right child is labeled by a special symbol $\top$. (All leaves of a decomposition tree are labeled either by $\bot$ or $\top$.) Figure 1 shows a graphic depiction[7]. It also shows the further decomposition of $\mathcal{R}(\mathcal{T}, j_1, x_1)$, here indicating that it has two empty children.

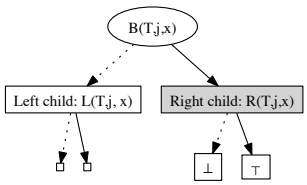


Figure 1. Basic scheme of a decomposition

Identical subtables may arise at different points in the decomposition tree. A goal of minimal representation is to represent these multiple occurrences only once by transforming the decomposition tree into a directed acyclic graph (*DAG*), which I call a *VDD* or *variant decision diagram*. All leaves can be identified with one of two predefined nodes also labeled $\bot$ and $\top$. Subsequently, all nodes labeled by the same proposition $p(v_j, x)$ that have identical children are represented by re-using one shared node. This reduction can be accomplished by an algorithm in the spirit of *Algorithm R* in [12].

[7] For simplicity in creating the graph, the label of the root node is given as $B(T, j, x)$ for $p(v_{j_1}, x_1)$

Figure 2 shows an entire VDD[8].[9]

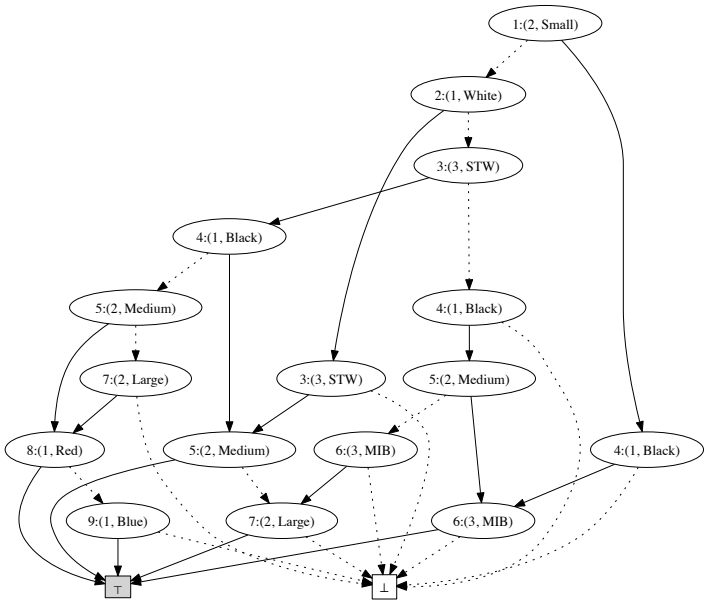


Figure 2. VDD of t-shirt using heuristic $h1$ (Section 4)

In Figure 2 each node is labeled in the form $\langle p : (j, val)\rangle$, where (j, val) is the column/value pair that denotes the proposition $p(v_j, x)$, and p is a unique identifier for the node/proposition. The identifiers are contiguously numbered. Thus, the set of propositions for $\mathcal{T}$ can be seen as totally ordered according to this numbering. The ordering underlying the graph in Figure 2 is

$$p(v_2, Small), p(v_1, x)White, p(v_3, x)STW, p(v_1, Black),$$
$$p(v_2, Medium), p(v_3, MIB), p(v_2, Large), p(v_1, Red),$$
$$p(v_1, Blue)$$

Given that such a total ordering can be identified in the decomposition, the resulting VDD may be seen as a ZDD (*zero-suppressed decision diagram*) [12]. The gist of the algorithms for evaluation of BDDs and their cousins given in [12], such as *Algorithm C* and *Algorithm B* would apply. However, I have not made any verbatim use of these so far.[10]

VDDs have certain special characteristics beyond ZDDs. A terminal *HI*-link always leads to $\top$, and a terminal *LO*-link always leads to $\bot$. This and further characteristics that are ensured by the heuristic $h2$ given in Algorithm 1 allow certain short-cuts in the implementation (see Sections 4 and 6).

[8] By conventions established in [12], I term a link to the left child as a *LO*-link, drawn with a dotted line, preferably to the left, and a link to the right child as a *HI*-link, drawn with a filled line, preferably to the *right*. A *LO*-link is followed when the proposition in the node label is disbelieved. A *HI*-link is followed when it is believed. The terminal nodes $\bot$ and $\top$ are called *sinks*

[9] The amount of compression achieved in figure 2 is not overwhelming. The heuristic $h2$ does better (see figure 3)

[10] Both heuristics $h1$ and $h2$ in Section 4 are designed to guarantee such an ordering, but this is doesn't seem essential to the VDD approach in general

3.3 Set-labeled Nodes

There is an additional reduction of a VDD I have implemented as an option. This is most easily described using the concept of an *l-chain*. Define the subgraph composed of a node and all its descendent nodes that can be reached from it following only *LO*-links as the l-chain of the node[11]. Nodes in an l-chain that pertain to the same column and have the same right child can be joined into a single node. Let $p(v_{j_1}, x_1), \ldots, p(v_h, x_h)$ be the propositions in the labels of members in an l-chain that can be joined. The resulting combined node is labeled with the disjunction of these propositions: $P := p(v_{j_1}, x_1) \vee \ldots \vee p(v_h, x_h)$. This disjunction is represented, for short, by the (non-atomic) set of values $X := \bigcup_{i=1}^{h} x_h$ occurring in P. In the sequel, I refer to a node by

$$\nu(j, X) \tag{7}$$

where j is the column index of the referenced product property v_j, and X is the set of values represented by the node[12]. In case the node label is a single atomic value $\{x\}$, I also denote this by $\nu(j, x)$.

Figure 3 is a graph of the t-shirt using set-labeled nodes[13]. It is not a reduction of Figure 2, but rather a reduction of the graph in Figure 4 (see Section 4).

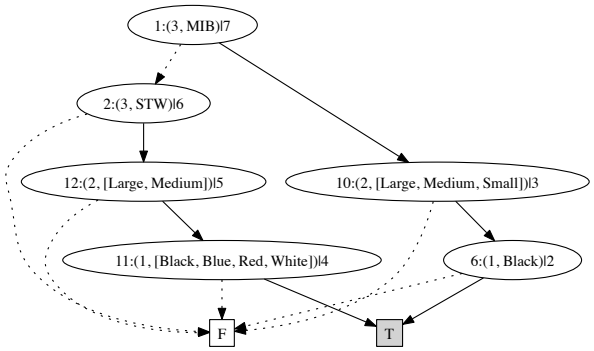


Figure 3. VDD of t-shirt with set-labeled nodes

By inspection, the complexity of the graph in Figure 3 is comparable to that obtained for the merged MDD in [2] (Figure 2 (b) there).

This further reduction is important from the viewpoint of compression, as each path from the root of the graph to a sink $\top$ can be seen as a c-tuple in the solution of $\mathcal{T}$, and a set-labeled node reduces the number of such c-tuples. It is also important from the viewpoint of run-time performance, if set intersection (*intersectsp*) is more efficient than multiple membership tests. A VDD using set-labeled nodes should result in similar c-tuples as the approach in [10] (depending of course on the heuristic). A key difference is that VDD paths may share nodes (c-tuples sharing common tails)[14], whereas this is not the case for a set representation of c-tuples. Hence, a VDD is a more compact representation.

Also, there may be external sources for set-labeled nodes if the maintained variant table is not relational. The SAP VC allows modeling variant tables with cells that contain real-valued intervals as well as a condensed form, where a cell may contain a set of values. Such cells can be directly represented as set-labeled nodes.

I close this section in noting that in [10] a Boolean function as in (4) is also used to construct a decomposition of a table into disjoint union of Cartesian products (c-tuples). There the resulting decomposition into c-tuples is the goal. Here the VDD is the goal, as I base the evaluation on it (see Section 8).

4 Heuristics

For the exposition in this section, I assume $\mathcal{T}$ to be in relational form with arity k.

The graph in Figure 2 is derived using on an initial heuristic $h1$, which I tried. $h1$ is based on trying to minimize splitting value blocks during decomposition. A value block for a proposition $p(v_j, x)$ is the rows in a table $\mathcal{T}$ that reference that proposition. Decomposing $\mathcal{T}$ on some other proposition $p(v_h, y)$ will split $p(v_j, x)$, if it has rows in both $\mathcal{L}(\mathcal{T}, j, y)$ and $\mathcal{R}(\mathcal{T}, j, y)$. Subsequently, both of these children need to be decomposed by $p(v_j, x)$, whereas a single decomposition would have handled $p(v_j, x)$ for $\mathcal{T}$ at its root level. It is assumed that keeping large value blocks intact is good, and the order of decomposition decisions should incur as little damage to these value blocks as possible. In order to apply this heuristic, all value blocks, their sizes, and the row indices that pertain to each one are initially calculated and stored. I do not go into further detail on this here.

The current implementation relies on characteristics of a VDD ensured by the decomposition heuristic $h2$ given in Algorithm 1. Recall that the total number of propositions is s, defined by (3), and that the number of propositions that pertain to the j-th column is s_j, defined by (2).

Algorithm 1 *(Heuristic $h2$)*
First, define $\boldsymbol{P}(\mathcal{T})$ as an ordered set of all s propositions $p(v_j, x)$ by
1. *Sorting the k columns of $\mathcal{T}$ by s_j, ascending (largest values last). $p(v_j, x)$, below, refers to the j-th column with respect to this ordering of the columns*
2. *Within each column, sort the values by their business order (any defined order the implementation can readily implement). x_{pj} refers to the p-th value in the j-th column domain $\pi_j(\mathcal{T})$.*

Then,
1. *Make a root node of the VDD for the first proposition in the first column: $p(v_1, x_{11})$*
2. *While nodes with a non-terminal subtable remain: split them always using the first proposition (value block) in their first column*
3. *Optionally, collect members of an l-chain with the same child nodes into an aggregated set-labeled node. I refer to this variant of the heuristic as $h2^*$*
4. *Reduce the nodes by unifying equivalent nodes as discussed in Section 3.2*

Figure 4 shows the graph of Table 1 produced using Algorithm 1 without set-labeled nodes ($h2$). Figure 3 shows the same graph produced with set-labeled nodes ($h2^*$).

A decomposition based on Algorithm 1 has the following characteristics:

- After k *HI*-links the $\top$-sink is always reached. (This is trivially true for all VDDs, as each row consists of k elements)

[11] A maximal l-chain would be one for a node which does not itself occur as the left child of any other node

[12] For the exposition, here, X represents a finite disjunction of propositions. In the implemented prototype it can also be an interval with continuous bounds

[13] The new set-labeled nodes are assigned a uniquely identifying node number outside the range used for numbering the propositions

[14] Figure 5 has shared nodes

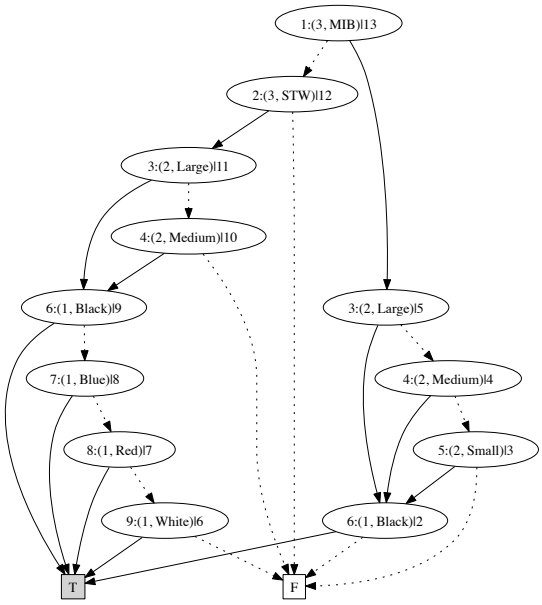


Figure 4. Basic VDD of t-shirt using algorithm 1

- All nodes in an l-chain (i.e., linked via *LO*-links) will always pertain to the same column. This follows from the fact that the columns of any non-empty left subtable of a table $\mathcal{T}$ are the same as the columns of $\mathcal{T}$. The heuristic says to always choose from the first column
- A node pertaining to the jth column is always $(j-1)$ *HI*-links distant from the root node. This follows by iterating the argument that if a table $\mathcal{T}$ is decomposed by a proposition $p(v_1, x)$ referencing its first column, then the right sub-table $\mathcal{T}$ has the second column of $\mathcal{T}$ as its first column (by construction))

Note that for BDDs an optimal ordering of $\boldsymbol{P}(\mathcal{T})$ is important to achieve a minimal graph. Thus, the search space for finding this is $s!$ (s factorial). In Algorithm 1 only the order of the columns is important. It is not important how the values are ordered within the column. To see this, note that the proposition $p(v_1, x_{11})$ used in decomposition *slices* $\mathcal{T}$ horizontally[15]. The slices obtained overall with respect to all values x_{p1} in the first column are the same, regardless of the order of the values in a column. Thus, the search space for an optimal column order is merely $k!$. As Algorithm 1 indicates, I am currently only exploring one ordering of columns, supposing that it will dominate the others. However, this still needs to be verified empirically.

5 Example of Extended T-shirt Model

In [8] I extended the t-shirt by adding the colors $Yellow$ and $DarkPurple$, the sizes XL and XXL, and the print $none$ to the global domains $\pi_j(\mathcal{T})$ (given in Section 2):

$$\pi_1(\mathcal{T}) = \{Black, Red, White, Blue, Yellow, DarkPurple\}$$
$$\pi_2(\mathcal{T}) = \{Large, Medium, Small, XL, XXL\}$$
$$\pi_3(\mathcal{T}) = \{MIB, STW, none\}$$

[15] The terminology is inspired by [6]

The new values combine with everything, except that no rows are added for combinations of MIB (print) and $Yellow$ (color). The table of all variants then has 73 rows (as opposed to the 11 of table 1)[16]. The graph is given in Figure 5

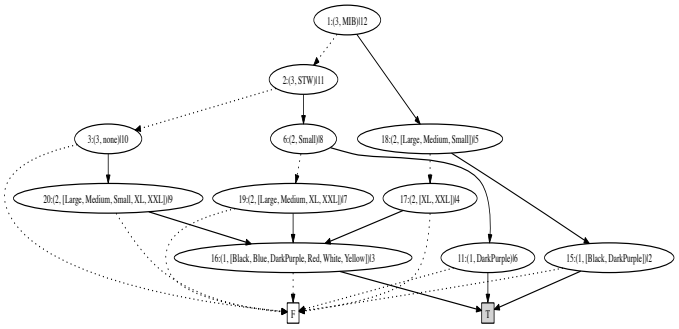


Figure 5. VDD of extended t-shirt with set-labeled nodes

Now 11 nodes are needed instead of the 6 nodes in Figure 3. The table is decomposed into 5 c-tuples. The node labeled $\langle 16 : (1, [Black, Blue, DarkPurple, Red, White, Yellow])\rangle$ is shared by three parents.

6 Empirical Compression Results

A prototype I have implemented in Java was tested for functional correctness against small exemplary tables such as the t-shirt model given in Table 1. This set of exemplary tables also contains some negative variant tables to test the approach in [8]. I refer to the implementation as the *VDD prototype*. It was further tested against 238 relational variant tables taken from three product models used at SAP in configurator maintenance issues. Since this data is proprietary, I give only summary statistics on the results. Testing on publicly available models, such as the Renault model [1] is a next step for future work.

It proved possible to successfully compile all 238 tables in this test base with all of the following three approaches:

- the heuristic $h1$ used to obtain the graph in Figure 2
- the heuristic $h2$ in Algorithm 1 - without merging nodes to set-labeled nodes
- the heuristic $h2^*$ in Algorithm 1 - with merging nodes to set-labeled nodes

Table 2 gives statistics on the table size and complexity. It lists the *minimal*, *maximal*, and *average* values, as well as the values for the four quartiles Q_1, Q_2, Q_3, Q_4, for each of the following parameters: k (*arity*), r (*number of rows*), s (*number of propositions*), and N (*number of cells* - $k * r$).

There is one table with arity one. This is used as a technique of dynamically defining a global domain for a product property in a variant table, rather than doing this directly in the declaration of the product property in the model. This technique has the disadvantage that it is more difficult to ensure translation of all relevant texts associated with a value in multi-lingual deployments of the configuration solution. It may, however, be used to dynamically restrict a large,

[16] I elaborate on the derivation of this example in [8]

Table 2. Size statistics on 238 SAP VC variant tables

Range	k	r	s	N
Minimum	1	1	2	2
Q_1	2	14	17	42
Q_2	3	56.5	46	176
Q_3	5	137.5	93.75	635.5
Q_4	16	21372	998	213720
Average	4.29	238.53	79.04	1590.51
Maximum	16	21372	998	213720

pre-defined, enterprise-wide global domain to the requirements of a particular product. General modeling considerations and experiences with the SAP VC are elaborated in [4].

The largest arity is 16. The associated table has only 76 rows. The largest table with 21372 rows has arity 10. The table with the largest number of propositions (998) has arity six and 469 rows.

Table 3 gives statistics on the achieved compression for the three compression techniques. (I discuss *double negation* separately, in Section 7.) The variant tables are partitioned into the four quartiles for s (*number of distinct node labels*[17]). These are denoted by Qs_1, Qs_2, Qs_3, and Qs_4. Averages are given for each of these four partitions for the following parameters: s, N (*number of cells in variant table*), n (*number of nodes*), $reduct$ (*reduction*: $(N - n)$), and t (*compilation time* in milli-seconds). Explicit results are also given for the table with largest number of cells (*Max* N), largest number of propositions (*Max* s), and largest arity (*Max* k), as well as the overall average.

Table 3. Average compression on 238 SAP VC variant tables

Range	N	*Heur*	s	n	$reduct$	t (msec)
		$h1$	17	20	15.75	4
Qs_1	42	$h2$	17	18	19.5	0
		$h2^*$	8	8	28.75	0
		$h1$	46	73.5	95.5	17.5
Qs_2	176	$h2$	46	65	102.5	1
		$h2^*$	17	19.5	155	1
		$h1$	93.75	184	418.5	112.75
Qs_3	635.5	$h2$	93.75	154.75	489.5	3
		$h2^*$	53.75	74.5	558.5	5
		$h1$	998	3756	213329	1192988
Qs_4	213720	$h2$	998	2728	213289	598
		$h2^*$	988	2381	213575	659
		$h1$	79.04	178.95	1411.57	5475.59
Average	1590.51	$h2$	79.04	149.95	1440.56	9.77
		$h2^*$	50.32	83.92	1506.59	12.43
		$h1$	152	391	54793	1192988
Max N	213720	$h2$	152	431	213289	598
		$h2^*$	70	145	213575	659
		$h1$	998	998	868	478
Max s	2814	$h2$	998	998	868	86
		$h2^*$	130	130	1736	91
		$h1$	181	885	331	874
Max k	1216	$h2$	181	653	563	5
		$h2^*$	182	649	567	5

The compilation times of $h2$ are drastically better for large tables than those for $h1$. This is because the information needed by $h1$ has some components that are non-linear to process, whereas $h2$ does not. Not surprisingly, using merged nodes further reduces both the number of nodes (n) in the VDD as well as the number of distinct labels of the nodes (s). The times are obtained on my Apple Mac mini with 2.5 GHz Intel Core i5 and 8GB memory. Times on other development PCs (both Windows and MacBook) are comparable. The time to compile the largest table with $h1$ is almost 20 minutes, but it takes less than one second using $h2$ with and without merging for the same table. Thus, compiling these tables into VDDs with $h2$ would almost be feasible at run-time.

Heuristic $h2$ strictly dominates $h1$ with respect to achieved compression (smaller number of nodes) for 143 of the 238 tables[18]. For 71 tables the same compression was achieved. For 24 tables $h1$ strictly dominates $h2$. Table 4 compares the advantages/disadvantages of $h2$ over $h1$. The three cases are labeled "$h2 > h1$" ($h2$ strictly dominates $h1$), "$h2 = h1$" (indifference), and "$h2 < h1$" ($h2$ is strictly dominated by $h1$). Table 4 lists averages for the following parameters: Tab, s, N , n, ΔR , and Δt. Tab is the number of tables that pertain to that case. The parameters s, N , and n are as defined above for Table 3. ΔR is the weighted difference in reduction: $\Delta R = Tab * (reduct_{h2} - reduct_{h1})$. It is positive where $h2$ has the advantage. Δt is the weighted difference in compile time: $\Delta t = Tab * (t_{h2} - t_{h1})$ in milli-seconds. It is negative where $h2$ has the advantage. The last row gives the averages per table $\Delta R/238$ and $\Delta t/238$ over all rows.

Table 4. Averages for dominated heuristics

Dominance	Tab	N	s	n	ΔR	Δt
$h2 > h1$	143	867.06	85.03	222.59	7702	-563065
$h2 = h1$	71	114.76	54.83	56.00	0	-1208
$h2 < h1$	24	10266.88	114.92	282.58	-992	-1206770
$Average$					28.19	-7446.43

The largest table is one where *h1* strictly dominates *h2*. However, the compile time using *h1* is almost 20 min. That for *h2* is 0.6 sec. Overall, $h2$ seems to prevail over $h1$. The average gain in reduction over all 238 tables is 28.19. The average gain in compilation time is 7446.43 (msec). The large gain in the latter is due to the non-linear performance of $h1$, which makes it grossly uncompetitive for large tables. Further experiments with other column orderings and with other heuristics remains a topic of future work.

7 Excursion: Negation

In [8] I describe *double negation* of a positive table as one approach to compression that is completely independent of the VDD mechanism. Being able to negate a table, i.e. calculate the complement with respect to a given domain restrictions is central in that approach.

My implementation of the VDD-prototype supports the approach in [8], and supports negation of a VDD. A BDD can be negated by switching the sinks $\bot$ and $\top$. This doesn't work for a VDD (and for a ZDD in general). Algorithm 2 gives the spirit of my implementation for negating a VDD produced using Algorithm 1 for a variant table $\mathcal{T}$ of arity k against a domain restriction tuple R. It produces a negated VDD that has set-labeled nodes.

[17] For VDDs without merged nodes this is just the number of propositions. For VDDs with merged nodes this is a different number, because labels for disjunctions of propositions are added, but not all original propositions still explicitly appear as node labels

[18] As merging could potentially also be done in conjunction with heuristic $h1$, it is not a fair comparison to compare the number of nodes achieved with $h1$ against that achieved with $h2^*$

Algorithm 2 *(Negation)*
Start with the root node of $\mathcal{V}$. Negate it as described below

- *If ν is a non-negated terminal node, i.e., ν references the last column index k, replace it with a node $\overline{\nu}$ that is assigned the complementary label $\overline{LC}(\nu) := \pi_j(\mathcal{T}) \setminus LC(\nu)$, where $LC(\nu)$ is defined as the union of all values/sets in the* l-chain[19] *of ν*
- *If ν is a non-negated non-terminal node that references column index $j < k$, negate it by doing the following in order:*
  - *negate each right child of each node in its l-chain in turn*
  - *add a node $\nu_\perp$ to the end of the l-chain of ν , where $\nu_\perp$ represents the c-tuple $\overline{LC}(\nu) \times R_{j+1} \times \ldots \times R_k$. $\nu_\perp$ itself is labeled with $\overline{LC}(\nu)$. Its right child is a node labeled R_{j+1} which has a right child R_{j+2} .... All* LO*-links of added nodes point to $\perp$*

Prune any unneeded nodes that have an empty set as a label or have a right child that is pruned, suitably rerouting a link to a pruned node to the left child of the pruned node

The VDD prototype actually does the pruning of empty nodes on the fly. If the root node itself is pruned, the entire resulting VDD is empty after negation.

The fact that double negation of a positive table needs to yield the same solution set as the original table (see [8]) provides a straightforward possibility to test for the correctness of this approach to negation.

In order to avoid complexity issues with very large complements, I so far applied double negation only in those cases where the number of tuples in the complement was smaller or equal to the number of tuples in the original table. 57 of the 238 SAP VC variant tables that are the basis for the results I presented in Section 6 proved amenable to double negation in this sense. Of these, 18 did not yield a smaller VDD than that obtained with heuristic $h2^*$. For the remaining 39 the maximal gain was 4 nodes, the average gain was 1.89 nodes.

Run-time performance tests have yet to be made, but these results raise the question, whether double negation will add value over direct compression. However, the concept of double negation is independent of the VDD concept, and could be applied on its own without using VDDs. Also, it remains to be seen, if the test on whether constraint propagation can be gainfully applied, given in [8], proves valuable.

8 Evaluation of a VDD

Given a run-time restriction R, a VDD $\mathcal{V}$, and a node $\nu(j, X)$ in $\mathcal{V}$ (using the notation in (7)). $\nu(j, X)$ can be marked as *out* if $X \cap R_j = \emptyset$. The admissible solutions of $\mathcal{V}$ are all paths from the root node to the sink $\top$ that do not contain any node marked *out*. I denote this set as $\mathcal{V} \cap R$, for short. If there are no such paths, then R is inconsistent given $\mathcal{V}$.

I do not go into further detail on how to determine $\mathcal{V} \cap R$. This follows the spirit of known algorithms for directed acyclic graphs. For example, see [12] for an exposition in the context of BDDs/ZDDs.

Concerning the general complexity of the calculations:

- Let s_j be the number of distinct node labels pertaining to the j-th column of $\mathcal{V}$ (as in (2), but modified to allow for set-valued labels). s_j node-labels must be intersected with R_j for each column index j to determine admissibility of all occurring node labels. Given that the domains are naturally ordered, binary search can be used to speed this up. The VDD prototype also imposes an ordering on the node labels to facilitate this
- Let n be the number of nodes in $\mathcal{V}$. The question of which nodes have admissible paths to $\top$, is related to the problem of counting the admissible paths. After determining which node labels are admissible, this has the remaining complexity of $O(n)$ (c.f., *Algorithm C* in [12])

In the case that $\mathcal{V}$ is a VDD without set-valued nodes, i.e., all node labels are of the form $X = \{x\}$, $\mathcal{V} \cap R$ is just the solution set of $\mathcal{T} \cap R$, where $\mathcal{T}$ is the variant table encoded by $\mathcal{V}$. But, if $\mathcal{V}$ has non-atomic set-valued nodes[20] $\mathcal{T} \cap R \subseteq \mathcal{V} \cap R$. Here, the evaluation comes at the slight additional cost of determining the solution set by additionally intersecting $\mathcal{V} \cap R$ with R.[21] The intersection of two c-tuples is easy to calculate. Thus, this additional cost is offset, because determining the admissibility of each node is now faster (a smaller number of *intersectsp* tests hopefully offsets the otherwise greater number of *memberp* tests).

Real run-time measurements have not yet been performed. However, in the beginning, in trying to determine whether the VDD approach is worthwile, I did an experimental integration with the (Java-based) SAP IPC configurator (see [4]) with the product models encompassing the 238 tables mentioned in Section 6. The software configuration I used was completely non-standard, so any results are not objectively meaningful, but they did encourage me to pursue this approach. The expectations on performance gains might be roughly oriented on the inverse of the compression ratio N/n (*total number of table cells*/*number of nodes*). For heuristics $h1$/$h2$/$h2^*$ the averages for N/n are 2.2/2.4/3.9, respectively. But, this does not account for losses due to the overhead of needing more complex set operations. In any case, real run-time measurements need to be performed as a future step.

I close this section with a remark on using a VDD as a simple database. A query with an instantiated database key is equivalent to a run-time restriction R, where the key's properties are restricted to singleton values, and the other properties are *unconstrained* (or have the domain that is established at the time of the query). The solution set of the VDD will contain only tuples consistent with the query (by construction). If, for example, the key is defined as unique in the database (and the table content is consistent with this definition), the result can contain at most the unique response (or the empty set, if no row for the key exists in the table). For queries with non-unique database keys, the solution set needs to be intersected with R, as discussed above.

9 Conclusion

Although it remains to explore other variants of Algorithm 1 with different orderings of the columns, the compression achieved with the current version (both $h2$ and $h2^*$) has been surprisingly satisfactory. The very short compile times may be of more practical advantage than a more expensive search for heuristics that provide (somewhat) better results. However, further investigation into search and heuristics of an altogether different type should to be done more completely and formally. This is future work.

[19] Defined in Section 3.3, the *l-chain* of a node is the sub-graph consisting of the node and all nodes reachable from it via *LO*-links, but excluding the sink $\perp$. All nodes in an l-chain reference the same column index j

[20] Either merged nodes or nodes representing continuous intervals

[21] To see this, suppose that $\mathcal{T}$ is itself a c-tuple, so that there is at most one admissible c-tuple consisting of $\mathcal{T}$ itself. Obviously R may be more restrictive

The goal of this work was not only to find a good compression technique, but also to provide a solution that can be used in conjunction with a legacy configurator to enhance the handling of variant tables, either as a whole or individually. The VDD prototype I implemented uses little machinery and adds little to software maintenance (training, size of the code base, etc.). It also conveys little risk. In the event that some tables cannot be compiled to a VDD, the legacy handling can be seamlessly kept. (This did not occur in the initial explorations using the $h1$ heuristic with the test models.) The SAP VC is a configurator with a very large user base. Thus, any change to it comes with large risk. This is the type of situation I had in mind when designing the VDD prototype.[22]

In the course of this work I have come to believe that all of the following three approaches to speed up variant table handling and to look for a compact representation yield very similar results:

- BDDs in various flavors ([9, 2])
- Compression into c-tuples and constraint slicing ([6, 10])
- Read optimized databases (such as column-oriented databases ([14])) in conjunction with a constraint propagation algorithm (e.g. the STR-algorithm [13])

The differences are in the machinery needed for the intended deployment and in the heuristics that suggest themselves. Although the representation as a VDD is central to my approach at compression and evaluation, functionally, the two important aspects in practice are that it functions as a (limited) replacement for a database, and that it performs constraint propagation. I would see as a topic of future work to both look more closely at read optimized databases and to investigate, if the VDD approach can be extended to support more complex database queries. Investigating the commonality between the three approaches (BDDs, compression, read-optimized databases) more formally could be another interesting topic. The VDD approach has elements of all three.

A note in closing: The compression algorithm in [10] is based on a very similar approach to table decomposition. I was not aware of this work until recently, so there are some unfortunate disconnects in conventions I use. I tend to follow [12], whereas in [10] left and right are used the other way around. I did adopt use of the term c-tuple. I think the column oriented view, here, is more intuitive and has resulted in more useful heuristics. Another major difference to [10], which I see, is that the I base evaluation directly on the VDD. Furthermore, the VDD supports nodes labeled with real-valued intervals, but his could also be added to [10] in a straightforward manner[23].

ACKNOWLEDGEMENTS

I would like to thank the anonymous reviewers and my daughter Laura for their constructive suggestions, on how to improve the intelligibility of this paper. I am afraid the current result may not yet meet their expectations, but I think it is a step forward from the previous version.

REFERENCES

[1] J. Amilhastre, H. Fargier, and P Marquis, 'Consistency restoration and explanations in dynamic csps application to configuration', *Artif. Intell.*, **135**(1-2), 199–234, (2002).

[2] H.R. Andersen, T. Hadzic, and D. Pisinger, 'Interactive cost configuration over decision diagrams', *J. Artif. Intell. Res. (JAIR)*, **37**, 99–139, (2010).

[3] C. Bessiere, 'Constraint propagation', in *Handbook of Constraint Programming*, eds., F. Rossi, P. van Beek, and T. Walsh, chapter 3, Elsevier, (2006).

[4] U. Blumöhr, M. Münch, and M. Ukalovic, *Variant Configuration with SAP, second edition*, SAP Press, Galileo Press, 2012.

[5] K. Ferland, *Discrete Mathematics*, Cengage Learning, 2008.

[6] Nebras Gharbi, Fred Hemery, Christophe Lecoutre, and Olivier Roussel, 'Sliced table constraints: Combining compression and tabular reduction', in *CPAIOR'14*, pp. 120–135, (2014).

[7] A. Haag, 'Chapter 27 - product configuration in sap: A retrospective', in *Knowledge-Based Configuration*, eds., Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, 319 – 337, Morgan Kaufmann, Boston, (2014).

[8] A. Haag, 'An approach to arc consistency with negative variant tables', in *Proceedings of the 17th International Configuration Workshop, Vienna, Austria, September 10-11, 2015.*, pp. 81–87, (2015).

[9] Tarik Hadzic, 'A bdd-based approach to interactive configuration', in *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, ed., Mark Wallace, volume 3258 of *Lecture Notes in Computer Science*, p. 797. Springer, (2004).

[10] G. Katsirelos and T. Walsh, 'A compression algorithm for large arity extensional constraints', in *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, ed., Christian Bessiere, volume 4741 of *Lecture Notes in Computer Science*, pp. 379–393. Springer, (2007).

[11] H. Keller, *The Official ABAP Reference*, number Bd. 1 in Galileo SAP Press, Galileo Press, 2005.

[12] D.E. Knuth, *The Art of Computer Programming*, volume 4A Combinatorial Algorithms Part 1, chapter Binary Decision Diagrams, 202–280, Pearson Education, Boston, 2011.

[13] C. Lecoutre, 'STR2: optimized simple tabular reduction for table constraints', *Constraints*, **16**(4), 341–371, (2011).

[14] Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alex Rasin, Nga Tran, and Stanley B. Zdonik, 'C-store: A column-oriented DBMS', in *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, eds., Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, pp. 553–564. ACM, (2005).

[22] In my estimation, the effort to reimplement the current VDD functionality in SAP ABAP ([11]) for use with the SAP VC would be feasible from a cost and risk perspective

[23] Basically treating an interval syntactically like a value in compilation, but evaluating it like a set at run-time

Inverse QUICKXPLAIN vs. MAXSAT — A Comparison in Theory and Practice

Rouven Walter[1] and **Alexander Felfernig**[2] and **Wolfgang Küchlin**[1]

Abstract. We compare the concepts of the INVQX algorithm for computing a Preferred Minimal Diagnosis vs. Partial Weighted MAXSAT in the context of Propositional Logic. In order to restore consistency of a Constraint Satisfaction Problem w.r.t. a strict total order of the user requirements, INVQX identifies a diagnosis. Partial Weighted MAXSAT aims to find a set of satisfiable clauses with the maximum total weight. It turns out that both concepts have similarities, i.e., both deliver a correction set. We point out these theoretical commonalities and prove the reducibility of both concepts to each other, i.e., both problems are $\mathrm{FP}^{\mathrm{NP}}$-complete, which was an open question. We evaluate the performance on problem instances based on real configuration data of the automotive industry from three different German car manufacturers and we compare the time and quality tradeoff.

1 Introduction

Constraint programming is successfully applied in many different areas, e.g., planning, scheduling, and configuration. Besides the usage of Constraint Satisfaction Problem (CSP) based knowledge bases, the usage of the more restrictive Propositional Logic has been successfully established in the context of automotive configuration and verification [11].

In many practical use cases the knowledge base can become over-constrained, e.g., by overly restrictive rules or user requirements. In the context of automotive configuration the knowledge base can become over-constrained, too [23]. A typical situation would be a customer configuring a car up to his wishes conflicting with the knowledge base. Another typical situation would be an engineer given the task that new features should be constructable for an existing type series by now which were not constructable before. In both situations we would like to have an automatic reasoning procedure for assistance in order to restore consistency.

One approach to restore consistency is to guide the user by computing minimal unsatisfiable cores (conflicts), which can be considered as a problem explanation. However, more than one conflict is involved in general. Another approach is to try to satisfy as many of the constraints as possible, i.e., finding a maximal satisfiable subset (MSS) or, the opposite, finding a minimum correction subset (MCS) which can be considered as a repair suggestion. The constraints of an MCS have to be removed or altered in order to restore consistency.

An MCS can be calculated in different ways: (i) MAXSAT is a generalization of the well-known SAT problem and computes the MCS of minimum cardinality; (ii) the Inverse QUICKXPLAIN (INVQX) algorithm (also denoted as FASTDIAG) [4] delivers a preferred minimal diagnosis w.r.t. a total order on the user requirements. Both approaches can be considered as an optimal repair suggestion w.r.t. their definition of optimum. In this paper, we study both approaches by giving the following contributions:

1. We point out theoretical similarities and suggest an improvement for INVQX.
2. We show that both problems, the computation of a preferred minimal diagnosis (INVQX) and the computation of an MCS of minimum cardinality (MaxSAT), are reducible to each other and that both are $\mathrm{FP}^{\mathrm{NP}}$-complete.
3. We provide experimental evaluations based on real automotive configuration data.

To the best of our knowledge, it has not been proven before, that the computation of a preferred minimal diagnosis in the context of Propositional Logic is $\mathrm{FP}^{\mathrm{NP}}$-hard.

The remainder of the paper is structured as follows: Section 2 introduces the formal background. Section 3 discusses related work. In Section 4 and Section 5 we introduce both approaches (MINUNSAT and INVQX) and give an overview of solving techniques, respectively. Section 6 points out the theoretical relationships and Section 7 shows how to reduce one problem to the other. In Section 8 we present experimental evaluations. Finally, Section 9 concludes the paper.

2 Preliminaries

Within the scope of this paper we focus on Propositional Logic over the standard operators $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ with constants $\bot$ and $\top$, representing false and true, respectively. For a Boolean formula φ we denote its evaluation by $\|\varphi\|_v \in \{0, 1\}$ for a variable assignment v. A Boolean formula is in CNF normal form iff it consists of a conjunction of clauses, where a clause is a disjunction of literals. A literal is a variable or its negation. A formula in CNF can be interpreted as a set of clauses and further a clause can be interpreted as a set of literals. The NP-complete SAT problem asks whether a Boolean formula is satisfiable or not.

Definition 1. *(MSS/MCS) Let φ be a set of clauses. A set $\psi \subseteq \varphi$ is a* Maximal Satisfiable Subset *(MSS) iff ψ is satisfiable and every $\psi' \subseteq \varphi$ with $\psi \subset \psi'$ is unsatisfiable.*

A set $\psi \subseteq \varphi$ is a Minimal Correction Subset *(MCS) iff $\varphi - \psi$ is satisfiable and for all $\psi' \subseteq \varphi$ with $\psi' \subset \psi$ the set difference $\varphi - \psi'$ is unsatisfiable.*

[1] Symbolic Computation Group, WSI Informatics, Universität Tübingen, Germany, www-sr.informatik.uni-tuebingen.de

[2] Institute for Software Technology, Graz University of Technology, Graz, Austria, www.felfernig.eu, email: alexander.felfernig@ist.tugraz.at

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

The definition of an MSS (resp. MCS) can be naturally extended by taking into account a set of hard clauses which have to be satisfied.

Clearly, for a given MSS (resp. MCS) ψ of φ, the complement $\varphi - \psi$ is an MCS (resp. MSS) of φ.

Analogous to [16] we introduce the following definitions:

Definition 2. *(L- and A-Preference) Let $<$ be a strict total order over a set $\varphi = \{c_1, \ldots, c_m\}$ of clauses with $c_i < c_{i+1}$ for $1 \leq i < m$, i.e., clause c_i is* preferred *to clause c_{i+1}.*

We define the lexicographical order $<_{\text{lex}}$ as follows: For two sets $\psi_1, \psi_2 \subseteq \varphi$ we say set ψ_1 is lexicographically preferred to ψ_2, denoted as $\psi_1 <_{\text{lex}} \psi_2$, iff $\exists_{1 \leq k \leq m} : c_k \in \psi_1 - \psi_2$ and $\psi_1 \cap \{c_1, \ldots, c_{k-1}\} = \psi_2 \cap \{c_1, \ldots, c_{k-1}\}$.

Furthermore, we define the anti-lexicographical order $<_{\text{antilex}}$ as follows: For two sets $\psi_1, \psi_2 \subseteq \varphi$ we say set ψ_1 is anti-lexicographically preferred to ψ_2, denoted as $\psi_1 <_{\text{antilex}} \psi_2$, iff $\exists_{1 \leq k \leq m} : c_k \in \psi_2 - \psi_1$ and $\psi_1 \cap \{c_{k+1}, \ldots, c_m\} = \psi_2 \cap \{c_{k+1}, \ldots, c_m\}$.

An MSS/MCS ψ_1 is L-preferred (resp. A-preferred) if for all MSS/MCS $\psi_2 \neq \psi_1$, $\psi_1 <_{\text{lex}} \psi_2$ (resp. $\psi_1 <_{\text{antilex}} \psi_2$).

The lexicographical order appears to be the more intuitive one. Whereas the most L-preferred set includes the most preferred clauses, the most A-preferred set excludes the most non-preferred clauses. We denote the inverse order of $<$ by $<^{-1}$.

If ψ is an L-preferred (resp. A-preferred) MSS/MCS of φ w.r.t. to the order $<$, then $\varphi - \psi$ is an A-preferred (resp. L-preferred) MSS/MCS of φ w.r.t. to the inverse order $<^{-1}$ (see Proposition 12 in [16]). Therefore, algorithms for the computation of an L-preferred MSS/MCS can also be used for the computation of the corresponding A-preferred MCS/MSS.

Note that we use the standard notation for the complexity class FP^{NP} (resp. $\text{FP}^{\text{NP}[\log n]}$), the class of function problems solvable in deterministic polynomial time using a polynomial (resp. logarithmic) number of calls to an NP oracle [19].

3 Related Work

The authors of [14] improve the computation of an MCS by newly introduced techniques, i.e., usage of backbone literals, disjoint unsatisfiable cores and satisfied clauses. Not all techniques can be applied to the computation of an A-preferred MCS, i.e., only the usage of backbone literals can be adopted. Hence the proposed enhanced version of INVQX (also denoted as FASTDIAG [4]) of the cited work can not be adopted for A-preferred MCS computation. The newly proposed MCS algorithm, called CLAUSED, exploits the fact that a falsified clause does not contain complementary literals, but it can not be adopted, either.

The INVQX algorithm (also known as FASTDIAG) [4] is based on the idea of divide-and-conquer that has been successfully exploited in the QUICKXPLAIN algorithm [9]. Whereas QUICKXPLAIN computes a preferred explanation (a minimal unsatisfiable subset (MUS) in the context of Propsitional Logic), the INVQX algorithm computes a preferred diagnosis (an MCS in the context of Propositional Logic), which can be interpreted as the *inverse* of QUICKXPLAIN.

The complexity of computing preferred sets is studied in [16]. Furthermore, the authors give an overview of established algorithms which can or can not be adopted to involve preferences.

The authors of [17] introduce improvements on computing an MCS in the context of CSP, i.e., they adopt the CLAUSED algorithm of [14]. The adopted variant is also not applicable for A-preferred MCS computation.

4 Preferred Minimal Diagnosis

The definition of a Preferred Minimal Diagnosis (PMD) used in [4] is in the context of a Constraint Satisfaction Problem (CSP). We will recap the essential definitions briefly. Let C_{KB} and C_{R} be sets of CSP constraints. The set C_{KB} (resp. C_{R}) represents the constraints of the knowledge base (resp. the user requirements).

Definition 3. *(CR Diagnosis) Let $(C_{\text{KB}}, C_{\text{R}})$ be a CR diagnosis problem. A set $\Delta \subseteq C_{\text{R}}$ is a* CR Diagnosis *if $C_{\text{KB}} \cup (C_{\text{R}} \setminus \Delta)$ is consistent. Furthermore, Δ is called* minimal *if no diagnosis $\Delta' \subset \Delta$ exists where $C_{\text{KB}} \cup (C_{\text{R}} \setminus \Delta')$ is consistent.*

In the context of Propositional Logic a CR diagnosis corresponds to the MCS problem with an additional hard part. The definitions for L- and A-Preference can analogously be defined for CSP. We will not write them out here.

Finally, we can define a preferred minimal diagnosis:

Definition 4. *(Preferred Minimal Diagnosis) Let $(C_{\text{KB}}, C_{\text{R}} = \{c_1, \ldots, c_m\})$ be a CR diagnosis problem with a strict total order s.t. $c_1 < \ldots < c_m$. A minimal diagnosis Δ is called a* Preferred Minimal Diagnosis *(PMD) if Δ is A-preferred w.r.t. the order $<^{-1}$.*

The strict total order in [4] is defined the other way round, i.e., if $c_i < c_j$ then constraint c_j is preferred to c_i. Our definition here is consistent with [16].

Remark 1. *In the context of Propositional Logic we assume, without loss of generality, that C_{KB} and C_{R} are clause sets:*

1. *C_{R}: Let s_i be a fresh variable. For each constraint $c_i \in C_{\text{R}}$ we add the clause set $\text{CNF}(s_i \rightarrow c_i)$ to the hard part and the unit clause $\{s_i\}$ replaces c_i within C_{R} (cf. [2, 7]).*
2. *C_{KB}: For any non-clausal constraint $c \in C_{\text{KB}}$ we apply the Tseitin-/Plaisted-Greenbaum Transformation [20, 21] which takes polynomial time and space. Furthermore, the constraint c and the resulting formula share the same models w.r.t. the original variables. The search space between both remains the same.*

Therefore we can interpret the problem of computing a PMD as the problem of computing an A-preferred MCS.

The computation of an A-preferred MCS is in FP^{NP} [16]. The question arises whether computing an A-preferred MCS is also FP^{NP}-hard? This is actually the case as we will show in Theorem 1.

4.1 Algorithms

A straightforward approach is Linear Search. We iterate in descending order through all constraints and check whether they conflict with the hard constraints and the previously added constraints or not. If there is a conflict, the constraint is part of the A-preferred MCS. Otherwise, the constraint will be added. The complexity of Linear Search in terms of the number of consistency checks is $\mathcal{O}(m)$, where $m = |C|$.

Algorithm 1 shows the procedure presented in [4]. The worst case complexity of INVQX in terms of the number of consistency checks is $\mathcal{O}(2d \cdot \log_2(\frac{m}{d}) + 2d)$, where d is the minimal diagnosis set size and $m = |C|$.

Remark 2. *(Exploiting inc-/decremental SAT interface) Modern SAT solvers often provide an inc-/decremental interface for adding clauses and removing them afterwards. This can be useful, e.g., when*

Algorithm 1: INVQX Algorithm

Input: $C \subseteq AC$, $AC = \{c_1, \ldots, c_t\}$
Output: Preferred minimal diagnosis Δ
if isEmpty(C) or inconsistent$(AC - C)$ **then**
  return $\emptyset$
else
  return FD$(\emptyset, C, AC)$

func FD$(D, C = \{c_1, \ldots, c_q\}, AC)$: diagnosis Δ
if $D \neq \emptyset$ and consistent(AC) **then**
  return $\emptyset$
if singleton(C) **then**
  return C
$k = \frac{q}{2}$; $C_1 = \{c_1, \ldots, c_k\}$; $C_2 = \{c_{k+1}, \ldots, c_q\}$
$D_1 = \mathrm{FD}(C_1, C_2, AC - C_1)$
$D_2 = \mathrm{FD}(D_1, C_1, AC - D_1)$
return $D_1 \cup D_2$

we have a huge formula representing the configuration model and small test instances to check against the configuration model.

Therefore we can improve Algorithm 1 by adding all constraints AC − C *first and do the consistency checks by using the inc-/decremental interface. Another improvement can be made for the second recursive call* $D_2 = \mathrm{FD}(D_1, C_1, AC - D_1)$*. All constraints in* $C_2 - D_1$ *have to be satisfied for this call. We add all constraints* $C_2 - D_1$ *before and remove them afterwards. We evaluated this improvement, see Section 8.*

5 Partial Weighted MINUNSAT

In the context of this paper, we will only focus on the Partial Weighted MINUNSAT problem. See [13] for analogous definitions of (Partial) (Weighted) MAXSAT.

Definition 5. *(*MINUNSAT*) Let* Hard *be a set of propositional clauses. Let* $\mathrm{Soft} = \{(c_1, w_1), \ldots, (c_m, w_m)\}$ *a set of tuples where* c_i *is a propositional clause and* $w_i \in \mathbb{N}_{\geq 1}$ *is a weight for all* $i = 1, \ldots, m$*. Let* $\varphi = (\mathrm{Hard}, \mathrm{Soft})$ *and* n *be the number of variables in* Hard ∪ Soft*. The* Partial Weighted Minimum Unsatisfiable Problem (MINUNSAT) *is defined as follows:*

$$\mathrm{MINUNSAT}(\varphi) := \min \left\{ \sum_{i=1}^{m} w_i (1 - \|c_i\|_v) \middle| v \in \{0,1\}^n \right\}$$

Partial Weighted MINUNSAT and Partial Weighted MAXSAT are closely connected: $\mathrm{MaxSAT}(\mathrm{Hard}, \mathrm{Soft}) = \sum_{i=1}^{m} w_i - \mathrm{MINUNSAT}(\mathrm{Hard}, \mathrm{Soft})$. A solution for one problem directly leads to a solution for the other one and vice verca. The Partial Weighted MaxSAT (resp. MINUNSAT) problem is $\mathrm{FP}^{\mathrm{NP}}$-complete [19]. The *unweighted* (partial) MAXSAT (resp. MINUNSAT) problem is $\mathrm{FP}^{\mathrm{NP}[\log n]}$-complete [10].

In the context of this paper, we will refer to Partial Weighted MINUNSAT just as MINUNSAT to simplify reading. Please note that in the literature often the name MAXSAT is used to refer to MINUNSAT. But we will use its original name to make the distinction clear.

5.1 Algorithms

Different approaches to solve the MINUNSAT problem have been developed: Branch-and-Bound for general optimization problems has been adopted to MINUNSAT, e.g., [8, 12]. In recent years, many MINUNSAT solvers make use of SAT solvers as a black box. A basic approach is to add a blocking variable to each soft clause, iteratively checking the instance for satisfiability and restricting the blocking variables further each time. Also binary search is possible this way. Nowadays solvers make usage of SAT solvers delivering an unsatisfiable core, which was first presented in [5] and extended by weights in [1]. Our list is not complete, see [18] for an overview. As an example, Algorithm 2 shows the WPM1 algorithm. If φ_c is not necessarily an MUS, the worst case complexity of WPM1 in terms of the number of consistency checks is $\mathcal{O}(d)$, where d is the minimal sum of weights of unsatisfied clauses, i.e., only costs of 1 are added in each iteration [6].

Algorithm 2: WPM1 Algorithm

Input: $(\mathrm{Hard}, \mathrm{Soft} = \{(c_1, w_1) \ldots, (c_m, w_m)\})$
Output: Sum of minimal unsatisfied weights: $cost$
if UNSAT(Hard) **then**
  return *No solution*
$cost \leftarrow 0$
while $true$ **do**
  $(st, \varphi_c) \leftarrow \mathrm{SAT}(\mathrm{Hard} \cup \{(c_i, w_i) \in \mathrm{Soft}\})$
  if $st = \mathrm{SAT}$ **then**
    return $cost$
  $\mathrm{BV} \leftarrow \emptyset$
  $w_{\min} \leftarrow \min\{w_i \mid c_i \in \varphi_c \wedge\ c_i \text{ is soft}\}$
  foreach $c_i \in \varphi_c \cap \mathrm{Soft}$ **do**
    $b_i \leftarrow$ fresh blocking variable
    Soft ←
    $\mathrm{Soft} - \{(c_i, w_i)\} \cup \{(c_i, w_i - w_{\min})\} \cup \{(c_i \vee b_i, w_{\min})\}$
    $\mathrm{BV} \leftarrow \mathrm{BV} \cup \{b_i\}$
  $\mathrm{Hard} \leftarrow \mathrm{Hard} \cup \mathrm{CNF}(\sum_{b \in \mathrm{BV}} b = 1)$
  $cost \leftarrow cost + w_{\min}$

6 Relationship

We want to identify and discuss similarities of the PMD problem and the MINUNSAT problem. As we have already shown in Remark 1 in the context of Propositional Logic the PMD problem can be interpreted as an A-preferred MCS problem. The hard parts of both problems are equal in terms of expressive power, since both are sets of clauses.

The interesting part is the set C_{R} with its strict total ordering $<$ and the set Soft of weighted clauses. Both can be defined as a special case of an MCS problem: The PMD problem can be interpreted as an A-preferred MCS problem (cf. Remark 1):

$$\min_{<^{-1}_{\text{antilex}}} \{S \mid S \text{ is MCS of } C_{\mathrm{R}} \text{ w.r.t. } C_{\mathrm{KB}}\}$$

Whereas the MINUNSAT problem is also an MCS with the minimum sum of weights:

$$\min_{\leq_{\sum_{c \in S} \mathrm{weight}(c)}} \left\{ S \middle| S \text{ is MCS of Soft w.r.t. Hard} \right\}$$

Where $\mathrm{weight}(c)$ denotes the weight of clause c.

Remark 3. *The same similarities hold for the corresponding opposite problems, i.e., the* MAXSAT *problem and the L-preferred MSS problem.*

6.1 Approximation of MINUNSAT

Solvers for the A-preferred MCS problem can be used to approximate the MINUNSAT problem. Let π be a permutation of the indices $1, \ldots, m$ such that the weights are sorted, i.e., if $i < j$ then $w_{\pi(i)} < w_{\pi(j)}$. We set:

$$\begin{aligned} C_{\mathrm{KB}} &:= \mathrm{Hard} \\ C_{\mathrm{R}} &:= \mathrm{Soft} \\ < &:= c_{\pi(1)}, \ldots, c_{\pi(m)} \end{aligned}$$

This transformation is actually an approximation: An A-preferred MCS will prefer to satisfy a clause with a high weight value more than to satisfy multiple clauses with low weight values. Whereas a MINUNSAT solution will minimize the sum of the weights of unsatisfied clauses in total. The transformation can be done in polynomial time: We sort the clauses w.r.t. their weights, which can be done in $\mathcal{O}(m \log m)$ where m is the number of soft clauses. We evaluated this approximation, see Section 8.

Example 1. *Consider* $\mathrm{Hard} = \emptyset$ *and* $\mathrm{Soft} = \{(x, 6), (\neg x \vee y, 5), (\neg y, 4), (\neg x, 3)\}$. *With a strict total ordering relying on the weights of the soft clauses (see above), the A-preferred MCS is* $\Delta = \{(\neg y, 4), (\neg x, 3)\}$ *resulting in costs of 7. But the* MINUNSAT *solution* $\{(x, 6)\}$ *has costs of 6.*

6.2 Approximation of A-preferred MCS

We can also use MINUNSAT to approximate the A-preferred MCS problem. The crucial question is which weights to assign to the soft clauses. We can assume C_{R} to be a set of clauses, see Remark 1. We set:

$$\begin{aligned} \mathrm{Hard} &:= C_{\mathrm{KB}} \\ \mathrm{Soft} &:= C_{\mathrm{R}} \\ \mathrm{weight}(c_i) &:= m - (i - 1) \end{aligned}$$

With this weight assignment, the most preferred clause is assigned to weight m, the next one to weight $m - 1$ and so on. The lexicographical order will be imitated somewhat but not sufficiently to be exact. The greater the distances of the weights of consecutive constraints c_i and c_{i+1} the better the approximation gets. Above we set the distance to 1. In Subsection 7.1 we will show how to determine distances that help to reach an exact reduction. The transformation can be done in polynomial time, too.

Example 2. *Consider* C_R *with* $x \vee y < \neg x < \neg y < x < z$. *The A-preferred MCS is* $\Delta = \{\neg y, x\}$, *but* $\mathrm{MINUNSAT}(\{(x \vee y, 5), (\neg x, 4), (\neg y, 3), (x, 2), (z, 1)\}) = 4$, *because* $\neg x$ *with weight 4 is less than clauses* x *and* $\neg y$ *with weights* $2 + 3 = 5$.

7 Reduction

In this section we will show how to polynomially reduce one problem to each other and finally see that both problems are $\mathrm{FP}^{\mathrm{NP}}$-complete and are therefore equally hard to solve.

7.1 From A-preferred MCS to MINUNSAT

Let (C_{KB}, C_R) be an A-preferred MCS problem with $C_R = \{c_1, \ldots, c_m\}$ and a total strict order s.t. $c_1 < \ldots < c_m$. We can assume C_{R} to be a set of clauses, see Remark 1. We can reduce the problem to a MINUNSAT problem by:

$$\begin{aligned} \mathrm{Hard} &:= C_{\mathrm{KB}} \\ \mathrm{Soft} &:= C_{\mathrm{R}} \end{aligned}$$

with weight w_i defined recursively:

$$w_i := \left(\sum_{j=i+1}^{m} w_j \right) + 1$$

With these weights assigned we achieve two important properties: (1) the ascending order of the constraints c_i and more important (2) the lexicographical ordering, because constraint c_i with weight $\left(\sum_{j=i+1}^{m} w_j\right) + 1 = w_{i+1} + \ldots + w_m + 1$ is greater than the sum of weights of all previous and less preferred constraints $c_{i+1}, \ldots, c_m$.

Each step requires polynomial time. But the downside of the above reduction is the exponential growth of the search space. It can be shown by induction that $(\sum_{j=i+1}^{m} w_j) + 1 = 2^{m-i}$. The most preferred clause has weight 2^{m-1}, so the weights are in $\mathcal{O}(2^m)$.

7.2 From MINUNSAT to A-preferred MCS

Firstly, we show how we can reduce the MINUNSAT problem easily to the A-preferred MCS problem if the weights comply with the following property:

Proposition 1. *Let* $\varphi = (\mathrm{Hard}, \mathrm{Soft})$ *be a* MINUNSAT *problem as defined in Definition 5 with* $\mathrm{Soft} = \{(c_1, w_1), \ldots, (c_m, w_m)\}$. *If there exists a permutation* π *of the indices* $\{1, \ldots, m\}$ *such that:*

$$w_{\pi(i)} > \sum_{j=i+1}^{m} w_{\pi(j)}$$

then the strict total ordering $<_\pi$ *with* $c_{\pi(1)} < \ldots < c_{\pi(m)}$ *with* $C_{\mathrm{KB}} = \mathrm{Hard}$ *and* $C_{\mathrm{R}} = \{c_{\pi(1)}, \ldots, c_{\pi(m)}\}$ *is a reduction to the A-preferred MCS problem.*

Proof. The reduction is correct since (1) it preserves the order of the soft clauses w.r.t. their weights and (2) the weights are in such a relation that for each clause $c_{\pi(i)}$ a MINUNSAT solution will try to satisfy $c_{\pi(i)}$ before satisfying all clauses $c_{\pi(i+1)}, \ldots, c_{\pi(m)}$ and so will a solution of A-preferred MCS due to the strict total clause ordering. □

We can check whether such a permutation can be found by: (1) Sorting soft clauses $c_1, \ldots, c_m$ in ascending order w.r.t. their weights for complexity $\mathcal{O}(m \log m)$, (2) Iterating through the sorted list and checking each clause c_i whether the inequality holds for complexity $\mathcal{O}(m)$.

The property of Proposition 1 is *sufficient* but *not necessary*. There are other classes of MINUNSAT instances without this property which are reducible in polynomial time, too.

Example 3. $\mathrm{Soft} = \{(x_1, m), \ldots, (x_m, 1)\}$, *i.e., a descending weight for each clause. We assume* $\mathrm{atMost}(x_1, \ldots, x_m) \subseteq \mathrm{Hard}$, *i.e., at most one soft clause is allowed to be true.* MINUNSAT *will try to satisfy the clause with the highest weight. The A-preferred MCS problem with the strict total ordering* $x_1 < \ldots < x_m$ *will be a diagnosis which contains clauses except for the most preferred one in the ordering which can be satisfied under* Hard. *Since at most one of the clauses can be true, the result is the same.*

Next we will show the newly result that actually any MINUNSAT instance is polynomially reducible to the A-preferred MCS problem, i.e., the A-preferred MCS is $\mathrm{FP}^{\mathrm{NP}}$-hard.

Theorem 1. *The A-preferred MCS problem is* $\mathrm{FP}^{\mathrm{NP}}$*-hard.*

Proof. Consider the *Maximum Satisfying Assignment* (MSA) problem: For a Boolean formula φ over the variables $x_1, \ldots, x_n$ find a satisfying assignment with the lexicographical maximum of the word $x_1 \cdots x_n \in \{0,1\}^n$ or 0 if not satisfiable. This problem is $\mathrm{FP}^{\mathrm{NP}}$-complete as proved in [10].

We can polynomially reduce the MSA problem to the A-preferred MCS problem:

$$C_{\mathrm{KB}} := \mathrm{Tseitin}(\varphi)$$
$$C_{\mathrm{R}} := \{\{x_1\}, \ldots, \{x_n\}\}$$
$$< := x_1, \ldots, x_n$$

Since the Tseitin-transformed formula $\mathrm{Tseitin}(\varphi)$ has the same models on the set of the original variables $x_1, \ldots, x_n$ as the original formula ψ (see Remark 1), our reduction is sound. Let $\mathrm{APreMCS}(C_{\mathrm{KB}}, C_{\mathrm{R}})$ be the solution of the constructed A-preferred MCS problem w.r.t. the order $<^{-1}$. Using Proposition 12 of [16], the corresponding L-preferred MSS, which is $\varphi - \mathrm{APreMCS}(C_{\mathrm{KB}}, C_{\mathrm{R}})$ w.r.t. the order $<$, is the solution for the MSA problem. Therefore, the A-preferred MCS problem is $\mathrm{FP}^{\mathrm{NP}}$-hard. □

Corollary 1. *The A-preferred MCS problem is* $\mathrm{FP}^{\mathrm{NP}}$*-complete.*

Proof. The A-preferred MCS problem is $\mathrm{FP}^{\mathrm{NP}}$-hard (Theorem 1) and in $\mathrm{FP}^{\mathrm{NP}}$ [16]. □

Remark 4. *With similar arguments one can prove that the L-preferred MSS problem is* $\mathrm{FP}^{\mathrm{NP}}$*-complete, too. Assuming* $\mathrm{P} \neq \mathrm{NP}$*, then* $\mathrm{FP}^{\mathrm{NP}[\log n]} \subset \mathrm{FP}^{\mathrm{NP}}$ *holds [10]. Hence* $\mathrm{FP}^{\mathrm{NP}}$*-complete problems are strictly harder than problems in* $\mathrm{FP}^{\mathrm{NP}[\log n]}$.

Theorem 1 negatively answers the open question whether computing L-preferred MSSes and A-preferred MCSes could be in $\mathrm{FP}^{\mathrm{NP}[\log n]}$ *or not stated in Remark 1 in [16].*

A practical encoding of a MINUNSAT problem instance as an A-preferred MCS problem could be to encode each soft clause as an unit clause with variable s_i (similiar to Remark 1) and to build the binary representation of the sum:

$$\sum_i w_i \cdot s_i = 2^{l+1} \cdot c_{l+1} + \ldots + 2^0 \cdot c_0$$

Where w_i is the weight of the unit soft clause $\{s_i\}$. Then, design an adder-network of this sum with the output variables $c_{l+1}, \ldots, c_0$ and set $C_{\mathrm{R}} := \{c_{l+1}, \ldots, c_0\}$. The strict total order is given by the order of the coefficients of the binary representation from the most significant bit c_{l+1} to the least significant bit c_0. Set C_{KB} is the union of the set of the hard clauses, the encoding of the soft clauses as unit clause and the encoding of the adder-network of the sum. We will not go into more detail in this work.

8 Experimental Evaluation

We evaluate both problems, the A-preferred MCS problem and the MINUNSAT problem, with real industrial datasets from the automotive domain. In the next subsections we describe the benchmark data and the considered use cases in more detail, afterwards we describe the used solver settings and finally discuss the results.

Table 1. Statistics about the considered POFs

		Rules		Families	
POF	#Var.	Qty.	Avg. #Var.	Qty.	Avg. #Var.
M1_1	996	**11,627**	5.9	**188**	6.3
M1_2	612	4,465	5.3	174	5.3
M2_1	483	495	4.3	60	8.7
M3_1_1	1,772	2,074	33.8	35	56.7
M3_1_2	1,586	1,496	4.6	35	48.9
M3_1_3	1,993	2,281	32.9	35	**61.6**
M3_2_1	**2,087**	2,430	34.0	42	57.2
M3_3_1	880	1,137	19.6	31	29.3
M3_3_2	884	1,121	**55.1**	31	29.4
M3_3_3	885	1,198	47.8	31	29.4

8.1 Benchmark Data

For our benchmarks we use test instances based on real automotive configuration data from three different major German car manufacturers. The configuration model of constructable cars is given as a product overview formula (POF) in Propositional Logic [11]. A POF is satisfiable and each satisfying assignment represents a valid configurable car. If the POF is over-constrained (unsatisfiable), then no cars are constructable. We denote a POF by `Mx_y_z`, where each `x` is a different car manufacturer, each `y` is a different type series and each `z` is a different model type. Each satisfying variable assignment is a constructable car configuration. Table 1 shows statistics about each used POF instance. Column #Var. shows the total variable number of each instance. Rules are Boolean formulas (not necessarily clauses) describing the dependencies between components. Column Qty. shows the number of rules and column Avg. #Var. shows the average number of variables of a rule. Families are groups of components where usually one element has to be chosen, e.g., one motor has to be chosen of the family of motors. But the condition of a family depends on the car manufacturer. Column Qty. shows the number of families and column Avg. #Var. shows the average number of components of a family.

We consider two use cases of re-configuration problems (see [23] for a detailed description of use cases regarding optimization and reconfiguration in the context of automotive configuration):

- **Re-Configuration of Selections:** The constraints of the POF are considered as hard constraints. We choose soft user requirements (variables) at random.
  Since not all user requirements are consistent w.r.t. the POF in general, such a random selection easily gets inconsistent. The goal is to optimize the user selections.
  By this use case we try to realistically imitate a user behavior when configuring a custom car.
- **Re-Configuration of Rules:** The constraints of the POF are considered as soft constraints. We choose user requirements (variables) at random.
  Similarly to the previous use case, such a random user selection easily leads to inconsistency. But in contrast to the previous use case, we want to optimize the POF constraints instead of the user selections.
  By this use case we try to realistically imitate, for example, an engineering situation where new hard requirements are given and the corresponding engineer wants to be guided by an optimized repair suggestion to adjust the rules.

Table 2. Results of MINUNSAT use case "Re-Configuration of Selections" (in seconds)

	30%					50%					70%				
Problem	WPM1	msu4	LSB	IQB	IQBO	WPM1	msu4	LSB	IQB	IQBO	WPM1	msu4	LSB	IQB	IQBO
M1_1	3.19	1.33	0.51	**0.47**	0.48	t/o	4.97	0.50	0.55	**0.49**	t/o	47.96	0.51	0.62	**0.50**
M1_2	2.13	**0.04**	0.24	0.23	0.20	0.77	**0.05**	0.23	0.23	0.22	5.20	**0.08**	0.25	0.24	0.24
M2_1	0.10	0.53	0.09	**0.07**	0.09	0.11	0.75	0.08	**0.07**	0.08	0.12	1.08	0.08	**0.07**	0.08
M3_1_1	1.13	**0.53**	0.89	1.01	0.89	1.19	**0.75**	0.95	1.00	1.00	1.24	1.08	**0.91**	0.98	1.04
M3_1_2	0.12	**0.05**	0.09	0.07	0.09	0.11	**0.07**	0.08	**0.07**	0.08	0.12	**0.08**	**0.08**	**0.08**	**0.08**
M3_1_3	1.34	**0.63**	1.16	1.08	1.17	1.36	**0.96**	1.19	1.09	1.17	1.43	1.30	1.18	**1.11**	1.15
M3_2_1	1.38	**0.71**	1.13	1.03	1.16	1.39	**0.92**	1.21	1.07	1.47	1.48	1.40	1.43	**1.05**	1.24
M3_3_1	0.88	**0.46**	1.00	0.75	1.16	0.87	**0.45**	0.88	0.75	0.94	0.90	**0.61**	0.89	0.79	0.89
M3_3_2	1.59	**0.78**	1.60	1.43	1.62	1.57	**0.90**	1.41	1.46	1.45	1.69	**1.41**	1.59	1.76	1.46
M3_3_3	1.34	**0.63**	1.22	1.23	1.18	1.25	**0.93**	1.18	1.23	1.18	1.31	**0.88**	1.20	1.22	1.21

Table 3. Results of MINUNSAT use case "Re-Configuration of Rules" (in seconds)

	30%					50%					70%				
Problem	WPM1	msu4	LSB	IQB	IQBO	WPM1	msu4	LSB	IQB	IQBO	WPM1	msu4	LSB	IQB	IQBO
M1_1	10.98	t/o	80.45	13.01	**4.97**	48.15	t/o	79.33	22.16	**8.01**	31.69	t/o	79.26	25.25	**9.32**
M1_2	3.24	35.08	7.43	1.78	**0.72**	4.24	89.95	7.36	2.47	**0.89**	5.88	64.91	7.10	3.23	**1.13**
M2_1	0.20	**0.11**	0.25	0.15	0.16	0.19	0.19	0.25	0.18	**0.15**	0.23	0.34	0.25	0.19	**0.16**
M3_1_1	1.88	13.35	9.36	1.97	**1.85**	2.17	41.86	9.49	2.30	**2.04**	**1.89**	14.00	9.13	2.42	2.18
M3_1_2	0.39	0.59	1.48	0.41	**0.27**	0.50	4.98	1.54	0.55	**0.33**	**0.33**	2.15	1.56	0.51	0.34
M3_1_3	2.11	23.69	12.95	2.25	**2.08**	2.44	59.11	12.32	2.75	**2.37**	**2.17**	32.55	12.51	2.84	2.57
M3_2_1	2.40	36.57	11.47	2.56	**2.22**	**2.37**	71.15	14.16	2.99	2.50	**2.29**	65.02	13.95	3.17	2.70
M3_3_1	**1.10**	2.07	5.14	1.24	1.22	**1.18**	10.80	5.24	1.49	1.36	**1.09**	5.26	5.18	1.44	1.36
M3_3_2	**2.04**	5.92	7.78	2.08	2.10	**2.03**	14.19	8.76	2.38	2.27	3.08	8.11	8.54	**2.30**	2.48
M3_3_3	**1.58**	4.62	6.41	1.79	1.78	**1.61**	5.54	6.72	1.89	1.83	**1.57**	6.41	6.73	2.00	1.98

Additionally, we considered three different levels of the user selections: 30%, 50% and 70%. Each level represents the percentage of components chosen from the families, where 100% are all families. By this distinction we want to compare the performance impact of less configured cars in contrast to highly configured cars. A more detailed description of these use cases can be found in [22].

Table 4. Avg. approx. quality of "Re-Config. of Selections"

Problem	30%	50%	70%
M1_1	93.39 %	93.60 %	91.78 %
M1_2	95.79 %	95.94 %	95.76 %
M2_1	100 %	99.47 %	98.79 %
M3_1_1	99.43 %	99.29 %	97.61 %
M3_1_2	100 %	100 %	97.49 %
M3_1_3	99.67 %	100 %	96.62 %
M3_2_1	100 %	98.21 %	96.43 %
M3_3_1	99.57 %	99.44 %	99.17 %
M3_3_2	100 %	97.56 %	98.85 %
M3_3_3	100 %	95.64 %	97.62 %
Avg.	98.79 %	97.92 %	97.01 %

We consider two problem categories:

- **Category I**: MINUNSAT
- **Category II**: A-preferred MCS

For MINUNSAT the weights were chosen between 1 to 10 by random and for A-preferred MCS the order was chosen by random. We assume a uniform distribution of the soft formulas to simulate the interaction of the user. For each POF, each use case and each percentage level we created 10 instances to get a reasonable distribution.

8.2 Implementation Techniques

On the MINUNSAT side we used the following solvers for our evaluation:

- **MSU4:** An unsat core-guided approach with iterative SAT calls using a reduced number of blocking variables [15].
- **WPM1**: An unsat core-guided with iterative SAT calls, see Algorithm 2 and [1]. In each iteration a new blocking variable will be added to each soft clause within the unsat core.

Whereas for the A-preferred MCS side we used:

- **LSB**: Linear search with backbone improvement plus usage of the in-/decremental SAT-Solving interface.
- **IQB**: Basic INVQX with backbone improvement plus usage of the first part of Remark 2.
- **IQBO**: Optimized INVQX with backbone improvement plus full usage of Remark 2.

Solver MSU4 is an external one due to [15]. All other solvers were implemented on top our uniform logic framework, which we use for commercial applications within the context of automotive configuration. Our SAT solver provides an inc-/decremental interface. We maintain two versions (Java and .NET) and decided to implement the solvers in C# using .NET 4.0.

Our experiments were run on the following settings: Processor: Intel Core i7-3520M, 2.90 GHz; Main memory: 8GB. All our .NET

Table 6. Results of A-preferred MCS use case "Re-Configuration of Selections" (in seconds)

	30%					50%					70%				
Problem	WPM1	msu4	LSB	IQB	IQBO	WPM1	msu4	LSB	IQB	IQBO	WPM1	msu4	LSB	IQB	IQBO
M1_1	0.76	2.15	**0.41**	**0.41**	0.44	–	–	**0.42**	0.47	0.49	–	–	**0.52**	0.56	**0.52**
M1_2	0.36	0.64	**0.18**	0.20	0.21	–	–	**0.19**	0.22	**0.19**	–	–	0.26	0.23	**0.22**
M2_1	0.10	**0.05**	0.06	0.06	0.06	0.11	**0.04**	0.06	0.06	0.05	0.11	0.06	0.06	0.07	**0.05**
M3_1_1	1.20	**0.62**	0.75	0.81	0.72	1.34	0.88	0.82	0.89	**0.73**	1.18	1.10	**0.76**	0.91	0.85
M3_1_2	0.10	**0.05**	0.06	0.06	0.06	0.11	**0.06**	**0.06**	**0.06**	**0.06**	0.11	0.08	**0.06**	0.07	**0.06**
M3_1_3	1.27	**0.69**	0.86	0.94	0.86	1.48	**0.73**	0.87	1.00	0.85	1.42	1.23	0.93	1.03	**0.88**
M3_2_1	1.31	0.84	**0.76**	0.98	0.85	1.33	1.10	0.85	0.96	**0.84**	1.39	1.44	**0.93**	0.99	**0.93**
M3_3_1	0.86	0.84	0.59	0.64	**0.56**	0.88	**0.56**	0.65	0.65	0.62	0.90	0.64	**0.63**	0.67	0.67
M3_3_2	1.63	**0.83**	1.08	1.19	1.07	1.59	1.13	**1.07**	1.15	1.08	1.62	**1.08**	1.10	1.17	1.09
M3_3_3	1.28	**0.66**	0.91	0.93	0.88	1.26	**0.86**	**0.86**	0.96	0.92	1.27	1.18	**0.93**	0.98	0.94

Table 7. Results of A-preferred MCS use case "Re-Configuration of Rules" (in seconds)

	30%			50%			70%		
Problem	LSB	IQB	IQBO	LSB	IQB	IQBO	LSB	IQB	IQBO
M1_1	77.00	10.69	**3.65**	73.83	17.95	**6.14**	73.75	19.90	**7.31**
M1_2	6.40	1.54	**0.56**	6.30	1.81	**0.67**	5.96	2.12	**0.81**
M2_1	0.22	0.14	**0.11**	0.22	0.15	**0.12**	0.22	0.18	**0.14**
M3_1_1	8.41	1.88	**1.62**	8.10	2.05	**1.63**	8.35	2.12	**1.71**
M3_1_2	1.36	0.33	**0.20**	1.33	0.46	**0.24**	1.38	0.49	**0.28**
M3_1_3	11.03	1.95	**1.69**	11.04	2.34	**1.94**	11.53	2.35	**2.07**
M3_2_1	9.87	2.20	**1.79**	12.48	2.51	**2.11**	12.12	2.69	**2.14**
M3_3_1	4.24	1.19	**1.02**	4.29	1.31	**1.08**	4.46	1.34	**1.16**
M3_3_2	6.79	2.01	**1.66**	7.25	2.06	**1.79**	7.05	2.10	**1.95**
M3_3_3	5.41	1.61	**1.42**	5.65	1.68	**1.55**	5.70	1.79	**1.61**

Table 5. Avg. approx. quality of "Re-Configuration of Rules"

Problem	30%	50%	70%
M1_1	99.90 %	99.91 %	99.93 %
M1_2	99.94 %	99.94 %	99.86 %
M2_1	99.82 %	99.56 %	99.70 %
M3_1_1	100 %	99.98 %	100 %
M3_1_2	100 %	99.96 %	99.99 %
M3_1_3	99.98 %	99.99 %	99.99 %
M3_2_1	100 %	99.99 %	99.97 %
M3_3_1	99.95 %	99.89 %	99.99 %
M3_3_2	99.96 %	99.94 %	99.97 %
M3_3_3	99.97 %	99.98 %	99.95 %
Avg.	99.95 %	99.91 %	99.94 %

based algorithms run under Windows 7 while MSU4 runs under Ubuntu 12.04.

8.3 Results

For all of the following result tables, each table entry shows the average time in seconds a solver needed to solve 10 different instances of the considered use case and POF. We set a timeout (t/o) of 600 seconds. Each table is separated in three levels (30%, 50% and 70%) which is the percentage of families where user selections were made from.

8.3.1 Category I: Partial Weighted MINUNSAT

Table 2 and Table 3 show the results of both use cases, respectively. MSU4 performs well on the first use case, whereas IQBO performs most often best on the second use case.

Table 4 and Table 5 show the average approximation quality of the A-preferred MCS approaches used as MINUNSAT approximation (cf. Subsection 6.1), which turns out to be quite good. Percentage p is calculated as follows:

$$p = \frac{(\sum_{i=1}^{m} w_i) - \text{approxOpt}}{(\sum_{i=1}^{m} w_i) - \text{exactOpt}}$$

Where approxOpt is the optimum of the A-preferred MCS solver and exactOpt is the optimum of the MINUNSAT solver.

8.3.2 Category II: A-preferred MCS

Table 6 does not contain entries for M1_1 and M1_2 in the columns 50% and 70% because the encoding of the weights exceeds the

native *long* data type. The MaxSAT competition format[3] permits a top weight lower than 2^{63}. Table 7 is missing an evaluation for MINUNSAT solvers for the same reason.

For instances where an encoding is possible, both MINUNSAT solvers can keep up with the native A-preferred MCS solvers. IQBO performs significantly better than IQB for the second use case.

9 Conclusions and Future Work

In this work, we compared the problem of finding a minimal preferred diagnosis (A-preferred MCS in the context of Propositional Logic) with the problem of finding a diagnosis of minium cardinality (MINUNSAT in the context of Propositional Logic). We proved the $\mathrm{FP}^{\mathrm{NP}}$-hardness of the A-preferred MCS problem and therefore showed, that both problems are equally hard to solve and reducible to each other.

Both optimization approaches (A-preferred MCS and MINUNSAT) complement each other. For use cases which can be expressed as an A-preferred MCS problem one should use INVQX as solving engine. For the other use cases, one should use MINUNSAT.

For instances where MINUNSAT is not able to find the solution in within a reasonable time or where fast responses are needed, we can fall back to INVQX if an approximated answer is reasonable for the considered use case.

We evaluated the performance of both problems with benchmarks based on real automotive configuration data. We assumed a uniform distribution of the soft formulas to simulate the interaction of the user. These benchmarks can be improved by choosing a more realistic distribution of the soft formulas.

For an exhaustive evaluation of both problems, we need to consider more complex benchmark data, e.g. unsatisfiable instances of the SAT[4] or MaxSAT[5] competitions. Also, we need to evaluate all solvers based on the same SAT solving engine, i.e. MINISAT [3], in order to build up an uniform environment.

Another interesting evaluation could be the reduction of the MINUNSAT problem to an A-preferred MCS problem by using an adder-network encoding as described in Subsection 7.2. With such an encoding, we could use every A-preferred MCS solver, like INVQX, to solve MINUNSAT problem instances.

INVQX and MINUNSAT in their original form compute only a single diagnosis. Both approaches can be extended to compute a set of all diagnoses. We plan to investigate the relationship of these extensions and to evaluate them, too.

REFERENCES

[1] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy, 'Solving (weighted) partial MaxSAT through satisfiability testing', in *Theory and Applications of Satisfiability Testing - SAT 2009*, ed., Oliver Kullmann, volume 5584 of *Lecture Notes in Computer Science*, 427–440, Springer Berlin Heidelberg, (2009).

[2] Josep Argelich and Felip Manyà, 'Exact Max-SAT solvers for over-constrained problems.', *Journal of Heuristics*, **12**(4–5), 375–392, (September 2006).

[3] Niklas Eén and Niklas Sörensson, 'An extensible SAT-solver', in *Theory and Applications of Satisfiability Testing—SAT 2003*, eds., Enrico Giunchiglia and Armando Tacchella, volume 2919 of *Lecture Notes in Computer Science*, 502–518, Springer Berlin Heidelberg, (2004).

[3] www.maxsat.udl.cat/13/requirements/index.html

[4] http://www.satcompetition.org/

[5] www.maxsat.udl.cat

[4] Alexander Felfernig, Monika Schubert, and Christoph Zehentner, 'An efficient diagnosis algorithm for inconsistent constraint sets', *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, **26**(1), 53–62, (February 2012).

[5] Zhaohui Fu and Sharad Malik, 'On solving the partial MAX-SAT problem', in *Theory and Applications of Satisfiability Testing—SAT 2006*, eds., Armin Biere and Carla P. Gomes, volume 4121 of *Lecture Notes in Computer Science*, 252–265, Springer Berlin Heidelberg, (2006).

[6] Federico Heras, António Morgado, and João Marques-Silva, 'Core-guided binary search algorithms for maximum satisfiability', in *AAAI*, eds., Wolfram Burgard and Dan Roth, pp. 36 – 41. AAAI Press, (August 2011).

[7] Federico Heras, António Morgado, and João Marques-Silva, 'An empirical study of encodings for group MaxSAT', in *Canadian Conference on AI*, eds., Leila Kosseim and Diana Inkpen, volume 7310 of *Lecture Notes in Computer Science*, pp. 85–96. Springer, (2012).

[8] Federico Heras, António Morgado, and João Marques-Silva, 'Lower bounds and upper bounds for MaxSAT', in *Learning and Intelligent Optimization – 6th International Conference, LION 6, Paris, France, January 16-20, 2012, Revised Selected Papers*, eds., Youssef Hamadi and Marc Schoenauer, volume 7219 of *Lecture Notes in Computer Science*, pp. 402–407. Springer Berlin Heidelberg, (2012).

[9] Ulrich Junker, 'QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems', in *Proceedings of the 19th National Conference on Artificial Intelligence*, pp. 167–172. AAAI Press / The MIT Press, (2004).

[10] Mark W. Krentel, 'The complexity of optimization problems', *Journal of Computer and System Sciences*, **36**(3), 490–509, (June 1988).

[11] Wolfgang Küchlin and Carsten Sinz, 'Proving consistency assertions for automotive product data management', *Journal of Automated Reasoning*, **24**(1–2), 145–163, (2000).

[12] Adrian Kügel, 'Improved exact solver for the weighted MAX-SAT problem', in *POS-10. Pragmatics of SAT*, ed., Daniel Le Berre, volume 8 of *EasyChair Proceedings in Computing*, pp. 15–27. EasyChair, (2012).

[13] Chu Min Li and Felip Manyà, 'MaxSAT, hard and soft constraints', in *Handbook of Satisfiability*, eds., Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 19, 613–631, IOS Press, (2009).

[14] João Marques-Silva, Federico Heras, Mikoláš Janota, Alessandro Previti, and Anton Belov, 'On computing minimal correction subsets.', in *IJCAI*, ed., Francesca Rossi, pp. 615–622. IJCAI/AAAI, (2013).

[15] João Marques-Silva and Jordi Planes, 'Algorithms for maximum satisfiability using unsatisfiable cores', in *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '08, pp. 408–413. IEEE, (2008).

[16] João Marques-Silva and Alessandro Previti, 'On computing preferred MUSes and MCSes', in *Theory and Applications of Satisfiability Testing – SAT 2014*, eds., Carsten Sinz and Uwe Egly, volume 8561 of *Lecture Notes in Computer Science*, pp. 58–74. Springer International Publishing, (July 2014).

[17] Carlos Mencía and João Marques-Silva, 'Efficient relaxations of over-constrained CSPs', in *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, pp. 725–732. IEEE, (2014).

[18] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva, 'Iterative and core-guided MaxSAT solving: A survey and assessment.', *Constraints*, **18**(4), 478–534, (2013).

[19] Christos M. Papadimitriou, *Computational complexity*, Addison-Wesley, Reading, Massachusetts, 1994.

[20] David A. Plaisted and Steven Greenbaum, 'A structure-preserving clause form translation', *Journal of Symbolic Computation*, **2**(3), 293–304, (September 1986).

[21] Grigorii S. Tseitin, 'On the complexity of derivations in the propositional calculus', *Studies in Constructive Mathematics and Mathematical Logic*, **Part II**, 115–125, (1968).

[22] Rouven Walter and Wolfgang Küchlin, 'ReMax – a MaxSAT aided product configurator', in *Proceedings of the 16th International Configuration Workshop*, eds., Alexander Felfernig, Cipriano Forza, and Albert Haag, pp. 59–66, Novi Sad, Serbia, (September 2014).

[23] Rouven Walter, Christoph Zengler, and Wolfgang Küchlin, 'Applications of MaxSAT in automotive configuration', in *Proceedings of the 15th International Configuration Workshop*, eds., Michel Aldanondo and Andreas Falkner, pp. 21–28, Vienna, Austria, (August 2013).

FLEXDIAG: AnyTime Diagnosis for Reconfiguration

Alexander Felfernig[1] and **Rouven Walter**[2] and **Stefan Reiterer**[1]

Abstract. Anytime diagnosis is able to determine solutions within predefined time limits. This is especially useful in realtime scenarios such as production scheduling, robot control, and communication networks management where diagnosis and corresponding reconfiguration capabilities play a major role. Anytime diagnosis in many cases comes along with a tradeoff between diagnosis quality and the efficiency of diagnostic reasoning. In this paper we introduce and analyze FLEXDIAG which is an anytime variant of existing direct diagnosis approaches. We evaluate the algorithm with regard to performance and diagnosis quality using a configuration benchmark.

Keywords: Anytime Diagnosis, Reconfiguration.

1 Introduction

Knowledge-based configuration is one of the most successful applications of Artificial Intelligence [7, 24]. There are many different applications of configuration technologies ranging from *telecommunication infrastructures* [11], *railway interlocking systems* [5], the *automotive domain* [22, 26, 28] to the *configuration of services* [27]. Configuration technologies must be able to deal with inconsistencies which can occur in different contexts. First, a configuration knowledge base can be inconsistent, i.e., no solution can be determined. In this context, the task of knowledge engineers is to figure out which constraints are responsible for the unintended behavior of the knowledge base. Bakker et al. [1] show the application of model-based diagnosis [19] to determine minimal sets of constraints in a knowledge base that are responsible for a given inconsistency. A variant thereof is documented in Felfernig et al. [6] where an approach to the automated debugging of knowledge bases with test cases is introduced. Felfernig et al. [6] also show how to diagnose customer requirements that are inconsistent with a configuration knowledge base. The underlying assumption is that the configuration knowledge base itself is consistent but combined with a set of requirements is inconsistent.

All diagnosis approaches mentioned so far are based on conflict-directed hitting set determination [15, 19]. These approaches typically determine diagnoses in a breadth-first search manner which allows the identification of minimal cardinality diagnoses. The major disadvantage of applying these approaches is the need of predetermining minimal conflicts which is inefficient especially in cases where only the leading diagnoses (the most relevant ones) are sought.

[1] Applied Software Engineering Group, Institute for Software Technology, TU Graz, Austria, email: {alexander.felfernig, stefan.reiterer}@ist.tugraz.at

[2] Symbolic Computation Group, WSI Informatics, Universität Tübingen, Germany, email: rouven.walter@uni-tuebingen.de

Anytime diagnosis algorithms are useful in scenarios where diagnoses have to be provided in real-time, i.e., within given time limits. If diagnosis is applied in *interactive configuration*, for example, to determine repairs for inconsistent customer requirements, response times should be below one second [2]. Efficient diagnosis and reconfiguration of *communication networks* is crucial to retain the quality of service [18, 25]. In today's production scenarios which are characterized by small batch sizes and high product variability, it is increasingly important to develop algorithms that support the efficient *reconfiguration of schedules*. Such functionalities support the paradigm of *smart production*, i.e., the flexible and efficient production of highly variant products. Further applications are the diagnosis and repair of *robot control software* [23], the *reconfiguration of cars* [29], and the *reconfiguration of buildings* [12].

Algorithmic approaches to provide efficient solutions for diagnosis problems are manyfold. Some approaches focus on improvements of Reiter's original hitting set directed acyclic graph (HSDAG) [19] in terms of a personalized computation of leading diagnoses [3] or other extensions that make the basic approach [19] more efficient [31]. Wang et al. [30] introduce an approach to derive binary decision diagrams (BDDs) on the basis of a pre-determined set of conflicts – diagnoses can then be determined by solving the BDD. A pre-defined set of conflicts can also be compiled into a corresponding linear optimization problem [10]; diagnoses can then be determined by solving the given problem. In knowledge-based recommendation scenarios, diagnoses for user requirements can be pre-compiled in such a way that for a given set of customer requirements, the diagnosis search task can be reduced to querying a relational table (see, for example, [14, 20]). All of the mentioned approaches either extend the approach of Reiter [19] or improve efficiency by exploiting pre-generated information about conflicts or diagnoses.

An alternative to conflict-directed diagnosis [19] are *direct diagnosis* algorithms that determine minimal diagnoses without the need of pre-determing minimal conflict sets [9, 17, 21]. The FASTDIAG algorithm [9] is a divide-and-conquer based algorithm that supports the determination of diagnoses without a preceding conflict detection. In this paper we show how this algorithm can be converted into an anytime diagnosis algorithm (FLEXDIAG) that is able to improve performance by disregarding the aspect of minimality, i.e., the algorithm allows for tradeoffs between diagnosis quality (e.g., minimality) and performance of diagnostic search. In this paper we focus on *reconfiguration scenarios*, i.e., we show how FLEXDIAG can be applied in situations where a given configuration (solution) has to be adapted conform to a changed set of customer requirements.

Our contributions in this paper are the following. First, we show how to solve reconfiguration tasks with direct diagnosis. Second, we make direct diagnosis anytime-aware by including a parametrization

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

that helps to systematically reduce the number of consistency checks. Finally, we report the results of a FLEXDIAG-related evaluation conducted on the basis of a configuration benchmark.

The remainder of this paper is organized as follows. In Section 2 we introduce an example configuration knowledge base from the domain of resource allocation. This knowledge base will serve as a working example throughout the paper. Thereafter (Section 3) we introduce a definition of a reconfiguration task. In Section 4 we discuss basic principles of direct diagnosis on the basis of FLEXDIAG and show how this algorithm can be applied in reconfiguration scenarios. In Section 5 we present the results of a performance analysis. A simple example of the application of FLEXDIAG in production environments is given in Section 6. In Section 7 we discuss major issues for future work. With Section 8 we conclude the paper.

2 Example Configuration Knowledge Base

A configuration system determines configurations (solutions) on the basis of a given set of customer requirements [13]. In many cases, constraint satisfaction problem (CSP) representations [16] are used for the definition of a configuration task. A configuration task and a corresponding configuration (solution) can be defined as follows.

Definition 1 (Configuration Task and Configuration). A configuration task can be defined as a CSP (V, D, C) where $V = \{v_1, v_2, ..., v_n\}$ is a set of variables, $D = \cup dom(v_i)$ represents domain definitions, and $C = \{c_1, c_2, ..., c_m\}$ is a set of constraints. Additionally, user requirements are represented by a set of constraints $R = \{r_1, r_2, ..., r_k\}$. A configuration (solution) for a configuration task is a set of assignments (constraints) $S = \{s_1 : v_1 = a_1, s_2 : v_2 = a_2, ..., s_n : v_n = a_n)\}$ where $a_i \in dom(v_i)$ which is consistent with $C \cup R$.

An example of a configuration task represented as a constraint satisfaction problem is the following.

Example (Configuration Task). In this resource allocation problem example, items (a *barrel of fuel*, a *stack of paper*, a *pallet of fireworks*, a *pallet of personal computers*, a *pallet of computer games*, a *barrel of oil*, a *palette of roof tiles*, and a *palette of rain pipes*) have to be assigned to three different containers. There are a couple of constraints (c_i) to be taken into account, for example, fireworks must not be combined with fuel (c_1). Furthermore, there is one requirement (r_1) which indicates that the palette of fireworks has to be assigned to container 1. On the basis of this configuration task definition, a configurator can determine a configuration S.

- $V = \{fuel, paper, fireworks, pc, games, oil, roof, pipes\}$
- $dom(fuel) = dom(paper) = dom(fireworks) = dom(pc) = dom(games) = dom(oil) = dom(roof) = dom(pipes) = \{1, 2, 3\}$
- $C = \{c_1 : fireworks \neq fuel, c_2 : fireworks \neq paper, c_3 : fireworks \neq oil, c_4 : pipes = roof, c_5 : paper \neq fuel\}$
- $R = \{r_1 : fireworks = 1\}$
- $S = \{s_1 : pc = 3, s_2 : games = 1, s_3 : paper = 2, s_4 : fuel = 3, s_5 : fireworks = 1, s_6 : oil = 2, s_7 : roof = 1, s_8 : pipes = 1\}$

On the basis of the given definition of a configuration task, we now introduce the concept of reconfiguration (see also [12, 18, 25, 28]).

3 Reconfiguration Task

It can be the case that an existing configuration S has to be adapted due to a change or extension of the given set of customer requirements. Examples thereof are changing requirements that have to be taken into account in production schedules, failing components or overloaded network infrastructures in a mobile phone network, and changes in the internal model of the environment of a robot. In the following we assume that the *palette of paper* should be reassigned to container 3 and the *personal computer* and *games palettes* should be assigned to the same container. Formally, the set of new requirements is represented by $R_\rho : \{r'_1 : pc = games, r'_2 : paper = 3\}$. In order to determine reconfigurations, we have to calculate a corresponding diagnosis Δ (see Definition 2).

Definition 2 (Diagnosis). A diagnosis Δ (correction subset) is a subset of $S = \{s_1 : v_1 = a_1, s_2 : v_2 = a_2, ..., s_n : v_n = a_n\}$ such that $S - \Delta \cup C \cup R_\rho$ is consistent. Δ is minimal if there does not exist a diagnosis Δ' with $\Delta' \subset \Delta$.

On the basis of the definition of a minimal diagnosis, we can introduce a formal definition of a reconfiguration task.

Definition 3 (Reconfiguration Task and Reconfiguration). A reconfiguration task can be defined as a CSP (V, D, C, S, R_ρ) where V is a set of variables, D represents variable domain definitions, C is a set of constraints, S represents an existing configuration, and $R_\rho = \{r'_1, r'_2, ..., r'_k\}$ (R_ρ consistent with C) represents a set of reconfiguration requirements. A reconfiguration is a variable assignment $S_\Delta = \{s_1 : v_1 = a'_1, s_2 : v_2 = a'_2, ..., s_l : v_l = a'_l\}$ where $s_i \in \Delta$, $a'_i \neq a_i$, and $S - \Delta \cup S_\Delta \cup C \cup R_\rho$ is consistent.

If R_ρ is inconsistent with C, the new requirements have to be analyzed and changed before a corresponding reconfiguration task can be triggered [4, 8]. An example of a reconfiguration task in the context of our configuration knowledge base is the following.

Example (Reconfiguration Task). In the resource allocation problem, the original customer requirements R are substituted by the requirements $R_\rho = \{r'_1 : pc = games, r'_2 : paper = 3\}$. The resulting reconfiguration task instance is the following.

- $V = \{fuel, paper, fireworks, pc, games, oil, roof, pipes\}$
- $dom(fuel) = dom(paper) = dom(fireworks) = dom(pc) = dom(games) = dom(oil) = dom(roof) = dom(pipes) = \{1, 2, 3\}$
- $C = \{c_1 : fireworks \neq fuel, c_2 : fireworks \neq paper, c_3 : fireworks \neq oil, c_4 : pipes = roof, c_5 : paper \neq fuel\}$
- $S = \{s_1 : pc = 3, s_2 : games = 1, s_3 : paper = 2, s_4 : fuel = 3, s_5 : fireworks = 1, s_6 : oil = 2, s_7 : roof = 1, s_8 : pipes = 1\}$
- $R_\rho = \{r'_1 : pc = games, r'_2 : paper = 3\}$

To solve a reconfiguration task (see Definition 3), conflict-directed diagnosis approaches [19] would determine a set of minimal conflicts and then determine a hitting set that resolves each of the identified conflicts. In this context, a minimal conflict set $CS \subseteq S$ is a minimal set of variable assignments that trigger an inconsistency with $C \cup R_\rho$, i.e., $CS \cup C \cup R_\rho$ is inconsistent and there does not exist a conflict set CS' with $CS' \subset CS$. In our working example, the minimal conflict sets are $CS_1 : \{s_1 : pc = 3, s_2 : games = 1\}$, $CS_2 : \{s_3 : paper = 2\}$, and $CS_3 : \{s_4 : fuel = 3\}$. The corresponding minimal diagnoses are $\Delta_1 : \{s_1, s_3, s_4\}$ and $\Delta_2 : \{s_2, s_3, s_4\}$. The elements in a diagnosis indicate which variable assignments have to be adapted such that a reconfiguration can be determined that takes into account the new requirements in R_ρ. If we choose Δ_1, the reconfigurations (reassignments) for the variable assignments in Δ_1 can be determined by a CSP solver call $C \cup R_\rho \cup (S - \Delta_1)$. The resulting configuration S' can be $\{s_1 : pc = 1, s_2 : games = 1, s_3 : paper = 3, s_4 : fuel = 2, s_5 : fireworks = 1, s_6 : oil = 2, s_7 : roof = 1, s_8 : pipes = 1\}$. For

a detailed discussion of conflict-based diagnosis we refer to Reiter [19]. In the following we introduce an approach to the determination of minimal reconfigurations which is based on a *direct diagnosis* algorithm, i.e., diagnoses are determined without the need of determining related minimal conflict sets.

4 Reconfiguration with FLEXDIAG

In the following discussions, the set $AC = C \cup R_\rho \cup S$ represents the union of all constraints that restrict the set of possible solutions for a given reconfiguration task. Furthermore, S represents a set of constraints that are considered as candidates for being included in a diagnosis Δ. The idea of FLEXDIAG (Algorithm 1) is to systematically filter out the constraints that become part of a minimal diagnosis using a divide-and-conquer based approach.

Algorithm 1 − FLEXDIAG.

```
1  func FLEXDIAG(S, AC = C ∪ R_ρ ∪ S) : Δ
2  if isEmpty(S) or inconsistent(AC − S) return ∅
3  else return FLEXD(∅, S, AC);

4  func FLEXD(D, S = {s_1..s_q}, AC) : Δ
5  if D ≠ 0 and consistent(AC) return ∅;
6  if size(S) ≤ m return S;
7  k = q/2;
8  S_1 = {s_1..s_k}; S_2 = {s_{k+1}..s_q};
9  D_1 = FLEXD(S_1, S_2, AC − S_1);
10 D_2 = FLEXD(D_1, S_1, AC − D_1);
11 return(D_1 ∪ D_2);
```

In our example reconfiguration task, the original configuration $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$ and the new set of customer requirements is $R_\rho = \{r'_1, r'_2\}$. Since $S \cup R_\rho \cup C$ is inconsistent, we are in the need of a minimal diagnosis Δ and a reconfiguration S_Δ such that $S - \Delta \cup S_\Delta \cup R_\rho \cup C$ is consistent. In the following we will show how the FLEXDIAG (Algorithm 1) can be applied to determine a minimal diagnosis Δ.

The FLEXDIAG algorithm is assumed to be activated under the assumption that AC is inconsistent, i.e., the consistency of AC is not checked by the algorithm. If AC is inconsistent but $AC - S$ is also inconsistent, FLEXDIAG will not be able to identify a diagnosis in S; therefore $\emptyset$ is returned. Otherwise, a recursive function FLEXD is activated which is in charge of determining one minimal diagnosis Δ. In each recursive step, the constraints in S are divided into two different subsets (S_1 and S_2) in order to figure out if already one of these subsets includes a diagnosis. If this is the case, the second set must not be inspected for diagnosis elements anymore.

FLEXDIAG is based on the concepts of FASTDIAG [9], i.e., it returns one diagnosis (Δ) at a time and is complete in the sense that if a diagnosis is contained in S, then the algorithm will find it. A corresponding reconfiguration can be determined by a solver call $C \cup R_\rho \cup (S - \Delta)$. The determination of multiple diagnoses at a time can be realized on the basis of the construction of a HSDAG [19]. If $m = 1$ (see Algorithm 1), the number of consistency checks needed for determining one minimal diagnosis is $2\delta \times log_2(\frac{n}{\delta}) + 2\delta$ in the worst case [9]. In this context, δ represents the set size of the minimal diagnosis Δ and n represents the number of constraints in solution S.

If $m > 1$, the number of needed consistency checks can be systematically reduced if we accept the tradeoff of possibly loosing the property of diagnosis minimality (see Definition 2). If we allow settings with $m > 1$, we can reduce the upper bound of the number of consistency checks to $2\delta \times log_2(\frac{2n}{\delta \times m})$ in the worst case. These upper bounds regarding the number of needed consistency checks allow to estimate the worst case runtime performance of the diagnosis algorithm which is extremely important for realtime scenarios. Consequently, if we are able to estimate the upper limit of the runtime needed for completing one consistency check (e.g., on the basis of simulations with an underlying constraint solver), we are also able to figure out lower bounds for m that must be chosen in order to guarantee a FLEXDIAG runtime within predefined time limits.

Table 1 depicts an overview of consistency checks needed depending on the setting of the parameter m and the diagnosis size δ for $|S| = 16$. For example, if $m = 2$ and the size of a minimal diagnosis is $\delta = 4$, then the upper bound for the number of needed consistency checks is 16. If the size of δ increases further, the number of corresponding consistency checks does not increase anymore. Figures 1 and 2 depict FLEXDIAG search trees depending on the setting of granularity parameter m.

δ	m=1	m=2	m=4	m=8
1	10	8	6	4
2	16	12	8	4
4	24	16	8	4
8	32	16	16	16

Table 1. Worst-case estimates for the number of needed consistency checks depending on the granularity parameter m and the diagnosis size δ for $|S| = 16$.

FLEXDIAG determines one diagnosis at a time which indicates variable assignments of the original configuration that have to be changed such that a reconfiguration conform to the new requirements (R_ρ) is possible. The algorithm supports the determination of *leading diagnoses*, i.e., diagnoses that are preferred with regard to given user preferences [9]. FLEXDIAG is based on a strict lexicographical ordering of the constraints in S: the lower the importance of a constraint $s_i \in S$ the lower the index of the constraint in S. For example, $s_1 : pc = 3$ has the lowest ranking. The lower the ranking, the higher the probability that the constraint will be part of a reconfiguration S_Δ. Since s_1 has the lowest priority and it is part of a conflict, it is element of the diagnosis returned by FLEXDIAG. For a discussion of the properties of lexicographical orderings we refer to [9, 15].

5 Evaluation

In order to evaluate FLEXDIAG, we analyzed the two major aspects of (1) *algorithm performance* and (2) *diagnosis quality* in terms of *minimality* and *accuracy*. We analyzed both aspects by varying the value of parameter m. Our hypothesis in this context was that the higher the value of m, the lower the number of needed consistency checks (the higher the efficiency of diagnosis search) and the lower diagnosis quality in terms of the share of diagnosis-relevant constraints returned by FLEXDIAG. Diagnosis quality can, for example, be measured by the degree of minimality of the constraints contained in a diagnosis Δ returned by FLEXDIAG (see Formula 1).

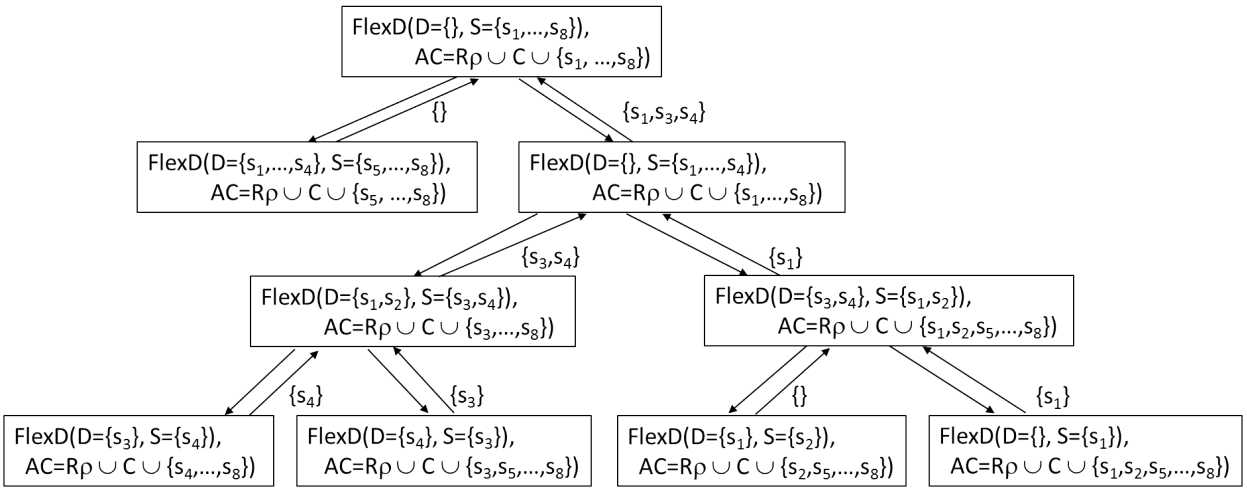


Figure 1. FLEXDIAG: determining one minimal diagnosis with $m = 1$ ($\Delta = \{s_1, s_3, s_4\}$).

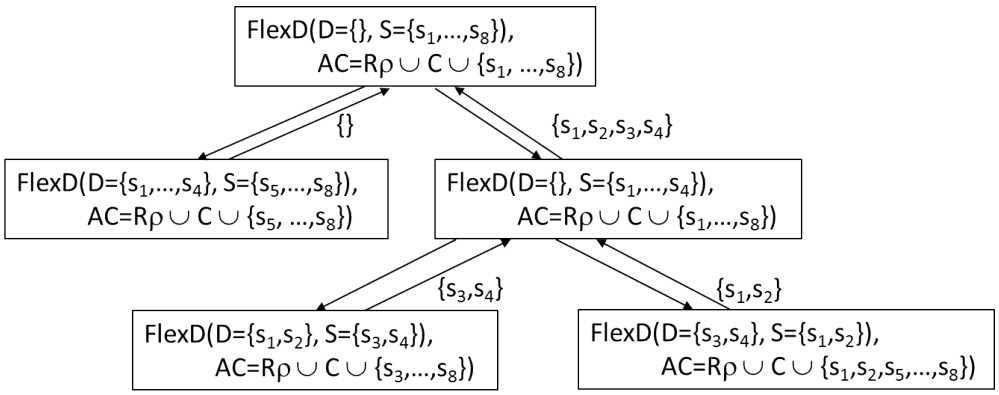


Figure 2. FLEXDIAG: determining a minimal diagnosis with $m = 2$ ($\Delta = \{s_1, s_2, s_3, s_4\}$).

id	$\|V\|$	$\|C\|$	$\|R_\rho\|$	$\|\Delta_{min}\|$	avg. runtime (ms)					minimality (accuracy)				
					m=1	m=2	m=4	m=6	m=10	m=1	m=2	m=4	m=6	m=10
1	47	285	5	4	772	647	561	429	350	1.0(1.0)	0.56(1.0)	0.45(1.0)	0.21(1.0)	0.15(1.0)
2	55	73	10	7	359	273	203	180	85	1.0(1.0)	0.55(0.91)	0.31(0.91)	0.30(0.83)	0.41(0.58)
3	73	150	10	5	733	593	343	304	195	1.0(1.0)	0.64(0.91)	0.83(0.55)	0.83(0.55)	1.07(0.30)
4	158	242	20	24	2176	1864	1451	1419	819	1.0(1.0)	0.66(0.89)	0.43(0.82)	0.41(0.67)	0.33(0.65)

Table 2. Avg. runtime and minimality of FLEXDIAG evaluated with feature models from www.splot-research.org (calculation of the first diagnosis: 1: Dell laptops, 2: Smarthomes, 3: Cars, 4: Xerox printers). $|R_\rho|$ is the number of changed requirements and $|\Delta_{min}|$ the average cardinality of minimal diagnoses.

$$minimality(\Delta) = \frac{|\Delta_{min}|}{|\Delta|} \quad (1)$$

If $m > 1$, there is no guarantee that the diagnosis Δ determined for S is a superset of the diagnosis Δ_{min} determined for S in the case $m = 1$. Besides minimality, we introduce accuracy as an additional quality indicator (see Formula 2). The higher the share of elements of Δ_{min} in Δ, the higher the corresponding accuracy (the algorithm is able to reproduce the elements of the minimal diagnosis for $m = 1$).

$$accuracy(\Delta) = \frac{|\Delta \cap \Delta_{min}|}{|\Delta_{min}|} \quad (2)$$

In order to evaluate FLEXDIAG with regard to both aspects we applied the algorithm to the configuration benchmark from www.splot-research.org - the configuration models are feature models which include requirement constraints, compatibility constraints, and different types of structural constraints such as mandatory relationships and alternatives. The feature models were represented as CSP on the basis of the Java-based Choco library.[3] For each setting (see Table 2) in the benchmark, we randomly generated $|R_\rho|$ new requirements that were inconsistent with an already determined configuration (10 iterations per setting). The average cardinality of a minimal diagnosis for $m = 1$ is $|\Delta_{min}|$. Related average runtimes (in milliseconds)[4] and degrees of minimality and accuracy (see Formula 1) are depicted in Table 2. As can be seen in Table 2, increasing the value of m leads to an improved runtime performance in our example cases. Minimality and accuracy depend on the configuration domain and are not necessarily monotonous. For example, since a diagnosis determined by FLEXDIAG is not necessarily a superset of a diagnosis determined with $m = 1$, it can be the case that the *minimality* of a diagnosis determined with $m > 1$ is greater than 1 (if FLEXDIAG determines a diagnosis with lower cardinality than the minimal diagnosis determined with $m = 1$).

Note that in this paper we did not compare FLEXDIAG with more traditional diagnosis approaches – for related evaluations we refer the reader to [9] were detailed related analyses can be found. The outcome of these analyses is that direct diagnosis approaches such as FLEXDIAG clearly outperform standard diagnosis approaches based on the resolution of minimal conflicts [19].

6 Reconfiguration in Production

The following simplified reconfiguration task is related to *scheduling in production* where it is often the case that, for example, schedules and corresponding production equipment has to be reconfigured. In this example setting we do not take into account configurable production equipment (configurable machines) and limit the reconfiguration to the assignment of orders to corresponding machines. The assignment of an order o_i to a certain machine m_j is represented by the corresponding variable $o_i m_j$. The domain of each such variable represents the different possible slots in which an order can be processed, for example, $o_1 m_1 = 1$ denotes the fact that the processing of order o_1 on machine m_1 is performed during and finished after time slot 1.

Further constraints restrict the way in which orders are allowed to be assigned to machines, for example, $o_1 m_1 < o_1 m_2$ denotes the fact that order o_1 must be completed on machine m_1 before a further processing is started on machine m_2. Furthermore, no two orders must be assigned to the same machine during the same time slot, for example, $o_1 m_1 \neq o_2 m_1$ denotes the fact that order o_1 and o_2 must not be processed on the same machine in the same time slot (slots 1..3). Finally, the definition of our reconfiguration task is completed with an already determined schedule S and a corresponding reconfiguration request represented by the reconfiguration requirement $R_\rho = \{r'_1 : o_3 m_3 < 5\}$, i.e., order o_3 should be completed within less than 5 time units.

- $V = \{o_1 m_1, o_1 m_2, o_1 m_3, o_2 m_1, o_2 m_2, o_2 m_3, o_3 m_1, o_3 m_2, o_3 m_3\}$
- $dom(o_1 m_1) = dom(o_2 m_1) = dom(o_3 m_1) = \{1, 2, 3\}$. $dom(o_1 m_2) = dom(o_2 m_2) = dom(o_3 m_2) = \{2, 3, 4\}$. $dom(o_1 m_3) = dom(o_2 m_3) = dom(o_3 m_3) = \{3, 4, 5\}$.
- $C = \{c_1 : o_1 m_1 < o_1 m_2, c_2 : o_1 m_2 < o_1 m_3, c_3 : o_2 m_1 < o_2 m_2, c_4 : o_2 m_2 < o_2 m_3, c_5 : o_3 m_1 < o_3 m_2, c_6 : o_3 m_2 < o_3 m_3, c_7 : o_1 m_1 \neq o_2 m_1, c_8 : o_1 m_1 \neq o_3 m_1, c_9 : o_2 m_1 \neq o_3 m_1, c_{10} : o_1 m_2 \neq o_2 m_2, c_{11} : o_1 m_2 \neq o_3 m_2, c_{12} : o_2 m_2 \neq o_3 m_2, c_{13} : o_1 m_3 \neq o_2 m_3, c_{14} : o_1 m_3 \neq o_3 m_3, c_{15} : o_2 m_3 \neq o_3 m_3\}$
- $S = \{s_1 : o_1 m_1 = 1, s_2 : o_1 m_2 = 2, s_3 : o_1 m_3 = 3, s_4 : o_2 m_1 = 2, s_5 : o_2 m_2 = 3, s_6 : o_2 m_3 = 4, s_7 : o_3 m_1 = 3, s_8 : o_3 m_2 = 4, s_9 : o_3 m_3 = 5\}$
- $R_\rho = \{r'_1 : o_3 m_3 < 5\}$

This reconfiguration task can be solved using FLEXDIAG. If we keep the ordering of the constraints as defined in S, FLEXDIAG (with $m = 1$) returns the diagnosis $\Delta : \{s_4, s_5, s_6, s_7, s_8\}$ which can be used to determine the new solution $S = \{s_1 : o_1 m_1 = 1, s_2 : o_1 m_2 = 2, s_3 : o_1 m_3 = 3, s_4 : o_2 m_1 = 3, s_5 : o_2 m_2 = 4, s_6 : o_2 m_3 = 5, s_7 : o_3 m_1 = 2, s_8 : o_3 m_2 = 3, s_9 : o_3 m_3 = 4\}$. Possible ordering criteria for constraints in such rescheduling scenarios can be, for example, customer value (changes related to orders of important customers should occur with a lower probability) and the importance of individual orders. If some orders in a schedule should not be changed, this can be achieved by simply defining such requests as requirements (R_ρ), i.e., change requests as well as stability

[3] choco-solver.org.

[4] Test platform: Windows 7 Professional 64 Bit, Intel(R) Core(TM) i5-2320 3.00 GHz CPU with 8.00 GB of memory.

requests can be included as constraints r'_i in R_ρ.

7 Ongoing And Future Work

We are currently evaluating FLEXDIAG with a more complex (industrial) benchmark from three different German car manufacturers on the level of type series. In this context we include further evaluation metrics that help to better estimate the quality of diagnoses (reconfigurations) – for example, the currently applied accuracy metric does not take into account the importance of the different constraints contained in a diagnosis. Furthermore, we will extend the FLEXDIAG algorithm in order to make it applicable in scenarios where knowledge bases are tested [6]. Our goal in this context is to improve the performance of existing automated debugging approaches and to investigate to which extent diagnoses resulting from $m > 1$ are considered as relevant by knowledge engineers. Finally, we will compare FLEXDIAG with local search approaches such as genetic algorithms.

8 Conclusions

Efficient reconfiguration functionalities are needed in various scenarios such as the reconfiguration of production schedules, the reconfiguration of the settings in mobile phone networks, and the reconfiguration of robot context information. We analyzed the FLEXDIAG algorithm with regard to potentials of improving existing direct diagnosis algorithms. When using FLEXDIAG, there is a clear tradeoff between performance of diagnosis calculation and diagnosis quality (measured, for example, in terms of minimality and accuracy).

REFERENCES

[1] R. Bakker, F. Dikker, F. Tempelman, and P. Wogmim, 'Diagnosing and solving over-determined constraint satisfaction problems', in *13th International Joint Conference on Artificial Intelligence*, pp. 276–281, Chambery, France, (1993).

[2] S.K. Card, G.G. Robertson, and J.D. Mackinlay, 'The information visualizer, an information workspace', in *CHI '91 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 181–188, New Orleans, Louisiana, USA, (1991). ACM.

[3] J. DeKleer, 'Using crude probability estimates to guide diagnosis', *AI Journal*, **45**(3), 381–391, (1990).

[4] A. Falkner, A. Felfernig, and A. Haag, 'Recommendation Technologies for Configurable Products', *AI Magazine*, **32**(3), 99–108, (2011).

[5] A. Falkner and H. Schreiner, 'SIEMENS: Configuration and Reconfiguration in Industry', in *Knowledge-based Configuration – From Research to Business Cases*, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, chapter 16, 251–264, Morgan Kaufmann Publishers, (2013).

[6] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner, 'Consistency-based diagnosis of configuration knowledge bases', *Artificial Intelligence*, **152**(2), 213–234, (2004).

[7] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Elsevier/Morgan Kaufmann Publishers, 1st edn., 2014.

[8] A. Felfernig, M. Schubert, G. Friedrich, M. Mandl, M. Mairitsch, and E. Teppan, 'Plausible repairs for inconsistent requirements', in *21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pp. 791–796, Pasadena, CA, USA, (2009).

[9] A. Felfernig, M. Schubert, and C. Zehentner, 'An efficient diagnosis algorithm for inconsistent constraint sets', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM)*, **26**(1), 53–62, (2012).

[10] A. Fijany and F. Vatan, 'New approaches for efficient solution of hitting set problem', in *Winter International Symposium on Information and Communication Technologies*, pp. 1–6, Cancun, Mexico, (2004). Trinity College Dublin.

[11] Gerhard Fleischanderl, Gerhard E. Friedrich, Alois Haselböck, Herwig Schreiner, and Markus Stumptner, 'Configuring large systems using generative constraint satisfaction', *IEEE Intelligent Systems*, **13**(4), 59–68, (1998).

[12] G. Friedrich, A. Ryabokon, A. Falkner, A. Haselböck, G. Schenner, and H. Schreiner, ' (Re)configuration using Answer Set Programming', in *IJCAI 2011 Workshop on Configuration*, pp. 17–24, (2011).

[13] L. Hotz, A. Felfernig, M. Stumptner, A. Ryabokon, C. Bagley, and K. Wolter, 'Configuration Knowledge Representation & Reasoning', in *Knowledge-based Configuration – From Research to Business Cases*, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, chapter 6, 59–96, Morgan Kaufmann Publishers, (2013).

[14] D. Jannach, 'Finding preferred query relaxations in content-based recommenders', in *3rd Intl. IEEE Conference on Intelligent Systems*, pp. 355–360, London, UK, (2006).

[15] Ulrich Junker, 'QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems', in *19th Intl. Conference on Artifical Intelligence (AAAI'04)*, eds., Deborah L. McGuinness and George Ferguson, pp. 167–172. AAAI Press, (2004).

[16] A. Mackworth, 'Consistency in Networks of Relations', *Artificial Intelligence*, **8**(1), 99–118, (1977).

[17] J. Marques-Silva, F. Heras, M. Janota, A. Previti, and A. Belov, 'On computing minimal correction subsets', in *IJCAI 2013*, pp. 615–622, Peking, China, (2013).

[18] I. Nica, F. Wotawa, R. Ochenbauer, C. Schober, H. Hofbauer, and S. Boltek, 'Kapsch: Reconfiguration of Mobile Phone Networks', in *Knowledge-based Configuration – From Research to Business Cases*, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, chapter 19, 287–300, Morgan Kaufmann Publishers, (2013).

[19] R. Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **32**(1), 57–95, (1987).

[20] M. Schubert and A. Felfernig, 'BFX: Diagnosing Conflicting Requirements in Constraint-based Recommendation', *International Journal on Artificial Intelligence Tools*, **20**(2), 297–312, (2011).

[21] I. Shah, 'Direct algorithms for finding minimal unsatisfiable subsets in over-constrained csps', *International Journal on Artificial Intelligence Tools*, **20**(1), 53–91, (2011).

[22] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin, 'Formal methods for the validation of automotive product configuration data', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **17**(1), 75–97, (2003).

[23] G. Steinbauer, M. Mörth, and F. Wotawa, 'Real-Time Diagnosis and Repair of Faults of Robot Control Software', in *RoboCup 2005*, LNAI, pp. 13–23. Springer, (2005).

[24] M. Stumptner, 'An Overview of Knowledge-based Configuration', *AI Communications*, **10**(2), 111–126, (1997).

[25] M. Stumptner and F. Wotawa, 'Reconfiguration using model-based diagnosis', in *10th International Workshop on Principles of Diagnosis (DX-99)*, pp. 266–271, (1999).

[26] J. Tiihonen and A. Anderson, 'VariSales', in *Knowledge-based Configuration – From Research to Business Cases*, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, chapter 26, 377–388, Morgan Kaufmann Publishers, (2013).

[27] J. Tiihonen, W. Mayer, M. Stumptner, and M. Heiskala, 'Configuring Services and Processes', in *Knowledge-based Configuration – From Research to Business Cases*, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, chapter 21, 313–324, Morgan Kaufmann Publishers, (2013).

[28] R. Walter and W. Küchlin, 'ReMax - A MaxSAT aided Product (Re-) Configurator', in *Workshop on Configuration 2014*, pp. 55–66, (2014).

[29] R. Walter, C. Zengler, and W. Küchlin, 'Applications of maxsat in automotive configuration', in *Workshop on Configuration 2013*, pp. 21–28, (2013).

[30] K. Wang, Z. Li, Y. Ai, and Y. Zhang, 'Computing Minimal Diagnosis with Binary Decision Diagrams Algorithm', in *6th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD'2009)*, pp. 145–149, (2009).

[31] F. Wotawa, 'A variant of reiter's hitting-set algorithm', *Information Processing Letters*, **79**(1), 45–51, (2001).

Short paper

Learning Games for Configuration and Diagnosis Tasks

Alexander Felfernig[1] and **Michael Jeran**[1] and **Thorsten Ruprechter**[1] and **Alexander Ziller**[1] and **Stefan Reiterer**[1] and **Martin Stettinger** [1]

Abstract. A goal of many Artificial Intelligence (AI) courses is to teach properties of synthesis and analysis tasks such as *configuration* and *diagnosis*. Configuration is a special case of design activity where the major goal is to identify configurations that satisfy the user requirements and are consistent with the configuration knowledge base. If the requirements are inconsistent with the knowledge base, changes (repairs) for the current requirements have to be identified. In this paper we present games that can, for example, be used within the scope of Artificial Intelligence courses to easier understand configuration and diagnosis concepts. We first present the CONFIGURATIONGAME and then continue with two further games (COLORSHOOTER and EATIT) that support the learning of model-based diagnosis concepts.

1 Introduction

Theoretical problems of combinatorial games have long been studied [8]. For example, Bodlaender [4] analyzes the properties of coloring games were players have to color vertices of a graphs in such a way that never two adjacent vertices have the same color. The player who was last able to color a vertex in a consistent fashion wins the game. Börner et al. [5] analyze complexity properties of different variants of two-person constraint satisfaction [12] games were, for example, two players alternately make moves and the first player tries to find a solution whereas the second player tries to make the constraint satisfaction problem (CSP) inconsistent. Different complexity classes of such games are analyzed which primarily depend on the allowed quantifiers – quantified constraint satisfaction problems (QCSPs) are constraint satisfaction problems were some of the variables are universally quantified [9].

Bayer et al. [2] present an application that models Minesweeper puzzles as a CSP [12]; the game supports players in finding a solution and is primarily used as means to support students in understanding the mechanisms of constraint-based reasoning. In a similar fashion, Simonis [14] shows how to solve Sudoku puzzles on the basis of constraint technologies. In addition to problem solving approaches, the authors also focus on mechanisms for puzzle generation and propose measures for evaluating puzzle complexity. Finally, we want to mention the application of constraint technologies in the context of the generation of crossword puzzles. In crossword puzzle generation [3], a crossword puzzle grid has to be filled with words from a dictionary in such a way that none of the words in the dictionary is included more than once in the grid.

In the line of previous work, we present the CONFIGURATION GAME which is based on conventional CSP representations [12] and was implemented with the goal to support the learning of basic concepts of knowledge-based configuration [6, 15]. Furthermore, we introduce two games which focus on analysis in terms of model-based diagnosis [13]. COLORSHOOTER and EATIT are based on the ideas of model-based diagnosis and were developed to support students in the understanding of the principles of hitting set determination [13]. To the best of our knowledge these are new types of games based on conflict detection [10] and model-based diagnosis [7, 13]. All the presented games are *serious games* [11] with the purpose of teaching AI knowledge and also domain knowledge (EATIT).

The remainder of this paper is organized as follows. In Section 2 we introduce definitions of a configuration and a corresponding diagnosis task. The subsequently presented games are discussed on the basis of these definitions. In Section 3 we introduce the CONFIGURATIONGAME Android app and present the results of a corresponding user study. In Section 4 we introduce the COLORSHOOTER diagnosis game and also present results of a user study. In Section 5 we introduce a new diagnosis game embedded in the domain of healthy eating. In Section 6 we discuss issues for future work. With Section 7 we conclude the paper.

2 Configuration and Diagnosis Task

Knowledge-based Configuration is one of the most successful technologies of Artificial Intelligence [6, 15]. Configurators determine configurations for a given set of user requirements, for example, on the basis of constraint technologies. In terms of a CSP, a configuration task and a corresponding solution can be defined as follows.

Definition 1 (Configuration Task and Solution). A configuration task can be defined as a constraint satisfaction problem (V, D, C) where $V = \{v_1, v_2, ..., v_n\}$ is a set of variables, $D = \cup dom(v_i)$ represents the corresponding domain definitions, and $C = \{c_1, c_2, ..., c_m\}$ is a set of constraints. Additionally, user requirements are represented by a set of constraints $CREQ = \{r_1, r_2, ..., r_k\}$. A solution for a configuration task is an assignment $S = \{inst(v_1), inst(v_2), ..., inst(v_n)\}$ where $inst(v_i) \in dom(v_i)$ which is consistent with the constraints in $C \cup CREQ$.

Example (Configuration Task and Solution). An example of a very simple configuration task (and a corresponding solution S) represented as a constraint satisfaction problem is the following. Such configuration tasks have to be solved by players of the CONFIGURATION GAME (see Section 3). This example represents a simple Map Coloring Problem were variables ($V = \{v_1, v_2, v_3\}$) represent, for example, countries on a map and the constraints ($C = \{c_1, c_2\}$) restrict solutions to colorings were neighborhood countries must be represented by different colors. In our example we assume that the neighborhood countries are $\{v_1, v_2\}$ and $\{v_2, v_3\}$ and the user requirements are $CREQ = \{r_1 : v_1 = 1\}$. A player of the CONFIG-

[1] Graz University of Technology, Institute for Software Technology, Austria, email: {felfernig, jeran, reiterer, stettinger}@ist.tugraz.at {thorsten.ruprechter, alexander.ziller}@student.tugraz.at

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

URATIONGAME has successfully solved a configuration task (found a solution $\underline{S}$) if consistent($S \cup CREQ \cup C$).

- $V = \{v_1, v_2, v_3\}$
- $dom(v_1) = dom(v_2) = dom(v_3) = \{1, 2\}$
- $C = \{c_1 : v_1 \neq v_2,\ c_2 : v_2 \neq v_3\}$
- $CREQ = \{r_1 : v_1 = 1\}$
- $\underline{S} = \{v_1 = 1,\ v_2 = 2,\ v_3 = 1\}$

In configuration scenarios it is often the case that no solution can be found for a given set of user requirements ($CREQ \cup C$ is inconsistent). In this context, users are in the need of additional support in order to be able to identify reasonable changes to the current set of requirements more efficiently. Model-based diagnosis [13] can help to automatically identify minimal sets of requirements that have to be deleted (or adapted) such that a solution can be identified. A diagnosis task related to the identification of faulty requirements can be defined as follows (see Definition 2).

Definition 2 (Diagnosis Task and Diagnosis). A diagnosis task can be defined by a tuple $(C, CREQ)$ where C represents a set of constraints and $CREQ$ represents a set of customer requirements. If the requirements in $CREQ$ are inconsistent with the constraints in C, a diagnosis Δ ($\Delta \subseteq CREQ$) represents a set of requirements such that $CREQ - \Delta \cup C$ is consistent (in this context we assume that the constraints in C are consistent). Δ is minimal if $\neg\exists\Delta' : \Delta' \subset \Delta$.

Example (Diagnosis Task and Diagnosis). An example of a simple diagnosis task and a corresponding diagnosis Δ is the following. Similar diagnosis tasks have to be solved by players of the the COLORSHOOTER and the EATIT game (see Sections 4 and 5). The following example represents a diagnosis task were the set of customer requirements ($CREQ$) is inconsistent with the constraints in C. A player of COLORSHOOTER and EATIT has successfully solved a diagnosis task (found a diagnosis $\underline{\Delta}$) if $CREQ - \Delta \cup C$ is consistent.

- $V = \{v_1, v_2, v_3\}$
- $dom(v_1) = dom(v_2) = dom(v_3) = \{0, 1\}$
- $C = \{c_1 : \neg(v_1 = 1) \vee \neg(v_2 = 1), c_2 : \neg(v_1 = 1) \vee \neg(v_3 = 1), c_3 : \neg(v_2 = 1) \vee \neg(v_3 = 1)\}$
- $CREQ = \{r_1 : v_1 = 1, r_2 : v_2 = 1, r_3 : v_3 = 1\}$
- $\underline{\Delta} = \{r_1, r_2\}$

A wide-spread approach to determine diagnoses for a given diagnosis task is to identify minimal conflict sets [10] in CREQ and to resolve these conflicts on the basis of a hitting set directed acyclic graph (HSDAG) approach [13]. A (minimal) conflict set can be defined as follows (see Definition 3).

Definition 3 (Conflict Set). A set $CS \subseteq CREQ$ is a conflict set if $CS \cup C$ is inconsistent (C is assumed to be consistent). CS is minimal if $\nexists CS'$ with $CS' \subset CS$.

On the basis of a set of identified minimal conflict sets [10] we are able to automatically determine the corresponding minimal diagnoses (see Figure 1). In our example, the minimal conflict sets are $CS_1 : \{r_1 : v_1 = 1, r_2 : v_2 = 1\}, CS_2 : \{r_2 : v_2 = 1, r_3 : v_3 = 1\}$, and $CS_3 : \{r_1 : v_1 = 1, r_3 : v_3 = 1\}$. The corresponding minimal diagnoses are $\Delta_1 : \{r_1, r_2\}, \Delta_2 : \{r_1, r_3\}$, and $\Delta_3 : \{r_2, r_3\}$. Exactly this example pattern is implemented in the diagnosis games presented in Section 4 and Section 5.

3 CONFIGURATIONGAME

User Interface. With this game (see Figure 2), one should be able to gain first insights into the basic concepts of knowledge-based con-

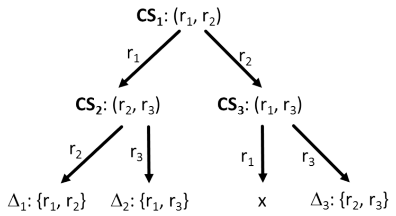


Figure 1. Example hitting set directed acyclic graph derived from the conflict sets CS_1, CS_2, and CS_3.

figuration. Constraints of the underlying CSP are depicted in the upper left corner. Constraints represent incompatible combinations of values, i.e., combinations that must not be positioned on adjacent vertices of the grid depicted in Figure 2. This way, tasks such as the map coloring problem [4] can be defined as a simple configuration problem (similar examples can also be found in [6]).

Each individual task to be solved by a player can be interpreted as a configuration task (V,D,C) (see Definition 1). In the setting shown in Figure 2, $V = \{v_1, v_2, ..., v_{14}\}$ represents a set of *14 interconnected hexagons* (in the center of the user interface). Furthermore, it is assumed that each variable has the same domain (in Figure 3, dom(v_i)={1,2,3,4}) and possible variable instantiations are represented by the values (hexagons) in the lower right corner. Constraints $C = \{c_1, c_2, ..., c_{16}\}$ represent incompatible colorings of adjacent vertices, for example, $c_1 : \neg(v_1 = 1 \wedge v_2 = 1) \wedge \neg(v_1 = 2 \wedge v_2 = 2) \wedge \neg(v_1 = 3 \wedge v_2 = 3) \wedge \neg(v_1 = 4 \wedge v_2 = 4) \wedge \neg(v_1 = 4 \wedge v_2 = 3) \wedge \neg(v_1 = 3 \wedge v_2 = 4) \wedge \neg(v_1 = 1 \wedge v_2 = 2) \wedge \neg(v_1 = 2 \wedge v_2 = 1)$. Incompatibilities are defined by red lines between individual values (left upper corner of Figure 2). A self-referring red line expresses an incompatibility on the same value, i.e., two adjacent vertices must not have to the same value. A proprietary constraint solver is used to generate individual tasks with increasing complexity in terms of the grid size (vertices and arcs) and the number of possible values and also to check proposed solutions for consistency.

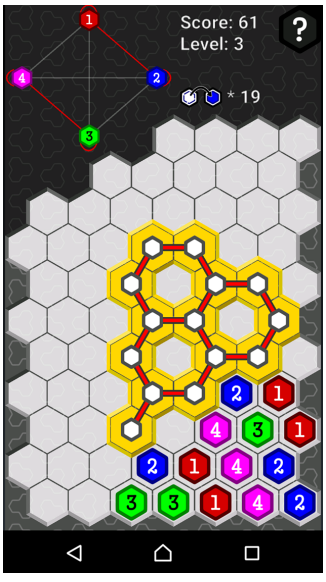


Figure 2. CONFIGURATIONGAME user interface.

The task of a player is to move values (hexagons) from the bottom to corresponding (empty) hexagons depicted in the middle of Figure 2. A player has found a solution if the grid instantiation S is

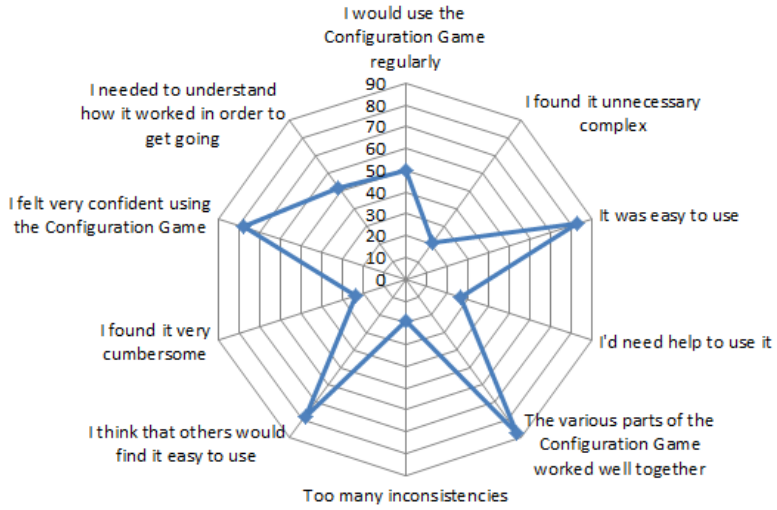


Figure 3. Results of a usability analysis of the CONFIGURATIONGAME on the basis of the system usability scale (SUS) [1].

consistent with the constraints in C.[2] A screenshot of an intermediate state of the CONFIGURATIONGAME is shown in Figure 4. The CONFIGURATIONGAME allows to define constraints that go beyond typical patterns of map coloring problems [4] since different types of incompatible adjacent vertices can be defined (in contrast to the map coloring problem where only incompatibilities regarding the same value (color) are defined). Instances of the configuration game are generated automatically.

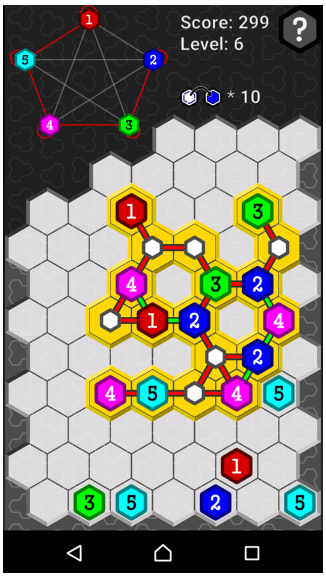


Figure 4. CONFIGURATIONGAME user interface (after the completion of some configuration steps).

Empirical Study. N=28 subjects of a usability study evaluated the CONFIGURATIONGAME. A first prototype of the game was made available to the subjects in the *Google Play Store*. The questionnaire was based on the system usability scale (SUS) [1] and thus focused on analyzing usability aspects of the system under investigation. The system was considered as easy to understand and well integrated. Results of the study are summarized in Figure 3.

4 COLORSHOOTER

User Interface. The COLORSHOOTER game (see Figure 5) focuses on providing first insights into the concepts of model-based diagnosis [13]. The game is available online in the *Apple App Store* (as an iOS application). The columns of the game represent minimal conflict sets (see Definition 3) – related diagnoses (see Definition 2) are represented by minimal color[3] sets such that at least one color from each row is included. The game consists of twenty different levels and inside each level of 30 different individual COLORSHOOTER tasks. Individual tasks are pre-generated in an automated fashion where the #colums, #rows, # of different colors, and diagnosis cardinality are major impact factors for determining the complexity of one COLORSHOOTER instance. Correct solutions (diagnoses) are pre-generated on the basis of a HSDAG [13].

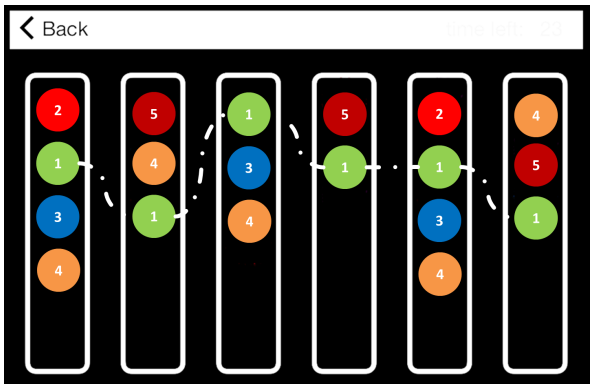


Figure 5. Example of a COLORSHOOTER diagnosis task.

Empirical Study. After two lecture units on model-based diagnosis [13], we investigated in which way COLORSHOOTER type games can actively support a better understanding of the principles

[2] In the CONFIGURATIONGAME we assume that $CREQ = \{\}$.

[3] For readability purposes we annotated the colored circles with numbers.

of model-based diagnosis. N=60 subjects (students) participated in a user study where each participant was assigned to one of three different settings (see Table 1).

Participants of the first setting used the COLORSHOOTER game directly before solving two diagnosis tasks. Participants of the second setting interacted with COLORSHOOTER one day before completing the two diagnosis tasks. Finally, participants of the third setting never interacted with COLORSHOOTER but only solved the two diagnosis tasks. The two diagnosis tasks where designed in such a way that a participant had to figure out all minimal diagnoses for each of two predefined inconsistent constraint sets (C_1 and C_2). C_1 included 4 constraints, 4 variables of domain size [1..3], and 3 related diagnoses. C_2 included 5 constraints, 4 variables of domain size [1..3] and 6 related diagnoses. Preliminary results in terms of the number of successfully completed diagnosis tasks are depicted in Table 1. In the case of the more complex constraint set C_2 we can observe a performance difference between users who applied COLORSHOOTER and those who did not.

setting	all minimal diagnoses found (C_1)	all minimal diagnoses found (C_2)
played directly before	33%	20%
played one day before	20%	20%
did not play before	23%	10%

Table 1. User study with three different settings: subjects played directly before solving two diagnosis tasks, subjects played one day before, and subjects did not use COLORSHOOTER.

5 EATIT

EATIT (see Figure 6) is an application currently under development, i.e., no related user studies have been conducted up to now. The major ideas of the game are the following: (1) similar to COLORSHOOTER, students should be able to more easily understand the concepts of model-based diagnosis. (2) there is a serious game [11] line of learning which is directly related to the underlying application domain: users of the system should learn about, for example, which vitamins are contained in which food. In EATIT, "conflicts" are represented by food items assigned to the same shelf (each food item contains the vitamin represented by the shelf) and diagnoses represent minimal sets of food items that are needed to cover all vitamins.

6 Future Work

In the CONFIGURATIONGAME our major goal for future work is to extend the expressiveness of constraints that can be defined for configuration tasks. A higher degree of expressiveness will allow the inclusion of further tasks such as scheduling and resource balancing. Furthermore, EATIT will be extended with functionalities that help to include user preferences and menu quality. In the current version of EATIT such aspects are not taken into account. In our future research we will also analyze in more detail which specific game types better help to increase understandability. Furthermore, we will analyze to which extent the games can be exploited to develop better configurator user interfaces and interaction schemes.

7 Conclusions

The overall goal of the (serious) games presented in this paper is to help to better understand the concepts of configuration and model-based diagnosis. Results of empirical studies are promising in the sense that the apps are applicable and can have a positive impact on the learning performance. Two of the presented games are already available: COLORSHOOTER in the Apple App Store and the CONFIGURATIONGAME in the Google Play Store.

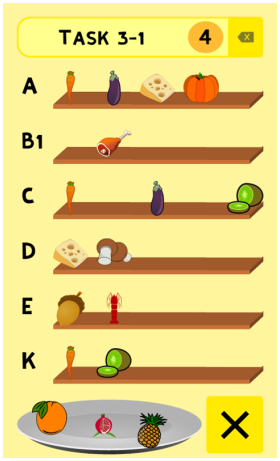


Figure 6. Screenshot of EATIT. Each shelf represents food that contains a specific vitamin (e.g., A, B1, ...). A solution (diagnosis) is found if the selected food on the plate covers all vitamins.

REFERENCES

[1] A. Bangor, P. Kortum, and J. Miller, 'An Empirical Evaluation of the System Usability Scale (SUS)', *International Journal of Human-Computer Interaction*, **24**(6), 574–594, (2008).

[2] K. Bayer, J. Snyder, and B. Choueiry, 'An Interactive Constraint-Based Approach to Minesweeper', in *AAAI 2006*, pp. 1933–1934, (2006).

[3] A. Beacham, X. Chen, J. Sillito, and P. vanBeek, 'Constraint Programming Lessons Learned from Crossword Puzzles', in *AI 2001*, LNAI, pp. 78–87. Springer, (2001).

[4] H. Bodlaender, 'On the Complexity of Some Coloring Games', *LNCS*, **484**, 30–40, (1991).

[5] F. Börner, A. Bulatow, H. Chen, P. Jeavons, and A. Krokhin, 'The Complexity of Constraint Satisfaction Games and QCSP', *Information and Computation*, **207**(9), 923–944, (2009).

[6] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Elsevier/Morgan Kaufmann Publishers, 1st edn., 2014.

[7] A. Felfernig, M. Schubert, and C. Zehentner, 'An efficient diagnosis algorithm for inconsistent constraint sets', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM)*, **26**(1), 53–62, (2012).

[8] A. Fraenkel, 'Complexity, Appeal and Challenges of Combinatorial Games', *Theoretical Computer Science*, **313**(3), 393–415, (2004).

[9] I. Gent, P. Nightingale, and K. Stergiou, 'A Solver for Quantified Constraint Satisfaction Problems', in *IJCAI 2005*, pp. 138–142, (2005).

[10] Ulrich Junker, 'QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems', in *19th Intl. Conference on Artifical Intelligence (AAAI'04)*, eds., Deborah L. McGuinness and George Ferguson, pp. 167–172. AAAI Press, (2004).

[11] H. Kelly, K. Howell, E. Glinert, L. Holding, C. Swain, A. Burrowbridge, and M. Roper, 'How to build Serious Games', *Communications of the ACM*, **50**(7), 44–49, (2007).

[12] A. Mackworth, 'Consistency in Networks of Relations', *Artificial Intelligence*, **8**(1), 99–118, (1977).

[13] R. Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **32**(1), 57–95, (1987).

[14] H. Simonis, ' Sudoku as a constraint problem', in *CP Workshop on Modeling and Reformulating Constraint Satisfaction Problems*, pp. 13–27, (2005).

[15] M. Stumptner, 'An Overview of Knowledge-based Configuration', *AI Communications*, **10**(2), 111–126, (1997).

Support for the social dimension of shopping through Web Based Sales Configurators

Chiara Grosso[1] and **Cipriano Forza**[1] and **Alessio Trentin**[1]

Abstract. Mass Customizers (MCs) often sell personalized products through Web Based Sales Configurators (WBSCs). Recently, a number of them have connected their WBSCs with Social Software Applications (SSAs). This is not surprising since SSAs provide an interactive and socially rich shopping experience, which makes shopping on WBSCs more similar to retail shopping experiences. Even though interaction with customers is a distinctive characteristic of MCs, research on the use of SSAs by MCs is very limited. The present paper reports the preliminary results of research that aims to (i) describe how existing WBSCs connect with SSAs, (ii) understand which stages of the configuration-shopping process are supported by the different WBSCs-SSAs connections, and (iii) explain how the different WBSCs-SSAs connections support fulfilment of the needs for social feedback and for social involvement that WBSC users perceive during their shopping experiences.

1 INTRODUCTION

Mass Customizers (MCs) often sell their products on the Web through Web Based Sales Configurators (WBSCs) [1]. This selling approach has proven to be beneficial to both MCs [2, 3] and their customers [4,5,6]. Selling through the Web is challenging not only because it is a new way of selling for many companies, but also because Web technologies are witnessing a number of innovations. One of these innovations refers to Social Software Applications (SSAs), which are new Web features that enhance connections between individuals and groups of Internet users. The social software phenomenon has grown in the past few years as millions of people have joined online communities and started using online social platforms [7, 8]. Globally, SSAs reach more than 1.5 billion members [7].

SSAs are increasingly being adopted by companies to improve their sales and their connections with actual and potential customers [9]. In this respect, MCs are not an exception and they are starting to include SSAs in their configuration sites as an aid for WBSC users during the shopping process.

Relatively few studies, however, have addressed this phenomenon. Previous research has highlighted the growing trend of social-media usage aimed to share information about configured products [10, 11] and has addressed the integration of customers in social networks [12]. In addition, prior research has indicated the combination of configuration toolkits with social networks as a promising field of inquiry [13], consistent with initial findings on how peer input improves the configuration process [14].

To our knowledge, however, the social benefits that WBSC users could derive from the connection of WBSCs with SSAs have not been investigated yet. This is a gap that research on WBSCs needs to address, for at least two reasons. First, socials benefits are recognized as enhancing users' quality perception of the configuration process [10]. Second, social needs are likely to be stronger while shopping for personalized products using WBSCs because, besides the perceived risk of online shopping [15], a customer can incur in some circumstances that enhance his/her need for being in contact with others, for interacting with them and for receiving feedback from trustworthy sources [16]. Among those circumstances, there may be the lack of experience in configuring a particular kind of object or in browsing into a WBSC, the lack of creativity or the lack of choice orientation about the product configuration one has created [14].

To narrow this research gap, we need to consider, first of all, the social needs that customers could experience while shopping for a personalized product on a WBSC and how an SSA would help satisfy those needs, if it were connected with the WBSC.

[1]Università degli Studi di Padova, Dipartimento di Tecnica e Gestione dei sistemi ind.li, Vicenza, Italy. E-mail addresses: chiara.grosso@unipd.it. alessio.trentin@unipd.it (A.Trentin), cipriano.forza@unipd.it (C.Forza).

2 BACKGROUND

2.1 The social dimension of the shopping process

2.1.1 The shopping process: the EBM model

The shopping experience involves a variety of consumer personal, situational and social variables that impacts on the consumer decision-making process. Engel, Blackwell and Miniard [17] developed a model (named from their initials as EBM model) to understand the variables implicated in the entire consumer decision-making process. One of the advantages of the EBM model is its applicability to a wide range of situations to explain and predict consumer behaviour. The EBM model's core is based on five decision-process stages that the customer goes through during his/her shopping experience.

The EBM's first stage is named: *problem recognition* and it refers to customer identification of a need or problem. The consumer experiences an unbalanced condition between the actual and the desired state of need [17].

Once the consumer has identified the need he/she wants to satisfy, the consumer goes through a second stage named: *information search*. The consumer starts gathering information on possible solutions. The information research involves both internal and external consumer sources. Internal sources are for example: the consumer's previous shopping experience, memories etc. External sources include interpersonal sources such as reference groups and relevant others who have direct or indirect influence on the individual's decision process (e.g. family, friends, colleagues, other consumers) [18]. The information search stage provides the consumer with the basis to evaluate the various alternatives to satisfy his/her need.

The EBM's third stage of the decision-making process is named: *the evaluation* of *alternatives.* At this point, the consumer evaluates the information previously gathered, develops a set of criteria to compare the alternatives and, finally, defines his/her own preferences.

The EBM's forth stage, namely: *purchase* stage, refers to the consumer's decision to buy a good or a service. The purchase stage also includes the consumer's decision about where and how to buy.

The EBM's fifth stage refers to post-purchase consumer decisions. Post-purchase is a crucial stage in understanding the entire consumer decision-making process and predicts how a consumer will act in the future. A consumer's good experience will

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

motivate him/her: to repeat the shopping experience, to provide positive feedback to others and positively influence other customer's intention to shop. On the contrary, post-purchase dissatisfaction will produce a negative impact on the consumer's personal and social attitudes [19].

A consumer shops not only for utilitarian reasons but shopping is frequently based on personal and social motives. Among personal motives, there are: individual self-gratification, learning about fashion trends, individual status, emulation of others, homologation to a trend and diversion from daily routine. Consumer's social motives are driven by: the affiliation to a group, the pleasure of being involved, the pleasure of sharing enjoyable moments with friends, the interest in sharing support between colleagues, the emulation of others (e.g. other consumers; reference groups as friends or colleagues; relevant others as people who can influence an individual's actions) [18].

During the shopping process, a consumer experiences the need to interact with other persons. In particular, he/she perceives two needs, namely: the need for social involvement, and the need for social feedback [20, 21]. To satisfy the above-mentioned needs during the shopping process the customer has to perceive the company of other persons that count for him/she (e.g. members from his/her reference groups, sales representatives, or other customers) and the possibility to interact with them (e.g. transmit information, receive feedback, receive help, or hints for his/her decision process, support in understanding if the selected alternative is the one that best fits his/her purchase purpose).

2.1.2 The need for social feedback while shopping

The consumer need for social feedback is based on the individual's need for social interaction [20]. Social feedback refers to the consumer preference of receiving feedback from trustworthy external sources that can orientate his/her decision process (e.g. family, friends, reference groups, company representatives, etc.) [19].

Consumer theorists have long recognized the influence that relevant others have on consumer's decision making [8, 17, 19, 21, 22]. The role of relevant others depends on the informational social influence. The informational social influence entails accepting feedback information from others to facilitate problem solving or cope with some aspect of the environment. It drives consumers to learn about some product/service by seeking information from peers, relevant others, reference groups [23]. Informational social influence is especially important when a consumer is faced with time constraints, possesses limited knowledge, perceives high risk in the action, or simply does not want take decision alone.

Feedback from others during a shopping experience enables a recommendation system based on real-time interaction (e.g. communication exchanges) that influences and guides the consumer in his/her decision process. Online users like to be in control, which means that they expect and want feedback to be available when needed. More specifically, they want that help to be at their request and to respond to their individual needs. Real-time feedback provided by others is especially relevant when the consumer has no clear ideas about his/her own preferences, thus about the product that better satisfies his/her needs. Research on online consumer behaviour highlighted the positive impact of informational social influence on the consumer's decision to shop online [22, 23].

Thanks to the Web evolution, consumers can easily communicate and exchange online information about their shopping experience. In particular, they can easily access to a large volumes of information on a product or service provided directly from those who have a recent experience with it [22, 23].

2.1.3 The need for social involvement while shopping

Offline shopping experiences encompass a wide range of social interactions between persons. Shopping online tends to be more impersonal, anonymous and automated than offline shopping [24]. In online consumer markets, there is still a reduced presence of social factors that supports the consumer's decision-making process. In particular, an online shopping experience typically lacks the sociability that characterizes the purchase stage. Previous research has highlighted how online stores tend to display their products with no social appeal, only providing a functional product description, attribute-based, and unemotional [25].

The need for social involvement refers to the consumer need for affiliation as a preference to act in accordance with persons that count for him/her and to be in contact with them during his/her shopping experience [26].

During online shopping, a consumer perceives of being in contact with others depending on the extent to which a communication medium (e.g. commercial web site) provides social presence to its users [23]. Social presence is an inherent quality of a medium to support its users in feeling others as being psychologically present by enabling various forms of interaction similar to the face-to face interactions [23, 25, 27]. According to the social presence theory, a medium with high social presence conveys a feeling of human contact, sociability, and sensitivity [23, 25, 27]. High social presence fulfils the individual need for social involvement.

Online consumers' perceptions of social presence have been shown to positively influence consumer intention to purchase from a commercial website [19].

2.2 Social software applications

According to Lawley [28], SSAs refer to computing tools to support, extend, or derive added value from social activity. In recent research, SSAs are defined as software applications that enable people to connect, collaborate, create online networks and manage contents in a social and bottom-up fashion. Bottom-up communication is a specific characteristic enabled by SSAs and differs from top-down companies communication or off-line media [29].

The distinctive characteristics of SSAs derive from their social purposes: (a) to intensify and extend online and offline social interaction, (b) to adapt to their users instead of the contrary, (c) to connect users to a network approach, and (d) to connect computing tools around users [29].

SSAs embrace a large number of tools for online interaction services including (but not limited to): weblogs, instant messaging, music and photo sharing, mailing lists and message boards, and online social network (SN) tools, internet forums, wikis, social guides, social bookmarking, social citations, social libraries, virtual worlds [30]. Online communication services supported by SSAs are named social media (SM). SM are network-based platforms that allow their users to interact online for different purposes (e.g. fun, professional support, share content, gaming, etc.) [30].

Hereafter we provide a brief description of the SSAs that will be recalled in the following sections of the present article.

Weblog (Blogs) are online platforms for a content management system. Blogs are website that allow the user to communicate online while maintaining control on contents and communications. Blogs are based on RSS (Really Simple Syndication or Rich Site Summary), a family of web feed formats used to publish frequently updated content [31]. A Blog supports both features of the reading and the writing of content. Blog's contents are visible also to external Internet users. The owner of the Blog can set the blog's features by allowing only the reading of the blog's content or by enabling Internet users to both read and write. In both cases, the

Blog's owner is the one that can decide about the management of the blog (e.g. delete contents, apply constraints to the blog's usage). Blog contents are mostly organized by thematic categories and are presented in a chronological order. The division by thematic categories avoids confusion and allows its users to focus their search for information on the topic of interest. Blogs are largely used by companies as communication tools for the construction and maintenance of relationships between companies and their customers [32].

Media sharing platforms are platforms that offer online services such as: loading, storing, sharing and browsing of different media contents and formats (e.g. photos, videos, slides, files, doc). Depending on the format of the shared content, the service platform assumes a corresponding name (e.g. photo-sharing platform, video-sharing platform, etc.). Media sharing platforms allow the co-viewing of content (multiple users can simultaneously see the same content). Media sharing also allows a collaborative filtering content (models and filters that select and propose content based on visions made by the user). Media sharing platforms support aggregation sharing and tagging of content on other external SM platforms.

Social networks (SN) are web-based services that allow users to build a public or semi-public personal profile within a bounded system, to articulate a list of other users with whom to share content. Depending on the SN, users contacts are named as friends, circles etc. The SN user can see and navigate through the list of his/her contacts and through other SN users' profiles within a defined system. SN services enable users to publish, share, modify, and endorse contents on his profile or other users' profiles following specific criteria. SN services support mechanisms for building relationships between users [30].

Discussion Forum (DF) or online community are online groups of users who interact supported by specific technologies. Community members can interact and share content depending on the enabling technologies of the different platforms (video, chat, mailing, comments and videos). Online communities define themselves by the topics and the purposes for which they are established (e.g. education, business) and for the kind of Software environment, that supports them (e.g. list servers, bulletin boards, forums, discussion list, various combinations of these). The DF includes also communities of practice, e-learning platforms, discussion groups, online brand communities, consumer communities [29]. A company DF allows interaction between both the company and its consumer (B2C) and between the DF's users (C2C), but only company representatives can manage the DF.

Internet Relay Chat (IRC) is an instant messaging protocol that allows real-time communication between two users, or the contemporary dialogue of entire groups of users (chat rooms). It provides a real-time communication via Internet. Once a chat has been initiated, involved users can enter text by typing on the keyboard and the entered text will appear on the other user's monitor. A large number of social networks and online services offer a chat feature.

E-mail systems are some of the most popular Internet-based applications, due to e-mail efficiency, low cost, and compatibility of diversified types of information. The Simple Mail Transfer Protocol (SMTP) is a transportation protocol used to transfer e-mail messages over the Internet. The e-mail service provides an a-synchronous messaging service (i.e. a- synchronous refers to a time shift between the message writing time and the message reading time of the recipient). E-mail enables its users to send and receive messages between each other both from inside and outside the local area network. Nowadays, SN platforms provide an e-mail service for its users (e.g. Facebook and LinkedIn).

2.3 SSA support to the social dimension of the online shopping process

In an online shopping process, the social presence provided by a commercial website enables the consumers in perceiving the proximity of relevant others (i.e. people that count for the consumer and able to influence his intention to purchase from a commercial website) [27].

Drawing on the consumer socialization framework, Lueg and Finney [21] argued that peer communication online could influence consumers so strongly that they convert each other into Internet shoppers. Given the risk perceived by consumers in online shopping [15], consumers will ask the opinion of their friends or online reference groups before make an online purchase decision. Thus, the online consumer will experience the need to be in contact with others so as to be supported by persons that count for him (i.e. relevant others).

In order to satisfy the need to be in contact with others (i.e. social interactions), SSAs support various forms of interactions between its users (e.g. share a comment, files, archives, text messages, chat, vote, endorse). Interactions are enabled both between person known and unknown [21; 28]. Interactions supported by SSAs are similar to face-to-face interaction (e.g. video call, chat), for this reason, SSAs provide a high level of social presence able to satisfy the need for social involvement of their users.

SSAs intensify and extend online and offline users' social interactions thus it provides its users different sources of feedback information mostly transmitted in real-time. Thanks to a real-time feedback, the SSAs user collects information provided by others and feels orientated in his/her choice process [21].

To improve the shopping website with SSAs makes the website as a highly interactive communication medium and therefore a medium able to provide a socially rich shopping experience similar to retail shopping.

2.4 Sales configurators

One peculiar shopping process is the online shopping of personalised products. This process happens more and more through WBSCs. Consistent with previous research, we define sales configurators as knowledge-based software applications that support a potential customer, or a sales-person interacting with the customer, in completely and correctly specifying a product solution within a company's product offer [5]. Franke et al. [14] described the MC self-customizing process through a WBSC as a problem solving process that includes the development of an initial idea, the generation of a preliminary design (interim design solution) and the final design evaluation [33].

WBSCs guide customers towards the purchase of a configured product that best fits their needs. The satisfaction of needs different from the functional need is important for the configuration process. The utilitarian value is not the only value that leads customers to purchase. Many other values and satisfaction of needs different from functional such as creative achievement and hedonic benefits [5, 34] or uniqueness and self-expressiveness [4] are important for the configuration process leading to a purchase or simply to leave a positive impression for user returns or at least to talk about it positively. The configuration experience has been recognized as important in concluding the shopping process with a purchase. Research has shown that up to 50% of the additional willingness to pay for configured products can be explained by positive perception of the co-design process itself [6].

3 RESEARCH AIMS AND METHOD

In order to achieve the first research aim, that is to describe how existing WBSCs connect with SSAs, we browsed into 250 existent configurators to first identify the presence of connections between WBSCs and SSAs and secondly to analyze the different modalities adopted to connect the WBSCs with SSAs. The WBSCs analyzed were largely drawn from the configurator database available on www.configurator-database.com, excluding those that were not in English, Italian or Spanish. To the considered dataset, other configurators available only in Italian have been added. A wide variety of products were considered, such as: jewellery, T-shirt, cars, bikes, notebooks, shoes, bags, etc.

To achieve the second research aim, that is to understand which stages of the configuration-shopping process are supported by the different WBSCs-SSAs connections, we analyzed the user's decision process during shopping via WBSC by using the EBM mode [l7]. We based the analysis on an analytical reasoning, however our reasoning was grounded on a number of configuration experiences we performed in different WBSCs for every identified WBSC-SSA connection modality. First, we identify which stages of the user's decision process are supported by each modality and after we describe how the connection of WBSC with SSA supports the self-configuration process. We adopt the technical terminology provided by Franke et al. [14] to describe the configuration process experienced by the user. When we refer to a partial product configuration, we mean a product configuration that has not been completed. Intermediate product configuration refers to a preliminary product configuration that has not yet been selected as the final one. Final product configuration refers to the product configuration that the user has chosen as the end result after his various trial configurations. Any reference to the company website refers only to a company website where the WBSC is enclosed.

To achieve the third research aim, that is to explain how the different WBSCs–SSAs connections support fulfilment of the needs for social feedback and for social involvement, we evaluate through analytical reasoning how social involvement and social feedback are provided by each modality. In doing so, we use the following three categories of people with whom the WBSC user is enabled to interact by a given modality: reference groups, refers to people the user already knows and is in contact with also via SM platforms; peers, refers to unknown people of equal standing, such as other customers; relevant others, refers to any people who can influence the user's decision process, which includes the two previous categories but also company representatives.

4 HOW WBSCs ARE CONNECTED WITH SSAs

We identify eight different modalities in which SSAs are connected to WBSCs. In the following sub-sections, we briefly describe each modality (M) and how it supports the configuration process, as well as the extent to which it supports fulfilment of the needs for Social Involvement (SI) and Social Feedback (SF).

4.1 M1: Icons in the company website to connect WBSC users to company SM profile(s)

SM icons are placed in the company website, external to the WBSC. The WBSC's user can connect to the corresponding company's SM profiles by clicking on the various icons. If the company has different SM profiles, the user can reach different company profiles by clicking on the corresponding icons. Example of WBSC: MY M&M'.

M1 supports the consumer in two stages of his decision process namely: (a) the search for information and (b) the evaluation of alternatives. M1 supports the configuration process by allowing the user to browse into the company's SM profile. Thus, the user can collect hints that can help him in the development of an initial configuration idea.

SI: ***indirectly provided***. M1 does not support the user in interacting with others, but it brings the user into the SM platforms where he can browse into the company's SM profile and interact with other SM users that share the same interest in the company. Thus, M1 satisfies the need for SI by providing an indirect social presence for the user.

SF: ***indirectly provided.*** M1 does not support the user in transmitting information from the WBSC to his SM profiles or vice versa. Thus, M1 does not directly provide SF linked with the configuration process. The user can gather hints or information by himself browsing into the company SM profile.

4.2 M2: Icons in the WBSC to connect WBSC users to their SM profile(s)

4.2.1. Variant M2.1

The WBSC contains one or more SM icons that bring the user to his own corresponding SM profile to automatically publish the link to the entry page of the configurator. The user can also publish additional information by placing it in his social SM while he is sharing the WBSC's link (e.g. information about his configuration experience, advices etc.). In the M2.1 modality as in the M1 the SSAs are not directly accessible during the configuration process. Example of WBSC: Jonathan Adler.

M2.1 supports the consumer in four stages of his shopping decision process, namely: the search for information (a), the evaluation of alternatives (b), the purchase (c), post-purchase (d). M2.1 supports the configuration process by triggering an information exchange between the user and his reference groups that helps the user during the development of an initial idea, the evaluation of an intermediate design and finally the configuration evaluation. In fact, the user can ask for advice about his configuration by sharing the WBSC's link and he can gather feedback on his SM profile (e.g. the user can ask if someone from his reference groups already knows the WBSC and how to better develop his configuration process).

SI: ***low***. M2.1 does not enable the users to interact with others in the configuration environment but it brings the user to the SM platforms where he can interact with his reference groups. Thus, M2.1 satisfies user need for SI by providing a low social presence to the user during his configuration process.

SF: ***low.*** M2.1 supports the user only in sharing the link to the entry page of the WBSC, the user is not supported in automatically share additional information about his configuration process. Moreover, the WBSC's user can gather feedback information only in the SM platforms outside of the configuration environment. Thus, M2.1 provides a low level of SF to the user.

4.2.2. Variant M2.2

One or more SM icons are placed in the WBSC. Each icon brings the user to his corresponding SM profile to automatically share a complete configuration. The WBSC's user can also add personal comments along with the complete configuration while he is still in the configuration environment (e.g. add details about his configuration experience on that specific WBSC). All shared contents will be visible to his SM reference groups since the SSAs of each SM platform enable this feature. Example of WBSC: Ethreads.

M2.2 supports the consumer at the fourth stage of his decision process: the purchase (c). M2.2 supports the evaluation of an intermediate configuration by enabling sharing of a complete

configuration from the WBSC to the SM environment. Thus, the user can interact (e.g. ask for advice) with his reference groups to gather advice and improve the intermediate configuration. Likewise, M2.2 supports the design evaluation by supporting the user in sharing a final configuration not yet purchased.

SI: ***medium***. M2.2 enables the user to interact with his reference groups in the SM platforms. Members of the user's reference groups are interested in supporting him and highly trustworthy. However, interactions between the user and his reference groups occur only outside the configuration environment. Thus, the M2.2 satisfies the SI need by providing a medium level of social presence to the user.

SF: ***medium***. M2.2 supports the user in transmitting his configuration from the WBSC to the SM platforms where he can ask and receive feedback information from his reference groups. The sharing of information is enabled only one-way: from the WBSC site to the SM platforms, not vice versa. The user can share his configuration outside the WBSC but he can collect feedback information only in the SM platforms. Thus, M2.2 provides a medium level of SF.

4.2.3. Variant M2.3

SM icons are inserted into the WBSC, each icon brings the user to his correspondent SM profile where he can automatically publish a partial configuration while the configuration is on-going. The user can also add personal comments along with the partial configuration that he is going to publish in his SM profiles. Example of WBSC: Nike.

M2.3 supports the consumer during his decision process at the purchase stage (c). The M2.3 supports the configuration process during the intermediate design evaluation since it enables the user to share his partial configuration with his references groups. Thus, the user can consult his reference groups about the configuration options previously chosen, and the options he is going to choose.

SI: ***medium-high***. M2.3 supports the user in interacting with his reference groups on the SM platforms while he is configuring a product. The user feels confident that if needed he can contact and be supported by his reference groups during the configuration process. By doing so, M2.3 recreates a shopping situation similar to retails shopping where a customer can shop in company of relevant others (i.e. friends, family, etc.). Any interaction between the user and his reference groups takes place only outside the WBSC. Thus, M2.3 satisfies the need for SI by providing a medium-high level of social presence to the user.

SF: ***medium***. M2.3 supports the transmission of information from the WBSC to the SM platforms but not vice versa. The user can gather feedback information about his configuration only in the SM platforms. Thus, M2.3 provides a medium level of SF.

4.3 M3: Direct browse/upload from the WBSC of files shared in the personal SM profile(s)

The WBSC embeds one or more SM icons, by clicking on them, each icon brings the user to his SM folders to browse and directly upload an item in the configured product (e.g. photo, image, drawing). Example of WBSC: Personalwine.

M3 supports the consumer during his decision process on both stages of the evaluation of alternatives (b) and the purchase (c). M3 supports the development of an initial idea and the evaluation of an intermediate configuration since it provides additional choice options from external sources (e.g. user's personal photos on SM). M3 supports the design evaluation by enabling browsing into the user's personal archives previously shared with his references groups on the SM platforms.

SI: ***not provided***. M3 does not enable the user in perceiving the company of others. No interaction is enabled while the user browses and uploads items from the SM folders to the WBSC. M3 does not provide social presence to the user thus it does not satisfy the SI need.

SF: ***indirectly provided***. M3 does not support the user in gathering feedback information, since the user cannot exchange information with others. However, M3 enables the user to collect a kind of indirect feedback hints (e.g. other SM users positive or negative comment on a certain photo of his SM folders, the number of likes etc.). In other words, by choosing items from his SM folders the user can select those items that have been previously positively evaluated by his reference groups on the SM platform and avoid the ones that received a negative response.

4.4 M4: Simplified WBSC embedded in company SN profile(s)

A simplified WBSC configurator is embedded into a company's SN profile (e.g. Facebook). The simplified WBSC is inserted as an application of the company's SN profile and it is visible as an ad hoc page. Since the configuration choices are very limited the simplified WBSC works as a demo-configurator by providing a very constrained configuration process. A complete configuration process is only possible on the full WBSC website. Often the link to the full WBSC is available on the company's SN profile. Example of WBSC: Vauxhall Facebook profile.

M4 supports the consumer in two stages of the decision process: the search for information (a) and (b) the evaluation of alternatives. M4 supports the configuration process during the development of an initial idea by enabling a user to experience the configuration in a highly interactive environment like a SN platform. In the SN platforms, a SN user is not specifically looking for a WBSC. The opportunity to face an existent simplified WBSC allows him to deal with a customization process, to start a sample of configuration or simply to be informed on how a WBSC works. M4 supports the evaluation of an interim configuration by enabling the user in sharing his intermediate configuration with his reference group on that specific SN.

SI: ***high.*** M4 allows the user to feels the support of his reference group and the company without leaving the configuration environment. Thus, M4 satisfies the need for SI by providing a high level of social presence.

SF: ***high***. M4 supports users in transmitting information and gathering feedback from reference groups and company representatives directly where the configuration occurs. The configuration process happens in an environment where the user feels confident because he already knows how to reach information easily from his reference groups. Moreover, the user can exchange information by using different communication features enabled by SSAs (e.g. publish or add comments, endorse, like a content, chat, etc.). M4 delivers a high SF to the user.

4.5 M5: Weblog (Blog) in the company website to connect WBSC users to relevant others

We observed two different types of Blogs, type 1 refers to a blog that supports only the reading features for external users. Type 2 refers to a blog type that supports both the features of reading and writing for the external users of the Blog. We named type 1: Blog-Diary and type 2: Blog-Post.

4.5.1 Variant M5.1

The company website provides a link to connect website users to the Blog Diary. The Blog-Diary mainly presents contents that report information about brand events, sponsorships, and competitions. Thus, Blog-Diary's contents inform the user not only about functional topics (e.g. product functionalities or features) but also ludic news (e.g. curiosity, unedited news about the company). M5.1 brings the user outside the configuration environment. Example: Renesim.

M5.1 supports the customer during the search for information (a) and the evaluation of alternatives (b). M5.1 supports the configuration process by providing hints that can inspire and guide the user in the development of his initial configuration idea (e.g. information about the company's new product, new fashion trends).

SI: ***not provided***. M5.1 does not enable the user to perceive proximity either of the company representatives or with other Blog-diary users.

SF: ***not provided***. M5.1 provides a one-way communication flow from the company to the user. The M5.1 does not provide SF since communication exchanges are not supported.

4.5.2 Variant M5.2

The company website provides a link to connect WBSC users with the company' Blog. The Blog-Post reports additional information not present in the WBSC environment but provided in the Blog-Post by the company and by other blog users. The Blog-Post's contents are mainly centred on utilitarian information (e.g. product's functionalities and features) that can help the user to gather hints, find answer to his questions or simply share his experiences with both company representatives and other Blog users. M5.2 brings the user outside the configuration environment. Example: Puget Systems.

M5.2 supports the customer decision process from the search for information (a) to the post purchase stage (d). M5.2 supports the stage of post purchase (d) because enables the user in share his feedback in the Blog-Post once he ends his shopping process (e.g. to give advice, details about the WBSC etc.). The user will share a positive or negative feedback based on the perceived quality of his shopping experience. M5.2 supports the development of the user's initial idea since it provides additional information useful for self-configuration. M5.2 supports the evaluation of an intermediate configuration and the design evaluation by enabling communication exchanges between the user and both the company representatives and peers (e.g. people not from his reference groups like other customers, brand followers, blog users). Thus, the additional information reported in the Blog-Post by the company and/or peer can help the user to improve his intermediate configuration or convince him to directly purchase his final configuration.

SI: ***medium-low***. M5.2 supports the user in interacting only with peers and company representatives, however, the user has to move outside the WBSC in order to perceive the proximity of others. The Blog-Post environment provides a site where the user does not know in advance with whom he will interact or if he can reach someone when he needs support. The chances for the WBSC user to interact with someone depend on the other Blog Post users availability to participate in the blog's activities as well as interest in the WBSC user request. Thus, M5.2 satisfies the need for SI by providing a medium-low social presence.

SF: ***low***. M5.2 allows the user to gather feedback information but the chance of receiving feedback will depend on the Blog-Posts users' availability in answering, as well as on their knowledge about the WBSC user's request. Thus, there are no guarantees that the WBSC user will receive feedback when he requires it or indeed that he will receive a feedback coherent with his request. For those reasons, M5.2 provides a low SF.

4.6 M6: Company Discussion Forum to connect WBSC users to relevant others

The company website provides a link to bring WBSC users to a company' Discussion Forum (DF). The link is placed outside the WBSC thus SSAs are not directly accessible during the configuration process. Example: Dell.

M6 supports the customer during the decision process from the search for information (a) to the post purchase stage (d). M6 supports the development of an initial configuration idea by providing hints and information that can guide the user since the contents are provided i both by the company and peers (e.g. other customers, blog users). M6 supports the evaluation of an intermediate configuration and the final design evaluation by allowing two-ways interactions between the user and the company or other peers. Thus, the user can feel supported to achieve advices about how to improve his configuration or decide to buy the final one.

SI: ***medium***. M6 enables the user to interact with unknown peers as well as with company representatives but not with his reference group. The user has to move outside the WBSC to perceive the proximity of others. The M6 brings the WBSC user to a site where even if he does not know in advance with whom he will interact, he can be confident that the other DF users are highly motivated to support each other as members of the same community. M6 satisfies the need for SI by providing a medium social presence.

SF: ***medium***. M6 supports users in receiving feedback information from trustworthy sources since most DF users are experienced consumers, professionals, or experts in a specific topic discussed in the forum. However, there are no guarantees that the WBSC' user will gather the feedback he requires when he needs it (the interaction between DF users depends on their availability and interest to interact). Only if the DF provides a chat as additional communication tool between its users, the interactions can occur in real-time exactly when a user asks for feedback. Thus, M6 provides a medium SF to the WBSC' user.

4.7 M7: E-mail service to connect WBSC users to relevant others

4.7.1 Variant M7.1

The WBSC provides an e-mail service directly accessible from the WBSC at the end of the configuration process. M7 supports the sending of a final configuration to one or more members of the user's reference groups. Example of WBSC: Ecremary.

M7.1 supports the customer in the purchase stage (c) of his decision process. M7.1 supports the configuration process at the final design evaluation by enabling the sharing of a final configuration with one or more members of the user's reference groups. M7.1 enables to send from the WBSC to outside but no vice versa. Thus, the user has the advantage to collect information, hints, or ask for advice to someone he already knows, but outside the configuration environment.

SI: ***low***. M7.1 supports users by sending his configuration by e-mail to someone from his reference groups. The user is confident in addressing his requests of interaction to someone already known and interested in help him. M7.1 satisfies the need for SI by providing a low social presence since e-mail is an online communication tool with a low level of social presence [35].

SF: ***low***. M7.1 enables the user to receive feedback from members of his reference groups, but there are no guarantees that feedback will be available when the user demands it. The communication exchanges via e-mail are a-synchronous, rarely the e-mail exchanges take place exactly when the user asks for feedback (i.e. on demand). Thus, SF provided by M7.1 is low.

4.7.2. Variant M7.2

The company website provides the e-mail service as customer's service. E-mail's exchanges are enabled only between company representatives and users. Example of WBSC: JL Hufford (almost all the analysed WBSCs provide an e-mail contact).

M7.2 supports the customer in each stages of his decision process. M7.2 supports the configuration process by providing an additional communication tool to the WBSC user, but it placed outside of the WBSC environment.

SI: ***low***. M7.2 provides the users the feeling of being in contact with the company by sending a request by e-mail. More and more often companies adopt e-mail automatic reply systems thus customers mainly perceive the company as a distant and impersonal interaction partner. M7.2 satisfies the need for SI by providing a low social presence to the user [35].

SF: ***low***. M7.2 supports the user in gathering feedback only from the company. The user has the advantage of receive feedback from a high trustworthy source but there are no guarantees that the user will receive feedback on demand. M7.2 satisfies the need for SI by providing a low SF to the user [35]

4.8 M8: Instant message services to connect WBSC users to company customer service

The company website provides a real-time messaging service (Chat) for customer service. M8 can be placed either outside or inside the WBSC, in both cases it enables real-time communication only between the company representative and the users. The user cannot interact in real-time with his reference groups or with other peers. Example of WBSC: CustomInk.

M8 supports the customer in real-time at each stages of his decision process from the search for information (a) to the post purchase stage (d). M8 supports the post-purchase stage by providing the user with a real-time communication channel to contact the supplier while he is waiting for delivery of his product. M8 supports the configuration process since it provides real-time professional support as well as additional information not presented in the WBSC product space.

SI: ***medium high***. M8 enables the user to interact in real-time only with company representatives. The user is confident that he will be in contact with professionals at each stage of the configuration-shopping process but he cannot be supported by his relevant others. M8 satisfies the SI need providing a medium-high social presence.

SF: ***high***. The M8 supports the user in gathering real-time feedback from company representatives whenever he asks for it during at each stage of his configuration-shopping process. Thus, the user is confident that feedback will properly fit with his request and it provided by a highly trustworthy source. However, the user cannot collect feedback from his reference groups. Thus, M8 provides a medium-high level of SF.

5 CONCLUSIONS

The present study investigated the connections of WBSCs with SSAs. More specifically, we (i) identified and described eight different connection modalities (Table 1 columns 1 & 2), (ii) explained which stages of the configuration-shopping process are supported by the different WBSCs-SSAs connections (Table 1 columns 3-6) and (iii) explained how the different connection modalities support the fulfilment of the needs for SF and for SI that WBSC users perceive during their shopping experiences (Table 1 columns 7-8).

Table 1 – *Synthesis of the research results*

Connection Modality	Variant	EBM stages				SF	SI
		a	b	c	d		
		1-2		3			
		CONF. process					
M1 - Icons in the company website to connect WBSC users to company SM profile(s)	-					Ip	Ip
M2 - Icons on the WBSC to connect WBSC users to their SM profile(s)	M2.1					Low	Low
	M2.2					Med	Med
	M2.3					Med-High	Med
M3 - Direct browse/upload from the WBSC of files shared in the personal SM profile	-					Np	Ip
M4 - Simplified WBSC embedded in company SN profile	-					High	High
M5 - Weblog (Blog) on the company website to connect WBSC users to relevant others	M5.1					Np	Np
	M5.2					Med-low	Med-low
M6 - Company Discussion Forum to connect WBSC users to relevant others	-					Med	Med
M7 – e-mail service to connect WBSC users to relevant others	M7.1					Low	Low
	M7.2					Low	Low
M8 - Instant message services to connect WBSC users to company customer service	-					Med-High	High

EBM Stages. *a: information search; b: alternative evaluation; c: purchase; d: post-purchase.*
CONF. process. *1: initial idea development; 2: intermediate evaluation; 3: configuration evaluation.* ***SF/ SI.*** *Ip: support for the fulfilment of SF/SI is indirectly provided; Np: support is not provided; Low: support is low; Med-low: support is medium-low; etc.*

While shopping, a WBSC user perceives greater SF in WBSC-SSA connections modalities: M4 and M8, which (a) deliver feedback to the user directly in the configuration environment, (b) make the user confident to achieve feedback when he asks for it (c) provide feedback from highly trustworthy source (e.g. from a member of his/her reference groups or from company representatives).

Likewise, a WBSC user perceives greater SI in WBSC-SSA connections modalities: M4 and M8, which (a) provide social support at each stage of the configuration-shopping process, (b) enable social support directly in the configuration environment, and (c) allow the user to interact in real-time while the configuration occurs.

Our findings, if confirmed by future research, provide the following guidelines to WBSC designers:

- Connect a WBSC with SSAs in a way that supports two-way exchange of information (i.e. from the WBSC to SSAs and vice versa).
- Insert a connection with SSAs in the WBSC, so as to provide support and feedback directly in the configuration environment. To prevent the user from leaving the WBSC, both support and feedback have to properly fit the user request.
- Insert SSAs that enhance the WBSC user's perception of being real-time supported by her/his reference groups.
- Insert SSAs that enable the user to choose from whom to be supported without leaving the configuration environment (e.g. from his/her reference groups, company representatives, peers, other users).

- Insert SSAs that provide support and feedback to the WBSC user exactly when he/she requires (on demand).

The present research, once completed, will allow new research to be started with the aim to: (a) describe the current frequency of adoption of the various WBSC-SSA connection modalities, eventually analysing possible dependences on the type of product; (b) empirically test the differences in SF and SI provided by the different WBSC-SSA connection modalities; (c) evaluate whether the different WBSC-SSA connections impact on benefits arising from self-personalized products and from experiences of self personalization, such as utilitarian, hedonic and creative achievement benefits.

ACKNOWLEDGEMENTS

We acknowledge the financial support of the University of Padua, Project ID CPDA129273.

REFERENCES

[1] F.S. Fogliatto, G.J.C. da Silveira, and D. Borenstein, "The mass customization decade: an updated review of the literature". *International Journal of Production Economics*, **138**(1), 14–25, (2012).

[2] M. Heiskala, J. Tiihonen, K.S. Paloheimo, and T. Soininen, "Mass customization with configurable products and configurators: a review of benefits and challenges", in: T. Blecker, G. Friedrich (Eds.), Mass Customization Information Systems in Business, IGI Global, London, UK, 1–32, (2007).

[3] C. Forza, and F. Salvador, "Application support to product variety management", *International Journal of Production Research,* **46**(3), 817–836, (2008).

[4] C. Grosso, A Trentin, and C. Forza, "Towards an understanding of how the capabilities deployed by a Web-based sales configurator can increase the benefits of possessing a mass-customized product". *In 16th International Configuration Workshop,* **21**, 81-88, (2014).

[5] A. Trentin, E. Perin, and C. Forza, "Increasing the consumer-perceived benefits of a mass-customization experience through sales-configurator capabilities". *Computers in Industry*, **65**(4), 693-705, (2014).

[6] N. Franke, M. Schreier and U. Kaiser, "The 'I designed it myself' effect in mass customization". *Management Science,* **56**(1), 125–140, (2010).

[7] M. Chui, J. Manyika, J.Bughin, R.Dobbs, H. Sarrazin, G.Sands, and M. Westergen, "The social economy: unlocking value and productivity through social technologies". *McKinsey Global Institute Report*, (2014).

[8] A.M. Kaplan and M. Haenlein, "Users of the world, unite! The challenges and opportunities of Social Media". *Business horizons*, **53**(1), 59-68 (2010).

[9] Z. Huang, and M. Benyoucef, "From e-commerce to social commerce: A close look at design features". *Electronic Commerce Research and Applications*, **12**(4), 246-259, (2013).

[10] F.T. Piller, and P. Blazek, "Core capabilities of sustainable mass customization". *Knowledgebased Configuration–From Research to Business Cases. Morgan Kaufmann Publishers, Waltham, MA*, 107-120, 2014.

[11] P. Blazek, M. Kolb, M. Partl, and C. Streichsbier, "The usage of social media applications in product configurators". *International Journal of Industrial Engineering and Management* (IJIEM), **3**(4), 179-183, (2012).

[12] F.T. Piller, A. Vossen, and C. Ihl, "From social media to social product development: the impact of social media on co-creation of innovation". (December 21, 2011). Die Unternehmung, **65**(1), (2012).

[13] N. Franke, and C. Hader, "Mass or Only "Niche Customization"? Why We Should Interpret Configuration Toolkits as Learning Instruments". *Journal of Product Innovation Management*, **31**(6), 1214-1234, (2014).

[14] N. Franke, P. Keinz, and M. Schreier, "Complementing Mass Customization Toolkits with User Communities: How Peer Input Improves Customer Self-Design". *Journal of Product Innovation Management*, **25**(6), 546-559, (2008).

[15] G. Pires, J. Stanton, and A. Eckford. "Influences on the perceived risk of purchasing online". *Journal of Consumer Behaviour*, **4**(2), 118–131, (2004).

[16] C.N. Ziegler, and J. Golbeck, "Investigating interactions of trust and interest similarity", *Decision Support Systems*, **43**(2), 460-475, (2007).

[17] J.F. Engel, R.D. Blackwell, and P.W. Miniard. Consumer behaviour, 8th ed. Fort Worth: Dryden Press, 1995.

[18] W.O. Bearden, and M.J. Etzel, "Reference group influence on product and brand purchase decisions". *Journal of Consumer Research*, 183-194, (1982).

[19] C.M.K. Cheung, and M.K.O. Lee, "Understanding consumer trust in internet shopping: a multidisciplinary approach". *Journal of the American Society for Information Science and Technology*, **57**(4), 479–492, (2006).

[20] T.S.H. Teo, and Y.D. Yeong, "Assessing the consumer decision process in the digital marketplace." *Omega The International Journal of Management Science*, **31**(5), 349–363, (2003).

[21] J.E. Lueg, and R.Z. Finney "Interpersonal Communication in the Consumer Socialization Process: Scale Development and Validation". *Journalof Marketing Theory and Practice*, **15**(1), 25–39, (2007).

[22] P. Butler, and J. Peppard, "Consumer purchasing on the Internet: Processes and prospects". *European Management Journal*, **16**(5), 600–610, (1998).

[23] M.K. Lee, N. Shi, N., C.M. Cheung, K.H. Lim, and C.L. Sia, "Consumer's decision to shop online: The moderating role of positive informational social influence". *Information & Management*, **48**(6), 185–191, (2011).

[24] X. Wang, C. Yu, and Y. Wei, "Social media peer communication and impacts on purchase intentions: A consumer socialization framework". *Journal of Interactive Marketing*, **26**(4), 198–208, (2012).

[25] N. Kumar, and I. Benbasat, "Para-social presence and communication capabilities of a web site: a theoretical perspective". *E-service Journal*, **1**(3), 5–24, (2002).

[26] J. Bloemer, G. Odekerken-Schröder, and L. Kestens,"The impact of need for social affiliation and consumer relationship proneness on behavioural intentions: An empirical study in a hairdresser's context". *Journal of Retailing and Consumer Services*, **10**(4), 231-240, (2003).

[27] K. Hassanein, and M. Head, "Manipulating perceived social presence through the web interface and its impact on attitude towards online shopping". *International Journal of Human-Computer Studies,* **65**(8), 689–708, (2007).

[28] E. Lawley (2004). Blog research issues. Retrieved, from http://www.corante.com/many/archives/2004/06/24/blog_research_issues.php

[29] W.A. Warr, "Social Software: fun and games, or business tools?". *Journal of Information Science*, **34**(4), 591–604, (2008).

[30] D.M. Boyd and N.B. Ellison, "Social network sites: Definition, history, and scholarship". *Engineering Management Review, IEEE,* **38**(3), 16-31, (2010).

[31] J. Schmidt, "Blogging practices: An analytical framework. *Journal of Computer Mediated Communication*, **12**(4), 1409–1427, (2007).

[32] T. Kelleher, and B.M. Miller, "Organizational blogs and the human voice: Relational strategies and relational outcomes". *Journal of Computer Mediated Communication*, **11**(2), 395–414, (2006).

[33] A. Newell, and H. Simon *Human problem solving*. Prentice-Hall, NJ, 1972.

[34] A. Merle, J.L. Chandon, E. Roux, and F. Alizon, "Perceived value of the mass-customized product and mass customization experience for individual consumers". *Production & Operations Management,* **19**(5), 503–514, (2010).

[35] G. Tang, J. Pei, and W.S. Luk, "Email mining: tasks, common techniques, and tools". *Knowledge and Information Systems*, **41**(1), 1–31, (2014).

A Goal-Question-Metrics Model for Configuration Knowledge Bases

Florian Reinfrank, Gerald Ninaus, Bernhard Peischl, Franz Wotawa
Institute for Software Technology
Graz University of Technology
Inffeldgasse 16b/II, 8010 Graz, Austria
{firstname.lastname}@ist.tugraz.at

Abstract. Configuration knowledge bases are a well-established technology for describing configurable products like cars, computers, and financial services. Such knowledge bases are characterized by sets of constraints, variables, and domains. Lot of research has been done for testing knowledge bases, finding conflicts, and recommending repair actions.

In contrast, less work has been done in the area of measuring the quality of configuration knowledge bases. Such quality measurements can help knowledge engineers to improve the maintainability, understandability, and functionality of knowledge bases. Based on a literature review we first give an overview of the state-of-the-art in knowledge base metrics. We will extend the current research by using the goal-question-metrics (GQM) approach of the software engineering discipline to find gaps for the characterization of knowledge bases. We will also identify further metrics to complete the model. The results of this paper help knowledge engineers to reduce the effort to develop and maintain configuration knowledge bases.

1 Introduction

Products like cars, computers, and software product lines can be customized according to consumers' preferences. For supporting users to get valid configurations and to support the manufacturing department in companies to get an overview of their production lines, models of their products are necessary [29]. Knowledge bases describe a part of the real world (e.g., the set of valid product configurations for bikes). The implementation of a knowledge base is typically done cooperatively between domain experts and knowledge engineers [5, 42]. Configuration knowledge bases can be represented, for example, as constraint satisfaction problems (CSP [39]).

Configuration knowledge bases (CKB) represent the complete set of valid product configurations. Adding, changing, and removing constraints of such knowledge bases is necessary, because the set of valid product configurations changes over time. Humans in general and knowledge engineers in particular tend to keep efforts related to knowledge acquisition and maintenance as low as possible. Due to cognitive limitations [11, 21, 37] anomalies such as inconsistencies, redundancies, and well-formedness violations are in CKBs. The CKB maintenance task gets even more complicated, if a couple of knowledge engineers has to develop and maintain the knowledge base.

In this paper we describe how to measure the quality of configuration knowledge bases.[1] Therefore, we use the goal question metric approach (GQM). The first step in the GQM approach is to define possible *goals* for the knowledge base, like understandability, maintainability, and functionality. The achievement of the goals will be measured by answers for a set of *questions*. These answers will be aggregated and weighted. After the aggregation and the weightings of answers, the quality of the current version of the configuration knowledge base can be measured in terms of fulfilling the goals. Third, questions will be answered by sets of *metrics*. In this paper we define goals, questions, and metrics and show, how we can measure them. These results will help knowledge engineers in focusing on relevant aspects of the configuration knowledge base to improve the management of a configuration knowledge base, e.g., the efforts for maintainability, understandability, and functionality of a knowledge base.

The remainder of this paper is organized as follows: Section 2 introduces a working example for this work, anomalies and their definitions. Section 3 gives an overview of the state of the art for goals, questions, and metrics for configuration knowledge bases and other research areas. In Section 4 we discuss the GQM model and compare it with the results of other research areas. Finally, Section 5 summarizes this paper and gives an overview of further research in this area.

2 Configuration Knowledge Base

A configuration knowledge base can be defined as a triple (V, D, C) with a set of variables V where each variable $v \in V$ has a domain $dom(v) \in D$. For example, a bike configuration knowledge base could contain the variables $V = \{Reflector, Pedal, Framecolor\}$ where each variable has a domain $D = \{dom(Reflector) = \{yes, no\}, dom(Pedal) = \{Standard,\ Clip\}, dom(Framecolor) = \{Green,\ Red\}\}$. The assignments to a variable (e.g., $Pedal = Clip$) are defined as constraints $c \in C$ [39]. While such assignments

[1] Please consider, that the approach presented in this paper can also be used in other models like knowledge-based recommendation [9] and feature models [4].

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

are defined by users, other constraints are defined by domain experts and are restricting the number of valid combinations of variable assignments. For example, a constraint $c_i = \neg(Pedal = Standard \wedge Framecolor = Red)$ does not allow a consistent configuration with $Pedal = Standard$ and $Framecolor = Red$. We call user assignments *customer requirements* $c \in C_R$ and constraints which are describing the relationship between customer requirements and product variables knowledge constraints $c \in C_{KB}$. The sets C_R and C_{KB} describe C, such that $C_R \cup C_{KB} = C$ [18]. Customers try to find a valid configuration for the configuration knowledge base which means, that they assign values to variables $\{C_R\}$ and the set of assignments is not in conflict with C_{KB} (see Definitions 1 and 2).

Definition1: A **consistent configuration** is defined as a set of customer constraints C_R. This set of constraints is not in conflict with C_{KB}, such that, there exists one possibility to assign a value to each variable, formally defined as $C_{KB} \cup C_R$ is consistent.

Definition2: A configuration is a **consistent complete configuration**, iff the knowledge base is a consistent configuration (see Definition 1) and each variable has an assignment, such that $\nexists_{v_i \in V} v_i = \emptyset$.

Figure 1 gives an example about a configuration knowledge base for a bike domain. The graphic is based on the notation of feature models [6] where each variable is represented as a feature and each feature has the same domain ($dom(v_i) = \{true, false\}$). The notation of constraints of feature models is described in Figure 1.

The following configuration knowledge base (CKB) reflects a constraint-based (CSP-based [39]) representation of the configuration model depicted in Figure 1. Each of the constraints in Figure 1 is part of the set C_{KB}.

$$
\begin{aligned}
&V = \{\\
&\quad Bike, Reflector, Pedal, Framecolor, Green, Red\\
&\quad Standard, Clip\\
&\}\\
&D = \{\\
&\quad dom(Bike) = dom(Reflector) =\\
&\quad dom(Pedal) = dom(Framecolor) =\\
&\quad dom(Green) = dom(Red) = dom(Standard) =\\
&\quad dom(Clip) = \{true, false\}\\
&\}\\
&C_{KB} = \{\\
&\quad c_0 : Bike = true;\\
&\quad c_1 : Bike = true \leftrightarrow Reflector = true;\\
&\quad c_2 : Bike = true \leftrightarrow Pedal = true;\\
&\quad c_3 : Bike = true \leftrightarrow Framecolor = true;\\
&\quad c_4 : Reflector = true \rightarrow Pedal = true;\\
&\quad c_5 : \neg(Pedal = true \wedge Framecolor = true);\\
&\quad c_6 : Framecolor = true \leftrightarrow (Green = true \vee\\
&\qquad Red = true);\\
&\quad c_7 : (Standard = true \leftrightarrow (Clip = false \wedge\\
&\qquad Pedal = true)) \wedge (Clip = true \leftrightarrow (Standard\\
&\qquad = false \wedge Pedal = true));\\
&\}
\end{aligned}
$$

In this example there are different types of anomalies. *Anomalies are patterns in data that do not conform to a well defined notion of normal behavior* [10]. Anomalies can be conflicts, redundancies and forms of well-formedness violations. For a detailed description of anomalies we refer the reader to [14, 16, 20, 28].

In our example, we can see that the set of constraints $\{c_0, c_2, c_3, c_5\}$ is in conflict because there does not exist a valid assignment for these constraints such that all constraints are fulfilled. We call such scenarios conflicts [23] and they are defined in the Definitions 3 and 4.

Definition 3: a **conflict** is given, iff there exists a set of constraints CS which can not result in a valid configuration (see Definitions 1 and 2), such that $CS \subseteq C$ is inconsistent.

Definition 4: a **minimal conflict** is given, if the constraint set CS leads to a conflict (see Definition 3), and there does not exist a subset of CS with the same property of being a conflict, such that, $\nexists CS' \subset CS$ is inconsistent.

The example of this paper contains one minimal conflict: $CS_1 = \{c_0, c_2, c_3, c_5\}$ because this constraint set leads to no valid configuration of the configuration knowledge base and it is not possible to remove a constraint from CS_1 without loosing the property of beeing a minimal conflict (see Definitions 3 and 4). A minimal conflict can be calculated with the QUICKXPLAIN algorithm [23]. If our model has more than one conflict and we want to have all minimal conflicts we need to calculate an acyclic graph (HSDAG) which is defined in [35].

Solutions for such conflicts are called diagnoses [35, 17]. By removing a set of constraints Δ from the configuration knowledge base, we receive at least one valid assignment for each variable of a configuration knowledge base, formally described in the Definitions 5 and 6.

Definition 5: A set of constraints Δ is called **diagnosis**, iff the removal of Δ from the knowledge base C leads to a valid configuration (see Definition 1 and 2), such that $C \setminus \Delta$ is consistent.

Definition 6: A set of constraints Δ is called **minimal diagnosis**, iff it is a diagnoses (see Definition 5) and it is not possible to remove a constraint from Δ without loosing the property of being a diagnoses, such that $C \setminus \Delta' \subset \Delta$ is consistent.

In our example the constraint sets $\Delta_1 = \{c_0\}$, $\Delta_2 = \{c_2\}$, $\Delta_3 = \{c_3\}$ and $\Delta_4 = \{c_5\}$ are minimal diagnosis because removing one of these constraint sets would lead to a consistent configuration knowledge base.

While conflicts and diagnosis are well discussed, little attention has been done to redundancies in configuration knowledge bases. Piette [28] and Felfernig et al. [20] focused on the problem of redundant constraint sets in knowledge bases and defined the term redundancy (see Definition 7).

Definition 7: A set of constraints R is **redundant**, if the removal of R leads to the same semantics of C, such that, $C \setminus R \models R$.

In our example, we have two different sets of redundant constraints: $R_1 = \{c_2\}$ and $R_2 = \{c_4\}$. A redundancy does not have an impact on the semantics of a knowledge base but probably leads to a higher effort for maintenance tasks of knowledge bases and decreases the performance of the configuration knowledge base. We can calculate such sets with the SEQUENTIAL [28] or CoreDiag [20] algorithm. Both algorithms use the negation of C ($\overline{C} = \neg c_0 \vee \neg c_1 \vee ... \vee \neg c_n\}$) for calculating redundant constraints. For checking the semantics of $C \setminus R$ the algorithms check, if $\overline{C \setminus R} \cup C$ is inconsistent. An

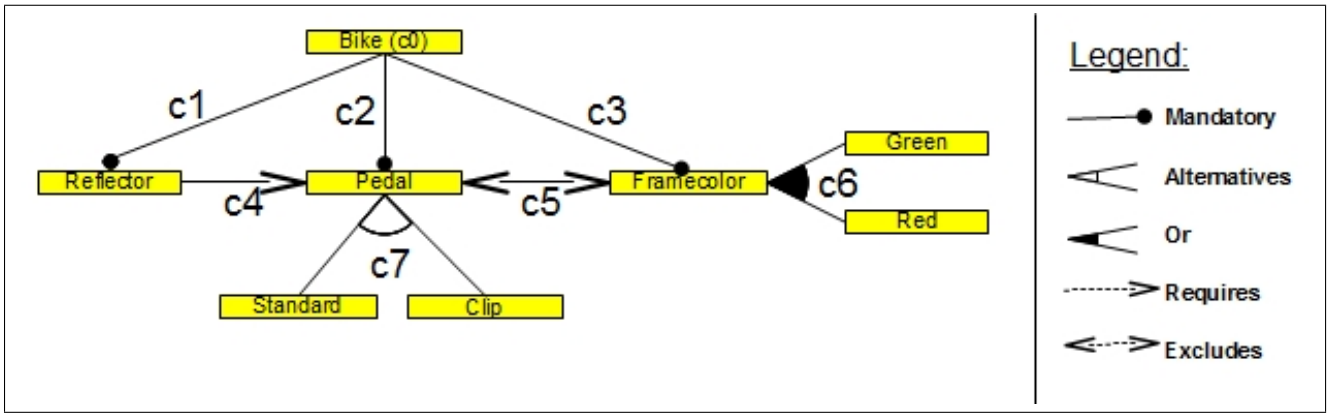


Figure 1. Feature Model of the bike configuration example.

inconsistency means, that the set R is redundant. For getting all different sets of redundant constraints we can also use HSDAG [35].

The third type of anomalies are **well-formedness violations**. Such violations do not have an impact on the consistency of a knowledge base (see Definitions 1 and 2). While well-formedness is well discussed in the research area of feature models (see e.g., [6]), less research has focused on well-formedness violations in configuration knowledge bases. How we use well-formedness violations for calculating metrics for configuration knowledge bases will be described in the next Section.

Anomalies and other aspects of a configuration knowledge base have impacts on the maintainability, understandability, and functionality of the knowledge base. In the following, we give an overview of goals, questions, and metrics for configuration knowledge bases which helps knowledge engineers to get an understanding of the quality of the knowledge base.

While conflicts, diagnoses, and redundancies focus on constraints, well-formedness violations identify anomalies based on variables and domain elements. We now introduce well-formedness violations in configuration knowledge bases.

The first well-formedness violation focuses on dead domain elements. A dead domain element is an element which can never be assigned to its variable in a consistent instance (see Definition 1). Definition 8 introduces a formal description of dead elements.

Definition 8:: A domain element $val_1 \in dom(v_i)$ is **dead** iff it is never part of a consistent instance, s.t. $C_{KB} \cup \{v_i = val_1;\}$ is inconsistent.

On the other hand, we can have domain elements which have to be assigned to each consistent instance. We denote such domain elements *full mandatory* (see Definition 9).

Definition 9:: A domain element $val_1 \in dom(v_i)$ is **full mandatory**, iff there is no consistent (complete or incomplete) instance where the variable v_i does not have the assignment val_1, s.t. $C_{KB} \cup \{v_i \neq val_1\}$ is inconsistent.

The configuration knowledge base can never be consistent, if $Bike = false$ or $Reflector = false$ or $Pedal = false$. For that domain elements, we can say that these domain elements are full mandatory and the other domain element $false$ is a dead domain element. Note that all domain elements of a domain are false optional if one domain element is full mandatory.

Another well-formedness violation is called unnecessary refinement. Such an unnecessary refinement consists of two variables. If the first variable has an assignment, it is possible to predict the assignment of the other variable. A formal definition is given in Definition 10.

Definition 10 (**Unnecessary refinement**): A configuration knowledge base contains a variable pair v_i, v_j. For each domain element val_1 of variable v_i, we can say that variable v_j always has the same assignment $v_j = val_2$, s.t. $\forall_{val_1 \in dom(v_i)} \exists_{val_2 \in dom(v_j)} v_i = val_1 \wedge v_j \neq val_2$ is inconsistent.

In our example the variable pair $Bike$ and $Reflector$ is unnecessary refined because whenever $Bike = true$ the $Reflector = true$, and $Bike = false$ respectively $Reflector = false$ leads to an inconsistency. If such a violation occurs, we can recommend the knowledge engineer to remove the variable $Reflector$ and rename the variable $Bike$ with $BikeWithReflectors$.

3 A GQM model for Configuration Knowledge Bases

For the overview of the metrics for configuration knowledge bases we use the GQM method. For each goal we use a set of questions to define the achievement of each goal. It is also necessary that the goals, questions, and metrics can be calculated automatically, and with explanations [2, 36].

In this section we first give an overview of the possible goals for configuration knowledge bases. Thereafter we give an overview of the questions in (configuration) knowledge bases. Finally we operationalize the questions by listing metrics for configuration knowledge bases.

3.1 Goals for Configuration Knowledge Bases

Nabil et al. [27] define five basic goals for knowledge bases. *Reusability* means, that the knowledge base can be reused in another application area. The *flexibility* defines the possibility to change the semantics of the configuration knowledge base. *Understandability* defines the possibility that knowledge engineers have correct assumptions. *Functionality* describes the applicability of the knowledge base. For example, if the model does not describe the real product assortment, the knowledge

base has no functionality. *Extendability* describes the possibility to extend the knowledge base. In our example (see Figure 1) we can extend the model by adding the bike type ($V' = V \cup \{Type, MountainBike, CityBike\}; D' = D \cup \{dom(Type) = dom(MountainBike) = dom(CityBike) = \{true, false\}\};$).

Lethbridge [25] identifies three goals for knowledge bases. First, it is necessary that knowledge engineers can *monitor* their work. Therefore it is necessary to offer baselines for its continous improvement. Another aspect is the support for knowledge engineers when they *maintain* a knowledge base. Finally, Lethbridge also focuses on the *understandability* of knowledge bases.

From the perspective of software product lines there are three goals: the *analyzability* focuses on the capability of a system to be diagnosed for anomalies. *Changeability* is the possibility and ease of change in a model when modifications are necessary. *Understandability* also means the likelihood that knowledge engineers and designers understand the knowledge base [2].

To sum up, we define the following goals for configuration knowledge bases:

- A configuration knowledge base must be **maintainable**, such that it is easy to change the semantics of the knowledge base in a desired manner [2, 27].
- A configuration knowledge base must be **understandable**, such that the effort for a maintainability task for a knowledge engineer can be evaluated [2, 25, 27].
- A configuration knowledge base must be **functional**, such that it represents a part of the real world (e.g. a bike configuration knowledge base) [27].

3.2 Questions for Configuration Knowledge Bases

After defining the goals for configuration knowledge bases we describe the questions relating to one or more goals. The concordance with the application will be defined by the *completeness*. It suggests the applicability of the current state of the knowledge base for representing the application area. For instance, in our example (see Figure 1) a *Bike* can have a green and a red frame color. If it is possible to combine those colors, the coverage is high. If a frame can only have either a red or a green frame color, the model does not represent the application area and the coverage will be low.

Q1: Is the configuration knowledge base complete?

Anomalies are a well researched area in the context of configuration knowledge bases [30]. The term 'anomalies' is used synonymously for errors and subsumes the terms inconsistencies, redundancies, and well-formedness violations. Errors can have an impact on each of the goals, since it has negative impacts on the reusability, maintainability, and understandability. It can also have a negative impact on the functionality, if there exists an inconsistency in the knowledge base.

Q2: Does the configuration knowledge base contain anomalies?

The *performance* describes the time which is required to calculate characteristics of a knowledge base. These characteristics are e.g., error checking, calculating consistent configurations, and generating user recommendations. This performance mainly influences the functionality of a system (latency) and the reusability.

Q3: Does the configuration knowledge base have an admissible performance?

If it is necessary to develop and maintain the knowledge base a high *modifiability* will help to reduce the effort for the update operation. The modifiability has a positive impact on the reusability and maintainability of a knowledge base. For example, when updating redundant constraints in a knowledge base (e.g. constraint c_2 in the example in Section 2) it's probably necessary to update the redundant constraints (c_4) too. This may lead to a low functionality because the knowledge base doesn't have the correct behavior.

Q4: Is the configuration knowledge base modifiable?

The development effort describes the effort when updating a configuration knowledge base. This effort contains the time for the update operation. This includes the update of the semantics of the knowledge base and the time, which is required to remove all new errors. This effort has an impact on the maintainability and reusability of a knowledge base and is mainly influenced by the *understandability* of a configuration knowledge base.

Q5: Is the configuration knowledge base understandable?

Not each goal has a relationship with each question. In Table 1 we give an overview of the relationship between goals and questions:

Question / Goal	MT	US	FT
Q1 (completeness)			+
Q2 (anomalies)	-	-	-
Q3 (performance)			+
Q4 (modifiability)	+		
Q5 (understandability)		+	

Table 1. Relations between goals and metrics (MT = maintainability, US = usability, FT = functionality)

3.3 Metrics for Configuration Knowledge Bases

The metrics are based on a literature review focusing on knowledge engineering [3, 5, 7, 16, 25, 26, 27, 30, 31, 32, 40] as well as on software product line engineering [2, 6, 22, 24]. The assumptions in this section are based on the literature of configuration knowledge bases and other research areas like feature models and software product lines, software engineering, and rule-based knowledge bases.

After having defined the questions for configuration knowledge bases, the next task is to quantify the metrics. Therefore, we describe possible metrics for configuration knowledge bases. Most of the metrics require a consistent CKB. The metrics are based on literature study in configuration, feature model and software engineering research areas.

The next list shows some metrics derived from MOOSE and function point analysis [2, 12, 25, 36]:

- **Number of variables** $|V|$: In the example (see Figure 1) $|V| = 8$.

- **Average domain size:** $domsize = \frac{\sum_{v_i \in V} |dom(v_i)|}{|V|} = 2$
- **Number of constraints:** $|C| = 8$

The **number of minimal conflicts** $|CS|$ is the first anomaly metric [23]. In our example (see Section 2), we have 1 minimal conflict, such that $|CS| = 1$. We can also evaluate the smallest number of constraints in a conflict set. The lowest number of constraints in a conflict set CS is the **minimal cardinality of conflict sets** $MCCS$ and can be defined as $\nexists CS_j : |CS_j| < |CS_i|$. The example in Section 2 has a minimal cardinality $MCCS = 4$.

We can also evaluate diagnoses for knowledge bases. For example, with the FastDiag algorithm [17, 19] we can calculate the **number of diagnoses** $|\Delta|$ and the number of constraints in a **minimal cardinality diagnosis** $MC\Delta$. A minimal cardinality diagnosis Δ_i is a minimal diagnosis which has the property of having the smallest number of constraints in the diagnosis, such that, $\nexists \Delta_j : |\Delta_j| < |\Delta_i|$. The example described in Section 2, contains 4 minimal diagnoses ($\Delta_1 = \{c_0\}, \Delta_2 = \{c_3\}, \Delta_3 = \{c_5\}, \Delta_4 = \{c_1, c_2\}, |\Delta| = 4$) and a minimal cardinality diagnosis of 1 ($MC\Delta = 1$).

The number of redundant constraints can also be used as a measure for knowledge bases [20, 28, 30, 31] if the configuration knowledge base is consistent.[2] The number of sets of **redundant constraints** is denoted as $|R|$ and the **maximum cardinality of a redundancy set** R_i is 1. We calculate the maximum cardinality for R_i by checking, if there exists another set R_j which has a bigger cardinality, such that R_i has the property of having the maximum cardinality, iff $\nexists R_j \, |R_j| < |R_i|$. The example in Section 2 contains one set with redundant constraints ($R_1 = \{c_4\}, |R| = 1$) and the maximum cardinality of these sets is 1 ($MCR = |1|$).

A domain element $dom_i \in dom(v_j)$ is a **dead domain element**, iff there does not exist a valid configuration, such that, $C \cup \{v_j = dom_i;\} \neq \emptyset$ [5, 6]. When assuming that $C' = C \cup \{c_8 : Standard = true;\}$ it is not possible, that $Clip$ is also $true$, such that, $C' \cup \{c_9 : Pedal = true\} = \emptyset$, such that, $DE = 1$. We use the sum of all dead elements as a metric $0 < DE < 1$ by using Equation 1 where a value nearer 0 means that there are no or less dead elements and a value nearer to 1 means that a high number of domain elements in the knowledge base can not be selected in a consistent configuration knowledge base.

$$DE = \frac{\sum_{v_i \in V} \sum_{d_j \in dom(v_i)} \begin{cases} 0 & C \cup \{v_i = d_j\} \neq \emptyset \\ 1 & else \end{cases}}{|V| \times domsize} \quad (1)$$

A domain element $dom_i \in dom(v_j)$ becomes dead if another domain element $dom_k \in dom(v_l), v_j \neq v_l$ is selected (*Conditionally dead domain elements CD* [6]). In the example (see Section 2) the constraint c_7 does not allow the configuration $Standard = true \wedge Clip = true$.

On the other hand, a domain element can be **full mandatory** (FM). Full mandatory means, that there does not exist a consistent instance of the knowledge base where this domain element isn't selected, formaly described as:

$$FM = \frac{\sum_{v_i \in V} \sum_{d_j \in dom(v_i)} \begin{cases} 0 & C \cup \{v_i \neq d_j\} = \emptyset \\ 1 & else \end{cases}}{|V| \times domsize} \quad (2)$$

Since each domain in our example knowledge base has two values ($true, false$) we can say, that whenever a domain element is dead, the other value becomes full mandatory automatically. When domains have more than two values, it can be the case, that a domain element is dead but there is no other domain value with the property of being a full mandatory domain element.

The third well-formedness violation is called **unnecessary refinement** (UR). Such a violation occurs when there are two variables and the domain element of the first variable in a valid configuration can be suggested by the assignment of a second variable. An unnecessary refinement can be described as $dom(v_i) \rightarrow dom(v_j)$.

In the example in Section 2 we can say that the variables $Standard$ and $Clip$ are an unnecessary refinement, because whenever $Standard = true$ $Clip = false$ and $Standard = false$ $Clip = true$. In that case, we can recommend, that the domain of the variable $Pedal$ can be replaced by $Standard, Clip$ and the variables $Standard$ and $Clip$ can be removed from the knowledge base without changing the semantics of the knowledge base.

The **restriction rate** RR compares the number of constraints with the number of variables. In the example described in Section 2 the restriction rate $RR = \frac{|C|}{|V|} = \frac{8}{8} = 1$. A value greater than 1 means that there is a high restriction [2, 25].

The metric RR is influenced by the design of the knowledge base. For example, while one knowledge engineer requires a single constraint for subsuming the constraints $c_0 \wedge c_1 \wedge c_2 \wedge c_3$ another knowledge engineer is using four single constraints. To consider these different design approaches in the metric, the **restriction rate** RR_2 is considering the number of variables in a constraint, such that, $RR_2 = \frac{\sum_{c_i \in C} \frac{\#vars(c_i)}{\#vars(C)} |C|}{|V|}$ where $\#vars(c_i)$ is the number of variables in c_i.

Another metric from the domain of software engineering is the **variable inheritance factor** VIF [1]. Adapted for configuration knowledge bases, we define VIF as the number of constraints in which a variable v_i appears related to the number of constraints, e.g., $VIF(Framecolor) = \frac{\sum_{c_i \in C} \begin{cases} 1 & v_{framecolor} \in c_i \\ 0 & else \end{cases}}{|C|} = 0.375$ because the variable $framecolor$ appears in three constraints and $|C| = 8$.

To receive a CKB metric we calculate the VIF_{all} for all variables. When calculating the arithmetic mean of the VIF_{all} of all variables, we can evaluate the importance distribution of all variables. A value near to 0 means, that all variables have the same importance and should be considered in the same way. On the other hand, a high value means that there are some important and less important variables in the knowledge base. In such cases, it makes sense to focus on the important variables when maintaining the knowledge base. $VIF_{all} = \sum_{v_i \in V} \frac{\sqrt{(VIF(v_i) - \frac{\sum_{v_j \in V} VIF(v_j)}{|V|})^2}}{|V|}$

Finally, we evaluate the metric **coverage**. The *coverage*

[2] To receive a consistent configuration knowledge base we remove the constraint c_5 from the set C.

measures the number of all consistent complete configurations (see Section 2) compared to the maximum number of complete configurations in a knowledge base. In our example in Section 2 the maximum number of configurations is $\prod_{i=0}^{|V|} |dom(v_i)| = 256$ (8 variables and each variable has a domain size 2). This will be compared with the number of consistent configurations. In our example we have the following consistent configurations:

$$
\begin{aligned}
&\{\\
&\quad Bike = true \wedge\\
&\quad Reflector = true \wedge\\
&\quad Pedal = true \wedge\\
&\quad Framecolor = true \wedge\\
&\quad (Standard = false \wedge Clip = true \wedge Green = true \wedge\\
&\qquad Red = false) \vee\\
&\quad (Standard = false \wedge Clip = true \wedge Green = false \wedge\\
&\qquad Red = true) \vee\\
&\quad (Standard = false \wedge Clip = true \wedge Green = true \wedge\\
&\qquad Red = true) \vee\\
&\quad (Standard = true \wedge Clip = false \wedge Green = true \wedge\\
&\qquad Red = false) \vee\\
&\quad (Standard = true \wedge Clip = false \wedge Green = false \wedge\\
&\qquad Red = true) \vee\\
&\quad (Standard = true \wedge Clip = false \wedge Green = true \wedge\\
&\qquad Red = true)\\
&\}
\end{aligned}
$$

Now we can compare the number of consistent configurations (= 6) with the number of all configurations (= 256). This leads to a coverage of $6/256 * 100 = 2.34375\%$ which is very low and the example configuration knowledge base is very restrictive. For the example knowledge base it is quite easy to evaluate all possible combinations of variables and domain elements. For knowledge bases with more variables, domain elements, and constraints we have millions and more possible combinations of variable assignments. For such scenarios we introduced the **simulation technique** in the context of knowledge based systems to approximate the *coverage*. For a detailed description to approximate this metric in large configuration knowledge bases we refer the reader to [33].

Finally, we can refer the questions to the metrics. Table 2 gives an overview of the relationships between the questions and the metrics.

	Q1	Q2	Q3	Q4	Q5
$\|V\|$	+		-		
$domsize$	+		-		
$\|C\|$	+		-		
$\|CS\|$	-	-		-	
$\|\Delta\|$	-	-		-	
$MCCS$					+
$MC\Delta$					+
$\|R\|$		-	-	-	
MCR					+
DE		-	-	-	-
FM		-	-	-	-
UR		-	-	-	-
RR				-	-
RR_2				-	-
VIF_{all}				-	-
$Coverage$				-	-

Table 2. Relations between metrics (rows) and questions (columns). A '-' means, that the metric has a negative impact on the question, '+' represents a positive impact.

For a detailed description of the calculation of metrics focusing on anomalies (conflicts, redundancies, and well-formedness violations) and the *coverage* metric we refer the reader to [33].

4 Discussion

In this Section we want to discuss relevant aspects of several metrics and give an insight in the implementation of the goal-question-metrics in the iCone interface.

Most of the research in the area of configuration knowledge engineering focuses on the area of verifying configuration knowledge bases (**functionality** goal, see Section 3) and ignores the question how to validate the knowledge base [30] (**maintainabilty** and **understandabilty**). Felfernig et al. [15] present an empirical study about the understandability of constraints in knowledge bases but there does not exist a metric for the understandability of constraints and the knowledge base.

Briand et al. [8] measured the effects of the structural complexity of software and its relationship to the **maintainability** of software. Bagheri and Gasevic [2] transferred this model into the area of feature models and found out, that the number of leaf features, the cyclomatic complexity, the flexibility of configuration, and the number of valid configurations influence the maintainability of feature models. While the simple metrics are easy to transfer into configuration knowledge bases, the depth of a tree or the number of valid configurations can not be calculated.

The number of **redundant constraints** is an important metric since a low number of redundant constraints can improve the maintenance task, simplify the understandability, and reduce the time for calculating valid configurations. An important issue in that case is, that redundant constraints can also improve the understandability of a configuration knowledge base. If a redundant constraint is declared as a desired redundant constraint, the metric should not contain such constraints, but should list it as a desired redundancy.

In a simple configuration knowledge base like the example in Section 2 it is easy to calculate the consistency of each possible configuration for the *coverage* metric. For example, in a configuration knowledge base with a medium number of variables (e.g. 10) and average domain size (e.g. 5) we have approximately $10M$ possible configurations. Since it is not possible to calculate so many possible configurations in realtime, we developed a simulation strategy to approximate the number of consistent configurations. For a detailed description of the simulation strategy, we refer the reader to [34].

While showing the GQM to knowledge engineers can help understand and maintain the configuration knowledge base, it is also important to interpret the results. Therefore we implemented a history for each metric in our iCone-interface[3]. When updates in a configuration knowledge base in the iCone-system are saved, a new version of the knowledge base will be created and metrics will be actualized. In Figure 2 we can see the changes of the value of the $DEAD$ elements metric.

[3] iCone is an 'intelligent environment for the development and maintenance of configuration knowledge bases' (`http://ase-projects-studies.ist.tugraz.at:8080/iCone/index.jsp`).

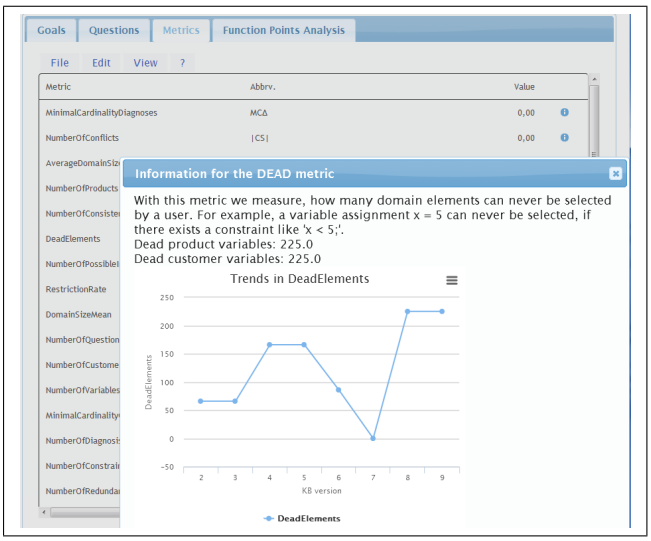


Figure 2. Visualization of changes for the metric *DEAD*. The y-axis shows the number of *DEAD* variables in each version of the configuration knowledge base (x-axis).

Felfernig [13] gives an overview of the usage of **function point analysis** for configuration knowledge bases. Therefore he analyzed the input of a configuration knowledge base from customers and the complexity of a configuration knowledge base. They use the customer requirements as external input (EI), the data, which are required by the user as external query (EQ), the consistent domain elements of variables as external output (EO), knowledge elements as internal logical files (ILF), and external information like the product assortment from an ERP-system as external interface file (EIF). While this approach takes input and output into account, it does not evaluate the quality (e.g., the number of *dead* domain elements) of the input and output.

We have implemented the GQM and the FPA approach in our iCone implementation [41]. Table 3 gives an overview of the performance. Note, that the time contains the calculation / approximation of the metrics and the calculation of all anomalies. The notebook domain is calculated six times and the mobile phone domain is calculated seven times.

	Notebooks	Mobile phones
Product variants	115.00	13,999.00
Product variables	28.00	34.00
product variable domain sizes	1.00 - 45.00	2.00 - 47.00
Customer variables	4.00	5.00
avg. customer variable dom. size	3.75	4.00
constraints	12.00	8.00
min. calc. time	669 msec.	6,811 msec.
max. calc. time	1,715 msec.	18,643 msec.
median calc. time	1,213 msec.	10,842 msec.
mean calc. time	1,252 msec.	11,307 msec.

Table 3. Duration for the calculation of all anomalies (conflicts, diagnoses, redundancies, well-formedness violations), metrics, goal-question-metrics and function-point-analysis for two configuration knowledge bases (notebooks and mobile phones)

In this paper we gave an overview of metrics in configuration knowledge bases, focusing on **knowledge base engineering processes** [38] is out of scope of this paper. We can measure the metrics of an existing configuration knowledge base, but we can not identify the causes of bad configuration knowledge base engineering. To give recommendations for optimizing the knowledge base engineering process, we have to observe the whole process.

5 Conclusion

This paper introduces a goal-question-metric approach to evaluate configuration knowledge bases. We gave an overview of configuration knowledge bases and introduced a running example for this paper and defined goals, questions, and metrics for configuration knowledge bases. Furthermore, we showed how to calculate time-consuming metrics efficiently and presented the iCone-visualization of metrics. It also points out some practical issues when dealing with metrics.

Future research should evaluate the relations between goals, questions, and metrics. Our future work will contain empirical evaluations about the correlation between the goals, questions, and metrics and their weightings in the aggregation process from metrics to questions and from questions to goals.

Future work should take a look at the knowledge engineering process. When calculating metrics for knowledge engineers, we can list some possible improvements. Preece gives an overview about verification and validation techniques for knowledge bases [30]. He aligns different V&V techniques to different models of a knowledge base, e.g., conceptual and design models and an implemented system. The metrics listed in Section 3 only focus on the implemented system. A GQM for the conceptual and design model does not exist.

Another relevant fact is the volatility of requirements [36]. If the requirements for the configuration knowledge base are changing frequently, it is also hard to keep the CKB up to date. For calculating metrics referring to the quality of a CKB and its requirements, it is necessary to integrate a requirements management system in the CKB maintenance tool or offer an interface for both tools.

Acknowledgements

The work presented in this paper has been conducted within the scope of the research project ICONE (Intelligent Assistance for Configuration Knowledge Base Development and Maintenance) funded by the Austrian Research Promotion Agency (827587).

REFERENCES

[1] F. B. Abreu and W. Melo. Evaluating the impact of object-oriented design on software quality. *Proceedings of the 3rd international software metrics symposium*, pages 90 – 99, 1996.

[2] Ebrahim Bagheri and Dragan Gasevic. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, 19(3):579–612, 2011.

[3] Valerie Barr. Applications of rule-base coverage measures to expert system evaluation. *AAAI*, 1997.

[4] Don Batory, David Benavides, and Antonio Ruiz-Cortes. Automated analysis of feature models: challenges ahead. *Commun. ACM*, 49:45–47, December 2006.

[5] Joachim Baumeister, Frank Puppe, and Dietmar Seipel. Refactoring methods for knowledge bases. In *Engineering Knowledge in the age of the Semantic Web: 14th international conference, EKAW, LNAI 3257*, pages 157–171. Springer, 2004.

[6] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35:615–636, September 2010.

[7] Jim Blythe, Jihie Kim, Surya Ramachandran, and Yolanda Gil. An integrated environment for knowledge acquisition. *IUI*, pages 14 – 17, 2001.

[8] L.C. Briand, J. Wust, S.V. Ikonomovski, and H. Lounis. Investigating quality factors in object-oriented designs: an industrial case study. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 345–354, 1999.

[9] Robin Burke. Knowledge-based recommender systems. In *Encyclopedia of library and information systems*, page 2000. Marcel Dekker, 2000.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41:15:1–15:58, July 2009.

[11] Yu-Chen Chen, Rong-An Shang, and Chen-Yu Kao. The effects of information overload on consumers' subjective state towards buying decision in the internet shopping environment. *Electronic Commerce Research and Applications*, 8:48 – 58, 2009.

[12] S. R. Chidamber and C. F. Kemerer. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, 20(6):476 – 493, 1994.

[13] Alexander Felfernig. Effort estimation for knowledge-based configuration systems. In Frank Maurer and Günther Ruhe, editors, *SEKE*, pages 148–154, 2004.

[14] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, editors. *Knowledge-based configuration. From research to business cases*, volume 1. Morgan Kaufmann, 2014.

[15] Alexander Felfernig, Monika Mandl, Anton Pum, and Monika Schubert. Empirical knowledge engineering: Cognitive aspects in the development of constraint-based recommenders. In Nicols Garca-Pedrajas, Francisco Herrera, Colin Fyfe, Jos Bentez, and Moonis Ali, editors, *Trends in Applied Intelligent Systems*, volume 6096 of *Lecture Notes in Computer Science*, pages 631–640. Springer Berlin / Heidelberg, 2010.

[16] Alexander Felfernig, Florian Reinfrank, and Gerald Ninaus. Resolving anomalies in configuration knowledge bases. *ISMIS*, 1(1):1 – 10, 2012.

[17] Alexander Felfernig and Monika Schubert. Personalized diagnoses for inconsistent user requirements. *AI EDAM*, 25(2):175–183, 2011.

[18] Alexander Felfernig, Monika Schubert, and Stefan Reiterer. Personalized diagnosis for over-constrained problems. *IJCAI*, pages 1990 – 1996, 2013.

[19] Alexander Felfernig, Monika Schubert, and Christoph Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *AI EDAM*, 26(1):53–62, 2012.

[20] Alexander Felfernig, Christoph Zehentner, and Paul Blazek. Corediag: Eliminating redundancy in constraint sets. In Martin Sachenbacher, Oskar Dressler, and Michael Hofbaur, editors, *DX 2011. 22nd International Workshop on Principles of Diagnosis*, pages 219 – 224, Murnau, GER, 2010.

[21] Yoav Ganzach and Yaacov Schul. The influence of quantity of information and goal framing on decision. *Acta Psychologica*, 89:23 – 36, 1995.

[22] Herman Hartmann and Tim Trew. Using feature diagrams with context variability to model multiple product lines for software supply chains. In *Proceedings of the 2008 12th International Software Product Line Conference*, pages 12–21, Washington, DC, USA, 2008. IEEE Computer Society.

[23] Ulrich Junker. Quickxplain: preferred explanations and relaxations for over-constrained problems. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, pages 167–172. AAAI Press, 2004.

[24] Kim Lauenroth and Klaus Pohl. Towards automated consistency checks of product line requirements specifications. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ASE '07, pages 373–376, New York, NY, USA, 2007. ACM.

[25] Timothy Lethbridge. Metrics for concept-oriented knowledge bases. *International Journal of Software Engineering and Knowledge Engineering*, 8:16–1, 1998.

[26] Mala Mehrotra, Dimitri Bobrovnikoff, Vinay Chaudhri, and Patrick Hayes. A clustering approach for knowledge base analysis. *American Association for Artificial Intelligence*, 2002.

[27] Doaa Nabil, Abeer El-Korany, and A. Sharaf Eldin. Towards a suite of quality metrics for kadss-domain knowledge. *Expert Systems with Applications*, 35:654 – 660, 2008.

[28] Cédric Piette. Let the solver deal with redundancy. In *Proceedings of the 2008 20th IEEE International Conference on Tools with Artificial Intelligence - Volume 01*, pages 67–73, Washington, DC, USA, 2008. IEEE Computer Society.

[29] Joseph B. Pine, editor. *Mass Customization. The new frontier in business competition*. Harvard Business School, 1992.

[30] Alun Preece. Building the right system right evaluating v&v methods in knowledge engineering, 1998.

[31] Alun D. Preece and Rajjan Shinghal. Foundation and application of knowledge base verification. *International Journal of Intelligent Systems 1994;9(8):683702. Duftschmid, S. Miksch /*, 22:23–41, 1994.

[32] ALUN D. Preece, STPHANE Talbot, and Laurence Vignollet. Evaluation of verification tools for knowledge-based systems. *International Journal of Human-Computer Studies*, 47(5):629 – 658, 1997.

[33] Florian Reinfrank, Gerald Ninaus, and Alexander Felfernig. Intelligent techniques for the maintenance of constraint-based systems. *Configuration Workshop*, 2015.

[34] Florian Reinfrank, Gerald Ninaus, Franz Wotawa, and Alexander Felfernig. Maintaining constraint-based configuration systems: Challenges ahead. *Configuration Workshop*, 2015.

[35] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[36] Pedro F. Salvetto, Milton F. Martinez, Carlos D. Luna, and Javier Segovia. A very early estimation of software development time and effort using neural networks. Workshop de Ingeniera de Software y Base de Datos, 2004.

[37] Cheri Speier. The influence of information presentation formats on complex task decision-making performance. *International Journal of Human-Computer Studies*, 64(11):1115 – 1131, 2006.

[38] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data & Knowlege Engineering*, 25:161 – 197, 1998.

[39] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[40] William van Melle, Edward H. Shortliffe, and Bruce G. Buchanan. Emycin: A knowledge engineer's tool for constructing rule-based expert systems. In Bruce G. Buchanan and Edward H. Shortliffe, editors, *Rule-Based Expert Systems. The Mycin Experiments of the Stanford Heuristic Programming Project*, pages 302–313. Addison-Wesley, 1984.

[41] Franz Wotawa, Florian Reinfrank, Gerald Ninaus, and Alexander Felfernig. icone: intelligent environment for the development and maintenance of configuration knowledge bases. *IJCAI 2015 Joint Workshop on Constraints and Preferences for Configuration and Recommendation*, 2015.

[42] Du Zhang and Doan Nguyen. Prepare: A tool for knowledge base verification. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):983–989, 1994.

Formal Analysis of the Linux Kernel Configuration with SAT Solving

Martin Walch[1] and **Rouven Walter**[1] and **Wolfgang Küchlin**[1]

Abstract. The Linux kernel is a highly configurable software system. The aim of this paper is to develop a formal method for the analysis of the configuration space. We first develop a Linux product overview formula (L-POF), which is a Boolean formula representing the high-level configuration constraints of the kernel. Using SAT solving on this L-POF, we can then answer many questions, such as which options are possible, mandatory, or impossible for any of the processor architectures for which the kernel may be configured. Other potential applications include building a configurator or counting the number of kernel configurations. Our approach is analogous to the methods we use for automobile configuration. However, in the Linux case the configuration options (e.g. the individual device drivers) are represented by symbols in Tristate Logic, a specialized three-valued logic system with several different data types, and the configuration constraints are encoded in a somewhat arcane language. We take great care to compile the L-POF directly from the files that hold the configuration constraints in order to achieve maximum flexibility and to be able to trace results directly back to the source.

1 Introduction

Linux is a kernel for a broad range of platforms with highly versatile configurations of peripheral components. Static configuration at compile time helps to adapt to the different requirements. The central tool for configuring this is *LinuxKernelConf*, abbreviated *LKC*. The input to LKC uses a domain specific language to describe configuration constraints. A common alternative name for LKC is *Kconfig*, and they are often used interchangeably. In this document, we refer to the configuration system as LKC and we denote the language as Kconfig. The input is large and stored in files which we call *Kconfig files*. They continuously change as kernel development goes on. Automatic checks for semantic consistency on Kconfig files are desirable, but LKC has no such checks implemented.

At the Workshop on Configuration 2010 in Lisbon, Zengler and Küchlin presented an approach [11] to encode the whole Linux kernel configuration in Propositional Logic. Conceptually this work parses the Kconfig files and stores the relevant information in a database. Subsequently, this database is translated into a *product overview formula*[2], abbreviated *POF*, in Propositional Logic. While it demonstrates central ideas and shows the technical feasability of the project, it is a first prototype with only a simplified view on the Kconfig files. As a consequence, the results were of the proof-of-concept type and of very limited use for verification purposes.

The comprehensive *VAMOS project*[3] at Friedrich-Alexander-Universität Erlangen-Nuremberg has lead to numerous results and publications, including the uncovering of hundreds of real world bugs. It also analyses the Linux kernel configuration with the means of Propositional Logic, but goes much further by considering the actual effects on the kernel code and applying the tools to other projects that use LKC as well. The PhD Thesis of Reinhard Tartler [9] from 2013 gives a detailed overview over the model, the implemented tools, and most of the applications and results.

The PhD Thesis of Sarah Nadi [5] from 2014 picks up the VAMOS project and extends it to not only consider the Kconfig files and the kernel code, but to additionally take the build system into account. Even more, it extracts configuration constraints from the implementation.

In this work, we focus solely on the Kconfig files. Although the VAMOS model proves to yield good results, it does not aim at being exact and relies on parts of LKC. We present a fairly precise model and translation process into a product overview formula in Propositional Logic with the goal to account for all details that are relevant in real-world use cases. Our implementation works independently from LKC and we show the results from running it against Linux 4.0.

This work is loosely based on the paper by Zengler and Küchlin from 2010 [11] in that it picks up and uses central ideas, but elaborates on many more details.

There is no precise specification of Kconfig. So we consider what information is available in the documentation, the implementation, and the way the input language is used. Section 2 gives a rough overview over this input language.

Our translation uses several intermediate stages. First we create what we call the *Zengler Model* in Section 3. In this step we abstract from technical details of reading the configuration input and isolate the data that is relevant for our purposes.

The Zengler Model retains some parts of the input structure that have an impact on the meaning. In Section 4, we transform it into the *Attributes Model*, resolving these parts and switching from a representation that focuses on input structure to one that revolves around the constraints.

LKC uses a three-valued logic, called *Tristate Logic*, that is uncommon in academic discourse. We take a look at this logic system in Section 5 and introduce some extensions to it. We then proceed by generating a product overview formula in this extended logic from the Attributes Model, encoding the set of all valid Linux kernel con-

[1] Symbolic Computation Group, WSI Informatics, Universität Tübingen, Germany, www-sr.informatik.uni-tuebingen.de

[2] A POF is a single Boolean formula which captures all configuration constraints of a high-level configuration model (cf. [3]). Historically it has been introduced in [4] to capture the high-level configuration model of Mercedes-Benz, which is called "Product overview" (German: Produktübersicht).

[3] VAMOS: Variability Management in Operating Systems, www4.cs.fau.de/Research/VAMOS/

figurations.

Eventually, we translate the product overview formula from the three-valued logic into Propositional Logic in Section 6. With common transformation into CNF and SAT solving, we can reason about the set of valid configurations, yielding first results in Section 7.

2 The Linux Kernel Configuration System

Kconfig is a specialized input language for LKC that describes available features in terms of symbols and offers several different means to encode constraints and interdependencies. Kernel developers maintain this information in hierarchically organized Kconfig files.

Listing 1. Example Kconfig file

```
config T0
        tristate
        prompt "feature T0"
        select T1
        depends on T2 && !C0

config T1
        tristate
        prompt "feature T1" if C0

config T2
        tristate
        default y

choice
        tristate
        prompt "CHOICE 0"

        config C0
                prompt "feature C0"

        config C1
                prompt "feature C1"

endchoice

menu "submenu"
        visible if T1

config S0
        string
        default "default string"

config B0
        bool
        prompt "custom S0"

if B0

config S0
        prompt "S0"

endif

endmenu
```

LKC reads these files with a scanner and a parser that are generated using *flex*[4] and *GNU Bison*.[5] Exploring the Bison grammar is clearly beyond the scope of this document as it consists of more than 100 production rules. Suffice it to say that the grammar we use very closely ressembles the grammar of LKC.

To give an overview over the relevant parts of the configuration language, we confine ourselves to explaining an artificial example. A concise formal description of the relevant parts was done by She and Berger [7].

Listing 1 contains most of the relevant language features. The central element everything else is built around is the *configuration block*. Its purpose is to declare and to describe symbols. The keyword `config` starts a configuration block and is followed by the name of the symbol it refers to. A configuration block contains one or more lines that further specify what we call *properties* of that symbol. Each symbol must have at least one configuration block. In practice, for the majority of symbols there is exactly one configuration block for each symbol, but there may be more. In Linux 4.0, there are up to six per symbol.

Each symbol has one of the five types `tristate`, `bool`, `string`, `int`, and `hex`. In our example, the symbols *T0*, *T1*, and *T2* have the type `tristate`, the symbol *B0* has the type `bool`, and the symbol *S0* has the type `string`. We call a symbol *declared* if it has at least one configuration block and one of the five types.

On actual configuration, each symbol is assigned an unambiguous value from their respective domains. The domain of `tristate` symbols is $\{0, 1, 2\}$ and the domain of `bool` symbols is $\{0, 2\}$. The purpose of the three-valued `tristate` type is to encode the three possible activation states that apply to many features in the kernel: Turn it off (0), compile it as a runtime-loadable module (1) or compile it into the kernel (2). The `bool` type is the same except that it is missing the value 1 for the module state.

To identify the three different states in syntax, there are the three constants `n`, `m`, and `y`, which evaluate to 0, 1, and 2, respectively.

The domain of `string` symbols is the set of all valid strings. Similar to the domain of `bool` being a subset of the domain of `tristate`, the domains of `int` and `hex` symbols both are subsets of the domain of `string` symbols, in that their elements are strings that read as valid integers or hexadecimal numbers, respectively.

There are different types of dependencies that affect properties, thus most properties can be active or inactive. The symbol type is the only unconditional property. It is always active and cannot dynamically change during configuration. It is also the only property that is mandatory for every symbol. However, to allow the user to directly assign a value, the symbol needs a `prompt` property. In our example, there are several such `prompt` properties, one of them for the symbol `T0`. The `prompt` keyword is followed by a string parameter *"feature T0"*. This parameter is mandatory and shows up as short description in the configuration interface.

The symbol `T0` also has a `select` property with the argument `T1`. This property is only valid with `bool` and `tristate` symbols. The `select` property sets up a direct relation between the values of two symbols. The value of the selected symbol is always at least as high as the value of the symbol that the `select` property belongs to. So, in our example the value of `T1` is always at least as high as the value of `T0`. The documentation calls this a *reverse dependency*.

The last line of the configuration block of `T0` starts with `depends on` and is followed by an expression in Tristate Logic `T2 && !C0`.

[4] flex: The Fast Lexical Analyzer, `flex.sourceforge.net`
[5] GNU Bison, `www.gnu.org/software/bison/`

This is a dependency that applies to the whole configuration block, i.e. in our case it is a dependency to the `prompt` and to the `select` property. The actual dependency is encoded in the expression. It has no direct influence on the value of the symbol, but rather on the properties which in turn set up constraints to the value.We take a closer look at Tristate Logic and the mechanisms of dependencies in section 5.

It is possible to add a dependency to a single property by appending an expression with `if` to the respective line, like for the configuration prompt of `T1`.

A `default` property is used to non-bindingly suggest a value, but it is also used to automatically set values of symbols that have no active `prompt`. In our example, the symbol `T2` uses the trivial expression `y` as `default` value, but more complicated expressions are possible.

Our example contains a *choice block*. This language construct encloses a series of configuration blocks with `choice` and `endchoice`, constituting a set of symbols that logically exclude each other. This is useful for features that serve the same purpose and which therefore naturally cannot be simultaneously active, like e.g. the compression algorithm of the kernel image or the preemption model. Like a symbol, a *choice block* has a type, but only the two data types `bool` and `tristate` are valid. A choice block of type `bool` enforces that exactly one of the enclosed symbols is set to 2. The `tristate` type relaxes this strict constraint and allows to not assign 2 to any of the symbols, while setting an arbitrary number of symbols to 1. There is a property `optional` that even allows setting all symbols to 0. In contrast to a symbol, a choice block must have a `prompt` property. There may also be `default` properties, suggesting one of the enclosed symbols, and dependencies using `depends on`, which analogously to configuration blocks apply to the whole choice block.

A hierarchical menu can be constructed using the `menu` blocks. Its primary purpose is organizing the features to ease navigation. Just like the `prompt` property, the `menu` keyword is followed by a string that serves as user visible label for the menu. For our point of view it is important that there may be dependencies added using `depends on` that apply to the whole menu, and a second type of dependency that is set up with `visible if`. That second kind of dependency toggles only the parts that concern direct configurability by the user, but leaves all other properties unaffected.

Finally it is also possible to add dependencies to an arbitrary sequence of configuration blocks, choice blocks, and menus as long as it does not violate the logical hierarchy. This is done with the enclosing keywords `if` and `endif` with the dependency expression right behind `if`.

Our example covers those language constructs that we regard as most important for variability, but there is more: For symbols with the `int` or `hex` type, there is a `range` property that fixes a lower and an upper bound for the value. Both bounds may depend on freely changeable values of other symbols. In practice nearly all bounds have constant values and those few bounds that are variable can be manually checked for inconsistencies. From our perspective the impact of `range` properties on the variability of the overall configuration is mostly negligible in current versions of Linux. Therefore, we incorporate `range` properties in our model, but ignore them when creating a product overview formula as we do not expect any observable consequences.

Another language feature that does not occur in our example is the generic `option` line in configuration blocks, which allows extending the language with only minimal changes to the grammar and possibly gaining forward compatibility. It starts with the `option` keyword, followed by one of a few option names, and depending on that, the meaning varies and an argument may be appended. As of Linux 4.0, two different options are defined that may be relevant in our context: `modules` and `env`. The `modules` option makes only sense with a `bool` symbol, and it may be only used once in the whole configuration. It associates the corresponding symbol with the availability of support for loading modules, i.e. as a consequence, deactivating that symbol basically prohibits assigning the value 1 from the `tristate` domain. The `env` option is usually only used for `string` symbols that have no `prompt` property as it imports a value from the runtime environment of the system the configuration system is running on. Only very few symbols use this option at all and even fewer have an effect on the configuration that is worth mentioning. Still, as we want to be precise, we account for them, too.

From our example it is also not apparent how Kconfig files are hierarchically organized. It works similarly to `#include` directives of the C preprocessor: A file inclusion is done with the keyword `string` and a string argument. The string argument is treated as path and the content of the file at that path is pasted at the position of the `string` line.

Kconfig still offers more constructs that we have not mentioned, but they are irrelevant at this point.

3 Zengler Model

Linux 4.0 supports 30 different main architectures. Each architecture has its own tree of Kconfig files, but there are big overlaps across all architectures by using common files. Accordingly we want to be able to reason about the configurations across all architectures. However, capturing several architectures in one data structure without losing precision is very complicated as this requires keeping track of which files are included for each architecture and in which order they are read. Therefore, we create a separate model for each architecture and consider them together if needed.

By considering one fixed architecture, we can deduce the full inclusion hierarchy of Kconfig files for that specific architecture. Accordingly, our parser moves through the file hierarchy in the same way as LKC does, without the need to store which files we include. However, it produces different data structures. We create two databases, a *symbol database* $\mathcal{D}_S$, and a *choice database* $\mathcal{D}_C$. They are heavily extended versions of the databases created by Zengler and Küchlin [11], hence we call these two databases the Zengler Model.

The symbol database contains *symbol descriptors*. Each configuration block corresponds to a symbol descriptor. They are grouped by the symbol they belong to. So, our *symbol database* is a set of pairs $\mathcal{D}_S = (s, \overrightarrow{\mathbf{desc}})$ where s is a symbol and $\overrightarrow{\mathbf{desc}}$ is a list of symbol descriptors $\mathbf{desc}$ with

$$
\begin{aligned}
\mathbf{desc} &= (t, \overrightarrow{\mathbf{pmpt}}, \overrightarrow{\mathbf{sel}}, \overrightarrow{\mathbf{def}}, \overrightarrow{\mathbf{rangé}}, \mathbf{dep}, \mathbf{opt}, k_{desc}) \\
\mathbf{pmpt} &= (e_p, k_p) \\
\mathbf{sel} &= (s_s, e_s, k_s) \\
\mathbf{def} &= (e_v, e_d, k_d) \\
\mathbf{range} &= (e_l^s, e_h^s, e_r, k_r) \\
\mathbf{dep} &= (\overrightarrow{e_{if}}, \overrightarrow{e_{dep}}, \overrightarrow{e_{vis}}) \\
\mathbf{opt} &= (b_{env}, b_{mod}).
\end{aligned}
$$

The type t is one of the five types or it is a special type "unknown" if the configuration block has no type property. It suffices if only one descriptor holds a regular type.

The properties `prompt`, `select`, `default`, and `range` appear in the descriptor as the tuples **pmpt**, **sel**, **def**, and **range**, respectively. As the input order plays a role, each of them and the descriptor contain unique indices k_{desc}, k_p, k_s, k_d, and k_r that are ascending in input order.

The string argument of a `prompt` property is unimportant for the configuration logic, so we only store the expression of the optional `if` dependency e_p. If there is no such dependency, we store a special symbol as placeholder for an empty expression. Analogously, the optional `if` dependencies for `select`, `default`, and `range` properties are stored as e_s, e_d, and e_r.

The target of a `select` property is a single symbol e_s. A `default` value may be the result of a non-trivial expression e_v. The lower bound of a `range` property e_l^s may be either another symbol or a constant value. The same goes for the upper bound e_h^s.

We do not store any menus or enclosing ifs explicitely. Instead we propagate their dependencies to the contained descriptors and store the dependency expressions in lists that distinguish between the different types. The dependencies of surrounding ifs are stored in $\overrightarrow{e_{if}}$. Those of `depends on` dependencies add up to $\overrightarrow{e_{dep}}$, including any such dependencies contained in the configuration block under consideration. The third list $\overrightarrow{e_{vis}}$ contains the `visible if` dependencies from menus.

The pair of bits **opt** indicates if there is a `module` or `env` option. Even though `env` carries an argument in the configuration block, we only care whether it is present.

Our choice database $\mathcal{D}_C$ is a set of tuples which we call *choice descriptors*:

$$(t, b_{opt}, \overrightarrow{\mathbf{pmpt}}, \mathbf{dep}, \overrightarrow{k_{desc}}, k_c)$$

Each choice block translates into a choice descriptor. The type t is one of `bool`, `tristate` or "unknown". The bit b_{opt} indicates whether the choice block carries the optional flag. Just like in a symbol descriptor, there is a list of **pmpt** tuples, dependencies lists **dep**, and an index k_c. Rather than storing the symbol names of the contained symbol descriptors, we create a list of their indices $\overrightarrow{k_{desc}}$.

In contrast to the symbol descriptors, the `default` properties of choice blocks have no influence on variability and we ignore them.

4 Attributes Model

Although the Zengler Model is an abstraction from the underlying Kconfig files, storing the relevant information in a normalized way, its focus lies on the input structure: The symbol database holds individual symbol descriptors **desc** and lists of dependencies **dep** reflecting artifacts from the input structure. Furthermore, we have yet to determine the final types of the symbols and choices and which symbols a choice includes as this is not trivial in all cases.

In the next step, we resolve all this and create a model that focuses on the effective impact of the descriptor properties on the symbols. For a better distinction from the properties of the descriptors in the Zengler Model, we refer to their resulting equivalents in the new model as attributes. Therefore we call the new model the *Attributes Model*.

First we associate the attributes directly with the dependencies from **dep** that control them. As already mentioned in Section 2, the dependencies in $\overrightarrow{e_{vis}}$ do not affect all types of attributes. In fact, they control only our `prompt` attributes a^P. We define them as

$$a^P = (E_p, k_p) = (\overrightarrow{e_{if}} \cup \overrightarrow{e_{dep}} \cup \overrightarrow{e_{vis}} \cup e_p, k_p)$$

We write E to denote sets of expressions. The other attributes are only affected by the other dependencies, hence we define them as follows:

$$\begin{aligned} a^S = (E_s, s, k_s) &= (\overrightarrow{e_{if}} \cup \overrightarrow{e_{dep}} \cup e_s, s, k_s) \\ a^D = (E_d, e_v, k_d) &= (\overrightarrow{e_{if}} \cup \overrightarrow{e_{dep}} \cup e_d, e_v, k_d) \\ a^R = (E_r, e_l^s, e_h^s, k_r) &= (\overrightarrow{e_{if}} \cup \overrightarrow{e_{dep}} \cup e_r, e_l^s, e_h^s, k_r) \end{aligned}$$

With these definitions we collect for each symbol s the `prompt` attributes a^P in a set $P(s)$, the `default` attributes a^D in a set $D(s)$, and the `range` attributes a^R in a set $R(s)$. We also create a set $S(s)$ for each symbol, but note that the symbol in the definition of a^S is the symbol of the descriptor that *contains* the property and not the symbol s_s that is the `select` target. Accordingly we add each a^S to the set of the target symbol. If there is no descriptor of the target symbol we discard the attribute.

To determine the symbol type that results from the types in the corresponding descriptors, we take the type of the symbol descriptor with the lowest k_{desc} that is not "unknown". If there is no such descriptor we also set the type of the symbol to "unkown".

Finally we concatenate the **opt** bitvectors component-wise with a logical or and store it with the symbol.

We do not need **dep** and k_{desc} anymore and do not transfer them into the Attributes Model.

Next we transform the choice descriptors from the Zengler Model into what we call *choice groups*. Like for symbols, we determine the type of the choice group. If t in the choice descriptor is `bool` or `tristate`, then this is also the type of the choice group. However, it is a frequently used feature to skip the explicit type declaration in choice blocks, resulting in choice descriptors with the special type "unknown". In that case the type is taken from the first symbol that is enclosed in the choice descriptor and has a regular type. If any symbol in the choice descriptor is lacking a regular type then it inherits the type of the choice group.

Now that we have completed the type resolution for choice groups, we determine which symbols are part of the choice group. This is not trivial as not all symbols that correspond to symbol descriptors in the choice descriptor are necessarily transferred. Symbols can be moved into their own submenu by depending on a symbol that is immediately above, excluding them from the choice group. We actually see this constellation intentionally used in Linux 4.0 and have to process it adequately. LKC involves an extensive logic to determine whether to move a symbol into a submenu, but a simple heuristic suffices to correctly capture any real-world case.

As LKC ignores the attributes $S(s)$ and $D(s)$ of all symbols of the choice group, we clear them if they have any content.

To complete the choice group, we generate a `prompt` attribute a^P the same way we do from a symbol descriptor and we keep the `optional` bit b_{opt}.

Our new databases $\mathcal{D}'_S$ and $\mathcal{D}'_C$ are now

$$\begin{aligned} \mathcal{D}'_S &= \overrightarrow{(s, t, P(s), S(s), D(s), R(s), \mathbf{opt})} \\ \mathcal{D}'_C &= \overrightarrow{(t, b_{opt}, a^P, \overrightarrow{s})} \end{aligned}$$

5 Tristate* Logic and POF

Our next step is to translate the Attributes Model into a coherent product overview formula. While our goal is to arrive at a POF in Propositional Logic, we prefer to split this translation into two steps. First we define Tristate* Logic, an extension to Tristate Logic that

we specifically design to create an initial POF. Then we translate the POF from Tristate* Logic into Propositional Logic in Section 6.

The following grammar describes the general syntax of expressions in Tristate Logic:

⟨*expression*⟩ → ⟨*expression symbol*⟩
| ⟨*expression symbol*⟩ '=' ⟨*expression symbol*⟩
| ⟨*expression symbol*⟩ '!=' ⟨*expression symbol*⟩
| '(' ⟨*expression*⟩ ')'
| '!' ⟨*expression*⟩
| ⟨*expression*⟩ '&&' ⟨*expression*⟩
| ⟨*expression*⟩ '||' ⟨*expression*⟩

⟨*expression symbol*⟩ → symbol name
| constant value

There are five operators and there are parentheses to override operator precedences. We distinguish two groups of operators: The `tristate` operators `!`, `&&`, and `||`, and the `string` operators = and `!=`. Both groups operate on their respective domains.

The `tristate` operators and their domain form a three-valued logic like those of Łukasiewicz and Kleene. A comprehensive overview of these logics has been done by Gabbay and Woods [2].

In fact, the three base operators `!`, `&&`, and `||` correspond to the operators $\neg$, $\wedge$ and $\vee$ in K_3 and $Ł_3$ from Kleene and Łukasiewicz. This observation has already been made in 2010 by Berger et al. [1]. However, Tristate Logic itself is not expressive enough for a full POF. We need to extend it for our purposes.

The nature of our dependencies is Boolean and not three-valued, because either they are met or they are not. There is no third value like the module state in Tristate Logic. Hence, when extending Tristate Logic to allow encoding all constraints in a POF, we look for operators that operate on `tristate` values, but only yield two different values. Such operators are not typically part of K_3 or $Ł_3$ and hence we exclude these logics from further consideration.

Instead we introduce our own new operators $\Rightarrow$ and $\Leftrightarrow$ and define their semantics as shown in Table 1 and Table 2.

Table 1. Truth table of tristate* operator $\Leftrightarrow$

$\Leftrightarrow$	0	1	2
0	2	0	0
1	0	2	0
2	0	0	2

Table 2. Truth table of tristate* operator $\Rightarrow$

$\Rightarrow$	0	1	2
0	2	2	2
1	0	2	2
2	0	0	2

They express equality and "less than or equal to" inequality on `tristate` values. We call this extended version of Tristate Logic the *Tristate* Logic*.

Note that Tristate* Logic also contains the `string` operators = and `!=`. They compare values from the `string` domain and also yield one of the two values 0 and 2. Mixing `tristate` and `string` operators and symbols in expressions is actually a feature, leading to frequent conversions between the two domains.

This may become quite complex. Consider the short expression `A=B`. If both, `A` and `B`, are declared `string` symbols, their values are compared, yielding 2 if they are exactly identical and 0 otherwise. However, if `A` is a `tristate` symbol and `B` is undeclared, then the values 0, 1, 2 of `A` are interpreted as the strings "*n*", "*m*", and "*y*" and *B* is interpreted as the `string` "*B*", i.e. in this case a string comparison is done on these letters, always yielding 0. We take all these details into account when producing our POF in Tristate* Logic.

Encoding the constraints of `bool` and `tristate` symbols that originate from their respective attributes, works in an indirect way: For each of these symbols, we add two auxiliary variables which represent a lower and an upper limit to the value of the symbol in consideration, and, using the $\Rightarrow$ operator, we append the constraint that the value of the symbol must not exceed the values set up by the auxiliary variables.

This is in contrast to encoding the choice groups: We encode the exclusiveness of symbols with expressions that directly relate to the symbols instead of their associated auxiliary variables.

Finally, we also take account of the mode without module support by adding two different subformulae for `tristate` symbols and `tristate` choice groups which depend on the symbol that has the b_{mod} bit set.

6 POF in Propositional Logic

Translating the POF from Tristate* Logic into Propositional Logic is mostly straightforward. We encode each symbol with the type `tristate` or `bool` and all auxiliary variables using two variables in Propositional Logic. The three values of the `tristate` domain correspond to three states that the two Propositional Variables can encode. We explicitely prohibit the fourth possible state.

Translating `tristate` constants, symbols, and operators works mostly the same way as in the paper by Zengler and Küchlin [11] with the two projections π_0 and π_1 as listed in table 3. The three constants `y`, `m`, and `n` map to corresponding combinations of $\top$ and $\bot$, and similarly each `tristate` symbol A^T maps to two propositional variables $p_0(A^T)$ and $p_1(A^T)$. Mapping the unary operator `!` is simple, but the entries for the binary operators operators `&&` and `||` stick out by being generalized to n-ary operators. This is an optimization to keep the size of the formulae smaller for long chains of the same operands.

Of course, the Tristate* operators $\Leftrightarrow$ and $\Rightarrow$ also have to be translated into Propositional Logic. However, due to our definition of these operands this is trivial for π_1 as they yield only 0 and 2, and π_0 is intuitive. We use these two translations to also cover the usage of the `string` operator = with `tristate` operands.

Finding a Propositional encoding for `string` symbols requires more work, because Propositional Logic is not well suited for encoding arbitrary strings. For each `string` symbol, we iterate over our POF in Tristate* Logic and collect all occurrences in expressions. In our method we consider only cases of equal strings and resulting other equalities and inequalities and define new propositional variables $P_{X^S \leftarrow \mathfrak{s}}$ to encode that the `string` variable X^S has the value $\mathfrak{s}$ and accordingly $(X^S = Y^S)$ to encode if X^S and Y^S have the same value. This suffices, because for the configuration space the actual value of a `string` is not important. The same goes for `int` and `hex` symbols. Finally we use the mappings for = and `!` to define the mappings for `!=`.

With these mappings we create our POF in Propositional Logic.

As it encodes the configuration space of the Linux kernel, we call it *L-POF*.

To use modern SAT solvers we do one final step: Our L-POF is not in CNF. So we use the Warthog Logic Framework[6] to generate an equisatisfiable formula using the Plaisted-Greenbaum algorithm [6].

7 Results

Our implementation consists of more than 7,000 lines of Java Code, using the features of Java SE 6. We measured execution times on a computer with an Intel Core2Quad Q6600 using the Java implementation of the IcedTea project in version 6.1.13.7 on Gentoo Linux. Creating the L-POFs from the Kconfig files for all 30 architectures, takes on average less than 3 seconds per architecture. Transformation into CNF varies between 20 and 42 seconds, depending on the architecture.

To give a rough impression of size of the configuration space, we show the distribution of symbol types for each architecture in Table 4. Across all architectures, the vast majority of symbols has one of the two types `tristate` and `bool` which has a major impact on the form of the formula.

Table 4. Distribution of symbol types in Linux 4.0

arch	tristate	bool	string	int	hex	total
alpha	6317	3402	34	192	27	9972
arc	6293	3352	36	194	29	9904
arm	6371	4724	38	209	41	11383
arm64	6366	3511	36	195	27	10135
avr32	6360	3462	36	193	30	10081
blackfin	6380	3676	36	702	43	10837
c6x	6292	3293	35	189	28	9837
cris	6345	3436	45	254	78	10158
frv	6316	3378	35	192	28	9949
hexagon	6292	3292	35	190	27	9836
ia64	6400	3519	36	195	27	10177
m32r	6324	3361	34	194	31	9944
m68k	6322	3453	35	192	37	10039
metag	6293	3363	37	193	28	9914
microblaze	6294	3335	37	195	32	9893
mips	6391	3964	36	198	28	10617
mn10300	6316	3426	35	199	33	10009
nios2	6292	3303	36	191	36	9858
openrisc	6292	3292	36	188	27	9835
parisc	6325	3384	34	191	27	9961
powerpc	6404	3933	37	205	40	10619
s390	6313	3453	35	195	27	10023
score	6292	3292	35	188	28	9835
sh	6374	3640	37	198	34	10283
sparc	6370	3444	37	193	30	10074
tile	6306	3393	36	191	29	9955
um	6297	3362	38	193	27	9917
unicore32	6363	3425	36	191	27	10042
x86	6420	3781	40	206	32	10479
xtensa	6326	3379	41	194	29	9969

Table 5 gives a rough overview of how big the formulae grow. It comes to no surprise that the translation from Tristate* Logic to Propositional Logic and the transformation into CNF using the Plaisted-Greenbaum algorithm both significantly increase the size. The fact that there are more regular variables in the POF in Tristate* Logic than there are declared symbols on the respective architecture comes from the fact that it still contains expressions with undeclared symbols. They are cleaned in the translation process into Propositional Logic. Each of our formulae in CNF contains roughly one million Propositional variables. For some very basics analyses with SAT solving, we use PicoSAT.[7] Processing the CNF formula without any additional clauses takes PicoSAT between 6 and 10 seconds. We search for redundant `bool` or `tristate` symbols, i.e. symbols that can never be active, and for symbols that are necessary, i.e. it is not possible to fully deactivate them. We find out if a symbol is redundant by assuming that one of the two corresponding Propositional variables is `true`. If this is not satisfiable, then the symbol is always inactive. Vice versa by assuming that both Propositional variables are `false` we find out whether a symbol must always be active. More than 99.8% of the individual tests run in less than three seconds.

Table 6. Redundant or necessary symbols in Linux 4.0

arch	redundant ($\Leftrightarrow$ `n`)	necessary (not $\Leftrightarrow$ `n`)
alpha	3223	55
arc	4263	63
arm	1691	75
arm64	3159	135
avr32	4522	54
blackfin	4430	47
c6x	4644	42
cris	3785	34
frv	3700	37
hexagon	4625	45
ia64	3454	74
m32r	4937	32
m68k	3741	32
metag	4301	67
microblaze	3251	71
mips	2773	64
mn10300	3702	39
nios2	4428	45
openrisc	4424	56
parisc	3499	51
powerpc	2652	94
s390	4149	107
score	7068	36
sh	3297	67
sparc	3201	51
tile	3633	57
um	7368	28
unicore32	3541	58
x86	2301	138
xtensa	3248	43
globally	135	1

Table 6 shows the results of these tests. The reason for the high numbers of features that cannot be activated on each architecture is that there are many features that run only on few architectures, but the corresponding Kconfig files are common for all architectures. Symbols that cannot be deactivated on the other side are less, but still a lot. They are symbols that are intentionally not deactivatable and in general they do not represent selectable features, but basic aspects of an architecture.

[6] Warthog Logic Framework: github.com/warthog-logic/warthog

[7] PicoSAT: fmv.jku.at/picosat/

Table 3. Translation rules for tristate* operators

e'	$\pi_0(e')$	$\pi_1(e')$
`y`	$\top$	$\bot$
`m`	$\bot$	$\top$
`n`	$\bot$	$\bot$
A^T	$p_0(A^T)$	$p_1(A^T)$
`!` e	$\neg\pi_0(e) \wedge \neg\pi_1(e)$	$\pi_1(e)$
e_0 `&&` $\cdots$ `&&` e_n	$\pi_0(e_0) \wedge \cdots \wedge \pi_0(e_n)$	$\bigwedge_{i\in\{0,\ldots,n\}}(\pi_0(e_i) \vee \pi_1(e_i)) \wedge \bigvee_{i\in\{0,\ldots,n\}} \pi_1(e_i)$
e_0 `||` $\cdots$ `||` e_n	$\pi_0(e_0) \vee \cdots \vee \pi_0(e_n)$	$\bigwedge_{i\in\{0,\ldots,n\}}(\neg\pi_0(e_i)) \wedge \bigvee_{i\in\{0,\ldots,n\}} \pi_1(e_i)$
$e_1 \Leftrightarrow e_2$	$(\pi_0(e_1) \leftrightarrow \pi_0(e_2)) \wedge (\pi_1(e_1) \leftrightarrow \pi_1(e_2))$	$\bot$
$e_1 \Rightarrow e_2$	$\pi_0(e_2) \vee \neg\pi_0(e_1) \wedge (\neg\pi_1(e_1) \vee \pi_1(e_2))$	$\bot$
$A^T = t$	$\pi_0\left(A^T \Leftrightarrow t\right)$	$\bot$
$A^T = B^T$	$\pi_0\left(A^T \Leftrightarrow B^T\right)$	$\bot$
$X^S = \mathfrak{s}$	$P_{X^S \leftarrow \mathfrak{s}}$	$\bot$
$X^S = Y^S$	$P_{X^S = Y^S}$	$\bot$
e_1^s `!=` e_2^s	$\neg\pi_0(e_1^s = e_2^s)$	$\bot$

To get meaningful results, we merge the results. We collect all symbols that do not have any architecture that allows activating, and we collect all symbols that are declared across all architectures, but may not be deactivated on any of them. These numbers are in the line globally. For most of the 135 symbols that cannot be activated, this is actually the intention of the maintainer. The one `tristate` symbol that can never be deactivated may intentionally only alternate between the two other possible states.

These results do not surprise as similar tests were also done in the context of the VAMOS project and hence many problems have already been uncovered and solved.

8 Conclusion

Our approach successfully leads to a precise product overview formula. Although Linux has reached more than 10.000 features, our implementation quickly creates the L-POF, a product overview formula of the Linux kernel in Propositional Logic. Despite its considerable size, fast SAT solving on the formula is in general possible. If needed there is still much room for optimization.

A future research direction could be to investigate the possibility of further verification tests beyond redundant and necessary symbols, e.g. specialized verification tests for a choice block analogously to verification tests for positions of a Bill of Materials as it is done in automotive configuration [8].

As we do not use parts of LKC in our implementation, we now have the option to extend our program to do fine-grained tests considering individual lines in Kconfig files and easily locate the origin of inconsistencies.

Another interesting topic is re-configuration. Although LKC does not permit invalid configurations by disabling options during the configuration process, it might be useful for users to select all wanted options first without caring about the validation. Afterwards, if the configuration is invalid, we can re-configure the selections of the user in an optimal way, i.e. by solving a MaxSAT optimization problem.

The reverse is also imaginable: If the selections of a user lead to an invalid configuration, the user might want to know which configuration constraints have to change in order to make the configuration valid. Thus, we want to find the minimal set of constraints to remove or change. Such MaxSAT re-configuration use cases have been described in the context of automotive configuration in [10] and could be adopted for the Linux kernel configuration.

Table 5. Sizes of POFs for Linux 4.0

arch	regular variables in POF in Tristate* Logic	auxiliary variables in POF in Tristate* Logic	total number of variables in POF inTristate* Logic	variables in L-POF	variables in CNF	clauses in CNF
alpha	10673	48845	59518	117417	996839	1812856
arc	10602	48481	59083	116538	954428	1718683
arm	11976	55760	67736	134270	1299812	2849653
arm64	10824	49640	60464	119333	1007563	1828031
avr32	10793	49366	60159	118671	1001036	1816464
blackfin	11539	51058	62597	123096	1026513	1861942
c6x	10548	48174	58722	115799	949031	1708805
cris	10867	49279	60146	118559	999006	1812066
frv	10666	48722	59388	117126	990489	1797565
hexagon	10540	48169	58709	115795	981542	1781667
ia64	10866	49850	60716	119837	1010856	1834072
m32r	10643	48681	59324	117039	993691	1804508
m68k	10717	49136	59853	118115	1008800	1836987
metag	10605	48535	59140	116671	955382	1720572
microblaze	10591	48406	58997	116370	985108	1788040
mips	11249	52034	63283	125090	1048937	1909971
mn10300	10721	48974	59695	117738	994158	1804015
nios2	10566	48235	58801	115951	950035	1710610
openrisc	10544	48168	58712	115783	949121	1709044
parisc	10658	48794	59452	117297	996712	1810111
powerpc	11247	51964	63211	124935	1055822	1917736
s390	10699	49084	59783	117997	998901	1813210
score	10539	48168	58707	115783	949788	1710461
sh	10955	50336	61291	121037	1020515	1854779
sparc	10774	49327	60101	118582	1004762	1823946
tile	10655	48748	59403	117195	990749	1798113
um	10606	48550	59156	116723	998908	1821791
unicore32	10753	49191	59944	118246	998333	1811566
x86	11135	51280	62415	123314	1051478	1913811
xtensa	10674	48786	59460	117278	993296	1802906

References

[1] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wąsowski, and Krzysztof Czarnecki, 'Variability Modeling in the Real: A Perspective from the Operating Systems Domain', in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, pp. 73–82, New York, NY, USA, (2010). ACM.

[2] *The Many Valued and Nonmonotonic Turn in Logic, Volume 8 (Handbook of the History of Logic)*, eds., Dov M. Gabbay and John Woods, North Holland, 1 edn., 8 2007.

[3] Albert Haag, 'Sales configuration in business processes', *IEEE Intelligent Systems*, **13**(4), 78–85, (July 1998).

[4] Wolfgang Küchlin and Carsten Sinz, 'Proving consistency assertions for automotive product data management', *J. Automated Reasoning*, **24**(1–2), 145–163, (February 2000).

[5] Sarah Nadi, *Variability Anomalies in Software Product Lines*, Ph.D. dissertation, University of Waterloo, 2014.

[6] David A. Plaisted and Steven Greenbaum, 'A structure-preserving clause form translation', *Journal of Symbolic Computation*, **2**(3), 293–304, (September 1986).

[7] Steven She and Thorsten Berger, 'Formal semantics of the kconfig language', *Technical note, University of Waterloo*, 24, (2010).

[8] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin, 'Formal methods for the validation of automotive product configuration data', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **17**(1), 75–97, (January 2003). Special issue on configuration.

[9] Reinhard Tartler, *Mastering Variability Challenges in Linux and Related Highly-Configurable System Software*, Ph.D. dissertation, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2013.

[10] Rouven Walter, Christoph Zengler, and Wolfgang Küchlin, 'Applications of MaxSAT in automotive configuration', in *Proceedings of the 15th International Configuration Workshop*, eds., Michel Aldanondo and Andreas Falkner, pp. 21–28, Vienna, Austria, (August 2013).

[11] Christoph Zengler and Wolfgang Küchlin, 'Encoding the Linux Kernel Configuration in Propositional Logic', in *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010) Workshop on Configuration*, eds., Lothar Hotz and Alois Haselböck, pp. 51–56, (2010).

How to Analyze and Quantify Similarities between Configured Engineer-To-Order Products by Comparing the Highlighted Features Utilizing the Configuration System Abilities

Sara Shafiee[1], **Lars Hvam**[2], **Katrin Kristjansdottir**[3]

Abstract. Engineering-To-Order (ETO) companies making complex and highly engineered products, face the challenge of delivering highly customized and engineered products with high quality and short delivery time. In order to respond to those challenges ETO companies strive to increase commonality between different projects and to reuse product related information. For that purpose companies need to be able to retrieve previously designed products and identify which parts of the design can be reused and which parts to redesign. This allows companies to reduce complexity in the product range, to decrease the engineering hours and to improve the accuracy of the product specifications. In this article we suggest a framework where product features from the company's configuration system are listed up in order to compare with previously made products by retrieving information from internal ERP/PLM systems. The list of features consists of defining features with potential sets of values e.g. capacity, dimensions, quality of material, energy consumptions, etc. When identifying a specific previously designed product, it allows access to all of the specifications of the existing product along with the engineering hours used, materials used, and hours used in the workshop. The aim of this paper is to make a framework for setting up a database before starting the comparison.

1 INTRODUCTION AND PROBLEM STATEMENT

A configurator supports the user in specifying different features of a product by defining how predefined entities (physical or non-physical) and their properties (fixed or variable) can be combined [1]. Improving the quotation process with the help of configuration systems is a great opportunity for enhancing the presale and production process efficiency in the companies [2]. There are several benefits that can be gained from utilizing product configuration systems, such as a shorter lead-time for generating quotation and fewer errors, increased ability to meet customer requirements with regards to functionality and quality of the products, increased customer satisfaction, etc. [3]. Theoretical elaboration of the empirical evidence suggests that, in order to reach all the advantages that can be gained from utilizing product configuration systems, changes in the organization and the supporting systems in the order acquisition and fulfillment process are needed [4]. These issues can be solved by double checking all the outputs generated by the configurator through an automated IT solution. "All designs are redesigns" has long been a popular cliché in design research [5]. More generally it has been observed that in many firms the reuse and generalization of past experiences (often called "lessons learned") is becoming a key factor for the improvement, in time and in quality, of operational processes [6]. It is rational to say that all the attributes of the products and all their relations are available in the configuration system; and for every received order from the customer, changes and specifications for the product are entered into the configuration system. The idea is to make a connection between ERP and the configuration system, when generating quotations in the product configuration systems and compare it with the previous done projects saved in the ERP system from different perspectives. ETO companies producing complex highly engineered products have a significant problem when calculating the prices for the presale and sale processes. That is especially the case when domain experts cannot determine accurate price curves or when vendors are not providing sufficient information to be modeled inside the configurator. Therefore estimates are used or markup factors are added. When underestimating costs in projects the company will lose profit and when overestimating the cost the customer might go elsewhere where he can buy the product at a reasonable price. The accuracy of calculations is affected, as previous projects are not easily accessible and it requires significant work to compare potential new projects with previous projects manually in order to find the relevant information.

Hvam et al. [1] explains this problem by using an example from F.L. Smidth, which is an ETO company selling cement plants. In this example, the company strives to reuse information from previously made projects to calculate the most accurate price based on weight and capacity. According to Hvam et al. [1], the price and weight curves are made by inserting the capacity, price and weight based on information from 3-5 previously produced machines. A curve is then drawn through the points as is demonstrated in

Figure 4. This allows identification of prices and weights for machines that have not previously been produced.

[1] Industrial PhD Student, Management Engineering department, Technical University of Denmark, 2800 Kgs.Lyngby, Denmark, sashaf@dtu.dk

[2] Professor, Centre for Product Modelling (CPM), Department of Management Engineering, Technical University of Denmark, 2800 Kgs.Lyngby, Denmark,lahv@dtu.dk

[3] PhD Student, Management Engineering department, Technical University of Denmark, 2800 Kgs.Lyngby, Denmark, katkr@dtu.dk

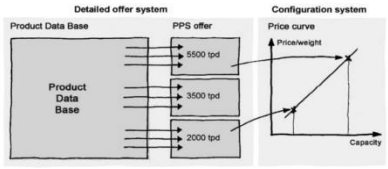


Figure 1. Price and weight curve for main machines in F.L Smith

However, with regards to highly complex products, the price curves are not thought to be the most accurate method as there are several dependent features and great numbers of neighbors on the curve. Another important drawback from the price curves is that the user is only provided access to some of the previously made projects. Therefore the most similar previous projects might be missed.

The first benefits of using an automated IT process, where an integration between the configuration system and the company's internal ERP in order to get access to previously saved project information, is to avoid time consuming redesigning activities in the production phase. This means that it will be possible to produce the same component or product while spending the least possible time and resources.

Salvador and Forza [7] offer much anecdotal evidence of the issues related to product configuration systems. These are listed in terms of: excessive errors, too long time between sales and installation due to inadequate product information supply to the sales office, an excess of repetitive activities within the technical office, and a high rate of configuration errors in production. Even if there are often concerns regarding product configuration projects and the possible errors in the early phases of deploying the systems, the confirmation of the configuration system is not the only benefit from the mentioned solution.

Salvador and Forza [7] describe product configuration systems as aid systems for the end users or customers for creating communication value. Comparing the new project with previous ones could also turn into a recommendation system in the companies. Felfernig [8] discusses different recommendation systems that are divided into Collaborative Filtering (CF), Content Based Filtering (CBF) and Knowledge Based Recommendations (KBR). The available recommendation technologies in e-commerce are potentially useful in helping customers to choose the optimal products configuration [9]. It seems that the mentioned idea is similar to the values that come from recommendation systems. This means that if a 95% similarity between the current project and a previous project is found, the previous project can be re-used and thereby cost related to making the product specification significantly reduced. This includes costs in the sales phase, engineering and production. Furthermore, this is likely to improve the quality and the accuracy of the cost estimations. It also makes it easy to reach an agreement with the customer, and to recommend to them a consultancy to confirm the success of the project by small changes in the order.

Furthermore, this approach enables companies to analyze the products statistically for future product development. Using the configuration systems and comparing different orders can provide valuable information to managers, as it helps them to keep track of product features and to get an overview of market demands. This helps companies to be more in control over the product assortment and eliminates the complexity related to the diversity of product features offered in the production line.

Modular architecture is a term that usually refers to the construction of a building from different instances of standardized components, and in manufacturing it is used for interchangeable units that are used to create the product variants [10]. Dahmus et al. [11] defines a Modularity Matrix to find the similarities between product platforms across columns for a single function in the matrix. Thereafter, architecting of the product portfolio is recommended to take advantage of possible commonalities through the reuse of modules across different product families. If an existing product has standardized and decoupled interfaces, the design of the next product can re-use heavily from the components of the previous product. Holmqvist [12] identifies existing modularization methods and analyses them with regards to their ability to deal with different degrees of product complexity. Based on that he proves that modularization methods are really useful for a simple product architecture but for higher degrees of product complexity, when several functions are allocated to several physical modules, or large variation of variants, these methods seem inefficient [12]. Zamirowski et al. [13] presents three additional heuristics to find common modules across products in a product family. By knowing the previously ordered products, there will be the opportunity of decoupling of design and production tasks.

The potential benefits that can be gained from using the comparison capabilities between configuration systems and other databases at the companies are summarized in Table 1.

Table 1. Benefits from reusing the previous projects

Area	**Benefits**
Management	1. Lean management by avoiding all the presales, production and sales activity that have been performed before.
Configuration system development	2. Reducing errors and increasing reliability of the configuration system.
	3. Facilitating the testing process for the configuration systems development.
Standardization, Product planning, Configuration system	4. Recommending previously successful projects to the end users.
	5. Basis for product standardization.
Product planning, management	6. Statistical approach to the information and market requirements of the product.
Product planning, Configuration system	7. Improve the quality of the configuration system, lead time, manufacturing, sales engineering.

Inakoshi et al. [14] propose a framework for product configuration that integrates a constraint satisfaction problem with a Case-based Reasoning tool (CBR), where the framework is applied to an on-line sales system. This framework contains the following steps:

1. **Case retrieval:** similar cases are retrieved from the case base in accordance with the similarities between the current query and the cases.
2. **Requirement formalization:** a well-defined requirement consists of the current query and the object function, and it is supplied to **a** Constraint Satisfaction Problem (CSP) solver.
3. **Requirement modification:** The well-defined requirement is modified only if there is no configuration and the CSP solver returns no solution back to the CBR Wrapper.

4. **Parts database:** a parts database that contains the definition of a product family. It defines the types of parts, the constraints on parts connectivity, and other kinds of restrictions on the products.
5. **CSP solver:** The CSP solver receives a well-defined user requirement and solves the problem.

The physical structure of the configuration system is illustrated in Figure 2.

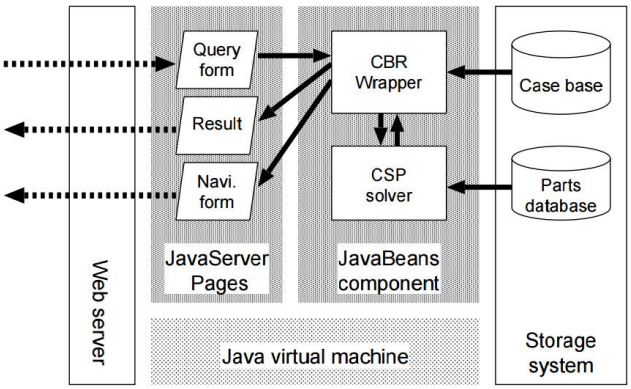


Figure 2. Physical architecture of configuration system [14]

This research work has been used as an inspiration for creating a database development framework and then doing a comparison and integrating it with the product configuration system. There is no discussion in detail on how to make a database from the ERP system, where all the previous projects are stored.

2 RESEARCH METHOD

In accordance with the overall objective, the first phase is focused on the development of the framework, devoted to selecting a framework for product configuration, which integrates a constraint satisfaction problem with Case-Based Reasoning tool (CBR) from previous literature.
The framework development is an ongoing research project to be developed further and tested by a group of researchers and practitioners with an applied research background in modelling products, product architecture, knowledge engineering and product configuration, software development, combining traditional domains of mechanical engineering with product configuration and software development. The framework will be tested in an ETO company specializing in production of catalysts.

3 SUGGESTED METHOD FOR IDENTIFICATIO AND COMPARISON BETWEEN PRODUCT FEATURES

Previous researchers define different tools and methods to measure the similarities between product features. Using configuration systems and techniques for comparing products, it is possible to compare different product features that have been ordered with the new coming orders. One of the prerequirements for using the automatic comparison is to have product configuration system in the sales process. The scenario is to use product features in the configuration system to compare with all the previously generated quotations, which are documented in a desired database. In Figure 3 the process needed for the comparison accomplishment is illustrated.

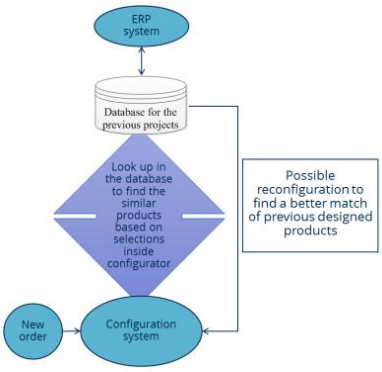


Figure 3. The process of comparing and find similar products

3.1 Set up of the database with previous projects, comparing the configured products with the previously designed products

Inakoshi et al. [14] introduce a framework for comparing a product configuration that integrates a constraint satisfaction problem with a Case-Based Reasoning tool (CBR) for a specific case and with specific tool. In this paper the aim is to make a framework in order to create a database for the comparison, which allows the comparison to be done in a standardized way where the currently available tools and methods can be utilized. Based on literature a seven step framework has been developed, the individual steps are illustrated in Figure 4. The process is not a complete waterfall process, as it is necessary to iterate some of the steps depending on the product.

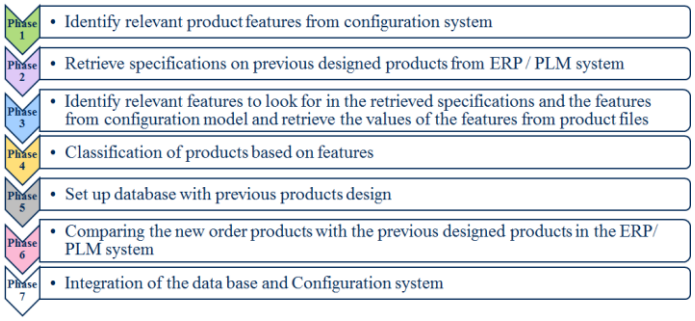


Figure 4. Database set up process in 7 phases

3.1.1 Identify relevant features according to features from configuration model

Previous research that describes how to use modules across different products [13] [15] will be used in order to compare different products. Commonality is best obtained by minimizing the non-value added variations across the products within the same product family without limiting the choices for the customers [16]. According to Ulrich [10], if an existing product has standardized and decoupled interfaces, the design of the next product can

borrow heavily from the components of the previous product. Thevenot and Simpson [16] discuss a framework where commonality indices are used for redesigning the product families to align with cost reductions in the product development process aligned with the standardized and modularized product structure incorporated to the configuration system, makes it easier to pick the relevant features or add them to the configurator.
E. Lopez-Herrejon et al. [17] introduce Software Product Line Engineering (SPLE) to represent the combinations of features that distinguish the system variants using feature models.

3.1.2 Retrieve specifications on previous designed products from ERP / PLM system

The current generation of database systems is designed mainly to support business applications and most of them offer discovery features using tree inducers, neural nets, and rule discovery algorithms [18]. One of the fundamental problems of information extraction from ERP systems is that the formats of available data sources are often incompatible, requiring extensive conversion efforts [19]. Knowledge discovery in databases represent the process for transformation of available data into strategic information, which is characterized by issues related to the nature of data and desired features [20] [21]. Brachman et al. [22] define Knowledge Discovery (KD) process elements to be in three steps:

1. Task discovery, data discovery, data cleansing, data segmentation
2. Model selection, parameter selection, model specification, model fitting
3. Model evaluation, model refinement, output evaluation

KD has a variety of meanings. It includes, at one end the derivation of useful information from a database like "which products are needed for the specific amount of engineering hours for installation?" [23].

3.1.3 Retrieve features from product files and determining the values

Most companies use the old technique called "British classification" when naming different components according to the product variants. However as the products get more complicated this technique becomes impractical. In this technique, as shown in Figure 5, there is a "surname" of five digits it is the general class of an item and the "Christian name" of three digits for an exact identity of for the particular item [24].

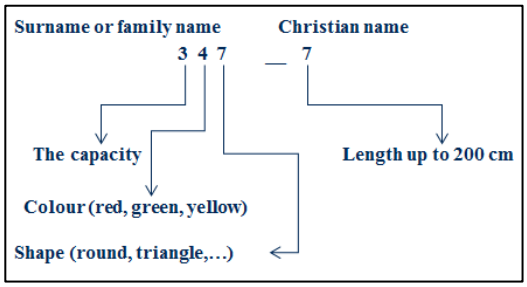


Figure 5. Expansion of a major class [24]

This technique could be used for finding the projects or products with the same specification but for a high level of similarities. This could help us to identify some of the product features and then search for their range of values.

3.1.4 Classifying the products based on features

For identifying and classifying relevant features in order to make a database, classification techniques are required. Burbidge describes how to classify the needs for the product components and coding them by introducing the Group Technology (GT) method [24]. Martinez et al. [25] then use the GT technique as a base for developing a new GT method [25] they provide an example where the GT technique is used in manufacturing plant where it help in the processes of minimizing unnecessary variety by making designers aware of existing components [24]. The aim of classification and coding is to provide an efficient method of information retrieval for decision making. To be efficient enough a code must be designed for the particular purpose for which it will be used [24]. Leukel et al. [26] discuss the design and components of product classification systems in B2B e-commerce and suggested a data model based on XML. Fairchild [27] discuss the application of classification systems and the requirements on them. Simpson [28] uses GT for adding, removing, or substituting one or more modules to the product platform for product platform design and customization. Sousa et al. [29] suggest an automated classification system for specialization of life cycle assessment. First of all they manage to have a conceptual framework for environmental performance of product concepts. Then, the hierarchical clustering has been used in several applications to show useful ways of grouping objects according to their similarities and product descriptors data. Finally, it is used to develop an automated classification system based on decision trees algorithms. Sousa et al. [29] also use Matlab and C4.5 decision tree algorithm, which seems to be applicable in all classification cases. C4.5 is an algorithm used to generate a classification in form of a decision tree that is either a leaf indicating a class or a decision node that specifies some test to be carried out on a single attribute value. This algorithm has a few base cases as below [30]:

1. All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.
2. None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class.
3. Instance of previously unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

Ho [21] introduces OSHAM system generated in hierarchical graphical browser which is competing with C4.5.

Magali and Geneste [6] propose object oriented modeling language, Unified Modeling Language (UML) as a standard modelling of domain knowledge for their research work to represent field data. The exploitation of the object modeling as an indexing base is suggested to allow a fast selection of potentially interesting objects during the similar case search [6].

Guillaume et al. [31] developed six heuristics for clustering and weighting the logical, syntactical and semantical relationships between feature names. The other representation introduced as the so-called Product Comparison Matrices (PCMs), can help to make a choice, where the aim is to visualize all the products characteristics through a metrical representation, [32].

3.1.5 Set up database with previous products design

Ramakrishnan et al. [33] give an overview of database design in the following three steps:

1. Requirement analysis: Understanding of what data is to be stored in the database, what applications must be built on top of it, what operations are most frequent and subject to performance requirements.
2. Conceptual database design: The information gathered in the requirements analysis step is used to develop a high-level description of the data along with the constraints to be stored in the database.
3. Logical database design: Database Management System (DBMS) has to be chosen to implement the database design, and convert the conceptual database design into a database schema in the data model of the chosen DBMS.

3.1.6 Comparing the new order products with the previous designed products in the ERP/ PLM system

There are extensive research works in the field of IT illustrating different methods to do the comparison in an automated way. Classical Case-based Reasoning tool (CBR) methodologies [34] [35] are based on four tasks, which are: Retrieve, Reuse, Revise and Retain are highly used for this purpose.

Navinchandra's [36] developed CYCLOPS, which was the first system to explore CBR in interactive design. Vareilles et al. [37] proposed an approach to use 'contextual knowledge corresponding to past cases' and 'general knowledge corresponding to relations, rules or constraints that link design variables'. In this research, Constraint Satisfaction Problem (CSP) is used regarding general knowledge and CBR operates with conceptual knowledge.

Magali and Geneste [6] propose a method to define the neighborhood of the retrieved case to propagate domain constraints. In this method they use Fuzzy Search is divided in two steps that are: rough filtering process and similarity measuring.

Coudert et al. [38] suggest an integrated case-based approach by using ontology of concepts for guiding project planning and system design processes.

3.1.7 Integration of the database with the product configuration system

According to Inakoshi [14], there is the possibility to integrate a constraint satisfaction problem with CBR for a product configuration system.

4. PLAN FOR THE CASE STUDY

The case study is planned based on a group of researchers from the Technical University of Denmark in collaboration with Haldor Topsoe. The aim is to test and make further developments to the proposed framework. The case study should aim to find the major and minor drawbacks in the current framework and refines it based on experiment. The main things that will be tested in the case study are listed below:

1. Can we retrieve the products' features out of the ERP system?
2. Can we classify the products?
3. Can we make a data base according to the product features?
4. How to do the comparison between the new product and the previous designed products?
5. How to integrate the data base and configuration systems? How to make the user interface in the configuration system?

5. CONCLUSION

In this paper we suggest an approach for comparing a new order that is being configured with previous made configurations, which are usually stored in various internal systems at the companies. This will lead to some advantages such as increased commonality across different products and reuse of modules across the family of products. To achieve the goal of comparing different products a database for the necessary features is needed. The proposed approach includes 7 separate phases. Finally after the database setup, the comparison method based on literature will be accomplished and the integration between the configuration system and database will be performed. The paper is just mentioning a problem realized as one of the configuration system drawbacks and suggests a framework for using comparison method to solve this problem. To have a generic framework to retrieve data from ERP/ PLM systems and compare them in configuration projects further research work is required as listed below:

1. Framework testing for a case study and test the available tools for retrieving and comparing the features.
2. Development of the possible ways to integrate database with product configuration system.

6. REFERENCES

[1] L. Hvam, N.H. Mortensen and J. Riis, *Product Customization*, Springer, Berlin, 2008.

[2] A. Felfernig, L. Hotz, C. Bagley and J. Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Morgan Kaufman, 2014.

[3] L. Hvam, S. Pape and M.K. Nielsen, "Improving the quotation process with product configuration," *Computers in Induustry*, 607-621, (2006).

[4] D.L. McGuinness and J.R. Wright, "Conceptual modelling for configuration: A description logic-based approach," *AI EDAM*, 12, 04, 333-344, (1998).

[5] A.K. Goel and S. Craw, "Design, innovation and case-based reasoning," *The Knowledge Engineering Review*, 271-276, (2005).

[6] R. Magali and L. Geneste, "Search and adaptation in a fuzzy object

oriented case base," *Advances in Case-Based Reasoning*, Springer, Berlin, 350-364, 2002.

[7] F. Salvador and C. Forza, *Product Information Management for Mass Customization: Connecting Customer, Front-office and Back-office for Fast and Efficient Customization*, New York: Palgrave Macmillan, 2007.

[8] A. Felfernig, L. Hotz, J. Tiihonen and C. Bagley, "Configuration-Related Topics.", in *Knowledge-based configuration: From research to business cases*, Morgan Kaufmann",. 21-28, 2014.

[9] L.L. Zhang, "Product configuration: a review of the state-of-the-art and future research," *International Journal of Production Research*, 52, 21, 6381-6398, (2014).

[10] H. Ulrich, "Fundamentals of product modularity", in "*Management of Design: Engineering and Management Perspectives*, Atlanta, GA, Springer, 219-231, (1994).

[11] J.B. Dahmus, J. P. Gonzalez-Zugasti and K. N. & Otto, "Modular product architecture," *Design studies*, 22, 5, 409-424, (2001).

[12] T.K. Holmqvist and M. L. Persson, "Analysis and improvement of product modularization methods: Their ability to deal with complex products.," *Systems Engineering*, 6,. 3, 195-209, (2003)

[13] E.J. Zamirowski and K. N. Otto, "Identifying product family architecture modularity using function and variety heuristics," in *11th International Conference on Design Theory and Methodology, ASME*, Las Vegas, (1999).

[14] H. Inakoshi, S. Okamoto, Y. Ohta and N. Yugami, "Effective decision support for product configuration by using CBR," in *Fourth International Conference on Case-Based Reasoning (ICCBR), Workshop Casebased Reasoning in Electronic Commerce*, Vancouver, Canada, 2001.

[15] A. Ericsson and G. Erixon, *Controlling design variants: modular product platform*, Society of Manufacturing Engineers, 1999.

[16] H. J. Thevenot and T. W. Simpson, "Commonality indices for product family design: a detailed comparison.," *Journal of Engineering Design*, 17, 2, 99-119, (2006).

[17] R.E. Lopez-Herrejon, L. Linsbauer, J.A. Galindo, J.A. Parejo, D. Benavides, S. Segura and A. Egyed, "An assessment of search-based techniques for reverse engineering feature models," *Journal of Systems and Software*, 103, 353-369, (2015).

[18] T. Imielinski and H. Mannila, "A database perspective on knowledge discovery," *Communications of the ACM*, 39, 11, 58-64, (1996).

[19] E. Bendoly, "Theory and support for process frameworks of knowledge discovery and data mining from ERP systems," *Information & Management*,. 40, 7, 639-647, (2003).

[20] R. Davies, "The creation of new knowledge by information retrieval and classification," *Journal of Documentation*, 45, 273–301, (1989).

[21] T. B. Ho, "Discovering and using knowledge from unsupervised data" *Decision Support Systems*, 21, 29–42, (1997).

[22] R. J. Brachman and T. Anand, "The process of knowledge discovery in databases," in *In Advances in knowledge discovery and data mining*, CA, (1996).

[23] F. H. Grupe, "Using domain knowledge to guide database knowledge discovery," *Expert Systems With Applications*, 10, 2, 173-180, (1996).

[24] J. L. Burbidge, *The introduction of group technology*, London: Heinemann, 1975.

[25] M. Martinez, J. Favrel and P. Ghodous, "Product Family Manufacturing Plan Generation and Classification," *Concurrent Engineering: Research & Applications*, 8,. 1, 12–23, (2000).

[26] J. Leukel, V. Schmitz and F. D. Dorloff, "A modeling approach for product classification systems," in *13th International Workshop on Database and Expert Systems Applications*, (2002).

[27] A. M. Fairchild and B. de Vuyst, "Coding standards benefiting product and service information in e-commerce," in *35th Annual Hawaii International Conference on System Sciences* , (2002).

[28] T. W. Simpson, "Product platform design and customization: Status and promise," *AI EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 18, no. 01, pp. 3-20, 2004.

[29] I. Sousa and D. Wallace, "Product classification to support approximate life-cycle assessment of design concepts," *Technological Forecasting and Social Change*, 73, 3, 186-189, (1980)..

[30] J. R. Quinlan, *C4. 5: programs for machine learning*, Morgan Kaufmann Publishers, 1993.

[31] G. Bécan, M. Acher, B. Baudry and S.B. Nasr, "Breathing ontological knowledge into feature model synthesis: an empirical study," *Empirical Software Engineering*, 1-48, (2015).

[32] G. Bécan, N. Sannier, M. Acher, O. Barais, A. Blouin and B. Baudry, "Automating the formalization of product comparison matrices," in *29th ACM/IEEE international conference on Automated software engineering*, 433-444, 2014.

[33] R. Ramakrishnan and J. Gehrke, *Database management systems*, Osborne: McGraw-Hill, 2000.

[34] Y. Avramenko and A. Kraslawski, "Similarity concept for case-based design in process engineering," *Computers & chemical engineering*, 30, 548-557, (2006).

[35] F. Grupe, R. Urwiler, N. Ramarapu and M. Owrang, "The application of case-based reasoning to the software development process," *Information and Software Technology*,. 40, 493–499, (1998).

[36] D. Navinchandra, "Exploration and innovation in design: towards a computational model", Springer Science & Business Media, New York, 2012.

[37] E. Vareilles, M. Aldanondo, A. C. De Boisse, T. Coudert, P. Gaborit and L. Geneste, "How to take into account general and contextual knowledge for interactive aiding design: Towards the coupling of CSP and CBR approaches," *Engineering Applications of Artificial Intelligence*, 25, 31-47, (2012).

[38] T. Coudert, E. Vareilles, L. Geneste, M. Aldanondo and J. Abeille, "Proposal for an integrated case based project planning and system design process," in *2nd International Conference en Complex Systems Design and Management, CSDM*, 2011.

[39] H. Fargier, J. Lang and T. Schiex, "Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge," *AAAI*, 1, (1996).

[40] A. K. Goel and S. Craw, "Design, innovation and case-based reasoning," *The Knowledge Engineering Review*, 20, 271-276, (2006).

[41] V. R. Basili and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEE transactions of software engineering*, 10, 728-738, (1984).

[42] A. Haug, "Representation of Industrial Knowledge – As a Basis for Developing and Maintaining Product Configurators," Technical University of Denmark, Lyngby, 2008.

[43] S. Shafiee and L. Hvam, "An agile documentation system for highly

engineered, complex product configuration systems," 22nd EurOMA Conference, NEUCHÂTEL, SWITZERLAND, 2015.

[44] P. Kruchten, The Rational Unified Process: An Introduction, New York: Addison-Wesley, 1998.