Dissertation

Knowledge-based Technologies for Wiki Environments

Stefan Reiterer

Graz, 2015

Institute for Software Technology
Graz University of Technology

Supervisor/First reviewer: Univ.-Prof. Dipl.-Ing. Dr. techn. Alexander Felfernig
Second reviewer: Assoc.Prof. David Benavides, Ph.D.

Abstract (English)

Knowledge-based recommendation and configuration systems support users in identifying relevant solutions from large solution spaces. In this context users enter requirements and, with respect to the constraints of a knowledge base, the solution space is restricted to a subset of solutions. Such systems are applicable in different domains such as consulting-intensive products, services, and industrial production. In the context of knowledge-based systems, this thesis introduces approaches to support the efficient creation of knowledge bases as well as the efficient handling of inconsistent user requirements. It is shown how diagnosis algorithms can be personalized and how these algorithms can be applied to support users (a) in the development of knowledge bases and (b) when interacting with recommender or configuration systems.

The process of acquiring knowledge for building knowledge bases is complex and time-consuming - this phenomenon is called the *Knowledge Acquisition Bottleneck*. This thesis introduces techniques to reduce the knowledge acquisition bottleneck: On the one hand by supporting users in tackling challenges that arise during the knowledge acquisition process by applying model-based diagnosis concepts. On the other hand by showing how to exploit wiki technologies for community-based knowledge base construction. The MediaWiki extension WEEVIS that was developed in the scope of this thesis allows users to embed knowledge-based recommenders into wiki pages.

During the interaction with knowledge-based systems, inconsistencies between the user requirements and the knowledge base can occur. Furthermore, inconsistencies of constraints with test cases and redundancies can arise during the creation of knowledge bases. The aim of this work is to help users of knowledge-based systems such as WEEVIS in such situations. For this purpose, we demonstrate how diagnosis techniques can be exploited to support knowledge engineers in collaboratively developing knowledge bases as well as to support users when interacting with recommender and configuration systems.

Abstract (German)

Wissensbasierte Empfehlungs- und Konfigurationssysteme unterstützen Benutzer beim Finden von relevanten Lösungen in großen Lösungsräumen. In diesem Kontext können Benutzer ihre Anforderungen eingeben und unter Beachtung der in einer Wissensbasis festgelegten Regeln wird der Lösungsraum auf eine Untermenge von Lösungen eingeschränkt. Derartige Systeme sind in verschiedenen Domänen wie zum Beispiel für beratungsintensive Produkte und Services sowie in der industriellen Fertigung einsetzbar. Im Kontext von wissensbasierten Systemen zeigt diese Arbeit Ansätze zur effizienten Erstellung von Wissensbasen, sowie zum effizienten Umgang mit inkonsistenten Kundenanforderungen. Es wird gezeigt, wie Diagnosealgorithmen personalisiert und zur Unterstützung von Benutzern (a) während der Entwicklung von Wissensbasen und (b) während der Interaktion mit einem Empfehlungs- oder Konfigurationssystem verwendet werden können.

Das Erstellen von Wissensbasen ist eine komplexe und zeitaufwändige Tätigkeit - dieses Phänomen wird auch *Knowledge Acquisition Bottleneck* genannt. In dieser Arbeit werden Techniken präsentiert mit welchen der Knowledge Acquisition Bottleneck reduziert werden kann: Zum einen werden Benutzer aktiv bei Herausforderungen, die im Kontext der Wissensakquisition entstehen durch das Anwenden von modellbasierter Diagnose unterstützt. Zum anderen wird eine Community von Benutzern herangezogen, um auf Basis von wiki Technologien Wissensbasen zu bauen. Dazu wurde im Rahmen dieser Arbeit WEEVIS entwickelt, eine MediaWiki Erweiterung, die es ermöglicht, wissensbasierte Empfehlungs- und Konfigurationssysteme in wiki Seiten einzubetten.

Beim Verwenden von wissensbasierten Systemen können beispielsweise Inkonsistenzen zwischen Anforderungen des Benutzers und der Wissensbasis auftreten. Weiters kommen bereits während der Erzeugung von Wissensbasen Inkonsistenzen mit Testfällen sowie Redundanzen vor. Ziel dieser Arbeit ist es, Benutzer von wissensbasierten Systemen wie zum Beispiel WEEVIS in derartigen Situationen zu unterstützen. Dazu zeigen wir, wie Diagnosetechnologien eingesetzt werden können, um *Knowledge Engineers* bei der kollaborativen Entwicklung von Wissensbasen zu helfen und um Benutzer während der Interaktion mit Empfehlungs- und Konfigurationssystemen zu unterstützen.

Acknowledgement

First and foremost I want to thank my supervisor Univ.-Prof. Dipl.-Ing. Dr.techn. Alexander Felfernig for the continuous support, his motivation and his prolific suggestions during the origin of this thesis. I also want to gratefully acknowledge the support of my colleagues Dipl.-Ing. Martin Stettinger, Michael Jeran, Dipl.-Ing. Gerald Ninaus, Dipl.-Ing Florian Reinfrank, Dipl.-Ing. Dr.techn. Monika Mandl, Dipl.-Ing. Dr.techn. Monika Schubert, Dr. Paul Blazek, Dipl.-Ing. Manfred Wundara, FH-Prof. Dr. Wolfgang Eixelsberger, Dipl.-Ing. Klaus Isak and Assoc.Prof. Mag. Dr. Gerhard Leitner.

Lastly, I would like to thank my family who supported me during the years of study. Especially I want to thank my father Ing. Werner Reiterer without whom I may have never stepped into the field of computer science.

Stefan Reiterer
Graz, 2015

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, ______________	______________
Place, Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ______________	______________
Ort, Datum	Unterschrift

Contents

Chapter 1

Introduction

Nowadays, users of online stores have become accustomed to a list of items entitled *Products that may be of interest to you* .... This list represents a personalized selection of items from an assortment of millions of products offered, for example, on Amazon.com[*]. The filtering of these items can be done by different kinds of recommender systems. Nowadays, recommender systems have become indispensable for online stores (see Jannach et al. (2010)). In 2009 the streaming platform Netflix[†] awarded one million dollars to dedicated researchers in a contest[‡] with the goal to improve the prediction quality of the Netflix recommender system by 10%. The high prize money and the fact that about three-fourths of the films sold by Netflix are a direct result of recommendations (Mayer-Schönberger and Cukier (2013)) underlines the importance of recommender systems today.

The aforementioned recommendation of *Products that may be of interest to you...* relies for many online stores on a *collaborative filtering* strategy (Schafer et al. (2007) and Goldberg et al. (1992)). The Netflix online store, for example, offers a portfolio of film streams. For collaboratively recommending a film to a user, let's give her the name *Alice*, the following steps are necessary: First, users (nearest neighbors), who have similar preferences to Alice must be identified. Based on the preferences of these users, a prediction of the preference of candidate films can be calculated. Another approach for recommending items is *content-based filtering*. Content-based recommenders (see e.g. Pazzani and Billsus (2007)) identify similar products by analyzing and comparing the meta data of other products to those of a given one. A basic goal of this type of recommendation is to recommend *more of the same*. Both strategies tend to have ramp up problems (Felfernig et al. (2014a)): Collaborative filtering is only applicable if each user has rated enough items. For applying content-based filtering, users must have selected (or purchased) at least one item.

Collaborative- and content-based filtering are applicable to simple items but are not directly ap-

[*]http://www.amazon.com
[†]http://www.netflix.com
[‡]http://www.netflixprize.com

plicable to the recommendation of complex products and services (Felfernig and Burke (2008)). To provide recommendation functionality for complex item domains such as financial services (Felfernig et al. (2007a)), diet plans (Reiterer et al. (2015b)), or digital cameras a different recommendation approach is needed. *Knowledge-based recommenders* (Burke (2000), Felfernig et al. (2006a)) provide dialog-based user interfaces for customers to specify their requirements. After entering the requirements, the system identifies items that fit the customers needs. This approach requires a knowledge base that describes the relation between items, their attributes, and the articulated requirements. The resulting solutions have to fulfill the users needs as well as to respect the constraints defined in the knowledge base. Knowledge-based recommenders help users to find the best fitting product within a large solution space.

In contrast to knowledge-based recommenders, where users receive recommendations from a list of predefined products, knowledge-based configuration systems (see Felfernig et al. (2014b) and Stumptner (1997)) help users building a *custom product* based on components and corresponding attributes. The different components fit together according to rules represented in a knowledge base. A configuration knowledge-base should be used to represent an item set if the amount of valid solutions is hardly enumerable, otherwise a recommendation knowledge base in terms of a list of products can be chosen (Falkner et al. (2011)). According to the constraints of the knowledge base and the user requirements, a constraint solver (like Choco[§] or JaCoP[¶]) is able to determine a list of possible configurations. Knowledge-based configuration is a key tool to enable mass customization (Aldanondo and Vareilles (2008) and Felfernig et al. (2014b)).

Knowledge-based configuration and recommendation systems can be considered from two different angles: On the one hand there is the *creation* process of the knowledge base, where knowledge engineers and domain experts are involved in creating the back end of the system. On the other hand there is the *consumption* part, where the end user engages with the system which is either used to identify solutions or to check the consistency of a given configuration.

Wiki environments such as MediaWiki[‖] can also be considered from these two angles: On the one hand wikis enable users to collaboratively *build* wiki pages (see Scardamalia and Bereiter (2003)). On the other hand the created pages are *consumed* for individual learning (see Kimmerle et al. (2009)). Knowledge building takes place when users extend/revise the content of a wiki page that was created by other users. Individual learning on the other hand takes place when a user acquires knowledge from a wiki page. The most prominent application of MediaWiki is Wikipedia[**], the most popular general reference work (with nearly five million wiki pages). Wikis are used for knowledge management and are applied in different domains. Each wiki page can be seen as a natural language knowledge base. A wiki page can be *created* and *edited* as well as *consumed* by users of the wiki system. The wiki

[§]http://choco-solver.org
[¶]http://www.jacop.eu
[‖]http://www.mediawiki.org
[**]http://www.wikipedia.org

environment is simple enough for a *domain expert* to structure a wiki page and fill it with knowledge. The wiki syntax is easy to learn and to understand which is the reason why many people all over the world have become contributors to Wikipedia.

In the line of the wiki principles, recommender and configuration knowledge bases can be created the same way as wiki pages. With a simple description language for knowledge bases, a community of *domain experts* can create and maintain knowledge bases within wiki pages. Considering the consumption of the created knowledge bases, an interactive user interface that guides users to the solution based on the stated requirements can be integrated into the wiki environment. With the inclusion of knowledge bases, natural language wiki pages can be enhanced by recommendation and configuration functionalities.

The guidance provided by knowledge-based systems can be taken even one step further by enriching them with the concept of model-based diagnosis (see Felfernig et al. (2004) and Reiter (1987)). Diagnosis algorithms are able to support users during the recommendation (or configuration) process if a situation occurs where the system is unable to come up with solutions for the given requirements. The situation is also known as the *no solution could be found dilemma*. In this case diagnosis algorithms such as FASTDIAG (see Felfernig et al. (2012a)) can be used to calculate diagnoses for the presentation of alternative solutions.

The remainder of this chapter is organized as follows: Section 1.1 motivates the research presented in this thesis. In Section 1.2 a detailed explanation of each research objective is presented. Section 1.3 contains the contributions to each of the aforementioned research objectives and Section 1.4 provides an outline of the following chapters.

1.1. Motivation

Knowledge-based configuration systems are applied in many domains for the purpose of supporting mass customization (Felfernig et al. (2014b)). With a configuration system a set of predefined components can be composed in such a way that user requirements as well as the constraints of a knowledge base are taken into account. A knowledge-based recommender system (Burke (2000)) relies on an assortment of explicitly defined solutions (items) that can easily be enumerated - in contrast to a configuration system that can deal with solution spaces that can't be represented explicitly in an efficient fashion. In the configuration process as well as in the recommendation process, user requirements and the knowledge base restrict the number of possible solutions (items) (see Falkner et al. (2011)). These systems support customers and companies by offering an interactive access to the background knowledge related to products and services. Model-based diagnosis (Reiter (1987)) improves the applicability of these systems by supporting users in the engineering process of knowledge bases as well as end users in the application of the recommender or configuration system (Felfernig et al. (2014b)). *The motivation for this thesis is to help users in building and using knowledge bases with the sup-*

port of personalized diagnosis algorithms on the basis of wiki environments. In this context two main challenges had to be tackled:

1. How to exploit a wiki environment for engineering configuration and recommendation knowledge?

2. How to support (a) end users during the interaction with knowledge bases and (b) domain experts during knowledge base development, both in a personalized fashion?

1. Expoiting Wikis for engineering configuration and recommendation knowledge. As a consequence of the rise of Web 2.0, social collaboration tools such as wikis emerged (see Cress and Kimmerle (2008)). Since Wikipedia became the worlds largest encyclopedia, nearly every world wide web user used the MediaWiki environment, the platform that serves as basis for Wikipedia. In the last decade many different wikis became popular, for example, DokuWiki[††], a wiki derivative especially useful for companies to serve as an internal knowledge base. Therefore the term *Enterprise Wiki* evolved as an umbrella term for wikis used for knowledge management in the intra-net of companies. The main goal of each wiki is to serve as a platform for knowledge exchange within a community. Domain experts can share their knowledge by creating an article (wiki page). Wikis are especially useful tools for managing knowledge transfer. Every created page acts as a source of knowledge for other users. Wiki platforms provide features to enhance the collaboration of multiple domain experts for the creation and maintenance of wiki pages. For example, wikis provide a forum to discuss contents and the relevance of articles. Mechanisms such as versioning make it easy to manage contents and also change articles without loosing contents.

However, *wikis don't provide the ability to retrieve the content of a page in a personalized and goal-oriented way.* For example, a wiki page that informs users about weight loss diets, such as the low carb diet, the ketogenic diet or others doesn't provide a functionality to support the user in receiving customized diet information from the page. Knowledge-based recommenders (e.g. Felfernig et al. (2006a)) in contrast are designed to support users in identifying items of interest in a personalized and efficient fashion. Available environments for knowledge base development such as the *CWAdvisor* (see Felfernig et al. (2006a)) and configurator environments such as presented in Blumöhr et al. (2010) and Tiihonen et al. (2013) are either too complex for mainstream users or only dedicated to commercial use.

2. Supporting (a) end users during the interaction with knowledge bases and (b) domain experts during knowledge base development, both in a personalized fashion. Model-based diagnosis algorithms (Reiter (1987)) can help to identify explanations for unintended behavior. Diagnoses can be seen as a special kind of explanation for over-constrained problems (O'Sullivan et al. (2007) and Felfernig et al. (2004)). Diagnosis algorithms can be applied in different contexts when *using*, *creating*, and *maintaining* knowledge bases:

[††]https://www.dokuwiki.org/

- During the interaction with knowledge-based recommender and configuration systems users can end up in a so-called *no solution could be found dilemma* (Felfernig et al. (2009b)). This situation occurs if the user requirements are inconsistent with the knowledge base.
- During the *creation* and *maintenance* of a knowledge base. In this phase, anomalies such as redundant constraints or inconsistencies between the knowledge base and test cases can occur (Felfernig et al. (2012b)).

In each of these contexts, model-based diagnosis supports the user by presenting constraints that can be either removed from the knowledge base or repaired. Depending on the size of the knowledge base and the number of faulty constraints, often many different diagnoses are available for resolving inconsistencies. The easiest way to restrict the amount of diagnoses is to calculate only minimal ones (there is no other diagnosis set Δ_1 that is a subset of the minimal diagnosis set Δ (see Felfernig et al. (2004)). But even then, for large knowledge bases it can be a difficult task to identify the best fitting diagnosis from the large set of minimal diagnoses (too many options lead to a phenomenon of *information overload* (Byung-Kwan and Wei-Na (2004))). A way to counteract the *information overload* is to order diagnoses in a way that the user will most likely be able to identify at least one acceptable diagnosis within the top elements of the list. The goal is to improve the *prediction* quality which increases the likelihood of presented diagnoses to be chosen by the user. Due to the fact that identifying *all possible diagnoses* and afterwards selecting the preferred ones is in most of the cases computationally inefficient (Felfernig et al. (2009a)), an alternative approach to the determination of diagnoses is introduced in this thesis.

1.2. Research Objectives

The mentioned challenges in the fields of collaborative knowledge acquisition and personalized diagnosis are a basis for the following research objectives:

1. **Exploiting the *wisdom of the crowds* for knowledge base development.**

   Wiki environments such as Wikipedia are platforms that enable a large author base to collaboratively build wiki pages. A comparison of Wikipedia with the Encyclopedia Britannica, where the source of wisdom are individual experts, demonstrates the power of the so-called *wisdom of the crowds* (see Surowiecki (2005) and Arazy et al. (2006)). Giles (2005) conducted a study where experts peer-reviewed articles from Wikipedia and the Encyclopedia Britannica that shows evidence of the high quality of Wikipedia. In the line of the Wikipedia idea, in every wiki users are collaboratively building on the knowledge of each other (see Scardamalia and Bereiter (2003)). Knowledge-based systems like recommenders and configurators can also be seen as a source of wisdom, but in comparison to wikis the knowledge is formalized. Due to this restriction, knowledge acquisition is more complex: not only domain experts are involved in the process but also knowledge engineers. Most established knowledge acquisition tools are based on the knowledge of individuals (e.g. Felfernig et al. (2007a) and Felfernig et al. (2006a)) and do not provide community support. Since knowledge acquisition using the *wisdom of the crowds* works for natural language wiki pages, the same idea can also be applied to knowledge base construction in the context of wiki pages, consequently the following research question emerges:

*(Q1) How to exploit the **wisdom of the crowds** for building knowledge bases?*

2. **Supporting users during the collaborative knowledge acquisition process in situations with a high cognitive load.**

   Building and maintaining knowledge bases is a very error-prone task. Especially, if many users collaborate to build or maintain the same knowledge base, inconsistencies and redundancies occur (see Reiterer (2015)). In software engineering it's common to use, for example, regression tests to prevent new faults in already tested software segments (see Liggesmeyer (2009)). These faults often occur if small updates or changes are made in code segments especially if many people collaborate in the development process. The same applies to knowledge bases: For example, if a constraint is initially integrated into the knowledge base, the solution set can be influenced accidentally by a new constraint integrated by another user who extends the knowledge base. Thus, development and maintenance of knowledge bases have similar challenges concerning faults. Additionally, knowledge bases can contain redundancies. The problem with redundancies is that they lead to a knowledge base that is unnecessary complex and decreases the execution speed of the underlying reasoning algorithms (see Felfernig et al. (2011a)). Consequently, the intelligent handling of inconsistencies and redundancies is inevitable for collaborative knowledge engineering, which leads to the next research question:

   (Q2) How to support users in managing anomalies (e.g., redundancies and inconsistencies) in knowledge bases during a collaborative knowledge engineering process?

3. **Increasing the performance of identifying alternative solutions in interactive settings.**

   The common way of identifying diagnoses is calculating conflict sets in the first place (Junker (2004)) and afterwards using the hitting set algorithm (Reiter (1987)), which is NP-hard (see Friedrich et al. (1990)) for identifying diagnoses. The basic node expansion strategy for the hitting set algorithm is *breath-first*, which results in the identification of all minimal diagnoses. Felfernig et al. (2009a) describe how to use *best-first* search for identifying preferred diagnoses, again with conflict detection in advance and afterwards determining diagnoses. Felfernig et al. (2012a) introduced the *direct diagnosis* algorithm FASTDIAG which is able to calculate exactly one minimal diagnosis at a time. This approach doesn't rely on conflict detection before determining the diagnosis. Users have very high expectations concerning the response time of user interfaces (see Nielsen (1994)). In WEEVIS the goal is to present not only diagnoses but also alternative products, which means that the system has to identify at least one minimal diagnosis in a first step, possible repair options in a second step, and the resulting alternative solutions in the last step. This scenario leads us to the following research question:

(Q3) How to optimize the performance of calculating diagnoses for interactive scenarios?

4. **Improving the prediction quality of personalized diagnosis as a way out of the no solution could be found dilemma.**

   Knowledge-based systems such as configurators and recommenders support users in constructing their preferences in a dialog-based fashion. Oftentimes during the preference construction phase, inconsistencies between user requirements and the knowledge base occur. Diagnosis algorithms support users in such situations by identifying a set of requirements (the diagnosis) which are the cause for the inconsistencies. Depending on the size of the set of requirements, there are often many diagnoses that can resolve the inconsistencies. The challenge of presenting diagnoses to the user is on the one hand to identify diagnoses which will most likely be accepted by the user and on the other hand to deliver diagnoses fast enough to be suitable for interactive settings. A common way of calculating diagnoses is conflict detection (Junker (2004)) and afterwards using the hitting set approach (Reiter (1987)) to detect minimal diagnoses in a breath-first manner. Using breath-first search is useful for delivering all possible minimal diagnoses but according to the large solution space breath-first strategies will often lead to information overload. The task of identifying a diagnosis by hand can be substituted by the task of automatically selecting the right diagnosis from a large list of possible diagnoses. The challenge of calculating diagnoses in a personalized fashion by integrating multiple personalization strategies into one *ensemble* leads to research Question 4.1. The last research question (4.2) addresses the applicability of the *ensemble-based* approach in interactive settings:

   (Q4.1) How to improve the prediction quality of personalized diagnoses by applying the ensemble-based approach?

   (Q4.2) Is the performance of calculating personalized diagnoses applicable for interactive settings?

1.3. Contributions

This section summarizes the contributions of this thesis. The focus of this thesis is on extending a wiki platform with knowledge-based technologies. The exploitation of wiki functionalities is explained and the extension WEEVIS for integrating knowledge bases into wiki pages is presented. Additionally, integrated diagnosis algorithms support users in resolving inconsistencies when building knowledge bases as well as applying them in configuration and recommender applications. To increase the acceptability of the presented diagnoses, personalization approaches are exploited to identify personalized diagnoses with a high predictive performance.

Research Question	**Contribution**
(Q1) How to exploit the wisdom of the crowds for building knowledge bases?	In this work we present WEEVIS, a MediaWiki-based knowledge acquisition environment (see Felfernig et al. (2014b)). The developed platform can be integrated seamlessly as an extension into the MediaWiki platform. With the extension, each wiki page can be extended by a knowledge base. After the knowledge base was built collaboratively, individual users can receive recommendations on the topic of the wiki page. The abilities of MediaWiki are exploited to support the collaborative knowledge acquisition process.
(Q2) How to support users in managing anomalies (e.g., redundancies and inconsistencies) in knowledge bases during a collaborative knowledge engineering process?	Anomaly management is crucial for efficient knowledge acquisition. In this thesis we focus on handling two types of anomalies in WeeVis (see Reiterer (2015) and Felfernig et al. (2014c)): The identification of inconsistencies and the identification of redundancies (see Felfernig et al. (2011a)) within knowledge bases. It's especially valuable to provide support for detecting these anomalies, because their handling is very time-consuming on the one hand and also leads to high cognitive overloads (Reiterer (2015)).
(Q3) How to optimize the performance of calculating diagnoses for interactive scenarios?	*Direct diagnosis* algorithms such as FASTDIAG (Felfernig et al. (2012a)) boost the performance of diagnosis calculation. However, the more diagnoses are taken into account, the more alternative products have to be calculated (see Reiterer et al. (2015a)). In this work we present a performance analysis of FASTDIAG when the algorithm is integrated with an approach to identify alternative products.

(Q4.1) How to improve the prediction quality of personalized diagnoses by applying the ensemble-based approach?	In this work we propose different personalized diagnosis algorithms for achieving a significant increase of the prediction quality (Felfernig et al. (2013a)). Similarity-based, utility-based, and probability-based prediction approaches are exploited for calculating personalized diagnoses. To demonstrate the results of our developments, we present an evaluation of an ensemble-based diagnosis approach on the basis of two datasets. The evaluation shows the benefits of the presented approach in terms of prediction quality.
(Q4.2) Is the performance of calculating personalized diagnoses applicable for interactive settings?	The PDIAG (Felfernig et al. (2013a)) algorithm presented in this thesis is based on the hitting set algorithm introduced in (Reiter (1987)). Although the algorithm is NP-hard (see Friedrich et al. (1990)), according to our evaluation the personalized diagnosis approach with the best first node expansion strategy can be applied in interactive settings. Furthermore a performance analysis that compares the run-times of the ensemble-based diagnosis and other personalization approaches is presented.

Table 1.1.: Overview of the research objectives and the corresponding contributions of this thesis.

1.4. Thesis Outline

This thesis consists of eight chapters:

Chapter 1 provides an introduction and motivation of the topics of this thesis. The research questions related to the fields of model-based diagnosis, personalization strategies, and collaborative knowledge acquisition are outlined. Finally, the chapter ends with a discussion of related research objectives.

Chapter 2 gives an overview of the topic of recommender systems and different types of recommendation approaches such as collaborative filtering, content-based filtering, knowledge-based recommendation, and group-based recommendation. Related applications are discussed and future research is outlined.

Chapter 3 introduces conflict detection and diagnosis methods for managing anomalies in knowledge bases. The algorithms and strategies in this chapter are key technologies for knowledge-based recommendation and configuration. Based on an example, algorithms for conflict detection and for determining diagnoses are explained in detail as well as their performance is discussed.

In Chapter 4 different strategies for personalized diagnosis are presented. These strategies are necessary for supporting users in finding a way out of the *no solution could be found dilemma* in a personalized fashion. The focus of this chapter is to combine different personalization strategies to one *ensemble-based* approach which is evaluated in terms of performance and prediction quality.

Chapter 5 is about WEEVIS, a wiki-based knowledge engineering platform for integrating knowledge bases into wiki pages. This chapter describes the WEEVIS syntax and advantages and limitations of the knowledge engineering environment.

In Chapter 6, the diagnosis and repair techniques of WEEVIS are presented in detail. The chapter outlines how the WEEVIS environment supports users in case of the *no solution could be found dilemma*.

Chapter 7 shows a practical application of the WEEVIS environment using an example from the e-government domain. The chapter also conducts a performance evaluation of WEEVIS.

With Chapter 8 this thesis is rounded up by a reflection on the research questions and contributions as well as topics for future research such as the parallelization of diagnosis algorithms and adopting further strategies for using the *wisdom of the crowd* for knowledge creation.

Chapter 2

Toward the Next Generation of Recommender Systems: Applications and Research Challenges

This chapter is based on the results documented in Felfernig et al. (2013b).

The author of this thesis contributed
the *analysis of related work* and *wrote major parts of the above mentioned paper.*

2.1. Abstract

Recommender systems are assisting users in the process of identifying items that fulfill their wishes and needs. These systems are successfully applied in different e-commerce settings, for example, to the recommendation of news, movies, music, books, and digital cameras. The major goal of this chapter is to discuss new and upcoming applications of recommendation technologies and to provide an outlook on major characteristics of future technological developments. Based on a literature analysis, we discuss new and upcoming applications in domains such as software engineering, data & knowledge engineering, configurable items, and persuasive technologies. Thereafter we sketch major properties of the next generation of recommendation technologies.

2.2. Introduction

Recommender systems support users in the identification of items that fulfill their wishes and needs. As a research discipline, *recommender systems* has been established in the early 1990's (see, e.g., Goldberg et al. (1992)) and since then has grown enormously in terms of algorithmic developments as well as in terms of deployed applications. A recommender system can be defined as a system that guides users in a personalized way to interesting or useful objects in a large space of possible objects or produces such objects as output (Burke (2000); Konstan and Riedl (2012)). Practical experiences from the successful deployment of recommendation technologies in e-commerce contexts (e.g., *amazon.com* Linden et al. (2003) and *netflix.com* Tuzhilin and Koren (2008)) contributed to the development of recommenders in new application domains. Especially such new and upcoming application domains are within the major focus of this chapter.

On an algorithmic level, there exist four basic recommendation approaches. First, *content-based filtering* (Pazzani and Billsus (1997)) is an information filtering approach where features of items a user liked in the past are exploited for the determination of new recommendations. Content-based filtering recommendation is applied, for example, by *amazon.com* for the recommendation of items which are similar to those that have already been purchased by the user. If a user has already purchased a book related to the *Linux* operating system, new (and similar) books will be recommended to her/him in the future.

Second, *collaborative filtering* is applied to the recommendation of items such as music and movies (Konstan et al. (1997); Koren et al. (2009); Linden et al. (2003)). It is based on the concept of analyzing the preferences of users with a similar item rating behavior. As such, it is a basic implementation of word-of-mouth promotion with the idea that a purchase decision is influenced by the opinions of friends and relatives. For example, if user *Joe* has purchased movies similar to the ones that have been purchased by user *Mary* then *amazon.com* would recommend items to *Joe* which have been purchased by *Mary* but not by *Joe* up to now. The major differences compared to content-based filtering are (a) no need of additional item information (in terms of categories or keywords describing the items) and (b) the need of information about the rating behavior of other users.

Third, high-involvement items such as cars, apartments, and financial services are typically recommended on the basis of *knowledge-based recommendation technologies* (Burke (2000); Felfernig et al. (2006a)). These technologies are based on the idea of exploiting explicitly defined recommendation knowledge (defined in terms of deep recommendation knowledge, e.g., as rules and constraints) for the determination of new recommendations. Rating-based recommendation approaches such as collaborative filtering and content-based filtering are not applicable in this context since items are not purchased very frequently. A consequence of this is that no up-to-date rating data of items is available. Since knowledge-based recommendation approaches rely on explicit knowledge representations the so-called knowledge acquisition bottleneck becomes a major challenge when developing and maintaining such systems (Felfernig et al. (2006a)).

Finally, *group recommendation* (Masthoff (2011)) is applied in situations where there is a need of recommendations dedicated to a group of users, for example, movies to be watched by a group of friends or software requirements to be implemented by a development team. The major difference compared to the afore mentioned recommendation approaches lies in the criteria used for determining recommendations: while in the case of content-based filtering, collaborative filtering, and knowledge-based recommendation the major goal is to identify recommendations that perfectly fit the preferences of the current user, group recommendation approaches have to find ways to satisfy the preferences of a user group.

The major focus of this chapter is to give an overview of new and upcoming applications of recommendation technologies and to provide insights into major requirements regarding the development of the next generation of recommender systems. Our work is based on an analysis of research published in workshops, conferences, and journals summarized in Table 2.1. In this context we do not attempt to provide an in-depth analysis of state-of-the-art recommendation algorithms (Adomavicius and Tuzhilin (2005); Jannach et al. (2010)) and traditional applications of recommendation technologies (Schafer et al. (2011)) but focus on the aspect of new and upcoming applications (Ducheneaut et al. (2009); Konstan and Riedl (2012)). An overview of these applications is given in Table 2.2.

The remainder of this chapter is organized as follows. In Sections 2.3–2.7 we provide an overview of new types of applications of recommendation technologies and give working examples. In Section 2.8 we sketch the upcoming generation of recommender systems as *Personal Assistants* which significantly improve the overall quality of recommendations in terms of better taking into account preferences of users – in this context we discuss major issues for future research.

Table 2.1.: Overview of journals and conferences (including workshops) used as the basis for the literature analysis (papers on recommender system applications 2005–2012).

Journals and Conferences
International Journal of Electronic Commerce (IJEC)
Journal of User Modeling and User-Adapted Interaction (UMUAI)
AI Magazine
IEEE Intelligent Systems
Communications of the ACM
Expert Systems with Applications
ACM Recommender Systems (RecSys)
User Modeling, Adaptation, and Personalization (UMAP)
ACM Symposium on Applied Computing (ACM SAC)
ACM International Conference on Intelligent User Interfaces (IUI)

2.3. Recommender Systems in Software Engineering

Recommender systems can support stakeholders in software projects by efficiently tackling the information overload immanent in software projects (Robillard et al. (2010)). They can provide stakeholder support throughout the whole software development process – examples are the *recommendation of* *methodological knowledge* (Burke and Ramezani (2010); Peischl et al. (2010)), the *recommendation of requirements* (Felfernig et al. (2011b); Mobasher and Cleland-Huang (2011)), and the *recommendation of code* (Cubranic et al. (2005); McCarey et al. (2005)).

Recommendation of Methodological Knowledge. Appropriate method selection is crucial for successful software projects, for example, the waterfall process is only applicable for risk-free types of projects but not applicable for high risk projects. Method recommendation is already applied in the context of different types of development activities such as the domain-dependent recommendation of algorithmic problem solving approaches (Burke and Ramezani (2010)) and the recommendation of appropriate effort estimation methods (Peischl et al. (2010)). These approaches are based on the knowledge-based recommendation paradigm (Burke (2000); Felfernig et al. (2006a)) since recommendations do not rely on the taste of users but on well-defined rules defining the criteria for method selection.

Recommendation of Code. Due to frequently changing development teams, the intelligent support of discovering, locating, and understanding code is crucial for efficient software development (McCarey et al. (2005)). Code recommendation can be applied, for example, in the context of (collaborative) call completion (which API methods from a large set of options are relevant in a certain context?) (McCarey et al. (2005)), recommending relevant example code fragments (which sequence of methods is needed in the current context?) (Holmes et al. (2006)), tailoring the displayed software artifacts to the current development context (Kersten and Murphy (2010)), and classification-based defect prediction (Misirli et al. (2011)). For an in-depth discussion of different types of code recommendation approaches and strategies we refer the reader to (Happel and Maalej (2008); Robillard et al. (2010)).

Table 2.2.: Overview of identified recommender applications not in the mainstream of e-commerce applications (Knowledge-based Recommendation: KBR, Collaborative Filtering: CF, Content-based Recommendation: CBR, Machine Learning: ML, Group Recommendation: GR, Probability-based Recommendation: PR, Data Mining: DM).

Domain	Recommended Items	Recommendation Approach	
Software Engineering	effort estimation methods	KBR	Peischl et al. (2010)
	problem solving approaches	KBR	Burke and Ramezani (2010)
	API call completions	CF	McCarey et al. (2005)
	example code fragments	CBR	Holmes et al. (2006)
	software defects	ML	Misirli et al. (2011)
	requirements prioritizations	GR	Felfernig et al. (2011b)
	software requirements	CF	Mobasher and Cleland-Huang (2011)

Domain	Recommended Items	Recommendation Approach	
Data & Knowledge Engineering	related constraints	KBR	Felfernig et al. (2012b, 2011a)
	explanations (hitting sets)	KBR	Felfernig et al. (2012a)
	constraint rewritings	KBR	Felfernig et al. (2010)
	database queries	CF	Chatzopoulou et al. (2009)
		CBF	Fakhraee and Fotouhi (2011)
Knowledge-based Configuration	relevant features	CF	Felfernig and Burke (2008)
	requirements repairs	CBF	Felfernig et al. (2009a)
Persuasive Technologies	game task complexities	CF	Berkovsky et al. (2010)
	development practices	KBR	Pribik and Felfernig (2012)
Smart Homes	equipment configurations	KBR	Leitner et al. (2012)
	smart home control actions	CF	LeMay et al. (2009d)
People	criminals	PR	Tayebi et al. (2011)
	reviewers	CF	Kapoor et al. (2007)
	physicians	CF	Hoens et al. (2010)
Points of Interest	tourism services	CF	Huang et al. (2010)
	passenger hotspots	PR	Yuan et al. (2012)
Help Services	schools	CBR	Wilson et al. (2009)
	financial services	KBR	Fano and Kurth (2003)
	lifestyles	KBR	Hammer et al. (2010)
	receipes	CBR	Pinxteren et al. (2011)
	health information	CBR	Wiesner and Pfeifer (2010)
Innovation Management	innovation teams	KBR	Brocco and Groh (2009)
	business plans	KBR	Jannach and Bundgaard-Joergensen (2007)
	ideas	DM	Thorleuchter et al. (2010)

Recommendation of Requirements. Requirements engineering is one of the most critical phases of a software development process and poorly implemented requirements engineering is one of the major reasons for project failure (Hofmann and Lehner (2001)). Core requirements engineering activities are elicitation & definition, quality assurance, negotiation, and release planning (Sommerville (2007)). All of these activities can be supported by recommendation technologies, for example, the (collaborative) recommendation of requirements to stakeholders working on similar requirements (Mobasher and Cleland-Huang (2011)) and the group-based recommendation of requirements prioritizations (Felfernig et al. (2011b); Ninaus et al. (2014)).

Example: Content-based Recommendation of Similar Requirements. In the following, we will exemplify the application of *content-based filtering* (Pazzani and Billsus (1997)) in the context of *requirements engineering*. A recommender can support stakeholders, for example, by recommending requirements that have been defined in already completed software projects (requirements reuse) or have been defined by other stakeholders of the same project (redundancy and dependency detection). Table 2.3 provides an overview of requirements defined in a software project. Each requirement req_i is characterized by a *category*, the number of estimated *person days* to implement the requirement, and a textual *description*.

Table 2.3.: Example of a content-based filtering recommendation problem: recommendation of similar requirements based on *category* and/or *keyword* information.

requirement	category	person days	description
req$_1$	database	170	store component configuration in DB
req$_2$	user interface	65	user interface with online help available
req$_3$	database	280	separate tier for DB independence
req$_4$	user interface	40	user interface with corporate identity

If we assume that the stakeholder currently interacting with the requirements engineering environment has already investigated the requirement req_1 (assigned to the category *database*), a content-based recommender system would recommend the requirement req_3 (if this one has not been investigated by the stakeholder up to now). If no explicitly defined categories are available, the textual description of each requirement can be exploited for the extraction of keywords which serve for the characterization of the requirement. Extracted keywords can then be used for the determination of similar requirements (R. Mooney (2004)). A basic metric for determining the similarity between two requirements is given in Formula 2.1. For example, sim(req$_1$, req$_3$) = 0.17, if we assume *keywords*(req$_1$) = {*store, component, configuration, DB*} and *keywords*(req$_3$) = {*tier, DB, independence*}.

$$sim(req_1, req_2) = \frac{|keywords(req_1) \cap keywords(req_2)|}{|keywords(req_1) \cup keywords(req_2)|} \tag{2.1}$$

Example: Group-based Recommendation of Requirements Prioritizations. Group recommenders include heuristics (Masthoff (2011)) that can help to find solution alternatives which will be accepted by the members of a group (with a high probability). Requirements prioritization is the task of deciding which of a given set of requirements should be implemented within the scope of the next software release – as such, this task has a clear need of group decision support: a stakeholder group has to decide which requirements should be taken into account. Different heuristics for coming up with a group recommendation are possible, for example, the *majority heuristic* proposes a recommendation that takes the majority of requirements-individual votes as group recommendation (see Table 2.4). In contrast, the *least misery* heuristic recommends the minimum of the requirements-individual ratings to avoid misery for individual group members. For a detailed discussion of different possible group recommendation heuristics we refer the reader to (Masthoff (2011)).

Table 2.4.: Example of a group recommendation problem: recommendation of requirements prioritizations to a group of stakeholders (used heuristics = majority voting).

requirement	$stakeholder_1$	$stakeholder_2$	$stakeholder_3$	$stakeholder_4$	*recommendation*
req_1	1	1	1	2	1
req_2	5	4	5	5	5
req_3	3	3	2	3	3
req_4	5	5	4	5	5

2.4. Recommender Systems in Data & Knowledge Engineering

Similar to conventional software development, knowledge-based systems development suffers from continuously changing organizational environments and personal. In this context, recommender systems can help database & knowledge engineers to better cope with the size and complexity of knowledge structures (Chatzopoulou et al. (2009); Felfernig et al. (2009a)). For example, recommender systems can support an improved understanding of the knowledge base by actively suggesting those parts of the knowledge base which are candidates for increased testing and debugging (Felfernig et al. (2009a, 2012b)). Furthermore, recommender systems can propose repair and refactoring actions for faulty and ill-formed knowledge bases (Felfernig et al. (2010, 2011a)). In the context of information search, recommender systems can improve the accessibility of databases (knowledge bases) by recommending queries (Chatzopoulou et al. (2009)).

Knowledge Base Understanding. Understanding the basic elements and the organizational structure of a knowledge base is a major precondition for efficient knowledge base development and maintenance. In this context, recommender systems can be applied for supporting knowledge engineers, for example, by (collaboratively) recommending constraints to be investigated when analyzing a knowledge base (recommendation of navigation paths through a knowledge base) or by recommending constraints which are related to each other, i.e., are referring to common variables (Felfernig et al. (2012b)).

Knowledge Base Testing and Debugging. Knowledge bases are frequently adapted and extended due to the fact that changes in the application domain have to be integrated into the corresponding knowledge base. Integrating such changes into a knowledge base in a consistent fashion is time-critical (Felfernig et al. (2006a)). Recommender systems can be applied for improving the efficiency of knowledge base testing and debugging by recommending minimal sets of changes to knowledge bases which allow to restore consistency (Felfernig et al. (2009a, 2012a)). Popular approaches to the identification of such *hitting sets* are model-based diagnosis introduced by Reiter (Reiter (1987)) and different variants thereof (Felfernig et al. (2012a)).

Knowledge Base Refactoring. Understandability and maintainability of knowledge bases are important quality characteristics which have to be taken into account within the scope of knowledge base development and maintenance processes (Barker et al. (1989)). Low quality knowledge structures can lead to enormous maintenance costs and thus have to be avoided by all available means. In this context, recommender systems can be applied for recommending relevant refactorings, i.e., semantics-preserving structural changes of the knowledge base. Such refactorings can be represented by simple constraint rewritings (Felfernig et al. (2010)) or by simplifications in terms of recommended removals of redundant constraints, i.e., constraints which do not have an impact on the semantics of the knowledge base (Felfernig et al. (2011a)).

Recommender Systems in Databases. Chatzopoulou et al. (2009) introduce a recommender application that supports users when interactively exploring relational databases (*complex SQL queries*). This application is based on collaborative filtering where information about the navigation behavior of the current user is exploited for the recommendation of relevant queries. These queries are generated on the basis of the similarity of the current user's navigation behavior and the navigation behavior of nearest neighbors (other users with a similar navigation behavior who already searched the database). Fakhraee and Fotouhi (2011) focus on recommending database queries which are derived from those database attributes that contain the keywords part of the initial user query. In contrast to Chatzopoulou et al. (2009), Fakhraee and Fotouhi (2011) do not support the recommendation of complex SQL queries but focus on basic lists of keywords (*keyword-based database queries*).

Example: Collaborative Recommendation of Relevant Constraints. When knowledge engineers try to *understand a given set of constraints* part of a knowledge base, recommender systems can provide support in terms of showing related constraints, for example, constraints that have been analyzed by knowledge engineers in a similar navigation context. In Table 2.5, the constraints $\{\text{constraint}_{1..4}\}$ have already partially been investigated by the knowledge engineers $\{\text{knowledge engineer}_{1..3}\}$. For example, knowledge engineer$_1$ has already investigated the constraints constraint$_1$ and constraint$_3$. Collaborative filtering (CF) can exploit the ratings (the rating = 1 if a knowledge engineer has already investigated a constraint and it is 0 if the constraint has not been investigated up to now) for identifying additional constraints relevant for the knowledge engineer.

Table 2.5.: Example of a collaborative recommendation problem. The entry *ij* with value 1 (0) denotes that fact that knowledge engineer$_i$ has (not) inspected constraint$_j$.

	constraint$_1$	constraint$_2$	constraint$_3$	constraint$_4$
knowledge engineer$_1$	1	0	1	?
knowledge engineer$_2$	1	0	1	1
knowledge engineer$_3$	1	1	0	1

User-based CF (Konstan et al. (1997)) identifies the k-nearest neighbors (knowledge engineers with a similar knowledge navigation behavior) and determines a prediction for the rating of a constraint the knowledge engineer had not to deal with up to now. This prediction can be determined, for example, on the basis of the majority of the k-nearest neighbors. In the example of Table 2.5 knowledge engineer$_2$ is the nearest neighbor (if we set k=1) since knowledge engineer$_2$ has analyzed (or changed) all the constraints investigated by knowledge engineer$_1$. At the same time, knowledge engineer$_1$ did not analyze (change) the constraint constraint$_4$. In our example, CF would recommend the constraint$_4$ to knowledge engineer$_1$.

2.5. Recommender Systems for Configurable Items

Configuration can be defined as a basic form of design activity where a target product is composed from a set of predefined components in such a way that it is consistent with a set of predefined constraints (Sabin and Weigel (1998)). Similar to knowledge-based recommender systems (Burke (2000); Felfernig et al. (2006a)), configuration systems support users in specifying their requirements and give feedback in terms of solutions and corresponding explanations (Falkner et al. (2011)). The major difference between configuration and recommender systems is the way in which these systems represent the product (configuration) knowledge: configurators are exploiting a configuration knowledge base whereas (knowledge-based) recommender systems are relying on a set of enumerated solution alternatives (items). Configuration knowledge bases are especially useful in scenarios with a large number of solution alternatives which would make an explicit representation extremely inefficient (Falkner et al. (2011)).

In many cases, the amount and complexity of options presented by a configurator outstrips the capability of a user to identify an appropriate solution. Recommendation technologies can be applied in various ways to improve the usability of configuration systems, for example, filtering out product features which are relevant for the user (Falkner et al. (2011); Garcia-Molina et al. (2011)), proposing concrete feature values and thus helping the user in situations where knowledge about certain product properties is not available (Falkner et al. (2011)), and by determining plausible repairs for inconsistent user requirements (Felfernig et al. (2009a)).

Selecting Relevant Features. In many cases users are not interested in specifying all the features offered by a configurator interface, for example, some users may be interested in the GPS functionality of digital cameras whereas this functionality is completely uninteresting for other users. In this context, recommender systems can help to determine features of relevance for the user, i.e., features the user is interested to specify. Such a feature recommendation can be implemented, for example, on the basis of collaborative filtering (Felfernig and Burke (2008)).

Determining Relevant Feature Values. Users try to avoid to specify features they are not interested in or about which they do not have the needed technical knowledge (Falkner et al. (2011)). In such a context, recommender systems can automatically recommend feature values and thus reduce the burden of interaction for the user. Feature value recommendation can be implemented, for example, on the basis of collaborative filtering (Falkner et al. (2011); Konstan et al. (1997)). Note that feature value recommendation can trigger biasing effects and – as a consequence – could also be exploited for manipulating users to select services which are not necessarily needed (Mandl et al. (2011)).

Determining Plausible Repairs for Inconsistent Requirements. In situations where no solution can be found for a given set of customer requirements, configuration systems determine a set of repair alternatives which guarantee the recovery of consistency. Typically many different repair actions are determined by the configurator which makes it nearly infeasible for the user to find one which exactly fits his/her wishes and needs. In such a situation, knowledge-based and collaborative recommendation technologies can be exploited for personalizing the user requirements repair search process (Felfernig et al. (2009a)).

Example: Collaborative Recommendation of Features. Table 2.6 represents an interaction log which indicates in which session which features have been selected by the user (in which order) – in session$_1$, feature$_1$ was selected first, then feature$_3$ and feature$_2$, and finally feature$_4$. One approach to determine relevant features for (the current) user in session$_5$ is to apply collaborative filtering. Assuming that the user in session$_5$ has specified the features$_{1,2}$, the most similar session would be session$_2$ and feature$_3$ would be recommended to the user since it had been selected by the nearest neighbor (user in session$_2$). For a discussion of further feature selection approaches we refer the reader to Falkner et al. (2011).

Table 2.6.: Example of collaboratively recommending relevant features. A table entry *x* denotes the order in which a user specified values for the given features.

	feature$_1$	feature$_2$	feature$_3$	feature$_4$
session$_1$	1	3	2	4
session$_2$	1	2	3	4
session$_3$	1	1	4	3
session$_4$	1	3	2	4
session$_5$	1	2	?	?

2.6. Recommender Systems for Persuasive Technologies

Persuasive technologies (Fogg (2003)) aim to trigger changes in a user's attitudes and behavior on the basis of the concepts of human computer interaction. The impact of persuasive technologies can be significantly increased by additionally integrating recommendation technologies into the design of persuasive systems. Such an approach moves persuasive technologies forward from a one-size-fits all approach to a personalized environment where user-specific circumstances are taken into account when generating persuasive messages (Berkovsky et al. (2012)). Examples of the application of recommendation technologies in the context of persuasive systems are the enforcement of physical activity while playing computer games (Berkovsky et al. (2010)) and encouraging software developers to improve the quality of their software components (Pribik and Felfernig (2012)).

Persuasive Games. Games focusing on the motivation of physical activities include additional reward mechanisms to encourage players to perform real physical activities. Berkovsky et al. (2010) show the successful application of collaborative filtering recommendation technologies (Konstan et al. (1997)) for estimating the personal difficulty of playing. This recommendation (estimation) is exploited to adapt the difficulty level of the current game session since the perceived degree of difficulty is correlated with the preparedness of a user to perform physical activities.

Persuasive Software Development Environments. Software development teams are often under the gun of developing software components under high time pressure which often has a direct impact on the corresponding software quality. However, in the long term software quality is strongly correlated with the degree of understandability and maintainability. In this context, software quality improvements can be achieved by recommendation technologies, for example, knowledge-based recommenders can be applied to inform programmers about critical code segments and also recommend actions to be performed in order to increase the software quality. Pribik and Felfernig (2012) introduce such an environment which has been implemented as an Eclipse plugin (*www.eclipse.org*).

Example: Collaborative Estimation of Personal Game Level Difficulties. Table 2.7 depicts a simple example of the application of collaborative filtering to the determination of personal difficulty levels. Let us assume that the current user in session$_5$ has already completed the tasks 1..3; by determining the nearest neighbor of session$_5$ we can infer a probable duration of task$_4$: session$_1$ is the nearest neighbor of session$_5$ and the user of session$_1$ needed 7 time units to complete task$_4$. This knowledge about the probable time efforts needed to complete a task can be exploited to automatically adapt the complexity level of the game (with the goal to increase the level of physical activity (Berkovsky et al. (2010))).

Table 2.7.: Collaborative filtering based determination of personal game difficulty levels. A table entry *x* denotes the time a user needed to complete a certain game task.

	task$_1$	task$_2$	task$_3$	task$_4$
session$_1$	4	6	5	7
session$_2$	1	2	2	2
session$_3$	4	5	5	6
session$_4$	1	1	1	2
session$_5$	4	6	4	?

2.7. Further Applications

Up to now we discussed a couple of new and innovative application domains for recommendation technologies. Admittedly, this enumeration is incomplete since it does not reflect wide-spread e-commerce applications – for a corresponding overview the interested reader is referred to Linden et al. (2003); Schafer et al. (2011). In this section we discuss further new and upcoming application domains for recommendation technologies that have been identified within the scope of our literature analysis.

Recommender Systems for Smart Homes. Smart homes are exploiting information technologies to improve the quality of life inside the home. Leitner et al. (2012) show the application of knowledge-based recommendation technologies in the context of ambient assisted living (AAL) scenarios where on the one hand recommenders are applied to the design a smart home, on the other hand to control the smart home equipment. During the design phase of a smart home, the user is guided through a preference construction process with the final outcome of a recommendation of the needed technical equipment for the envisioned smart home. For already deployed smart home installations recommendation technologies support the people living in the smart home by recommending certain activities such as activating the air conditioning or informing other relatives about dangerous situations (e.g., for some relevant time period the status of the elderly people living in the house is unclear). A further application of recommendation technologies within the context of AAL is reported by LeMay et al. (2009d) who introduce an application of collaborative recommendation technologies for supporting the control of complex smart home installations.

People Recommender. Social networks such as *facebook.com* or *linkedin.com* are increasingly popular communication platforms. These platforms also include different types of recommender applications that support users in retrieving interesting music, travel destinations, and connecting to other people. Recommendations determined by these platforms are often exploiting the information contained in the underlying social network (Golbeck (2009)). A new and upcoming application which exploits the information of social networks is the identification of crime suspects. In this context, social networks are representing the relationships between criminals. The *CrimeWalker* system introduced by Tayebi et al. (2011) is based on a random-walk based method which recommends (provides) a list of the top-k potential suspects of a crime. Similar to *CrimeWalker*, *TechLens* is a recommender system which focuses on the recommendation of persons who could act – for example – as reviewers within the scope of a scientific conference organization. In contrast to *CrimeWalker*, the current version of *TechLens* does not exploit information from social networks – it is based on a collaborative filtering recommendation approach. Yuan et al. (2012) present their approach to the recommendation of passenger hotspots, i.e., the recommender system is responsible for improving the prediction quality of hotspots and with this decreases the idle time of taxi drivers. Finally, Hoens et al. (2010) present their approach to the recommendation of physicians – the major goal behind this application is to provide mechanisms which improve the quality of physician selection which otherwise is based on simple heuristics such as the opinion of friends or the information found on websites.

RFID based Recommenders. Personalized services become ubiquitous, for example, in tourism many destinations such as museums and art galleries are providing personalized access to help customers to better cope with the large amount of available services. An alternative to force users to explicitly declare their preferences before receiving a recommendation of potentially interesting services is to observe a user's navigation behavior and – on the basis of this information – to infer plausible recommendations. Such a recommendation approach is in the need of location and object information in order to be able to store the user navigation behavior. A collaborative filtering based handheld guide for art museums is introduced in Huang et al. (2010) where Radiofrequency Identification (RFID) serves as a basis for preference data acquisition. RFID is a non-contact tracking system which exploits radio-frequency electromagnetic fields to transfer tag data for the purposes of object identification and object tracking (Thiesse and Michahelles (2009)).

Help Agents. Howto's are an important mechanism to provide guidance for users who are non-experts in a certain problem domain – such a support can be implemented, for example, on the basis of recommendation technologies (Terveen and Hill (2001)). One example for such a type of *help agent* is *SmartChoice* which is a (content-based) recommender system that supports representatives of low-income families in the choice of a public school for their children. Such a recommendation support is crucial since in many cases (a) parents do not dispose of detailed knowledge about the advantages and disadvantages of the different school types and (b) a false decision can have a very negative impact on the future life of the children. Another example of a help agent is *Personal Choice Point* (Fano and Kurth (2003)) which is a financial service recommendation environment focusing on the visualization of the impact of different financial decisions on a user's life. Hammer et al. (2010) present *MED-StyleR* which is a lifestyle recommender dedicated to the support of diabetes patients with the goal of improving care provision, enhancing the quality of a patient's life, and also to lower costs of public health institutions and patients. Another lifestyle related recommender application is presented by Pinxteren et al. (2011) which focuses on determining health-supporting recipes that also fit the lifestyle of the user. In the same line, Wiesner and Pfeifer (2010) introduce a content-based recommender application dedicated to the identification of relevant health information for a specific user.

Recommender Systems in Open Innovation. Integrating consumer knowledge into a company's innovation processes (also denoted as Open Innovation (Chesbrough (2003))) is in many cases crucial for efficient new product and service development. Innovation process quality has a huge impact on the ability of a company to achieve sustainable growth. Innovations are very often triggered by consumers who are becoming active contributors in the process of developing new products. Platforms such as *sourceforge.net* or *ideastorm.com* confirm this trend of progressive customer integration. These platforms exploit community knowledge and preferences to come up with new requirements, ideas, and products. In this context, the size and complexity of the generated knowledge (informal descriptions of requirements and ideas as well as knowledge bases describing new products on a formal level) outstrips the capability of community members to retain a clear view. Recommender systems can provide help in terms of finding other users with similar interests and ideas (*team recommendation* (Brocco and Groh (2009))) and to (semi-) automatically filter out the most promising ideas (*idea mining* (Jannach and Bundgaard-Joergensen (2007); Thorleuchter et al. (2010))).

2.8. Issues for Future Research

Recommendation technologies have been successfully applied for almost two decades primarily with the goal of increasing the revenue of different types of online services. In most of the existing systems the primary focus is to help to achieve business goals, rarely the viewpoint of the customer is taken into account in the first place (Martin et al. (2011)). For example, *amazon.com* recommenders inform users about new books of interest for them; a more customer-centered recommendation approach would also take into account (if available) information about books that have been bought by friends or relatives and thus are potentially available for the customer (Martin et al. (2011)). As a consequence, we are in the need of new recommendation technologies that allow more customer-centered recommendation approaches. In this context, the following research challenges have to tackled.

Focusing on the User Perspective. There are many other scenarios quite similar to the above mentioned *amazon.com* one where the recommender system is clearly focused on increasing business revenues. For example, consumer packaged goods (CPG) are already offered on the basis of recommender systems (Dias et al. (2008)), however, these systems are domain-specific, i.e., do not take into account information regarding goods and services offered by the grocer nearby. Digital camera recommenders recommend the newest technology but in most cases do not take into account the current portfolio of the user, for example, if a user has a complete lens assortment of camera provider *X* it does not make sense to recommend a new camera of provider *Y* in the first place. An approach which is in the line of the idea of a stronger focus on the quality of user support is the RADAR personal assistant introduced by Faulring et al. (2009) that supports multi-task coordination of personal emails.

Sharing Recommendation Knowledge. Besides commercial interests, one of the major reasons for the low level of customer orientation of todays recommender solutions is the lack of the needed recommendation knowledge. In order to recommend books already read by friends the recommender would need the information of the social network of the customer. The global availability of CPG goods information seems to be theoretically possible but is definitely in the need of a corresponding cloud and mobile computing infrastructure. More customer-centered recommender systems will follow the paradigm of personal assistants which does not focus on specific recommendation services but rather provides an integrated and multi-domain recommendation service (Chung et al. (2007); Martin et al. (2011)). Following the idea of *ambient intelligence* (Ramos et al. (2008)), such systems will be based on global object information (Ramiez-Gonzales et al. (2010); Thiesse and Michahelles (2009)) and support users in different application contexts in a cross-domain fashion.

Context Awareness. New recommendation technologies will intensively exploit the infrastructure of mobile services to determine and take into account the context of the current user (Adomavicius and Tuzhilin (2005)). Information such as the users shorttime and longterm preferences, geographical position, movement data, calendar information, information from social networks can be exploited for detecting the current context of the person and exploit this information for coming up with intelligent recommendations (Ballatore et al. (2010)).

Unobtrusive Preference Identification. Knowledge about user preferences is a key preliminary for determining recommendations of relevance for the user. A major issue in this context is the development of new technologies which allow the elicitation of preferences in an unobtrusive fashion (Foster and Oberlander (2010); Lee et al. (2008); Winoto and Tang (2010)). The three major modalities to support such a type of preference elicitation are the detection of facial expressions, the interpretation of recorded speech, and the analysis of physiological signals. An example of the derivation of user preferences from the analysis of eye tracking patterns is presented by Xu et al. (2008) who exploit eye tracking technologies by interpreting attention times to improve the quality of a content-based filtering recommender. An approach to preference elicitation from physiological signals is presented by Janssen et al. (2011) who exploit the information about skin temperature for measuring valence which is applicable to mood measurement.

Psychological Aspects of Recommender Systems. Building efficient recommendation algorithms and the corresponding user interfaces requires a deep understanding of human decision processes. This goal can be achieved by analyzing existing psychological theories of human decision making and their impact on the construction of recommender systems. Cosley et al. (2003) already showed that the style of item rating presentation has a direct impact on a users' rating behavior. Adomavicius et al. (2011) showed the existence of anchoring effects in different collaborative filtering scenarios. With their work, Adomavicius et al. (2011) confirm the results presented by Cosley et al. (2003) but they show in more detail in which way rating drifts can have an impact on the rating behavior of a user. As already mentioned, recommendation technologies improve the quality of persuasive interfaces (Berkovsky et al. (2010); Pribik and Felfernig (2012)). Future recommenders should exploit the information provided by the mentioned preference elicitation methods (Janssen et al. (2011)).

2.9. Conclusions

With this chapter we provide an overview of new and upcoming applications of recommendation technologies. This overview does not claim to be complete but is the result of an analysis of work published in recommender systems related workshops, conferences, and journals. Beside providing insights into new and upcoming applications of recommendation technologies we also provide a discussion of issues for future research with the goal of advancing the state of the art in recommender systems which is characterized by a more user-focused and personal assistance based recommendation paradigm.

Chapter 3

Conflict Detection and Diagnosis Techniques for Anomaly Management

This chapter is based on the results documented in Felfernig et al. (2014d).

Besides *writing major parts of the above mentioned paper*, the author of this thesis contributed the *examples and the empirical evaluation*.

3.1. Abstract

The widespread industrial application of configuration technologies triggers an increasing demand for intelligent techniques that efficiently support anomaly management operations for configuration knowledge bases. Examples of such operations are the testing and debugging of faulty knowledge bases (see (Friedrich et al., 2014)) and the detection of redundancies in configuration knowledge bases (see (Felfernig et al., 2014e)). The goal of this chapter is to discuss techniques and algorithms which form the technological basis for the aforementioned anomaly management operations.

3.2. Introduction

Anomalies can be characterized as parts of a knowledge base that conform to a defined *pattern of unintended structures* – see also (Chandola et al., 2009). Anomaly management operations presented in this chapter are the *automated testing and debugging* of configuration knowledge bases (see (Friedrich et al., 2014)) and the *automated detection of redundancies* in configuration knowledge bases (see (Felfernig et al., 2014e)). Anomaly management operations are very important since configuration knowledge bases are often complex and subject to frequent changes (Barker et al., 1989; Fleischanderl et al., 1998).

The foundation for anomaly management are *conflict detection* and *diagnosis* algorithms that help to detect (a) *minimal subsets* of the knowledge base that are responsible for a faulty behavior (Junker, 2004; Felfernig et al., 2004, 2012b, 2013c) and (b) *maximal subsets* of the knowledge base which are redundancy-free (Felfernig et al., 2011a). More precisely, conflict detection algorithms are able to determine minimal sets of constraints that are inconsistent, i.e., do not allow the determination of a solution (configuration). In addition, diagnosis algorithms can determine minimal sets of constraints in the configuration knowledge base that have to be deleted or adapted such that the remaining set of constraints is consistent (see (Friedrich et al., 2014)), i.e., a solution can be determined. Typically, diagnoses are determined from a given set of (minimal) conflicts and – vice-versa – minimal conflicts can as well be determined on the basis of a given set of (minimal) diagnoses (see Section 3.5).

The major focus of this chapter is to provide an introduction to some of the existing conflict detection and diagnosis techniques which are the foundation for the aforementioned anomaly management operations. The remainder of this chapter is organized as follows. In Section 3.3, we introduce the working example used in this chapter – it is a simplified version of the example configuration knowledge base introduced in (Hotz et al., 2014). We discuss selected conflict detection algorithms in Section 3.4 and the corresponding diagnosis algorithms in Section 3.5. On the basis of the results of a performance analysis, we discuss the advantages and disadvantages of the presented conflict detection and diagnosis algorithms. Section 3.6 concludes this chapter.

3.3. Example

We introduce a working example which is based on the configuration knowledge base introduced in (Hotz et al., 2014). For the following discussions we introduce the configuration knowledge base (C_{KB}) of a fragment of a personal computer (`PC`) which consists of the component types `MB` (motherboard) and `CPU` (central processing unit) and their specific subtypes {`MBSilver`, `MBDiamond`} and {`CPUS`, `CPUD`}.

The following personal computer (`PC`) configuration knowledge base (configuration model) is *inconsistent*, i.e., it does not allow the calculation of a solution (configuration). Such situations often occur as a result of maintenance operations in a knowledge base where, for example, conflicts are introduced due to the insertion of new elements or due to the adaptation of existing ones. In our working example we assume that constraint c_2 (`CPUS` requires `MBSilver`) has been added by the knowledge engineer in order to replace the faulty constraint c_1 (`CPUS` requires `MBDiamond`). Unfortunately, the knowledge engineer inserted c_2 without deleting c_1. Our example also assumes that a constraint c_5 (instances of the component type `CPUD` should not be part of any configuration) was introduced earlier for testing purpose with no intent to have it pertaining to the configuration model. However the knowledge engineer forgot to disable it after testing the model. Constraint c_3 states that an `MBSilver` must not be combined with a `CPUD` and constraint c_4 states that an `MBDiamond` must not be combined with a `CPUS`.

$C_{KB} = \{$

$c_\alpha : \forall X : MB(X) \rightarrow \exists_1^1 Y :$ cpu-of-mb(Y,X).

$c_\beta : \forall X : CPU(X) \rightarrow \exists_1^1 Y :$ mb-of-cpu(Y,X).

$c_\gamma : \forall X : MB(X) \leftrightarrow MBSilver(X) \vee MBDiamond(X).$

$c_\delta : \forall X : \neg MBSilver(X) \vee \neg MBDiamond(X).$

$c_\varepsilon : \forall X : CPU(X) \leftrightarrow CPUD(X) \vee CPUS(X).$

$c_\phi : \forall X,Y :$ cpu-of-mb(X,Y) $\leftrightarrow$ mb-of-cpu(Y,X).

$c_\iota : \forall X : \neg CPUD(X) \vee \neg CPUS(X).$

$c_1 : \forall X,Y :$ cpu-of-mb(Y,X) $\wedge CPUS(Y) \rightarrow MBDiamond(X).$

$c_2 : \forall X,Y :$ cpu-of-mb$(Y,X) \wedge CPUS(Y) \rightarrow MBSilver(X).$

$c_3 : \forall X,Y :$ cpu-of-mb$(Y,X) \wedge CPUD(Y) \ \wedge \ MBSilver(X) \rightarrow false.$

$c_4 : \forall X,Y :$ cpu-of-mb$(Y,X) \wedge CPUS(Y) \ \wedge \ MBDiamond(X) \rightarrow false.$

$c_5 : \forall X : CPUD(X) \rightarrow false.$

$\}$

We will now start with a discussion of algorithms that focus on conflict detection and then show how to exploit identified minimal conflict sets in order to determine corresponding diagnoses.

3.4. Determining Minimal Conflict Sets

Before discussing basic conflict detection algorithms useful for (but not limited to) configuration scenarios, we introduce the definition of a conflict (conflict set) in the context of an inconsistent configuration knowledge base.

Definition (Minimal Conflict Set) *A conflict set $CS = \{c_a, c_b, ..., c_z\}$ is a subset of C such that* inconsistent*($B \cup CS$). $AC = B \cup C$ represents the set of all constraints in the knowledge base ($AC = \{c_1, c_2, ..., c_n\}$), B represents the background knowledge (no conflict elements are assumed to be included in B), and C represents the set of constraints subject of conflict search. A conflict set CS is* minimal *if there does not exist a $CS' \subset CS$ which has the conflict property.*

Note that if we want to analyse the whole knowledge base (taking into account the complete set of constraints in conflict set search), we simply have to define $C = AC$ and – as a consequence – $B = \emptyset$.

In our working example we are able to identify the following minimal conflict sets: $CS_1 = \{c_1, c_4, c_5\}$ and $CS_2 = \{c_1, c_2, c_5\}$. These sets do not allow the determination of a solution, i.e., *inconsistent*($\{c_1, c_4, c_5\} \cup B$) and *inconsistent*($\{c_1, c_2, c_5\} \cup B$). CS_1 and CS_2 are minimal since none of their subsets fulfills the conflict set property. Each minimal conflict set (conflict) can simply be resolved by deleting one of it's elements. This approach is used by some of the diagnosis algorithms discussed in Section 3.5.

In the remainder of this section we introduce and evaluate two basic conflict detection algorithms: SIMPLECONFLICTDETECTION and QUICKXPLAIN.

3.4.1. Simple Conflict Detection

The first approach to the determination of minimal conflict sets is SIMPLECONFLICTDETECTION (see Algorithm 1). Conform with the introduced definition of a *conflict set*, the algorithm focuses the search for a minimal conflict by explicitly specifying the set of constraints $C \subseteq AC$ which might induce a conflict. Note that SIMPLECONFLICTDETECTION determines exactly one conflict set per computation. In order to identify all minimal conflicts in C, this algorithm has to be combined with a HSDAG-based diagnosis algorithm (see Section 3.5).

Algorithm 1 – SIMPLECONFLICTDETECTION

```
func SIMPLECONFLICTDETECTION($C \subseteq AC, B = AC - C\}$) : $CS$
$CS \leftarrow \emptyset$;
if $inconsistent(B)$ or $consistent(B \cup C)$ $return(\emptyset)$;
else
repeat
  $\Phi = CS$;
  repeat
    $c \leftarrow element(C - \Phi)$;
    $\Phi \leftarrow \Phi \cup \{c\}$;
  until $inconsistent(\Phi)$
  $CS \leftarrow CS \cup \{c\}$;
until $inconsistent(CS)$
$return(CS)$;
```

The set CS collects all relevant elements (constraints) of the minimal conflict. If the deletion of all constraints of C does not allow the determination of a solution (*inconsistent*(B)) or AC ($B \cup C$) itself is consistent, there is no need for searching for a conflict (the empty set $\emptyset$ is returned). In all other cases, the algorithm tries to identify a minimal conflict. In order to achieve this goal, elements c_i of C

are extracted and used to figure out whether they are triggering a conflict. If such an element has been identified, it is stored in the set *CS* which collects the set of elements contributing to the conflict.

In order to demonstrate the functionality of SIMPLECONFLICTDETECTION, the basic steps of this algorithm are shown in Table 3.1. For this example (see also Figure 3.1) we take the constraints of our domain description (DD) and assume that $C = \{c_1, c_2, c_3, c_4, c_5\}$ and $AC = \{c_\alpha, c_\beta, c_\gamma, c_\delta, c_\varepsilon, c_\phi, c_\iota, c_1, c_2, c_3, c_4, c_5\}$ ($B = \{c_\alpha, c_\beta, c_\gamma, c_\delta, c_\varepsilon, c_\phi, c_\iota\}$). The inner loop is responsible for detecting individual elements that participate in the conflict, the outer loop is responsible for checking wether the generated set *CS* is already inconsistent, i.e., a minimal conflict set could be found.

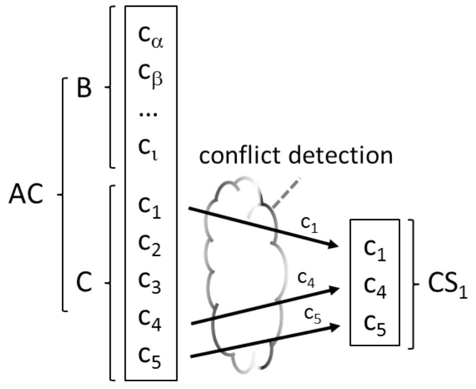


Figure 3.1.: A conflict set CS is a subset of C ($AC = C \cup B$) which is inconsistent with B. CS is minimal if no subset of CS fulfills the conflict set property. In this context, B is the background knowledge which includes all constraints considered correct. An example conflict set is $CS_1 = \{c_1, c_4, c_5\}$.

Algorithm Analysis. If a minimal conflict set can be determined ($AC - C$ consistent and AC inconsistent), SIMPLECONFLICTDETECTION needs O(2) additional consistency checks in the best case (a constraint c_a that represents a singleton conflict and is located at the first position of the constraint list C). In the worst case, each element of C is also part of the minimal conflict – in this case, $O(\frac{(n \times (n+1))}{2} + n)$ additional consistency checks are needed.

3.4.2. QuickXPlain

A more efficient approach to the determination of minimal conflict sets is QUICKXPLAIN introduced by Junker (Junker, 2004) (see Algorithm 2). This algorithm is based on the concept of *divide and conquer*: each time it detects that the first half of a constraint set C is already inconsistent, the second half of the constraint set is omitted, i.e., not further taken into account when determining the conflict. This is an efficient way to get rid of constraints that do not participate in the conflict.

QUICKXPLAIN determines exactly one conflict set per computation. In order to identify all minimal conflicts in C, this algorithm has to be combined with a HSDAG-based diagnosis algorithm (see

step	CS	c	Φ
1	$\emptyset$	c_5	$\{c_5\}$
2	$\emptyset$	c_4	$\{c_5, c_4\}$
3	$\emptyset$	c_3	$\{c_5, c_4, c_3\}$
4	$\emptyset$	c_2	$\{c_5, c_4, c_3, c_2\}$
5	$\emptyset$	c_1	$\{c_5, c_4, c_3, c_2, c_1\}$
6	$\{c_1\}$	c_5	$\{c_1, c_5\}$
7	$\{c_1\}$	c_4	$\{c_1, c_5, c_4\}$
8	$\{c_1, c_4\}$	c_5	$\{c_1, c_4, c_5\}$
9	$\underline{\{c_1, c_4, c_5\}}$	–	–

Table 3.1.: Example of the application of SIMPLECONFLICTDETECTION. CS = $\{c_1, c_4, c_5\}$ is returned as minimal conflict set (CS) for C = $\{c_5, c_4, c_3, c_2, c_1\}$ and B = $\{c_\alpha, c_\beta, c_\gamma, c_\delta, c_\varepsilon, c_\phi, c_\iota\}$.

Algorithm 2 – QUICKXPLAIN

```
func QUICKXPLAIN($C \subseteq AC, B = AC - C$) : $CS$
if $isEmpty(C)$ or $inconsistent(B)$ return $\emptyset$;
else return QX($\emptyset, C, B$);

func QX($D, C = \{c_1..c_q\}, B$) : $CS$
if $D \neq \emptyset$ and $inconsistent(B)$ return $\emptyset$;
if $singleton(C)$ return $C$;
$k = \lceil \frac{q}{2} \rceil$;
$C_1 = \{c_1..c_k\}; C_2 = \{c_{k+1}..c_q\}$;
$CS_1 = QX(C_2, C_1, B \cup C_2)$;
$CS_2 = QX(CS_1, C_2, B \cup CS_1)$;
return($CS_1 \cup CS_2$);
```

Section 3.5). How QUICKXPLAIN determines a minimal conflict can best be explained on the basis of our working example (see Table 3.2). The activation hierarchy for the recursive function QX is depicted in Figure 3.2.

The function QX adds additional constraints (from C_2) to the background knowledge as long as the resulting constraint set remains consistent. If it becomes inconsistent, the algorithm leaves out the remaining constraints. For example, in Table 3.2 (step 5) the set $\{c_1, c_3, c_4, c_5\}$ is inconsistent and the remaining constraint c_2 can be left out. On the other hand, if the background knowledge is consistent and only one constraint remains that induces the inconsistency, this constraint must be part of the conflict set. For example, in Table 3.2 (step 4) the background knowledge consists of the constraints $\{c_2, ..., c_5\}$ and c_1 remains as single constraint. It is clear that c_1 is part of the conflict since $\{c_2, ..., c_5\}$ is consistent but $\{c_2, ..., c_5\} \cup \{c_1\}$ is inconsistent.

Algorithm Analysis. When comparing the performance of SIMPLECONFLICTDETECTION with QUICKXPLAIN we can see that QUICKXPLAIN has the potential to reduce the number of needed

step	D	C	B	C_1	C_2	return
1	$\emptyset$	$\{c_1,...,c_5\}$	Γ	$\{c_1,c_2,c_3\}$	$\{c_4,c_5\}$	$\{c_1,c_4,c_5\}$
2	$\{c_4,c_5\}$	$\{c_1,c_2,c_3\}$	$\Gamma\cup\{c_4,c_5\}$	$\{c_1,c_2\}$	$\{c_3\}$	$\{c_1\}$
3	$\{c_3\}$	$\{c_1,c_2\}$	$\Gamma\cup\{c_3,...,c_5\}$	$\{c_1\}$	$\{c_2\}$	$\{c_1\}$
4	$\{c_2\}$	$\{c_1\}$	$\Gamma\cup\{c_2,...,c_5\}$	$\emptyset$	$\emptyset$	$\{c_1\}$
5	$\{c_1\}$	$\{c_2\}$	$\Gamma\cup\{c_1,c_3,c_4,c_5\}$	$\emptyset$	$\emptyset$	$\emptyset$
6	$\{c_1\}$	$\{c_3\}$	$\Gamma\cup\{c_1,c_4,c_5\}$	$\emptyset$	$\emptyset$	$\emptyset$
7	$\{c_1\}$	$\{c_4,c_5\}$	$\Gamma\cup\{c_1\}$	$\{c_4\}$	$\{c_5\}$	$\{c_4,c_5\}$
8	$\{c_5\}$	$\{c_4\}$	$\Gamma\cup\{c_1,c_5\}$	$\emptyset$	$\emptyset$	$\{c_4\}$
9	$\{c_4\}$	$\{c_5\}$	$\Gamma\cup\{c_1,c_4\}$	$\emptyset$	$\emptyset$	$\{c_5\}$

Table 3.2.: Example of QUICKXPLAIN: $\Gamma = \{c_\alpha, c_\beta, c_\gamma, c_\delta, c_\varepsilon, c_\phi, c_\iota\}$ is the (original) background knowledge and CS = $\{c_1,c_4,c_5\}$ is the returned conflict set. The sequence of the different QX activations is depicted in Figure 3.2.

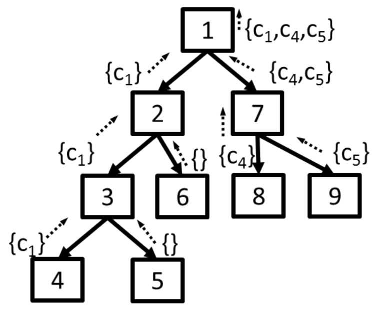


Figure 3.2.: Activation sequence of the different QUICKXPLAIN instances (for details see Table 3.2).

consistency checks. In our working example, SIMPLECONFLICTDETECTION needs 11 consistency checks whereas QUICKXPLAIN only needs 8 consistency checks. In the worst case, the algorithm needs $2k \times log_2(\frac{n}{k}) + 2k$ where k is the set size of the minimal conflict and n is the number of constraints in C (Junker, 2004). The best case complexity with regard to the number of consistency checks is $log_2(\frac{n}{k}) + 2k$.

Note that both of the presented conflict detection algorithms determine conflicts depending on the constraint ordering. If we reverse the ordering of the constraints of both algorithms, we would receive the conflict set $CS_2 = \{c_1,c_2,c_5\}$ first. For a more detailed discussion of issues related to constraint orderings and their role in conflict detection we refer the reader to (Junker, 2004).

3.4.3. Runtime Performance of Conflict Detection Algorithms

We conclude our discussion of conflict detection issues with an analysis of the runtime performance of the presented algorithms when executed on real-world configuration datasets taken from www.splot-research.org. The average runtime of the two algorithms was measured in milliseconds (ms) where in each setting the ordering of the constraints was randomized and each setting was executed 100 times.

It becomes clear that QUICKXPLAIN clearly outperforms SIMPLECONFLICTDETECTION in all of the analyzed settings (see Table 3.3).

domain	#con.	#var.	QUICKXPLAIN (ms)	SCD (ms)
DELL laptops	285	47	75.7	643.2
smarthomes	73	55	42.6	89.6
cars	150	73	42.9	406.2
Xerox printers	242	158	78.1	812.2

Table 3.3.: Runtime evaluation: the average runtime in milliseconds (ms) needed by SIMPLECONFLICTDETECTION (SCD) and QUICKXPLAIN to calculate one minimal conflict set (on a standard PC). The basis for this evaluation are knowledge bases from www.splot-research.org.

A major influence factor for the performance of QUICKXPLAIN is the size of conflict sets – the more elements in the conflicts, the more consistency checks are needed for determining one minimal conflict set. Another factor that has an impact on the performance of QUICKXPLAIN is ordering of the constraints in the consideration set *C*. The more these constraints are spread over *C* the more consistency checks can be expected since less constraints can be omitted in early phases of QX execution.

3.5. Determining Minimal Diagnoses

Conflict sets help to identify areas of inconsistencies within a set of constraints. If we want to know the minimal set of constraints that have to be adapted (or deleted from the configuration knowledge base) such that a solution can be found for the given configuration task, we need to determine the (minimal) diagnoses with regard to the determined conflict sets. Based on our definition of a *conflict set* in Section 3.4, we now introduce the definition of a diagnosis task and the corresponding diagnosis (solution).

Definition (Diagnosis Task) *A* diagnosis task *can be defined by the tuple (C,AC) where $AC = B \cup C$, B is the background knowledge, and C is the set of constraints to be analyzed.*

Before discussing potential algorithms that support the calculation of minimal diagnoses, we introduce the definition of a diagnosis which represents a solution to a given diagnosis task (C,AC). The basic idea of diagnosis determination is depicted in Figure 3.3.

Definition (Diagnosis) *A* diagnosis *for a given diagnosis task (C,AC) is a set of constraints $\Delta \subseteq C$ such that $B \cup C - \Delta$ is consistent. A diagnosis Δ is* minimal *if there does not exist a diagnosis $\Delta' \subset \Delta$ with the diagnosis property. Finally, a minimal diagnosis Δ is denoted as* minimal cardinality diagnosis *if there does not exist a minimal diagnosis Δ' with $|\Delta'| < |\Delta|$.*

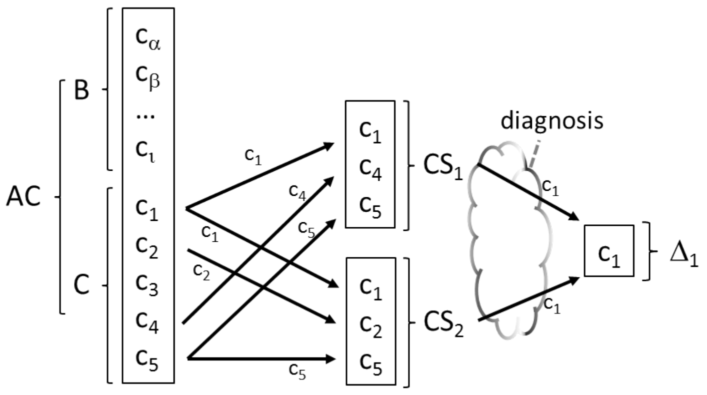


Figure 3.3.: A diagnosis Δ is a subset of C ($AC = C \cup B$) such that $B \cup C - \Delta$ is consistent. Δ is minimal if no subset of Δ fulfills the diagnosis property. B again represents the background knowledge. An example diagnosis is $\Delta_1 = \{c_1\}$.

A widespread approach to the determination of diagnoses (*hitting sets*) is the construction of a HSDAG. The algorithm is known as *hitting set directed acyclic graph* algorithm which has been introduced by (Reiter, 1987).

3.5.1. Hitting Set Directed Acyclic Graph (HSDAG)

The HSDAG algorithm supports the determination of minimal diagnoses. Since the algorithm performs diagnosis search typically in breadth-first fashion, it supports the determination of *minimal cardinality diagnoses*, i.e., minimal diagnoses with the lowest possible number of included constraints. However, the algorithm is complete, i.e., after returning minimal cardinality diagnoses, it returns all other diagnoses contained in the consideration set C. HSDAG structures can be established by repeatedly activating a conflict detection algorithm (e.g., SIMPLECONFLICTDETECTION or QUICKXPLAIN) and to analyze the different possibilities to resolve the returned conflict. Since this is a simple implementation of a *breadth-first* search strategy, we do not provide an algorithm here.

For example, in Figure 3.4 the first minimal conflict set returned by the conflict detection algorithm is $CS_1 : \{c_1, c_4, c_5\}$. If we resolve the conflict CS_1, by deleting the constraint c_4, another conflict set will be identified – $CS_2 : \{c_1, c_2, c_5\}$.

Due to the minimality of CS_1 we have exactly three different alternatives to resolve the conflict (by deleting one of its elements). The HSDAG algorithm is complete, i.e., all different alternatives to resolve the given conflicts are analyzed. For example, if we delete c_1 from CS_1, we have already determined our first diagnosis which is $\Delta_1 = \{c_1\}$. The second *minimal cardinality diagnosis* that can be derived from the HSDAG is $\Delta_2 = \{c_5\}$. There is a third minimal diagnosis (which is not a minimal cardinality one): $\Delta_3 = \{c_2, c_4\}$.

The HSDAG algorithm (Reiter, 1987) does not only support the determination of hitting sets in

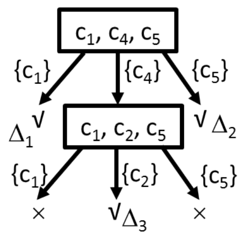


Figure 3.4.: Breadth-first based search for diagnoses on the basis of the minimal conflict sets $CS_1 = \{c_1, c_4, c_5\}$ and $CS_2 = \{c_1, c_2, c_5\}$. The resulting minimal diagnoses are $\Delta_1 = \{c_1\}$, $\Delta_2 = \{c_5\}$, and $\Delta_3 = \{c_2, c_4\}$.

terms of diagnoses. Given a predefined set of diagnoses we are also able to derive all corresponding minimal conflicts (see, e.g., Figure 3.5). Typical HSDAG implementations are based on a number of additional concepts which help to improve the performance of diagnosis (conflict set) detection. First, conflicts (diagnoses) can be *reused* in the sense that already determined conflicts (diagnoses) can be reused for those paths of the HSDAG which do not include the corresponding elements. In our working example (see Figure 3.5) there is no possibility to reuse a conflict set due to the fact that there is only one path to the leaf node of the HSDAG.

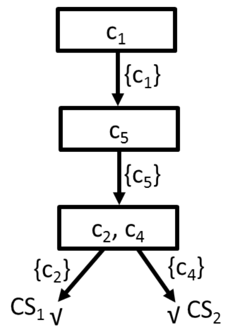


Figure 3.5.: Breadth-first based search for conflicts on the basis of the minimal diagnoses $\Delta_1 = \{c_1\}$, $\Delta_2 = \{c_5\}$, and $\Delta_3 = \{c_2, c_4\}$. The resulting minimal conflict sets are $CS_1 = \{c_1, c_2, c_5\}$, $CS_2 = \{c_1, c_4, c_5\}$.

A second possibility to improve the performance of diagnosis search is to close specific paths in the tree which have already been expanded in other parts of the tree. In our working example (see Figure 3.3), the path $\{c_1\}$ directly leads to a diagnosis. For this reason it does not make sense to further expand other paths that include c_1. We can also take into account situations where two paths contain the same set of elements. In such a situation, one of these paths can be closed, i.e., does not have to be expanded further.

3.5.2. FastDiag

There are many situations where not all the existing diagnoses are of relevance for the user. For example, when interacting with a configurator, the user is not interested in having to handle a possible huge number of alternative diagnoses. There is a need for algorithms that are able to determine so-called *leading diagnoses* quickly which can then be analyzed and evaluated by the user. Since in many cases not all possible diagnoses can be determined (for performance reasons), diagnoses assumed to be relevant for the user are determined first – these diagnoses are denoted as *leading diagnoses*. FASTDIAG is an algorithm that efficiently determines leading diagnoses without the additional overhead of determining conflict sets. It is easy to implement (no conflict detection and HSDAG algorithm are needed for determining one diagnosis) and is specifically useful in interactive configuration settings, for example, to support the user in situations where no solution can be found (Felfernig et al., 2012b).

In the line of QUICKXPLAIN, FASTDIAG heavily relies on the concept of *divide and conquer*. Assuming that the set AC is inconsistent, the major idea is to divide the set of constraints C into two different subsets C_1 and C_2. If, for example, $AC - C_1$ is consistent, we already know that the set C_1 includes a diagnosis and – as a consequence – there is no need to further analyze C_2 with regard to further diagnosis elements. A major advantage of FASTDIAG is that it is based on conflict-independent search strategies, i.e., no conflicts are needed for determining a diagnosis. FASTDIAG determines exactly one diagnosis at a time (if one exists). If we are interested in all diagnosis for a given set of constraints AC, we have to combine FASTDIAG with a corresponding HSDAG algorithm (see the example in Figure 3.5).

Algorithm 3 – FASTDIAG

```
1  func FASTDIAG($C \subseteq AC, AC = \{c_1..c_t\}$) : $diagnosis\ \Delta$
2  if $isEmpty(C)$ or $inconsistent(AC - C)$ return $\emptyset$
3  else return FD($\emptyset, C, AC$);
4  func FD($D, C = \{c_1..c_q\}, AC$) : $diagnosis\ \Delta$
5  if $D \neq \emptyset$ and $consistent(AC)$ return $\emptyset$;
6  if $singleton(C)$ return $C$;
7  $k = \frac{q}{2}$;
8  $C_1 = \{c_1..c_k\}; C_2 = \{c_{k+1}..c_q\}$;
9  $D_1 = FD(C_2, C_1, AC - C_2)$;
10 $D_2 = FD(D_1, C_2, AC - D_1)$;
11 $return(D_1 \cup D_2)$;
```

How FASTDIAG determines minimal diagnoses can best be explained using our working example (see Table 3.4). The activation hierarchy for the FASTDIAG function is depicted in Figure 3.6. The algorithm checks whether the deletion of constraints (from AC) makes the remaining constraints consistent. For example, if we delete C_2 from AC in step 1 (Table 3.4) we are able to restore the consistency. At the same time we also know that there must be a diagnosis in C_2 since its deletion from *AC* contributed to the restoration of consistency. Furthermore, there is no need to search for a diagnosis in C_1.

step	D	C	AC	C_1	C_2	return
1	$\emptyset$	$\{c_1,...,c_5\}$	$\Gamma\cup\{c_1,...,c_5\}$	$\{c_1,c_2,c_3\}$	$\{c_4,c_5\}$	c_5
2	$\{c_4,c_5\}$	$\{c_1,c_2,c_3\}$	$\Gamma\cup\{c_1,c_2,c_3\}$	$\emptyset$	$\emptyset$	$\emptyset$
3	$\emptyset$	$\{c_4,c_5\}$	$\Gamma\cup\{c_1,...,c_5\}$	$\{c_4\}$	$\{c_5\}$	c_5
4	$\{c_5\}$	$\{c_4\}$	$\Gamma\cup\{c_1,...,c_4\}$	$\emptyset$	$\emptyset$	$\emptyset$
5	$\emptyset$	$\{c_5\}$	$\Gamma\cup\{c_1,...,c_5\}$	$\emptyset$	$\emptyset$	c_5

Table 3.4.: Example of FASTDIAG: $\Gamma = \{c_\alpha,c_\beta,c_\gamma,c_\delta,c_\varepsilon,c_\phi,c_\iota\}$ is the (original) background knowledge and $\Delta = \{c_5\}$ is the returned diagnosis. The activation sequence of the different FASTDIAG instances is depicted in Figure 3.6.

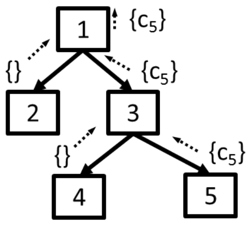


Figure 3.6.: Activation sequence of the different FASTDIAG instances (for the details see Table 3.4).

Algorithm Analysis. When comparing the performance of HSDAG with FASTDIAG we can see that FASTDIAG has the potential the reduce the number of needed consistency checks especially in scenarios where there is a need for identifying so-called leading diagnoses (i.e., not the complete set of diagnoses). In the worst case, FASTDIAG needs $2d \times log_2(\frac{n}{d}) + 2d$ where d is the set size of the minimal diagnosis and n is the number of constraints in C (Junker, 2004). The best case complexity with regard to the number of consistency checks is $log_2(\frac{n}{d}) + 2d$.

The efficiency of FASTDIAG can best be documented by the fact that the number of consistency checks needed for determining one diagnosis is similar to the number of consistency checks needed by QUICKXPLAIN to determine exactly one conflict set. Typically, there is more than one conflict set in an inconsistent configuration knowledge base. Let n_{cs} be the number of minimal conflict sets in a constraint set and n_{diag} be the number of minimal diagnoses, then we need exactly n_{diag} calls of the function *FD* (see Algorithm 3) plus n_{cs} additional consistency checks and n_{cs} activations of QUICKXPLAIN with n_{diag} additional consistency checks for determining all diagnoses. The results of a performance evaluation (comparison of the HSDAG-based approach with FASTDIAG) are depicted in Table 3.5.

domain	#con.	#var.	FASTDIAG (ms)			HSDAG (ms)		
$\#\Delta_i$			1	5	10	1	5	10
laptops	285	47	1638.7	2792.3	3464.1	2668.9	4977.6	5336.3
homes	73	55	593.1	2433.5	3167.8	2074.2	2151.4	2241.2
cars	150	73	1404.4	2730.8	3606.0	5741.1	6347.9	6942.0
printers	242	158	2871.9	6927.2	12091.0	>100k	>100k	>100k

Table 3.5.: Runtime evaluation: the average runtime in milliseconds (ms) needed by HSDAG and FASTDIAG to calculate one minimal diagnosis (on a standard PC). The basis for this evaluation are knowledge bases from www.splot-research.org (Dell laptops (laptops), smarthomes (homes), cars, and Xerox printers (printers).

3.5.3. Further Approaches

We now sketch two further approaches than can be used for the determination of diagnoses. First, we show how to represent the task of identifying minimal cardinality diagnoses as an optimization problem – this approach is based on the assumption that all minimal conflicts have been determined before the start of the diagnosis process – see, for example, (Fijany and Vatan, 2004). Second, we show how to determine minimal cardinality diagnoses in the simple case that the complete set of possible configurations is available – in this case, all diagnoses are known beforehand and the task is to identify the minimal ones – see, for example, (Jannach, 2008; Schubert et al., 2010; Schubert and Felfernig, 2011).

Diagnosis as Optimization Problem. We explain the approach to represent a diagnosis task as an optimization problem on the basis of the example depicted in Table 3.6. Each minimal conflict set CS_i is represented by a tuple that describes for each constraint c_i whether it is part of the conflict set or not (1 = c_i is part of the conflict set, 0 = c_i is *not* part of the conflict set).

Conflict Set	c_1	c_2	c_3	c_4	c_5
CS_1	1	0	0	1	1
CS_2	1	1	0	0	0
CS_3	1	0	1	0	0
CS_4	0	1	1	0	0

Table 3.6.: Representation of a diagnosis task as optimization problem – in this case, all minimal conflict sets ($CS_1,...,CS_4$) have to be determined before the optimization can start (1 (0) denotes the fact that c_i is part (not part) of the minimal conflict set).

On the basis of this information we are able to define a corresponding optimization problem. Each constraint c_i of Table 3.6 is represented by a corresponding variable x_i with $dom(x_i)$=$\{0,1\}$. The conflict sets CS_i are represented in the form of constaints cs_i, for example, $cs_1 : x_1 + x_4 + x_5 \geq 1$ requires that at least one constraint out of $\{x_1,x_4,x_5\}$ has to be *deactivated*, i.e., $x_i = 1$ denotes that fact that constraint c_i is inactive (which also means that the corresponding conflict has been resolved).

The complete set of constraints that corresponds to the conflict sets in Table 3.6 is the following.

$cs_1 : x_1 + x_4 + x_5 \geq 1.$

$cs_2 : x_1 + x_2 \geq 1.$

$cs_3 : x_1 + x_3 \geq 1.$

$cs_4 : x_2 + x_3 \geq 1.$

To complete the definition of the optimization problem, we introduce an optimization criterion. This criterion (see Formula 3.1) expresses the fact that the number of constraints part of at least one minimal conflict set and that are deactivated should be minimized. For further details on optimization-based diagnosis we refer the reader to (Fijany and Vatan, 2004).

$$minimize : x_1 + x_2 + x_3 + x_4 + x_5. \tag{3.1}$$

Filtering Minimal Cardinality Diagnoses. We explain the approach to determine minimal cardinality diagnoses in the case that all possible configurations are already available – see the information in Tables 3.7 and 3.8. Table 3.7 depicts the definition of a configuration task defined by a set of variables (V), their domains (dom(v_i)), and a constraint c_p that describes the set of possible configurations ($c_p = conf_1 \vee conf_2 \vee conf_3 \vee conf_4$).

Variables	$conf_1$	$conf_2$	$conf_3$	$conf_4$
v_1	1	3	1	1
v_2	2	2	2	2
v_3	3	1	2	1

Table 3.7.: A simple configuration problem defined by the variables $V = \{v_1, v_2, v_3\}$, dom(v_i) = $\{1,2,3\}$, and the constraint $c_p = conf_1 \vee conf_2 \vee conf_3 \vee conf_4 \in C_{KB}$.

and each configuration ($conf_i$) whether the requirement is supported by $conf_i$ or not (see Table 3.8). The *support* value indicates how many of the user requirements are supported by the configuration, for example, configuration $conf_1$ supports the user requirement $v_1 = 1$ but not the remaining ones; consequently the support of $conf_1$ = 1. It should be clear now that we are interested in diagnoses for the given set of requirements, i.e., which is the minimal (cardinality) set of requirements that have to be deleted such that a solution can be identified.

On the basis of the intermediate representation (see Table 3.8) it is straightforward to determine, for example, one minimal cardinality diagnosis. The configuration $conf_i$ with the maximum support also defines a minimal cardinality diagnosis Δ since there does not exist a diagnosis Δ' with $|\Delta'| < |\Delta|$ (if this would be the case then Δ would not have the maximum support value). In our working example, the only minimal cardinality diagnosis is $\Delta = \{v_2 = 1\}$.

User Requirements	$conf_1$	$conf_2$	$conf_3$	$conf_4$
$v_1 = 1$	1	0	1	1
$v_2 = 1$	0	0	0	0
$v_3 = 1$	0	1	0	1
support	1	1	1	2

Table 3.8.: Example user requirements C_R and their relationship to the configurations $conf_1, ..., conf_4$ (1 = requirement supported, 0 = not supported).

3.6. Conclusion

In this chapter, we sketched major concepts, techniques, and algorithms that support anomaly management in constraint-based application development, especially configurator application development. These concepts are the basis for other chapters in this thesis (see the Chapter 4 and Chapter 6).

Chapter 4

Personalized Diagnosis for Over-Constrained Problems

This chapter is based on the results documented in Felfernig et al. (2013a).

The author of this thesis contributed the *development of the ensemble-based approach* and *wrote major parts of the above mentioned paper.*

4.1. Abstract

Constraint-based applications such as configurators, recommenders, and scheduling systems support users in complex decision making scenarios. Typically, these systems try to identify a solution that satisfies all articulated user requirements. If the requirements are inconsistent with the underlying constraint set, users have to be actively supported in finding a way out from the *no solution could be found* dilemma. In this chapter we introduce techniques that support the calculation of personalized diagnoses for inconsistent constraint sets. These techniques significantly improve the diagnosis prediction quality compared to approaches based on the calculation of minimal cardinality diagnoses. In order to show the applicability of our approach we present the results of an empirical study and a corresponding performance analysis.

4.2. Introduction

Constraint-based applications such as configurators, recommenders, and scheduling systems support users in complex decision making scenarios. Interacting with constraint-based applications often means to *specify* a set of requirements (e.g., when interacting with a car configurator, required components such as *car type* and *park distance control*), to *adapt* inconsistent requirements, and to *evaluate* different alternative solutions. In this chapter we focus on situations where the constraint solver is not able to identify a solution and it is difficult for the user (customer) to identify minimal sets of requirements that need to be changed such that a solution for the underlying constraint satisfaction problem (CSP) can be identified. In order to improve the prediction quality of diagnosis algorithms in such contexts, we show how to exploit personalization techniques (Felfernig et al. (2007b)).

Existing approaches to the determination of diagnoses for inconsistent requirements are primarily focusing on *minimal-cardinality diagnoses* (Felfernig et al. (2004)) which are determined on the basis of breadth-first search. In the context of recommender systems (Felfernig et al. (2007b)) the complement of a diagnosis is denoted as *maximally successful sub-query* (Godfrey (1997); McSherry (2004)). Such a maximally successful subquery contains maximal sets of elements (requirements) that guarantee the identification of a solution, i.e., elements which are not part of the minimal diagnosis. In the context of constraint-based systems (Tsang (1993)) diagnoses are also interpreted as a specific type of *explanation* (O'Sullivan et al. (2007)).

Especially in interactive settings the determination of all diagnoses is infeasible due to unacceptable run times of the underlying diagnosis algorithms (Felfernig et al. (2009a)). Furthermore, we are not able to guarantee that standard breadth-first search leads us to explanations that are acceptable for the user (O'Sullivan et al. (2007)). The work of (O'Sullivan et al. (2007)) contributes to the tailoring of diagnoses in a way that makes the identification of acceptable diagnoses easier for the user – O'Sullivan et al. (2007) denote this type of diagnosis *representative explanations*. Representative explanations are diagnosis sets that fulfill the criteria that each element contained in at least one diagnosis is also contained in the set of diagnoses presented to the user. Felfernig et al. (2009a) show how to exploit concepts of collaborative recommendation for improving diagnosis prediction quality – the concepts have been developed for a knowledge-based recommendation environment (Burke (2000)).

On the basis of this existing work, we show how to exploit different recommendation algorithms for the personalized identification of diagnoses. In our approach we exploit these algorithms for guiding best-first search in the construction of Hitting Set Directed Acyclic Graphs (HSDAGs). The major contribution of this chapter is the significant improvement of *diagnosis prediction quality* by the integration of state-of-the-art recommendation approaches (similarity-based, utility-based, probability-based, ensemble-based) with standard model-based diagnosis (Reiter (1987); DeKleer et al. (1992)). Furthermore, we provide an empirical evaluation on the basis of two configuration datasets.

The remainder of this chapter is organized as follows. In Section 4.3 we introduce a working example from the domain of car configuration. In Section 4.4 we show how recommendation algorithms can be exploited for personalized model-based diagnosis. In Section 4.5 we present the results of evaluations conducted with two datasets. In Section 4.6 we discuss related work. The chapter is concluded with Section 4.7.

4.3. Working Example

The configuration of cars will serve for illustration purposes throughout this chapter. A configuration task can be defined as a constraint satisfaction problem (CSP) (Tsang (1993)):[*]

Definition 1 (Configuration Task). A configuration task can be defined as a CSP (V, D, C). V = $\{v_1, v_2, \ldots, v_n\}$ represents a set of finite domain variables. D = {dom(v_1), dom(v_2), …, dom(v_n)} represents a set of variable domains dom(v_k) where dom(v_k) represents the domain of variable v_k. C = $C_{KB} \cup C_R$ where $C_{KB} = \{c_1, c_2, \ldots, c_q\}$ is a set of domain specific constraints (the configuration knowledge base) that restrict the possible combinations of values assigned to the variables in V. C_R = $\{c_{q+1}, c_{q+2}, \ldots, c_t\}$ is a set of customer requirements also represented as constraints.

A simple example of a configuration task is the following. The variable *type* represents the car type, *pdc* is the parc distance control feature, *fuel* represents the average fuel consumption per 100 kilometers, a *skibag* supports a convenient ski stowage inside a car, and *4-wheel* represents the actuation type (4-wheel supported or not supported). These variables are representing all possible user (customer) requirements. The possible combinations of customer requirements are restricted by C_{KB} which is in our case $\{c_1, c_2, c_3, c_4, c_5\}$. Finally, we assume C_R to be $\{c_6, c_7, c_8, c_9, c_{10}\}$.

- *V* = {***type***, ***fuel***, ***skibag***, ***4-wheel***, ***pdc***}
- *D* = {dom(***type***)={*city*, *limo*, *combi*, *xdrive*}, dom(***fuel***) = {*4l*, *6l*, *10l*}, dom(***skibag***)={*yes*, *no*}, dom(***4-wheel***)={*yes*, *no*}, dom(***pdc***)= {*yes*, *no*}}
- C_{KB} = {c_1: ***4-wheel*** = *yes* ⇒ ***type*** = *xdrive*, c_2: ***skibag*** = *yes* ⇒ ***type*** ≠ *city*, c_3: ***fuel*** = *4l* ⇒ ***type*** = *city*, c_4: ***fuel*** = *6l* ⇒ ***type*** ≠ *xdrive*, c_5: ***type*** = *city* ⇒ ***fuel*** ≠ *10l*}
- C_R = {c_6: ***4-wheel*** = *yes*, c_7: ***fuel*** = *6l*, c_8: ***type*** = city, c_9: ***skibag*** = yes, c_{10}: ***pdc*** = yes}

On the basis of this simple example of a configuration task, we can now introduce the definition of a corresponding *configuration* (solution to a configuration task).

Definition 2 (Configuration). A configuration for a given configuration task (V, D, C) is an instantiation I = {v_1=ins_1, v_2=ins_2, …, v_n=ins_n} where $\mathrm{ins}_k \in \mathrm{dom}(\mathrm{v}_k)$.

A solution (configuration) for a given configuration task is *consistent* if the assignments in I are consistent with the $\bigcup \mathrm{c}_i \in$ C. A solution is *complete* if all $v_i \in$ V are instantiated. Finally, a solution is *valid* if it is consistent and complete.

[*]Note that the presented concepts are applicable to different knowledge representations such as SAT solving (Marques-Silva and Sakallah (1996)) and description logics (Friedrich and Shchekotykhin (2005)).

4.4. Calculating Personalized Diagnoses

Users do not want and are not able to evaluate large sets of diagnosis alternatives. For this reason we are now introducing alternative approaches that help to systematically reduce the number of diagnosis alternatives. Our goal is to identify diagnoses that are relevant for users and thus *keep the process of evaluating and selecting diagnoses as simple as possible*. The first approach to reduce the number of diagnoses (which is the only non-personalized one we consider here) is to perform *breadth first search* which returns minimal cardinality diagnoses first (Reiter (1987)). In addition to this breadth first search approach we will discuss four approaches to the *personalized ranking of diagnoses*: *similarity-based*, *utility-based*, *probability-based*, and *ensemble-based* search.

Cardinality-based diagnosis (not personalized). Our example configuration task (car configuration) is defined in a way which does *not* allow the calculation of a solution, for example, the requirements c_6 and c_8 are incompatible. For identifying minimal sets of constraints which have to be deleted from the given set of customer requirements we use the concepts of Model-Based Diagnosis (MBD) (Reiter (1987); DeKleer et al. (1992)). MBD diagnosis exploits the description of a system – in our case the configuration knowledge base C_{KB} which describes a set of possible configurations (solutions). If we detect that the behavior of the system conflicts with its intended behavior (at least one solution can be identified), the task of a diagnosis component is to determine components (constraints) in the given set of customer requirements (C_R) which, when assumed to function abnormally, sufficiently explain the discrepancy between actual and expected system behavior. An identified *minimal diagnosis* is a minimal set of faulty constraints that need to be relaxed or deleted in order to be able to calculate a configuration.

Assuming the existence of $C_{KB} = \{c_1, c_2, ..., c_q\}$ and $C_R = \{c_{q+1}, c_{q+2}, ..., c_t\}$ which is *inconsistent* with C_{KB}, breadth first search based diagnosis algorithms (Reiter (1987); DeKleer et al. (1992)) determine minimal diagnoses $DIAGS = \{\Delta_1, \Delta_2, ..., \Delta_k\}$ in the order of their cardinality such that $\forall \Delta_i \in DIAGS : C_{KB} \cup (C_R - \Delta_i)$ is consistent. A *User Requirements Diagnosis Problem (UR Diagnosis Problem)* can be defined as follows:

Definition 3 (User Requirements (UR) Diagnosis Problem): A UR Diagnosis Problem is defined as a tuple (C_{KB}, C_R); C_{KB} represents the constraints of the configuration knowledge base and C_R is a set of user requirements.

Based on the definition of a UR Diagnosis Problem, a *UR Diagnosis* can be defined as follows:

Definition 4 (UR Diagnosis): A User Requirements Diagnosis (UR Diagnosis) for (C_{KB}, C_R) is a set of constraints $\Delta \subseteq C_R$ such that $C_{KB} \cup (C_R - \Delta)$ is consistent. A diagnosis Δ is minimal iff there does not exist a diagnosis $\Delta' \subset \Delta$ s.t. $C_{KB} \cup (C_R - \Delta')$ is consistent.

Following the basic principles of Model-Based Diagnosis (MBD) (Reiter (1987); DeKleer et al. (1992)), the calculation of diagnoses is based on the identification and resolution of conflict sets. A *conflict set* in C_R can be defined as follows:

Definition 5 (UR Conflict Set): A User Requirements Conflict Set (UR Conflict Set) is defined as $CS \subseteq C_R$ s.t. $CS \cup C_{KB}$ is inconsistent. CS is minimal iff there does not exist a conflict set CS' with CS' $\subset$ CS.

In our working configuration example, $C_R = \{c_6,..,c_{10}\}$ is inconsistent with $C_{KB} = \{c_1,..,c_5\}$, i.e., there does not exist a configuration (solution) that completely fulfills the requirements in C_R. The *minimal conflict sets* are $CS_1 = \{c_6, c_7\}$, $CS_2 = \{c_8, c_9\}$, and $CS_3 = \{c_6, c_8\}$ since each of these conflict sets is inconsistent with C_{KB} and there do not exist conflict sets CS_1', CS_2', and CS_3' with $CS_1' \subset CS_1$, $CS_2' \subset CS_2$, and $CS_3' \subset CS_3$.

In MBD (Reiter (1987); DeKleer et al. (1992)), the standard algorithm for determining minimal diagnoses is the *hitting set directed acyclic graph* (HSDAG). UR diagnoses $\Delta_i \in DIAGS$ are determined by conflict resolution in the set of requirements C_R. Due to its minimality property, one conflict can simple be resolved by deleting one of the elements from the conflict set. After one element has been retracted from each of the given conflict sets, we are able to present a corresponding diagnosis. The original HSDAG approach employs breadth-first search. In our example, the diagnoses derived from CS_1, CS_2, and CS_3 are $DIAGS = \{\Delta_1 : \{c_6, c_8\}, \Delta_2 : \{c_6, c_9\}, \Delta_3 : \{c_7, c_8\}\}$.

The HSDAG construction for our working example is shown in Figure 7.1. In our implementation we employ the QUICKXPLAIN conflict detection algorithm which has been developed by Junker (2004). Following a strict *breadth first* search regime, we resolve the first conflict set (CS_1) by checking whether one of its elements already represents a diagnosis. Both alternatives (c_6 and c_7) do not lead to a diagnosis due to the inconsistency of (C_R - $\{c_6\}$) $\cup$ C_{KB} and (C_R - $\{c_7\}$) $\cup$ C_{KB}. The next minimal conflict set returned by QUICKXPLAIN is $CS_2 = \{c_8, c_9\}$. C_R - ($\{c_6\} \cup \{c_8\}$)) $\cup$ C_{KB} allows the determination of a solution; consequently we have identified the first minimal diagnosis: $\Delta_1 = \{c_6, c_8\}$.

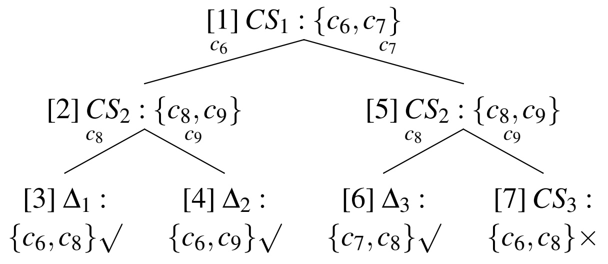

[1] $CS_1 : \{c_6, c_7\}$
c_6
c_7
[2] $CS_2 : \{c_8, c_9\}$
c_8
c_9
[5] $CS_2 : \{c_8, c_9\}$
c_8
c_9
[3] $\Delta_1 :$ $\{c_6, c_8\}\surd$
[4] $\Delta_2 :$ $\{c_6, c_9\}\surd$
[6] $\Delta_3 :$ $\{c_7, c_8\}\surd$
[7] $CS_3 :$ $\{c_6, c_8\}\times$

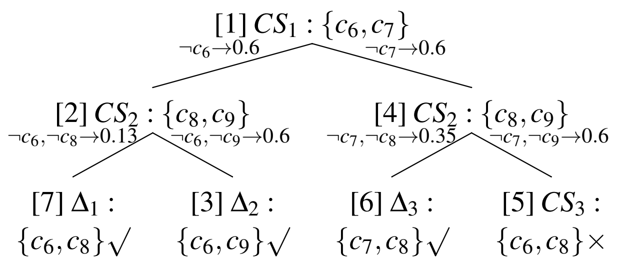
Figure 4.1.: Cardinality-based diagnosis (breadth-first): the diagnoses ranking is $\{\Delta_1, \Delta_2, \Delta_3\}$. The expression $[\{c_6, c_8\} \times]$ denotes *containment*, i.e., the node can be closed.

Similarity-based diagnosis. The similarity-based ranking of diagnoses is based on the idea of preferring minimal diagnoses which lead to solutions (configurations) that resemble the orginial set of requirements as much as possible. In this context we exploit the information contained in already existing configurations (see, e.g., Table 4.1). For each configuration contained in this table we deter-

Algorithm 4 PDIAG(C_R, C_{KB}, H, *crit*): Δ

```
{$C_R$: user requirements}
{$C_{KB}$: configuration knowledge base}
{$H$: paths of the diagnosis search tree (initially empty)}
{$crit$: expansion criteria ($card$, $sim$, $utility$, $prob$)}
{$\Delta$: identified diagnosis}
$\Delta \leftarrow first(H)$
$CS \leftarrow TP((C_R - \Delta) \cup C_{KB})$
**if** $isEmpty(CS)$ **then**
    $return\ \Delta$
**else**
    **for all** $X\ in\ CS$ **do**
        $H \leftarrow H \cup \{\Delta \cup \{X\}\}$
    **end for**
    $H \leftarrow delete(\Delta, H)$
    $H \leftarrow sort(H, crit)$
    PDIAG($C_R$, $C_{KB}$, $H$, $crit$)
**end if**
```

mine its similarity with the given set of requirements – the similarity values of our working example are depicted in Table 4.3. The similarity-based determination of diagnoses is based on Algorithm 1 – a generic algorithm which is applicable with different node expansion strategies (in our case, *cardinality*, *similarity*, *utility*, and *probability-based* search).



[1] $CS_1 : \{c_6, c_7\}$
$\neg c_6 \rightarrow 0.6$
$\neg c_7 \rightarrow 0.6$
[2] $CS_2 : \{c_8, c_9\}$
[4] $CS_2 : \{c_8, c_9\}$
$\neg c_6, \neg c_8 \rightarrow 0.13$
$\neg c_6, \neg c_9 \rightarrow 0.6$
$\neg c_7, \neg c_8 \rightarrow 0.35$
$\neg c_7, \neg c_9 \rightarrow 0.6$
[7] Δ_1 : $\{c_6, c_8\} \surd$
[3] Δ_2 : $\{c_6, c_9\} \surd$
[6] Δ_3 : $\{c_7, c_8\} \surd$
[5] CS_3 : $\{c_6, c_8\} \times$

Figure 4.2.: Similarity-based diagnosis: the diagnoses order is $\{\Delta_2, \Delta_3, \Delta_1\}$. The term $\neg c_6 \rightarrow 0.6$ denotes the fact that the highest similarity between C_R and the tuples of the sessions s_i in Table 1 consistent with $\neg c_6$ is 0.6 (in our case $i = 1$).

We determine similarity values based on three different *attribute-level similarity measures* which are predominantly applied in knowledge-based recommender applications (McSherry (2004)). The attribute-level measures determine the similiarity of each attribute value a_i of session s_k and the corresponding requirement c_i (e.g., the similarity between the attribute *type* of session s_1 and the corresponding requirement c_8). Depending on the characteristics of the attribute, one of the following attribute-level similarity measures has to be selected (see Formulae 4.1–4.3): *More-Is-Better* (MIB), *Less-Is-Better* (LIB) or *Nearer-Is-Better* (NIB) (McSherry (2004)). The overall similiarity between $c = C_R$ and a tuple a in Table 1 is defined by Formula 4.4. The similarity between C_R (c) and a tuple

Table 4.1.: Example user interaction data from already completed configuration sessions.

SESSION s_i	TYPE	FUEL	SKIBAG	4-WHEEL	PDC
s_1	city	4l	no	no	yes
s_2	city	4l	no	no	no
s_3	xdrive	10l	yes	yes	yes
s_4	limo	6l	no	no	yes
s_5	combi	6l	no	no	no
s_6	xdrive	10l	no	yes	yes
s_7	limo	6l	yes	no	no
s_8	combi	6l	yes	no	no

Table 4.2.: Example importance values ($w(c_i)$ in %).

TYPE	FUEL	SKIBAG	4-WHEEL	PDC
50.0	5.0	10.0	30.0	5.0

a of user interaction data is represented by the sum of weighted ($w(c_i)$ – see Table 2) attribute level similarities.[†]

For attributes such as *fuel*, the lower the value the more satisfied the user is (LIB). When the user specifies a certain car type (no intrinsic value scale), we suppose the most similar is the preferred one. In such cases, the nearer-is-better (NIB) similarity measure is used.[‡]

$$MIB : s(c_i, a_i) = \frac{val(c_i) - min(a_i)}{max(a_i) - min(a_i)} \tag{4.1}$$

$$LIB : s(c_i, a_i) = \frac{max(a_i) - val(c_i)}{max(a_i) - min(a_i)} \tag{4.2}$$

$$NIB : s(c_i, a_i) = \begin{cases} 1 & if\ val(c_i) = val(a_i) \\ 0 & else \end{cases} \tag{4.3}$$

$$sim(c, a) = \sum_{i=1}^{n} \mathrm{s}(c_i, a_i) * w(c_i) \tag{4.4}$$

[†]Our preference ($w(c_i)$) determination method is typically based on *multi attribute utility theory* (Winterfeldt and Edwards (1986)).

[‡]For a detailed discussion of different types of similarity measures see, for example, (McSherry (2004)). In Formulae 4.1 – 4.3, $val(c_i)$ denotes the value of user requirement c_i, $min(a_i)$ denotes the minimal possible value of configuration attribute a_i, and $max(a_i)$ denotes the maximal possible value of a_i.

Table 4.3.: Similarity ($sim(c,a)$) between requirements ($c = C_R$) and user interaction data (a) from configuration sessions.

s_i	TYPE	FUEL	SKIBAG	4-WHEEL	PDC	*sim(c,a)*
s_1	1.0	1.0	0.0	0.0	1.0	0.6
s_2	1.0	1.0	0.0	0.0	0.0	0.55
s_3	0.0	0.0	1.0	1.0	1.0	0.18
s_4	0.0	0.67	0.0	0.0	1.0	0.08
s_5	0.0	0.67	0.0	0.0	0.0	0.03
s_6	0.0	0.0	0.0	1.0	1.0	0.35
s_7	0.0	0.67	1.0	0.0	0.0	0.13
s_8	0.0	0.67	1.0	0.0	0.0	0.13

Utility-based diagnosis. Utility-based diagnosis prefers minimal diagnoses that are predominantly composed of requirements which are of *low importance* (a low $w(c_i)$ value) for the customer (user). Based on the concepts of *multi attribute utility theory* (Winterfeldt and Edwards (1986)), individual importance values (see Table 4.2) of user requirements that are part of a diagnosis are summed up – the lower this sum, the lower is the overall importance of the parameters contained in the diagnosis and the higher is the ranking of the corresponding diagnosis. The function *utility*($C \subseteq C_R$) returns a utility value for each set C which is a subset of C_R (see Formula 4.5). Note that in the case of our working example the diagnosis ranking on the basis of the utility-based approach is the same as with the discussed similarity-based approach; therefore we omit a graphical representation of utility-based diagnosis search.

$$u(C \subseteq C_R) = \frac{1}{\sum_{c_i \in C} w(c_i)} \tag{4.5}$$

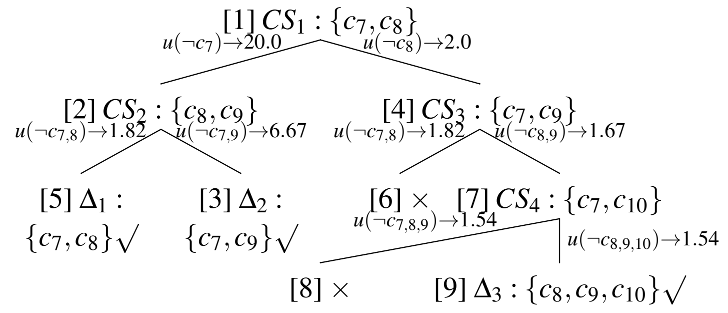

[1] $CS_1 : \{c_7, c_8\}$
$u(\neg c_7) \to 20.0$
$u(\neg c_8) \to 2.0$
[2] $CS_2 : \{c_8, c_9\}$
$u(\neg c_{7,8}) \to 1.82$
$u(\neg c_{7,9}) \to 6.67$
[4] $CS_3 : \{c_7, c_9\}$
$u(\neg c_{7,8}) \to 1.82$
$u(\neg c_{8,9}) \to 1.67$
[5] Δ_1 : $\{c_7, c_8\}\surd$
[3] Δ_2 : $\{c_7, c_9\}\surd$
[6] $\times$
[7] $CS_4 : \{c_7, c_{10}\}$
$u(\neg c_{7,8,9}) \to 1.54$
$u(\neg c_{8,9,10}) \to 1.54$
[8] $\times$
[9] $\Delta_3 : \{c_8, c_9, c_{10}\}\surd$

Figure 4.3.: Utility-based diagnosis search: the order of identified diagnoses is $\{\Delta_2, \Delta_3, \Delta_1\}$.

Probability-based diagnosis. Probability-based best first search for diagnoses prefers minimal diagnoses with a high *probability* of being selected by the user. For the determination of diagnosis probabilities we rely on joint probabilities that a particular diagnosis (or part of a diagnosis) will be

Table 4.4.: Example diagnoses selected by users, the individual probabilities are: p($\neg c_6$)=0.30, p($\neg c_7$)=0.50, p($\neg c_8$)=0.60, p($\neg c_9$)=0.20 where, for example, p($\neg c_6$)=0.30 denotes the probability of c_6 being part of a diagnosis.

Δ_i	TYPE	FUEL	SKIBAG	4-WHEEL	PDC
Δ_{log1}	≠city	–	–	=no	–
Δ_{log2}	–	–	=no	=no	–
Δ_{log3}	≠city	≠6l	–	–	–
Δ_{log4}	≠xdrive	–	–	–	–
Δ_{log5}	–	–	=no	=no	–
Δ_{log6}	≠city	≠6l	–	–	–
Δ_{log7}	≠city	≠6l	–	–	–
Δ_{log8}	≠xdrive	≠4l	–	=yes	–
Δ_{log9}	≠city	≠6l	–	–	–
Δ_{log10}	≠city	≠6l	–	–	–

selected by the user. Formula 4.6 is used for determining the joint probabilities for a given set of constraints $C \subseteq C_R$. Figure 4.4 shows the application of this approach in the context of our working example. The probabilities are determined on the basis of user-selected diagnoses (see Table 4.4). The assumption of *independence of failure* made here is one widely made in model-based diagnosis (DeKleer (1990)).

$$p(C \subseteq C_R) = \prod_{c_i \in C} p(c_i) \tag{4.6}$$

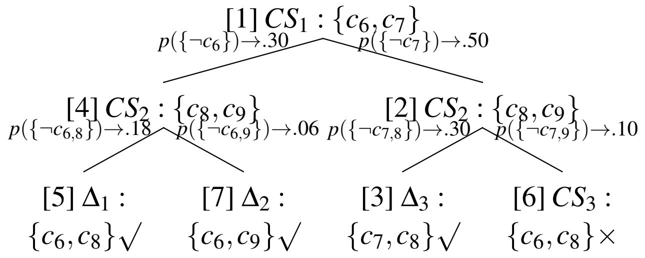

[1] $CS_1 : \{c_6, c_7\}$
$p(\{\neg c_6\}) \rightarrow .30$
$p(\{\neg c_7\}) \rightarrow .50$
[4] $CS_2 : \{c_8, c_9\}$
$p(\{\neg c_{6,8}\}) \rightarrow .18$
$p(\{\neg c_{6,9}\}) \rightarrow .06$
[2] $CS_2 : \{c_8, c_9\}$
$p(\{\neg c_{7,8}\}) \rightarrow .30$
$p(\{\neg c_{7,9}\}) \rightarrow .10$
[5] $\Delta_1 : \{c_6, c_8\}\surd$
[7] $\Delta_2 : \{c_6, c_9\}\surd$
[3] $\Delta_3 : \{c_7, c_8\}\surd$
[6] $CS_3 : \{c_6, c_8\}\times$

Figure 4.4.: Probability-based diagnosis: the order of identified diagnoses is $\{\Delta_3, \Delta_1, \Delta_2\}$.

Ensemble-based diagnosis. In the case of cardinality-based, similarity-based, utility-based, and probability-based diagnosis search, diagnosis predictions (the rankings) are based on a single hypothesis. The idea of ensemble-based diagnosis search is to exploit a set of hypotheses (an ensemble) for making the prediction. For the ranking of diagnoses we apply a basic *majority voting* approach (see Table 4.5); assuming that the errors made by each individual prediction mechanism are not the same, ensemble-based methods can be very useful for improving the prediction quality (see Section 4).

Table 4.5.: Example diagnoses selected by ensemble method (implemented as a basic form of *majority voting*).

METHOD / POSITION	1	2	3
utility-based	Δ_2	Δ_3	Δ_1
probability-based	Δ_3	Δ_1	Δ_2
similarity-based	Δ_2	Δ_3	Δ_1
ensemble-based	Δ_2	Δ_3	Δ_1

4.5. Evaluation

Prediction Quality. We now demonstrate the improvements achieved by the application of our personalized diagnosis approaches on the basis of an empirical study with *two datasets*.

Dataset 1: Computer Configuration. This dataset has been composed on the basis of an online user experiment conducted at the Graz University of Technology. 415 subjects participated in the study (82,4% male and 17,6% female). Participants had to define their requirements (C_R) regarding a set of 12 *computer properties*. The task of the subjects was to define their product requirements (including requirement importance). After having completed the requirement specification phase, *each* participant was informed about the fact that no solution could be found. The configurator then presented a list of max. 50 different repair configurations (solution alternatives) – at least one property of each configuration was inconsistent with C_R. The configurations were extracted from *www.dell.at*. The ranking of the solution alternatives was randomized and the subjects were enabled to navigate in order to evaluate the solution alternatives regarding criteria such as *price*, *harddisk size* or *number of fulfilled requirements*. The subjects then had to select one out of the presented repair configurations that appeared to be the most acceptable one for them. Since no solution has been made available for C_R (only configurations inconsistent with C_R were shown) we could calculate conflicts (in C_R induced by the repair configurations) and the corresponding diagnoses with our diagnosis techniques (the average number of diagnoses per C_R was: 5.32 (std.dev. 1.67)). *Precision* (see Formula 4.7) was measured then in terms of how often a repair configuration selected by the participant was consistent with one of the top-N ranked diagnoses.

Dataset 2: Financial Services. This dataset belongs to a financial service configurator we deployed at *www.hypo-alpe-adria.at*. In this commercial application, inconsistent states and selected diagnoses had been recorded (N=1.703 sessions out of which in 418 sessions a diagnosis process had been activated – average number of diagnoses per C_R: 20.42 (std.dev. 4.51)). In the case of the financial services configurator, the importance of a user requirement ($w(c_i)$) was (is) determined on the basis of *multi attribute utility theory* (Winterfeldt and Edwards (1986)).

Based on the two datasets (*computer* and *financial services*) we evaluated our diagnosis approaches w.r.t. their *precision* (Formula 4.7). The idea of this precision measure is to figure out how often a diagnosis that corresponds to a *diagnosis selected by the user* or leads to a *repair configuration* *selected by a user* is among the top-n ranked diagnoses.

$$precision = \frac{\#(correct\ predictions)}{\#(predictions)} \tag{4.7}$$

As can be seen in Table 4.6 and Table 4.7, in our two empirical settings the ensemble-based approach (majority voting) outperforms the other prediction methods in terms of *precision*. Breadth first search has the lowest precision. Based on a *two-sample t-test* we tried to figure out whether there exist statistically significant differences between the diagnosis approaches in terms of their *mean square error* ($\frac{1}{n}\sum_{i=1}^{n}|(1-diagpos(i))^2|$) where *diagpos* denotes the position of the diagnosis selected by the user and *n*=#(*diagnosis processes started*). In both datasets we detected a significant difference between breadth-first search and all other approaches (computer: $p = 2.2e^{-16}$, financial services: $p < 0.05$). Furthermore, there is a significant difference between the ensemble-based and the other personalized approaches in the case of the *computer* dataset ($p = 7.55e^{-6}$); in the case of the *financial services* dataset we can observe a tendency ($p < 0.07$).

Table 4.6.: Predictive quality (precision) of used diagnosis selection methods for *computer* configuration dataset.

METHOD / N	1	2	3	4	5
breadth first	0.55	0.76	0.82	0.88	0.96
utility-based	0.66	0.83	0.94	0.95	0.98
similarity-based	0.65	0.81	0.90	0.93	0.98
probability-based	0.64	0.85	0.93	0.95	0.99
ensemble-based	0.68	0.86	0.94	0.96	0.99

Table 4.7.: Predictive quality (precision) of used diagnosis selection methods for *financial services* configuration dataset.

METHOD / N	1	2	3	4	5
breadth first	0.12	0.27	0.39	0.52	0.62
utility-based	0.17	0.37	0.48	0.65	0.74
similarity-based	0.17	0.37	0.49	0.65	0.73
probability-based	0.15	0.33	0.47	0.57	0.74
ensemble-based	0.17	0.35	0.50	0.63	0.76

Performance. The PDIAG algorithm (Algorithm 1) has been implemented on the basis of the standard hitting set algorithm introduced in Reiter (1987) – it is NP-hard in the general case but is applicable for interactive configuration settings (see the following evaluation). In PDIAG minimal conflict sets are determined on the basis of QUICKXPLAIN (Junker (2004)) – the worst case complexity in terms of the number of consistency checks of QUICKXPLAIN is O($2k * log(\frac{n}{k}) + 2k$) where k represents the size of the minimal conflict set and n is the number of constraints.

We conducted a performance analysis in order to show the applicability of our approach (see Table 4.8). The tests have been executed on a standard desktop computer (*Intel®Core^{TM}2 Quad CPU*

Q9400 CPU with *2.66GHz* and *2GB* RAM). In addition to our datasets we evaluated our diagnosis algorithms with the Renault configuration knowledge base part of the *configuration benchmark suite*.[§] Even for complex settings (Renault benchmark) we can expect a performance acceptable for interactive settings.

Table 4.8.: Avg. runtime(*msec*) for determining the first-N diagnoses (bf = *breadth first*, pers=*all personalized approaches*).

DATASET	APPROACH	N=1	N=5
computer configuration	bf	64.1	70.0
computer configuration	pers	64.2	71.1
financial services	bf	23.3	36.8
financial services	pers	23.9	37.1
car (Renault)	bf	921.3	1510.1
car (Renault)	pers	952.7	1581.9

4.6. Related Work

The increasing size and complexity of knowledge bases led to the application of model-based diagnosis (Reiter (1987); DeKleer et al. (1992)) to automated knowledge base debugging (Felfernig et al. (2004)). The contribution of (Felfernig et al. (2004)) has a special relationship to the concepts presented in this chapter: (Felfernig et al. (2004)) identify faulty constraints in configuration knowledge bases, furthermore, they present a first approach to the identification of minimal sets of faulty user requirements (following a breadth-frist search regime). The work presented in this chapter extends existing research results by demonstrating the application of recommendation algorithms for improving the prediction quality of diagnosis algorithms. One approach to determine personalized diagnoses for inconsistent requirements has been proposed for knowledge-based recommendation scenarios (Felfernig et al. (2009a)) where repair proposals for inconsistent features requests are generated on the basis of the similarity between the feature requests and the items in a product table. O'Sullivan et al. (2007) introduce minimal exclusion sets. On the basis of such sets, O'Sullivan et al. (2007) discuss the concept of representative explanations which can be interpreted as sets of minimal diagnoses covering all constraints part of at least one of the existing diagnoses. In contrast to the work presented in this chapter, the approach of O'Sullivan et al. (2007) does not explicitly take into account the preferences of the current user (customer). In knowledge-based recommendation maximally successful sub-queries (Godfrey (1997); McSherry (2004)) represent the complement to minimal diagnoses (DeKleer et al. (1992); Reiter (1987)). Note that our work relies on the assumption of an *open configuration* (*trade-off exploration*) based scenario where the user is free to specify preferred requirements – the configuration system then provides the corresponding feedback in terms of diagnoses.

[§]www.itu.dk/research/cla/externals/clib.

4.7. Conclusions

We have introduced techniques that help to calculate *personalized diagnoses*. In this context we proposed different personalization strategies which can help to significantly increase prediction quality. Within the scope of an empirical study we compared five search strategies (cardinality-based, similarity-based, utility-based, probability-based, and ensemble-based). The results show clear advantages of personalized diagnosis calculation in terms of precision. Thus the results presented in this chapter provide a solid basis for improving existing constraint-based applications in terms of a lower number of needed interaction cycles for the user and a lower number of needed diagnosis calculations.

Chapter 5

WeeVis

This chapter is based on the results documented in Reiterer et al. (2014).

The author of this thesis *developed the examples mentioned in this chapter* and *wrote major parts of the above mentioned paper.*

5.1. Abstract

Configuration is a thriving application area for AI technologies. As a consequence, there is a need for advancing knowledge acquisition practices in order to make configuration technologies more accessible. In this chapter we introduce WEEVIS which is a freely available Wiki-based environment for defining and solving basic configuration tasks. In the line of the idea of Wikipedia, knowledge bases are regarded as Wiki pages which can be created, edited, and versionized by a community of users. WEEVIS configurators can be integrated into standard Wikipedia pages and are thus more easily accessible compared to proprietary knowledge representations. WEEVIS technologies are easily accessible and therefore well-suited for the application in educational contexts.

5.2. Introduction

In the line of the Wikipedia spirit of allowing communities to cooperatively develop and maintain Wiki pages, we introduce the WEEVIS environment that supports the development and maintenance of configuration knowledge bases on the basis of wide-spread and well-known Wiki technologies. WEEVIS[*] has been developed on the basis of MediaWiki (*www.mediawiki.org*) which is the established standard Wiki platform.

[*]www.weevis.org.

For a long period of time the engineering of configuration knowledge bases required that knowledge engineers are technical experts (in the majority of the cases computer scientists) with the needed technical capabilities (Stumptner, 1997; Fleischanderl et al., 1998). Recent developments in the field moved one step further and also provided graphical engineering environments (Felfernig, 2007; Tiihonen et al., 2013) which improve the accessibility and maintainability of knowledge bases. However, potential users of these technologies still have to deal with additional tools and technologies which is in many cases a reason for not applying configuration technologies.

The WEEVIS environment tackles this challenge by providing new technologies that allow users to develop and maintain knowledge bases without the need of getting acquainted with the specific properties of a configurator development environment and with a corresponding programming language. In its current version, WEEVIS supports scenarios where user requirements can be defined in terms of *functional requirements* (Mittal and Frayman, 1989) and the corresponding solutions (configurations) are typically selected from a set of pre-defined configurations. User requirements are checked with regard to their consistency and – in the case of inconsistent requirements – WEEVIS proposes a set of possible repair alternatives. This way, WEEVIS does not only support product selection but also consistency maintenance processes on the basis of intelligent repair mechanisms (Felfernig et al., 2009a). Currently, WEEVIS does not provide a graphical development environment for configuration knowledge bases but relies on a textual knowledge representation language. The idea is that a community of engineers (users) cooperatively contributes to the development of a configuration knowledge base. The inclusion of graphical concepts supporting community-based knowledge base development and maintenance is within the scope of future work. WEEVIS comes along with the following innovations:

- *Knowledge Acquisition.* WEEVIS users are enabled to articulate their domain knowledge by defining the basic properties of the configuration task in a Wikipedia style. WEEVIS provides a set of predefined tags that can be used for creating, defining, and maintaining a configuration problem definition. By exploiting the basic functionalities of the underlying MediaWiki infrastructure, configuration knowledge bases can be versioned and restored in the case of the creation of unintended versions.
- *Knowledge Distribution and Collaborative Development.* WEEVIS users can develop and publish their configuration knowledge to a community of interested users. Other users interested in contributing to the further development of the knowledge base can engage in the development and maintenance process as it is the case with standard Wiki pages. WEEVIS and WEEVIS knowledge bases are freely available and accessible in the Internet.
- *Rapid Prototyping.* By using the basic functionalities provided by Wikipedia, WEEVIS allows rapid prototyping processes where the result of a change can immediately be seen by simply switching from the *edit* mode to the corresponding *read* mode. This approach allows an easy understanding of the WEEVIS tags and also of the semantics of the provided WEEVIS language.

The major contributions of this chapter are the following. We introduce a Wiki-based environment for the development and maintenance of configuration problems. In this context we provide an overview of the functionalities supported by the WEEVIS environment. This serves as a basis for further application and exploitation.

The remainder of this chapter is organized as follows. On the basis of an example from the domain of *personal computers* (see (Hotz et al., 2014)) we introduce the WEEVIS knowledge representation concepts (Section 5.3). In Section 5.4 we give an overview of how WEEVIS is embedded in the Media Wiki environment. In Section 5.5 we discuss the limitations of the WEEVIS approach and shed light on related work by comparing existing engineering practices with the WEEVIS functionalities. With Section 5.6 we conclude the chapter.

5.3. Modeling of the Working Example

For demonstration purposes we will now show how to define the configuration model introduced in (Hotz et al., 2014) on the basis of the WEEVIS knowledge representation concepts. In this context we will see that not all of the semantic properties defined in that configuration model can be represent directly within WEEVIS. Note that due to the supported knowledge representation concepts, WEEVIS can also be interpreted as a basic constraint-based recommendation environment (see (Felfernig et al., 2006a)), however, in its current version specific personalization concepts such as solution ranking and personalized repair for inconsistent requirements (see (Felfernig et al., 2009a)) are not supported. The expressivity of WEEVIS (which properties can be represented?) is very similar to those of feature models (see (Hotz et al., 2014)), i.e., the upper bound of the multiplicity of the relationship between different variables (features) is restricted to 1. However, variables in WEEVIS are not restricted to the values 1 (*true*) and 0 (*false*). WEEVIS relies on a CSP-based knowledge representation – see, for example, (Mackworth, 1977; Tsang, 1993). Constraints are currently restricted to types typically used for configuration knowledge representation (e.g., compatibilities, incompatibilities, and requirement relationships). Restricted to such basic knowledge types, WEEVIS has not been designed for the development of large-scale and complex component-oriented configurator applications (see, e.g., (Hotz et al., 2014)). Reasonable upper bounds regarding the complexity of knowledge bases that can be developed in WEEVIS have to be figured out in future work.

In the following we provide a short overview of the knowledge representation concepts currently available in WEEVIS.

Customer Properties. Customer properties describe the way in which users are enabled to specify their requirements with regard to the configurable product. Examples of customer properties are the primary usage (*usage*), the degree of energy efficency (*efficiency*), the maximum price accepted by the user (*maxprice*), the resident country (country), the selected motherboard (*mb*), and the selected central processing unit (*cpu*). The list of customer properties used in this working example is shown in Table 5.1.

Customer Property	Domain
usage	Internet, Scientific, Multimedia
efficiency	A, B, C
maxprice	Integer
country	Austria, Germany, ...
mb	MBSilver, MBDiamond
cpu	CPUS, CPUD

Table 5.1.: Customer properties of working example. For example, *maxprice* allows the customer to specify his/her requirements regarding the upper bound of the price.

Product Properties. In WEEVIS, product properties are representing the basic properties of items that can be part of a solution. Examples of product properties in the personal computer domain are the name of the product (*pname*), type of the cpu (*pcpu*), the type of the motherboard (*pmb*), the type of operating system (*pos*), the price of the computer (*pprice*). The list of product properties used in our working example is shown in Table 5.2.

Product Property	Domain
pname	Text
pcpu	CPUS, CPUD
pmb	MBSilver, MBDiamond
pos	OSAlpha, OSBeta
pprice	Integer

Table 5.2.: Product properties of working example. For example, each computer configuration has a specified cpu type *pcpu* (*CPUS* or *CPUD*).

Incompatibility Constraints. These constraints describe exclusion relationships between different customer properties and/or product properties. For example, if the value of *efficiency* is *A* then the value of *mb* must not be *MBSilver*. Furthermore, if the value of *efficiency* is *C* then the value of *mb* must not be *MBDiamond*. The complete list of incompatibility constraints in our working example is shown in Table 5.3.

Attribute	Property	Attribute	Property
efficiency	A	mb	MBSilver
efficiency	C	mb	MBDiamond

Table 5.3.: Incompatibility constraints of working example. For example, a motherboard *MBSilver* is incompatible with the energy efficiency class *A*.

Filter Constraints. In WEEVIS, filter constraints are used for expressing the relationship between selected customer requirements (instantiations of customer properties) and the corresponding product properties. For example, the price of a solution must not exceed the maximum price defined by the user or the selected cpu should be contained in the configuration. The complete list of filter constraints used in this working example is shown in Table 5.4.

Attribute	Operator	Attribute
maxprice	$\geq$	price
cpu	=	pcpu
mb	=	pmb

Table 5.4.: Filter constraints of working example. For example, the *cpu* selected by the customer has to be included in the corresponding configuration.

Includes. This is an additional constraint type that allows the definition of *alternative inclusions* that are not taken into account by incompatibility and filter constraints. An example of such an inclusion constraint is the following: if the user is interested in a highly energy-efficient computer, include the newest model named *energystar* (even if the maximum price accepted by the user is lower than the price of *energystar*) – see Table 5.5.

Attribute	Property	*pname*
efficiency	A	energystar

Table 5.5.: Includes constraint of working example. For example, if a user is interested in an energy class *A* configuration, then the item (product) *energystar* should be included.

Excludes. In the line of includes constraints, WEEVIS also supports the definition of so-called excludes constraints. A solution could be excluded because the needed hardware is currently not available (e.g., pname = hw1) or a solution can not be delivered due to the fact that certain personal computers can not be delivered to all destination countries (e.g., country = Austria and pname = hw2). An overview of the excludes constraints is given in Table 5.6.

Attribute	Property	*pname*
-	-	hw1
country	Austria	hw2

Table 5.6.: Excludes constraint of working example. For example, if the destination country is *Austria* then the computer *hw2* is not allowed to be contained in a configuration. Furthermore, *hw1* is currently not available.

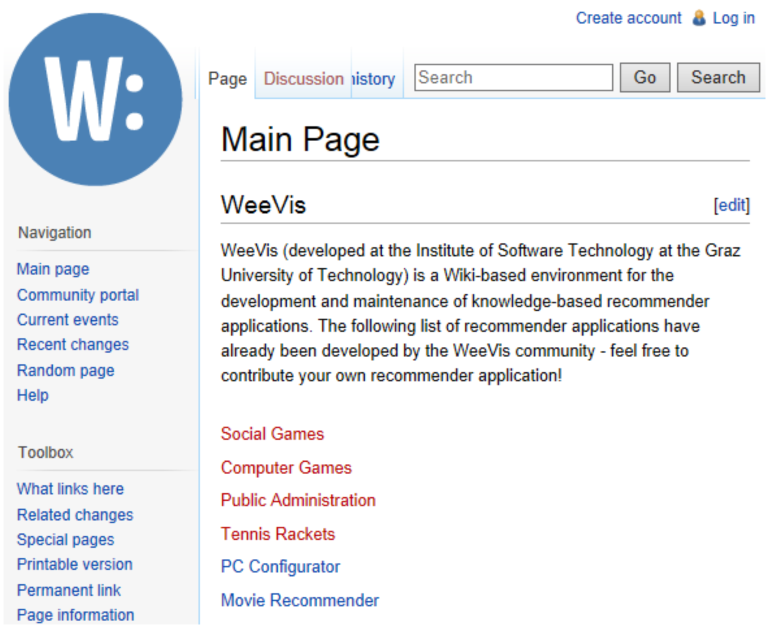
Products. In WEEVIS, possible solution alternatives can be defined in a product table (see Table 5.7). Using such a product table makes sense if the set of alternatives is low. This way, search can be made more efficient. In the case of a large search space, the explicit representation of solution alternatives makes knowledge engineering and the underlying search processes inefficient. In such a situation, WEEVIS also allows for the definition of the solution space in a implicit (constraint-based) form.

pname	pcpu	pmb	pos	pprice
hw1	CPUD	MBSilver	OSAlpha	600
hw2	CPUS	MBDiamond	OSBeta	600
engergystar	CPUS	MBDiamond	OSBeta	800

Table 5.7.: Supported solutions of working example.

5.4. User Interface

The entry page of the WEEVIS Wiki environment is depicted in Figure 5.1. The idea of this entry page is similar to the entry page of Wikipedia, i.e., to support the user in easily identifying the most relevant information units (in our context configuration-enhanced Wikipedia pages).



Main Page

WeeVis [edit]

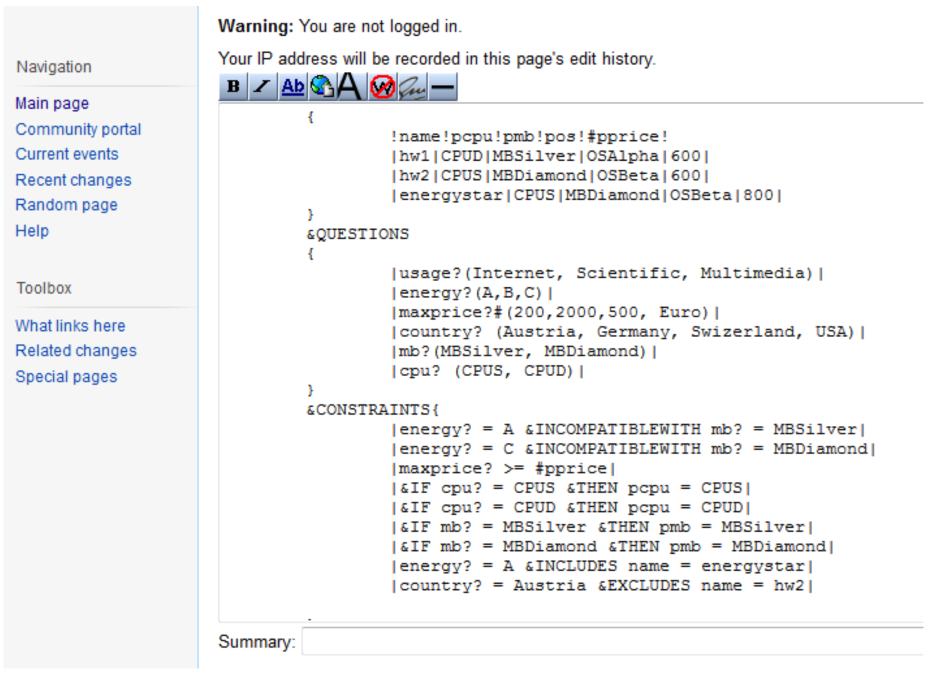
WeeVis (developed at the Institute of Software Technology at the Graz University of Technology) is a Wiki-based environment for the development and maintenance of knowledge-based recommender applications. The following list of recommender applications have already been developed by the WeeVis community - feel free to contribute your own recommender application!

Social Games
Computer Games
Public Administration
Tennis Rackets
PC Configurator
Movie Recommender

Figure 5.1.: The WeeVis entry page (www.selectionarts.org/weevis).

After having selected (or created) a configurator application (this is done by simply inserting a link to a new Wiki page), the new configurator application can be defined where the *MediaWiki Tag Extension System* has been used to define a custom *recommender tag*. WEEVIS configuration knowledge bases can be defined on a textual level with a syntax that is similar to the syntax of standard Wiki pages. A screenshot that shows the definition of WEEVIS configurator applications is depicted in Figure 5.2.

Finally, a configurator application can be activated by simply switching from the *edit* view to the corresponding *read* view. A screenshot of a corresponding WEEVIS configurator application is shown in Figure 5.3. The interface supports the definition of customer requirements (on the left hand side) – corresponding solutions are displayed on the right hand side. For each solution, a so-called support



Warning: You are not logged in.

Your IP address will be recorded in this page's edit history.

```
        {
                !name!pcpu!pmb!pos!#pprice!
                |hw1|CPUD|MBSilver|OSAlpha|600|
                |hw2|CPUS|MBDiamond|OSBeta|600|
                |energystar|CPUS|MBDiamond|OSBeta|800|
        }
        &QUESTIONS
        {
                |usage?(Internet, Scientific, Multimedia)|
                |energy?(A,B,C)|
                |maxprice?#(200,2000,500, Euro)|
                |country? (Austria, Germany, Swizerland, USA)|
                |mb?(MBSilver, MBDiamond)|
                |cpu? (CPUS, CPUD)|
        }
        &CONSTRAINTS{
                |energy? = A &INCOMPATIBLEWITH mb? = MBSilver|
                |energy? = C &INCOMPATIBLEWITH mb? = MBDiamond|
                |maxprice? >= #pprice|
                |&IF cpu? = CPUS &THEN pcpu = CPUS|
                |&IF cpu? = CPUD &THEN pcpu = CPUD|
                |&IF mb? = MBSilver &THEN pmb = MBSilver|
                |&IF mb? = MBDiamond &THEN pmb = MBDiamond|
                |energy? = A &INCLUDES name = energystar|
                |country? = Austria &EXCLUDES name = hw2|
```

Summary:

Figure 5.2.: WeeVis: example configurator knowledge base.

score is determined. If a solution fulfills all requirements, this score is 100%, otherwise it is lower and – when clicking on the score value – a corresponding repair action is displayed on the left hand side (see Figure 5.3).

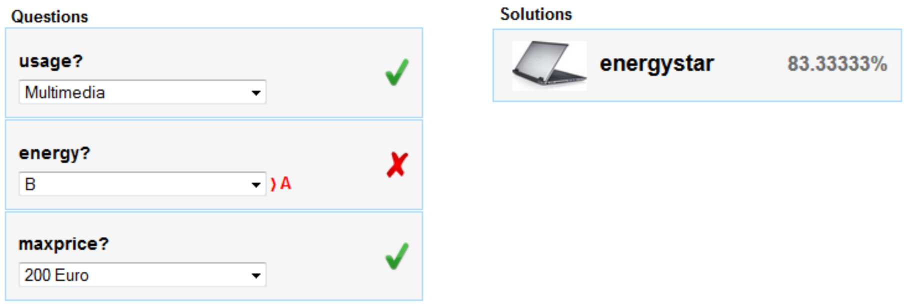


Figure 5.3.: WeeVis: example personal computer configurator. User specify their requirements in terms of providing answers to questions. WEEVIS determines solutions and – in the case of inconsistent requirements – presents a set of repair actions.

5.5. Related Work

Semantic Wikis. The integration of knowledge-based concepts into Wikipedia platforms has already been realized by different approaches to the development of so-called *Semantic Wikis*. For example, (Baumeister et al., 2011) introduce a Semantic Wiki based approach were domain ontologies can be specified as a basis of the corresponding wiki-based application. Domain ontologies are defined in terms of an ontology representation language. Compared to WEEVIS, (Baumeister et al., 2011) focus on the application of ontology representation languages which are less accessible to users who are not experts in the knowledge engineering field. Furthermore, the WEEVIS constraint representation is focused on configuration domain specific knowledge representations which makes it easier to define the corresponding domain knowledge.

Configuration Knowledge Representations. There is a long history of developing representation languages for configuration problems. One of the first approaches to a consistency-based characterization of a configuration problem has been proposed by (Mittal and Frayman, 1989). Thereafter different types of constraint-based representations have been developed, ranging from dynamic constraint satisfaction problems (see (Mittal and Falkenhainer, 1990)) to so-called generative constraint satisfaction problems – see (Stumptner et al., 1998). An in-depth overview of different types of basic configuration knowledge representations can be found in (Stumptner, 1997). (Felfernig, 2007) introduced an approach to define configuration knowledge bases on the basis of the Unified Modeling Language (UML) and (Felfernig et al., 2003) also show how to represent such models on the basis of description logics based concepts. (Soininen et al., 1998) introduce a general ontology that includes relevant concepts needed for configuration knowledge representation.

Configuration Environments. (Felfernig et al., 2006a) introduce an environment for the graphical development of knowledge-based recommender applications. This environment has commonalities with the basic functionalities of WEEVIS: product alternatives are explicitly specified and then selected and ranked on the basis of a given set of user requirements. (Tiihonen et al., 2013) introduce their answer-set programming based configuration environment WECOTIN which is an environment dedicated to the graphical definition of large-scale configuration knowledge bases. Major configuration-related constraints types can be defined on a graphical level. More complex constraints have to be defined in a proprietary logic-based knowledge representation language. Other graphical development environments are presented in (Haselböck and Schenner, 2014), (Krebs, 2014), and (Hotz and Günter, 2014). The major difference between WEEVIS and other configuration knowledge base development environments is that it can be expected that more users are able to deal with modeling configurators in Wikipedia compared to the alternative of defining knowledge bases with (provider-) specific environments. However, admittedly, the major drawback of WEEVIS is its limited expressiveness compared to more general knowledge representation formalisms (see (Hotz et al., 2014)). Furthermore, graphical constraint management facilities have to be integrated in order to achieve a wide-spread use.

Anomaly Management. Most of the existing configuration environments include different types of anomaly detection (see (Felfernig et al., 2014d)) or at least consistency management functionalities. In a similar fashion, WEEVIS includes a basic implementation of the FASTDIAG diagnosis algorithm (Felfernig et al., 2012b) which supports users in situations where no solution can be found. Furthermore, WEEVIS allows the specification of test cases which can be exploited for the purposes of regression testing. The integration of further quality assurance mechanisms such as the automated debugging of WEEVIS knowledge bases, redundancy detection (Felfernig et al., 2011a), and diagnosis discrimination (Shchekotykhin et al., 2012) is within the scope of future work.

5.6. Conclusion

In this chapter we have introduced the WEEVIS environment for the Wiki-based development and maintenance of simple configuration knowledge bases. WEEVIS is especially useful in scenarios were groups or communities of users are interested in commonly providing a configurator application to the whole public or to a specific group of users. From the technological point of view the major advantage of WEEVIS is the development and maintenance of configuration knowledge bases without the overhead of getting acquainted with a completely new technology. Major issues related to the further development of WEEVIS are the integration of personalization concepts (Ardissono et al., 2003; Tiihonen and Felfernig, 2010) that help to rank the set of determined solutions (configurations), the integration of additional quality assurance mechanisms (beside the basic versioning concept of Wikipedia), and the further development of the WEEVIS language itself in the sense of improving the understandability and reducing the cognitive complexity of the provided modeling concepts.

Chapter 6

An Overview of Direct Diagnosis and Repair Techniques in the WEEVIS Environment

This chapter is based on the results documented in Felfernig et al. (2014c).

The author of this thesis contributed the *related work*, *developed the working example and wrote major parts of the above mentioned paper.*

6.1. Abstract

Constraint-based recommenders support users in the identification of items (products) fitting their wishes and needs. Example domains are financial services and electronic equipment. In this chapter we show how divide-and-conquer based (direct) diagnosis algorithms (no conflict detection is needed) can be exploited in constraint-based recommendation scenarios. In this context, we provide an overview of the MediaWiki-based recommendation environment WEEVIS.

6.2. Introduction

Constraint-based recommenders (Felfernig et al. (2006b); Felfernig and Burke (2008)) support the identification of relevant items from large and often complex assortments. Example item domains are electronic equipment (Felfernig et al. (2006a)) and financial services (Felfernig et al. (2007a)). In contrast to collaborative filtering (Konstan et al. (1997)) and content-based filtering (Pazzani and Billsus (1997)), constraint-based recommendation relies on an explicit representation of recommendation knowledge. Two major types of knowledge sources are exploited for the definition of a constraint-based recommendation task (Felfernig and Burke (2008)). First, knowledge about the given set of customer requirements. Second, recommendation knowledge that is represented as a set of items and a set of constraints that help to establish a relationship between requirements and the item assortment.

Diagnosis techniques can be useful in the following situations: (1) in situations where it is not possible to find a solution for a given set of user (customer) requirements, i.e., the requirements are inconsistent with the recommendation knowledge base and the user is in the need for repair proposals to find a way out from the *no solution could be found* dilemma; (2) if a recommendation knowledge base is inconsistent with a set of test cases that has been defined for the purpose of regression testing, the knowledge engineer needs support in figuring out the responsible faulty constraints.

For situation (1) we sketch how model-based diagnosis (Reiter (1987)) can be applied for the identification of faulty constraints in a given set of customer requirements. In this context efficient divide-and-conquer based algorithms can be applied to the diagnosis and repair of inconsistent requirements. In a similar fashion, such algorithms can be applied for the diagnosis of inconsistent recommender knowledge bases (the knowledge base itself can be inconsistent, or alternatively, inconsistencies can be induced by test cases used for regression testing).

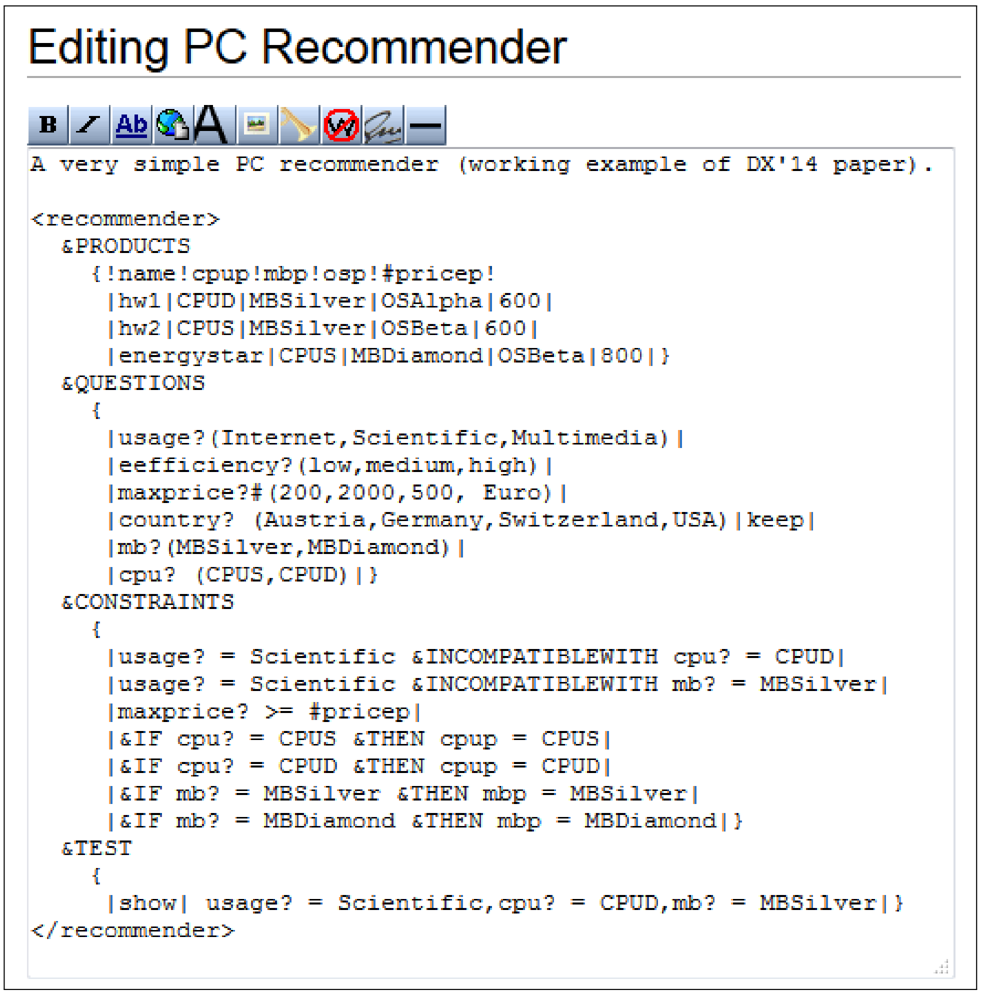
The diagnosis approaches presented in this chapter have been integrated into WEEVIS[*] which is a MediaWiki-based recommendation environment for complex products and services. In the line of the Wikipedia[†] idea to support communities of users in the cooperative development of Web content, WEEVIS is an environment that supports all the functionalities available for the creation of Wiki pages. Additionally, it allows the inclusion of constraint-based recommender applications that help to work up existing knowledge and present this in a compressed and intuitive fashion.

The contributions of this chapter are the following. First, we sketch how efficient divide-and-conquer based algorithms can be applied for solving diagnosis and repair tasks in constraint-based recommendation scenarios. Second, we sketch how diagnosis and repair approaches can be integrated into Wiki technologies[‡] and with this be made accessible to a large user group. Third, we discuss challenges for future research that have to be tackled to advance the state-of-the-art in constraint-based recommendation.

The remainder of this chapter is organized as follows. In Section 6.3 we discuss properties of constraint-based recommendation tasks. Thereafter, we introduce an example recommendation knowledge base. In Section 6.4 we show how divide-and-conquer based algorithms can be applied for the diagnosis and repair of inconsistent requirements. Thereafter we show how such algorithms can be applied to the identification of faulty constraints in knowledge bases (see Section 6.5). Related and future work are discussed in Section 6.6. We conclude the chapter with Section 6.7.

[*]www.weevis.org.
[†]www.wikipedia.org.
[‡]www.mediawiki.org.

6.3. Working Example


Editing PC Recommender

```
A very simple PC recommender (working example of DX'14 paper).

<recommender>
  &PRODUCTS
    {!name!cpup!mbp!osp!#pricep!
     |hw1|CPUD|MBSilver|OSAlpha|600|
     |hw2|CPUS|MBSilver|OSBeta|600|
     |energystar|CPUS|MBDiamond|OSBeta|800|}
  &QUESTIONS
    {
     |usage?(Internet,Scientific,Multimedia)|
     |eefficiency?(low,medium,high)|
     |maxprice?#(200,2000,500, Euro)|
     |country? (Austria,Germany,Switzerland,USA)|keep|
     |mb?(MBSilver,MBDiamond)|
     |cpu? (CPUS,CPUD)|}
  &CONSTRAINTS
    {
     |usage? = Scientific &INCOMPATIBLEWITH cpu? = CPUD|
     |usage? = Scientific &INCOMPATIBLEWITH mb? = MBSilver|
     |maxprice? >= #pricep|
     |&IF cpu? = CPUS &THEN cpup = CPUS|
     |&IF cpu? = CPUD &THEN cpup = CPUD|
     |&IF mb? = MBSilver &THEN mbp = MBSilver|
     |&IF mb? = MBDiamond &THEN mbp = MBDiamond|}
  &TEST
    {
     |show| usage? = Scientific,cpu? = CPUD,mb? = MBSilver|}
</recommender>
```

Figure 6.1.: Example WEEVIS PC Recommender definition (MediaWiki 'Edit' mode).

In the remainder of this chapter we will use *personal computer recommendation* as working example. Roughly speaking, a recommendation task consists of selecting those items that match the user requirements. In the context of personal computers, the recommender user has to specify his/her requirements regarding, for example, the intended usage, the maximum accepted price, and the cpu type. Since WEEVIS is a MediaWiki-based environment, the definition of a recommender knowledge base is supported in a textual fashion (see Figure 6.1).

On the basis of a set of requirements, the recommender system determines alternative solutions (the consideration set) and presents these to the user. If no solution could be found for the given requirements, repair alternatives are determined which support users in getting out of the *no solution could be found* dilemma (see Figure 6.3).

Constraint-based recommendation requires the explicit definition of questions (representing alternatives for user requirements), properties of the items, and constraints. An example of a recommendation knowledge base is shown in Figure 6.1. The WEEVIS tag &QUESTIONS enumerates variables that describe user requirements where *usage* specifies the intended use of the computer, *eefficiency* represents the required energy efficiency, *maxprice* denotes the upper price limit specified by the user, *country* represents the country of the user, *mb* represents the type of motherboard, and *cpu* the requested central processing unit. If a variable is associated with a *keep* tag, this variable is not taken into account in the diagnosis process. For example, *country?* is associated with a *keep* tag; for this reason, it will not be part of any diagnosis presented to the recommender user. Other examples of such attributes are a person's age and gender.

In addition to variables representing potential user requirements, a recommendation knowledge base includes the definition of variables that represent item properties (represented by the WEEVIS tag &PRODUCTS). In our example, cpu_p represents the CPU included in the item, mb_p specifies the included motherboard, os_p represents the installed operating system, and price_p is the overall price. Furthermore, the set of items (products) must be specified that can be recommended to users. A simplified item assortment is included in Figure 6.1 as part of the item properties. Our example assortment of items consists of the entries *hw1*, *hw2*, and *energystar*.

Incompatibility constraints describe combinations of requirements that lead to an inconsistency. The description related to the WEEVIS tag &CONSTRAINTS includes an incompatibility relationship between the variable *usage* and the variable *cpu*. For example, computers with a CPUD must not be sold to users interested in scientific calculations.

Filter constraints describe the relationship between user requirements and items. A simple example of such a filter constraint is $maxprice \geq price_p$, i.e., the price of an recommended item must be equal or below the maximum accepted price specified by the customer (see the WEEVIS tag &CONSTRAINTS in Figure 6.1).

Finally, WEEVIS supports the definition of test cases (see also Section 6.5) which can be used to specify the intended behavior of a recommender knowledge base (WEEVIS tag &TEST). After changes to the knowledge base, regression tests can be triggered on the basis of the defined test suite. The —show— tag specifies whether the recommender system user interface should show the status of the test case (satisfied or not) – see, for example, Figure 6.4.

On a formal level, a recommendation knowledge base can be represented as a constraint satisfaction problem (Mackworth (1977)) with two sets of variables V = U ∪ P and the corresponding constraints C = COMP ∪ PROD ∪ FILT. In this context, $u_i \in U$ are variables describing possible user requirements (e.g., *usage* or *maxprice*) and $p_i \in P$ are variables describing item (product) properties (e.g., mb_p or $price_p$).

The recommendation knowledge base specified in Figure 6.1 can be transformed into a constraint satisfaction problem where &QUESTIONS represents U, &PRODUCTS represents P and $PROD$, and

&CONSTRAINTS represents $COMP$ and $FILT$.[§] Given such a recommendation knowledge base we are able to determine concrete recommendations on the basis of a specified set of user (customer) requirements. Requirements collected are represented in terms of constraints, i.e., R = $\{r_1, r_2, ..., r_k\}$ represents a set of user requirements.

After having identified the set of alternative solutions (recommended items or consideration set), this result is presented to the user. In constraint-based recommendation scenarios, the ranking of items is often performed on the basis of Multi-Attribute Utility Theory (MAUT) where items are evaluated on the basis of a given set of interest dimensions. For further details on the ranking of items in constraint-based recommendation scenarios we refer to Felfernig et al. (2013c).

6.4. Diagnosis and Repair of Requirements

In situations where the given set of requirements $r_i \in R$ (unary constraints defined on variables of U such as *maxprice* $\leq$ *500*) become inconsistent with the recommendation knowledge base (C), we are interested in repair proposals that indicate for a subset of these requirements change operations with a high probability of being accepted by the user. On a more formal level we now introduce a definition of a customer requirements diagnosis task and a corresponding diagnosis (see Definition 1).

Definition 1 (Requirements Diagnosis Task). Given a set of requirements R and a set of constraints C (the recommendation knowledge base), the diagnosis task it to identify a minimal set Δ of constraints (the diagnosis) that have to be removed from R such that R - Δ ∪ C is consistent.

An example of a set of requirements for which no solution can be identified is R = $\{r_1$: *usage = Scientific*, r_2 :*eefficiency = high*, r_3: *maxprice = 1700*, r_4: *country = Austria*, r_5:*mb = MBSilver*, r_6: *cpu = CPUD*$\}$. The recommendation knowledge base induces two minimal conflict sets (CS) (Junker (2004)) in R which are CS_1: $\{r_1, r_6\}$ and CS_2: $\{r_1, r_5\}$. For these conflict sets we have two alternative diagnoses which are Δ_1:$\{r_5, r_6\}$ and Δ_2:$\{r_1\}$. The pragmatics, for example, of Δ_1 is that at least r_5 and r_6 have to be adapted in order to be able to find a solution. How to determine such diagnoses on the basis of a HSDAG (hitting set directed acyclic graph) is shown, for example, in Felfernig et al. (2004).

Approaches based on the construction of hitting sets typically rely on conflict detection (Junker (2004); Felfernig et al. (2004)). In interactive settings, where only preferred diagnoses (leading diagnoses) should be presented, hitting set based approaches tend to become too inefficient since conflict sets have to be determined before a diagnosis can be presented (Felfernig et al. (2012a); Felfernig and Schubert (2010)). This was the major motivation for the development of the FASTDIAG algorithm (Felfernig et al. (2012a); Felfernig and Schubert (2010); Felfernig et al. (2013d)), which is a divide-and-conquer based algorithm that enables the determination of minimal diagnoses without the need of conflict determination and HSDAG construction. This way of determining minimal diagnoses can also be denoted as *direct diagnosis* since no conflict set determination is needed in this context.

[§]PROD is assumed to be represented as a single constraint in disjunctive normal form where each conjunct is an item.

FASTDIAG can be seen as an inverse QUICKXPLAIN (Junker (2004)) type algorithm which relies on the following basic principle (see Figure 6.2). Given, for example, a set R = $\{r_6, r_5, ..., r_1\}$ and a diagnosis (see Definition 1) is contained in $\{r_6, r_5, r_4\}$ (first part of the split), then there is no need of further evaluating $\{r_3, r_2, r_1\}$, i.e., the latter set is consistent. The similarity to QUICKXPLAIN is the following. If a minimal conflict is contained in $\{r_6, r_5, r_4\}$ there is no need to further search for conflicts in $\{r_3, r_2, r_1\}$ since the algorithm determines one minimal conflict set at a time. Both algorithms (FASTDIAG and QUICKXPLAIN) rely on a total lexicographical ordering (Junker (2004); Felfernig et al. (2012a)) which allows the determination of preferred minimal diagnoses (minimal conflict sets).

A minimal (preferred) diagnosis Δ can be used as a basis for the determination of corresponding repair actions, i.e., concrete measures to change user requirements in R in a fashion such that the resulting R' is consistent with C.

Definition 2 (Repair Task). Given a set of requirements R = $\{r_1, r_2, ..., r_k\}$ inconsistent with the constraints in C and a corresponding diagnosis $\Delta \subseteq R$ ($\Delta = \{r_l, ..., r_o\}$), the corresponding repair task is to determine an adaption A = $\{r_l', ..., r_o'\}$ such that R - Δ ∪ A is consistent with C.

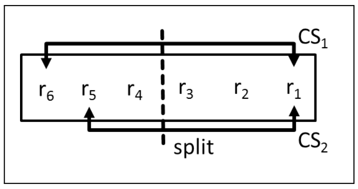
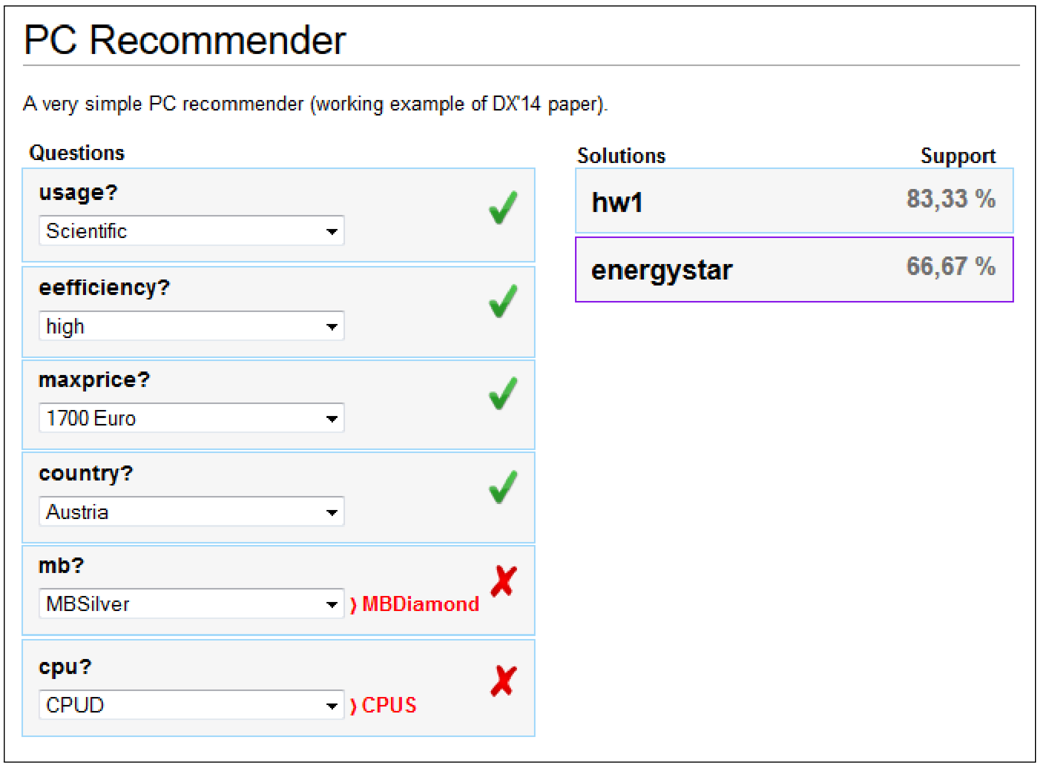


Figure 6.2.: Divide-and-conquer principle of FASTDIAG (CS_1 and CS_2 are assumed to be conflict sets). The set of requirements $R = \{r_1, ..., r_6\}$ is split in the middle. If a diagnosis is already contained in the first part of the split (R - $\{r_6, r_5, r_4\}$ is consistent), there is no need to further investigate the right part for further diagnosis elements. This way, half of the potential diagnosis elements can be eliminated in one step (consistency check).

In WEEVIS, repair actions are determined conform to Definition 2. For each diagnosis Δ determined by FASTDIAG (currently, the first n=3 leading diagnoses are determined – for details see Felfernig et al. (2012a)), the corresponding solution search for R - Δ ∪ C returns a set of alternative repair actions (represented as adaptation A). In the following, all products that satisfy R - Δ ∪ A are shown to the user (see the right hand side of Figure 6.3).

In the current WEEVIS implementation, the total lexicographical ordering is derived from the order in which a user has entered his/her requirements. For example, if r_1: *usage = Scientific* has been entered before r_5: *mb = MBSilver* and r_6: *cpu = CPUD* then the underlying assumption is that r_5 and r_6 are of lower importance for the user and thus have a higher probability of being part of a diagnosis. In our working example $\Delta_1 = \{r_5, r_6\}$. The corresponding set of repair actions (solutions for R-Δ_1 ∪ C) is A = $\{r_5$':mb=MBDiamond, r_6':cpu=CPUS$\}$, i.e., $\{r_1, r_2, r_3, r_4, r_5, r_6\}$ - $\{r_5, r_6\}$ ∪ $\{r_5$', r_6'$\}$ is consistent. The item that satisfies R - Δ_1 ∪ A is {hw1} (see the first entry in Figure 6.3). In a similar



PC Recommender

A very simple PC recommender (working example of DX'14 paper).

Questions			
usage?	Scientific		✓
eefficiency?	high		✓
maxprice?	1700 Euro		✓
country?	Austria		✓
mb?	MBSilver	) MBDiamond	✗
cpu?	CPUD	) CPUS	✗

Solutions	Support
hw1	83,33 %
energystar	66,67 %

Figure 6.3.: PC recommender UI (MediaWiki Readmode). If the user selects the item *energystar* on the right-hand side, a diagnosis with corresponding repair actions is depicted on the left-hand side.

fashion, repair actions are determined for Δ_2 - the recommended item is {energystar}. The identified items (p) are finally ranked according to their support value (see Formula 6.1).

$$support(p) = \frac{\#repair\ actions\ in\ R'}{\#\ requirements\ in\ R} \tag{6.1}$$

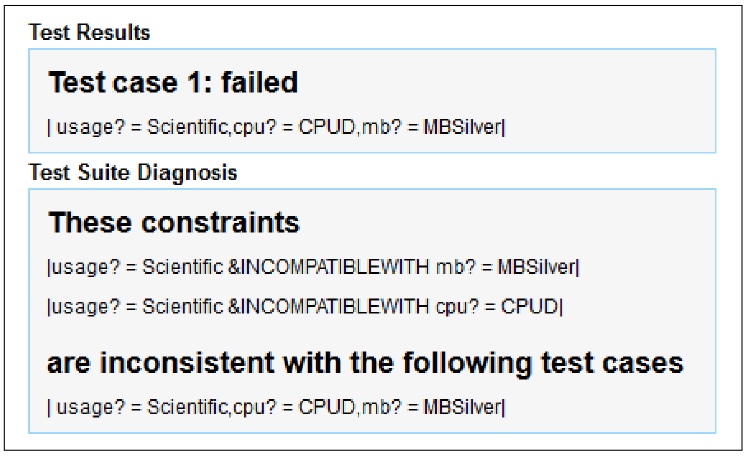
6.5. Knowledge Base Diagnosis

Recommendation knowledge is often subject to change operations. Due to frequent changes it is important to support quality assurance of recommendation knowledge. WEEVIS supports the definition and execution of test cases[¶] which define the intended behavior of the recommender knowledge base. If some test cases become inconsistent with a new version of the knowledge base, the causes of the unintended behavior must be identified. On a formal level a recommendation knowledge base (RKB) diagnosis task can be defined as follows (see Definition 3).

[¶]WEEVIS supports the definition of positive test cases (test cases that should be consistent with the knowledge base).

Definition 3 (RKB Diagnosis Task). Given a set C (the recommendation knowledge base) and a set T = $\{t_1, t_2, ..., t_q\}$ of test cases t_i, the corresponding diagnosis task is it to identify a minimal set Δ of constraints (the diagnosis) that have to be removed from C such that $\forall t_i \in T$: $C - \Delta \cup t_i$ is consistent.

An example test case which induces an inconsistency with the constraints in C is t: *usage* = *Scientific* and *cpu* = *CPUD* and *mb* = *MBSilver* (see Figure 6.1). t induces two conflicts in the recommendation knowledge base which are CS_1: $\neg$(*usage* = *Scientific* $\wedge$ *cpu* = *CPUD*) and CS_2: $\neg$(*usage* = *Scientific* $\wedge$ *mb* = *MBSilver*). In order to make C consistent with t, both incompatibility constraints have to be deleted from C, i.e., are part of the diagnosis Δ.


Test Results

Test case 1: failed

| usage? = Scientific,cpu? = CPUD,mb? = MBSilver|

Test Suite Diagnosis

These constraints

|usage? = Scientific &INCOMPATIBLEWITH mb? = MBSilver|

|usage? = Scientific &INCOMPATIBLEWITH cpu? = CPUD|

are inconsistent with the following test cases

| usage? = Scientific,cpu? = CPUD,mb? = MBSilver|

Figure 6.4.: PC recommender knowledge base: result of the diagnosis process presented in WEEVIS.

Similar to the diagnosis of inconsistent requirements, the hitting set based determination of diagnoses for inconsistent knowledge bases is shown in Felfernig et al. (2004). This approach relies on the construction of a HSDAG determined on the basis of minimal conflict sets provided by conflict detection algorithm such as QUICKXPLAIN. Diagnoses are determined in a breadth-first fashion, i.e., *minimal cardinality* diagnoses of faulty constraints in C are returned first.

In contrast to Felfernig et al. (2004), WEEVIS includes a FASTDIAG based approach to knowledge base debugging that can also be applied in interactive settings. In this case, diagnoses are searched in C. In the case of requirements diagnosis, the total ordering of the requirements is related to user preferences (in WEEVIS derived from the instantiation order of variables). Total orderings of constraints in the context of knowledge base diagnosis are determined using criteria different from the diagnosis of inconsistent requirements, for example, age of constraints, frequency of quality assurance, and structural constraint complexity (see Felfernig et al. (2013e)). An example screenshot of the WEEVIS diagnosis presentation is depicted in Figure 6.4.

6.6. Related and Future Work

Diagnosing Inconsistent Requirements. Junker (2004) introduced the QUICKXPLAIN algorithm which is a divide-and-conquer based approach to the determination of minimal conflict sets (one conflict set at a time). Combining QUICKXPLAIN with the hitting set directed acyclic graph (HS-DAG) algorithm (Reiter (1987)) allows for the calculation of the complete set of minimal conflicts. O'Sullivan et al. (2007) show how to determine representative explanations (diagnoses) which fulfill the requirement that minimal subsets Δ_S of the complete set of diagnoses Δ_C should be determined that fulfill the criteria that if a constraint c_i is contained in a diagnosis of Δ_C it must also be part of at least one diagnosis in Δ_S. Felfernig et al. (2009b, 2013a) show how to integrate similarity metrics, utility-, and probability-based approaches to the determination of leading diagnoses on the basis HSDAG-based search.

Schubert and Felfernig (2010) introduce FLEXDIAG which is a top-down version of FASTDIAG allowing a kind of anytime diagnosis due to the fact that diagnosis granularity (size of constraints regarded as one component in the diagnosis process) can be parametrized. Felfernig et al. (2012a); Schubert and Felfernig (2010) introduce the FASTDIAG algorithm that allows for a more efficient determination of diagnoses due to the fact the there is no need for determining conflict sets (= *direct diagnosis*). FASTDIAG is a QUICKXPLAIN style algorithm that follows a divide-and-conquer approach for the determination of minimal diagnoses. Note that in contrast to traditional HSDAG based approaches, FASTDIAG does not focus on the determination of minimal cardinality but preferred minimal diagnoses. A major issue for future work will be the development of diagnosis algorithms that are capable of performing intra-constraint debugging an thus help to better focus on the sources of inconsistencies. FASTDIAG is not restricted to the application in knowledge-based recommendation scenarios but generally applicable in consistency-based settings (Hotz et al. (2014)). For example, the same principles can be applied in knowledge-based configuration (Stumptner (1997); Sabin and Weigel (1998); Felfernig et al. (2014b)). Further approaches to the determination of diagnoses for inconsistent knowledge bases can be found, for example, in K. McAreavey and Miller (2014); Marques-Silva and Previti (2014); Marques-Silva et al. (2013); Y. Malitsky and Marques-Silva (2014); Walter et al. (2013).

Knowledge Base Maintenance. The application of model-based diagnosis for the debugging of inconsistent constraint sets was first presented in Bakker et al. (1993). Felfernig et al. (2004) show how to exploit test cases for the induction of conflict sets in knowledge bases which are then resolved on the basis of a hitting set based approach. In the line of the work of Felfernig et al. (2012a); Felfernig and Schubert (2010) the performance of knowledge debugging can be improved on the basis of FASTDIAG. A detailed evaluation of the performance gains of FASTDIAG in the context of knowledge base debugging is within the focus of our future work. A detailed comparison between the performance of FASTDIAG and conflict-driven diagnosis of inconsistent requirements can be found, for example, in Felfernig et al. (2012a).

Identifying redundant constraints is an additional issue in the context of knowledge base development and maintenance. Redundant constraints can deteriorate runtime performance and also be the cause of additional overheads in development and maintenance operations (Felfernig et al. (2011a)). Redundancy detection can be based on QUICKXPLAIN especially in the case of an increasing number of redundant constraints. For a detailed discussion of alternative algorithms for redundancy detection in knowledge bases we refer to Felfernig et al. (2011a). A major focus of our future research will be the development of an intra-constraint redundancy detection, i.e., it will be possible to identify redundant subexpressions.

6.7. Conclusions

In this chapter we provide an overview of the WEEVIS environment with a special focus on the integrated diagnosis support. Diagnosis techniques integrated in WEEVIS are the result of research in model-based diagnosis with a special focus on divide-and-conquer based (direct) algorithms that make diagnosis search more efficient in the case that leading diagnoses are required. WEEVIS is a publicly available MediaWiki-based environment for developing and maintaining constraint-based recommender applications.

Chapter 7

The WEEVIS Environment applied in the E-Government Domain

This chapter is based on the results documented in Reiterer et al. (2015a).

The author of this thesis *developed the examples mentioned in this chapter* and *wrote major parts of the above mentioned paper.*

7.1. Abstract

Constraint-based recommenders support customers in identifying relevant items from complex item assortments. In this chapter we present WEEVIS, a constraint-based environment that can be applied in different scenarios in the e-government domain. WEEVIS supports collaborative knowledge acquisition for recommender applications in a MediaWiki-based context. This chapter shows how Wiki pages can be extended with recommender applications and how the environment uses intelligent mechanisms to support users in identifying the optimal solutions to their needs. An evaluation shows a performance overview with different knowledge bases.

7.2. Introduction

Constraint-based recommender applications help users navigating in complex product and service assortments like digital cameras, computers, financial services and municipality services. The calculation of the recommendations is based on a knowledge base of explicitly defined rules. The engineering of the rules for recommender knowledge bases (for constraint-based recommenders) is typically done by knowledge engineers, mostly computer scientists (Felfernig and Burke (2008)). For building high quality knowledge bases there are domain experts involved who serve the knowledge engineers with deep domain knowledge (Felfernig and Burke (2008)). Graphical knowledge engineering interfaces like Felfernig et al. (2006a) improved the maintainability and accessability and moved the field one step further.

Other recommendation approaches like collaborative filtering use information about the rating behavior of other users to identify recommendations (Linden et al. (2003); Konstan et al. (1997)). Content-based filtering (Pazzani and Billsus (1997)) exploits features of items for the determination of recommendations. Compared to these approaches, constraint-based recommenders are more applicable for complex products and services due to their explicit knowledge representation.

In the line of Wikipedia[*] where users build and maintain Wiki pages collaboratively we introduce WEEVIS[†]. WEEVIS is a MediaWiki[‡] based environment that exploits the properties of MediaWiki and enables community based development and maintenance of knowledge bases for constraint-based recommenders. WEEVIS is freely available as a platform and successfully applied by four Austrian universities (in lectures about recommender systems), in the financial services domain and in e-government .

In the e-government domain officials as well as the community residents can take numerous advantages of knowledge-based recommenders:

- WEEVIS can be used as an online advisory service for citizens for example for documents that are necessary to apply for a private construction project. The online recommendation of necessary documents in advance to on-site appointments can lead to a time reduction for community residents and community officials.
- WEEVIS can be used for modeling internal processes like the signing of travel applications for example a community official wants to visit a conference, based on different parameters like the conference type, or if it's abroad or in the domestic area, different officials have to sign the travel request. In WEEVIS the appropriate rules for such internal processes can be mapped and especially for new employees WEEVIS recommenders can provide substantial assistance.
- WEEVIS can be used as an information platform for example with integrated knowledge-based recommenders for community residents e.g. to identify the optimal waste disposal strategy for a household (this example is used as a running example in this chapter, see Section 2). Instead of providing plain text information, like common municipality web pages, the knowledge representation as a recommender provides an easier way for community members to identify the optimal solution for their situation.

A recommender development environment for single users is introduced in Felfernig et al. (2006a). This work is based on a Java platform and focuses on constraint-based recommender applications for

[*]www.wikipedia.org
[†]www.weevis.org
[‡]www.mediawiki.org

online selling. Compared to Felfernig et al. (2006a), WEEVIS provides a wiki-based user interface that allows user communities to develop recommender applications collaboratively. Instead of an incremental dialog, where the user answers one question after the other, like Felfernig et al. (2006a), WEEVIS provides an integrated interface where the user is free to answer questions in any order.

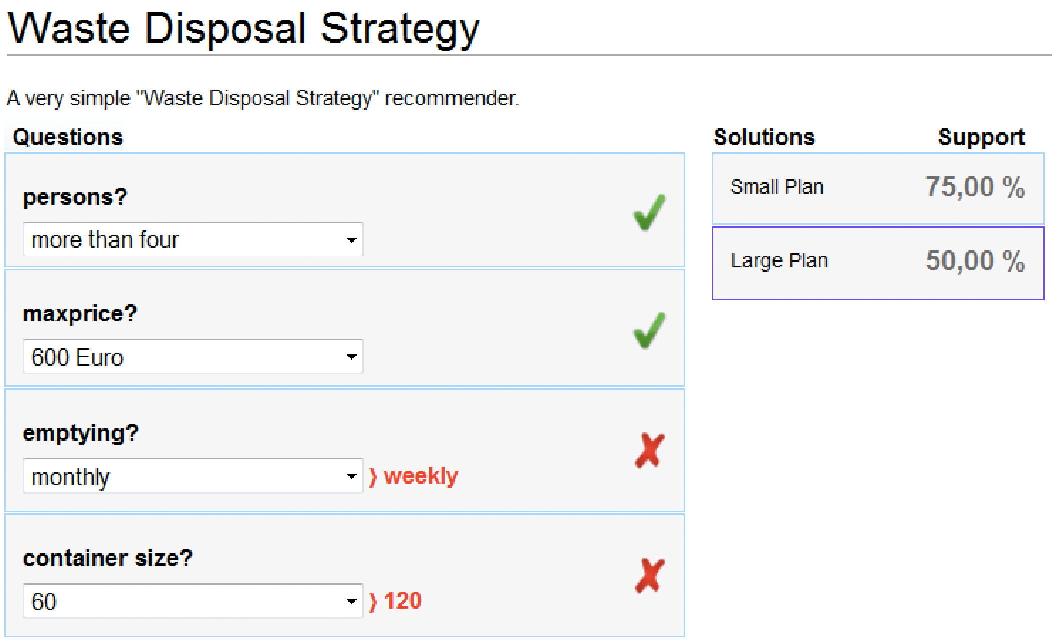
The WEEVIS interface also provides intelligent mechanisms for an instant presentation of alternative solutions in situations where it is not possible to find a solution for a given set of user (customer) requirements, i.e., the requirements are inconsistent with the recommendation knowledge base and the user is in the need for repair proposals to find a way out from the *no solution could be found dilemma*. Model-based diagnosis (Reiter (1987)) can be applied for the identification of faulty constraints in a given set of customer requirements. In this context efficient divide-and-conquer based algorithms (Junker (2004)) can be applied to the diagnosis and repair of inconsistent requirements. The environment supports the user with integrated model-based diagnosis techniques (Reiter (1987); Felfernig et al. (2012a)). A first approach to a conflict-directed search for hitting sets in inconsistent CSP definitions was introduced by Bakker et al. (1993). With regard to diagnosis techniques, WEEVIS is based on more efficient techniques that make the environment applicable in interactive settings (Felfernig et al. (2012a); Friedrich (2014)).

A Semantic Wiki-based approach to knowledge acquisition for collaborative ontology development is introduced, for example, in Baumeister et al. (2011). Compared to Baumeister et al. (2011), WEEVIS is based on a recommendation domain specific knowledge representation (in contrast to ontology representation languages) which makes the definition of domain knowledge more accessible also for domain experts.

The remainder of this chapter is organized as follows. In Section 2 we present an overview of the recommendation environment WEEVIS and it's application in the e-government domain. In Section 3 we present results of a performance evaluation that illustrates the performance of the integrated diagnosis technologies. With Section 4 we conclude the chapter.

7.3. WEEVIS Overview

Since WEEVIS is based on the MediaWiki platform, it can be installed on freely available web servers. On the website *www.weevis.org* a selection of different WEEVIS recommenders is publicly available. For internal processes WEEVIS can be deployed in the local intranet. Standard wiki pages can be complemented easily by recommender knowledge bases. Currently, WEEVIS calculates recommendations based on previously entered requirements. If the requirements would result in a *no solution could be found message* WEEVIS calculates alternative solutions based on diagnoses (see Section 2.4). In line with the Wiki idea, WEEVIS provides the ability to build knowledge bases collaboratively, a valuable feature in e-government domain, because depending on the community department multiple people are responsible for data management and administration. Furthermore, WEEVIS exploits the basic functionalities provided by MediaWiki and allows rapid prototyping processes where the result of a change can immediately be seen by simply switching from the *edit mode* to the corresponding *read mode*. This approach allows an easy understanding of the WeeVis tags and also of the semantics of the provided WeeVis language.


Waste Disposal Strategy

A very simple "Waste Disposal Strategy" recommender.

Questions		
persons?	more than four	✓
maxprice?	600 Euro	✓
emptying?	monthly ⟩ weekly	✗
container size?	60 ⟩ 120	✗

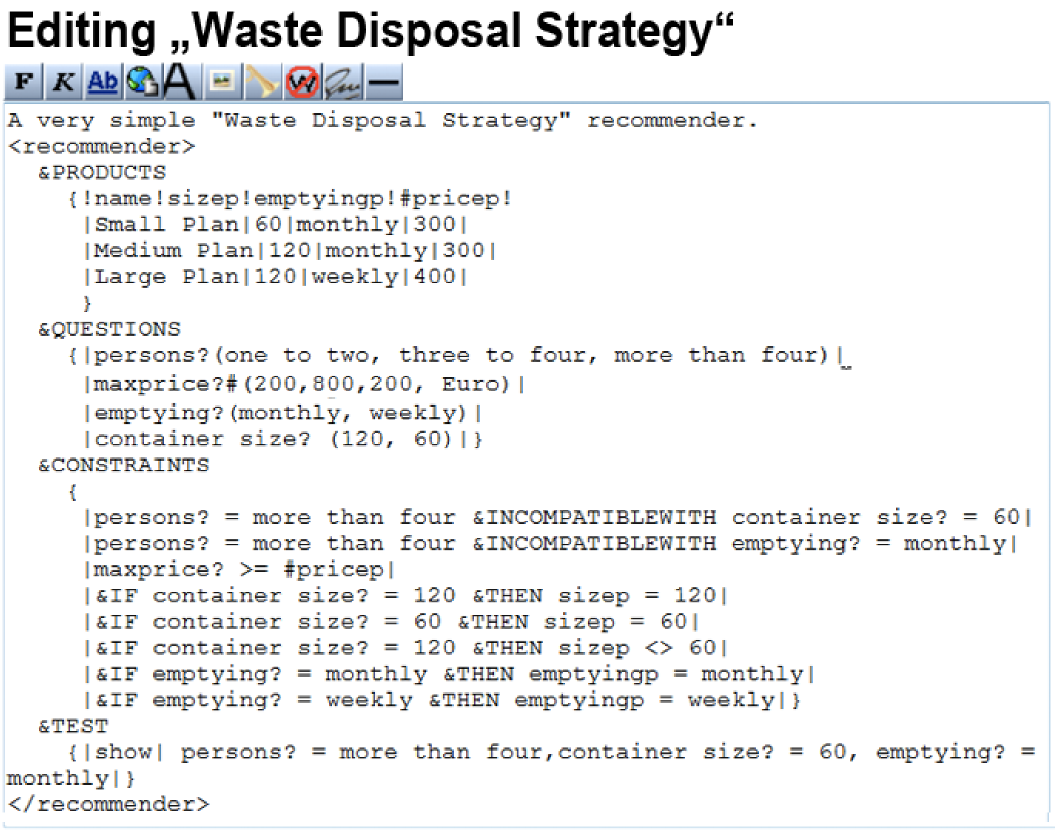
Solutions	Support
Small Plan	75,00 %
Large Plan	50,00 %

Figure 7.1.: *Waste Disposal Strategy* a simple recommender knowledge base from the *e-government* domain (WeeVis *read* mode).

7.3.1. WeeVis User Interface

Since WeeVis is a MediaWiki-based environment the user interface relies on the common Wiki principle of the *read* mode (see Figure 7.1) for executing a recommender and the *write* mode (see Figure 7.2) for defining a recommender knowledge base. The development and maintenance of a knowledge base is supported a textual fashion with a syntax that is similar to the standard Wiki syntax (see Figure 7.2). In the following we will present the concepts integrated in the WeeVis environment on the basis of a working example from the e-government domain. More specifically we present a recommender that supports households in identifying their optimal waste disposal strategy. In this recommendation scenario, a user has to specify his/her requirements regarding, for example, the number of persons living in the household or how frequently the containers should be emptied. A corresponding WeeVis user interface is depicted in Figure 7.1. Requirements are specified on the left hand side and the corresponding recommendations for the optimal waste disposal plan are displayed in the right hand side.

For each solution, a so-called support score is determined. If a solution fulfills all requirements,



Editing „Waste Disposal Strategy“

```
A very simple "Waste Disposal Strategy" recommender.
<recommender>
  &PRODUCTS
    {!name!sizep!emptyingp!#pricep!
     |Small Plan|60|monthly|300|
     |Medium Plan|120|monthly|300|
     |Large Plan|120|weekly|400|
     }
  &QUESTIONS
    {|persons?(one to two, three to four, more than four)|
     |maxprice?#(200,800,200, Euro)|
     |emptying?(monthly, weekly)|
     |container size? (120, 60)|}
  &CONSTRAINTS
    {
     |persons? = more than four &INCOMPATIBLEWITH container size? = 60|
     |persons? = more than four &INCOMPATIBLEWITH emptying? = monthly|
     |maxprice? >= #pricep|
     |&IF container size? = 120 &THEN sizep = 120|
     |&IF container size? = 60 &THEN sizep = 60|
     |&IF container size? = 120 &THEN sizep <> 60|
     |&IF emptying? = monthly &THEN emptyingp = monthly|
     |&IF emptying? = weekly &THEN emptyingp = weekly|}
  &TEST
    {|show| persons? = more than four,container size? = 60, emptying? =
monthly|}
</recommender>
```

Figure 7.2.: The *Waste Disposal Strategy* recommender knowledge base (*view source (edit)* mode).

this score is 100%, otherwise it is lower and, when clicking on the score value, a corresponding repair action is displayed on the left-hand side (see Figure 7.1). Due to the automated alternative determination, WEEVIS is always able to present a solution and users are never ending up in the *no solution could be found* dilemma (see Figure 7.1).

An example of the definition of a (simplified) e-government recommender knowledge base is depicted in Figure 7.2. The definition of a recommender knowledge base is supported in a textual fashion on the basis of a syntax similar to MediaWiki. Basic syntactical elements provided in WEEVIS will be introduced in the next subsection.

7.3.2. WEEVIS Syntax

A WEEVIS recommender consists of three necessary aspects, the definition of questions and possible answers, items and their properties, and constraints (see Figure 7.2).

The definition of an item assortment in WEEVIS starts with the *&PRODUCTS* tag (see Figure 7.2). The first line represents the attributes separated by the exclamation mark. In our example, the item assortment is specified by the *name*, *sizep*, the container size, *emptyingp*, the emptying frequency, and *pricep*, the price of the waste disposal plan. Each of the next lines represents an item with the values related to the attributes, in our example there are three items specified: *Small Plan*, *Medium Plan*, and *Large Plan*.

The second aspect starts with the &QUESTIONS tag. In our example the following user requirements are defined: *persons*, specifies the number of persons living in the household (one to two, three to four, more than four) and *maxprice* specifies the upper limit regarding the price of the waste disposal plan. Furthermore, *emptying* represents the sequence in which the dustbins will be emptied, weekly or monthly, and *container size*, the preferred size of the dust container, 120 or 60.

The third aspect represents the definition of the constraints. Starting with the &CONSTRAINTS tag in WEEVIS different types of constraints can be defined. For the first constraint in our example the &INCOMPATIBLE keyword is used to describe incompatible combinations of requirements. The first incompatibility constraint describes an incompatibility between the number of persons in the household (*persons*) and the *container size*. For example, a waste disposal plan with (*container size*) 60 must not be recommended to users who live in a household with more than four persons. Filter constraints describe relationships between requirements and items, for example, $maxprice \geq pricep$, i.e., the price of a waste disposal plan must be equal or below the maximum accepted price.

7.3.3. Recommender Knowledge Base

A recommendation knowledge base can be represented as a CSP (Constraint Satisfaction Problem) (Mackworth (1977)) on a formal level. The CSP has two sets of variables V ($V = U \cup P$) and the constraints $C = PROD \cup COMP \cup FILT$ where $u_i \in U$ are variables describing possible user requirements (e.g., *persons*) and $p_i \in P$ are describing item properties (e.g., *emptyingp*). Each variable v_i has a domain d_j of values that can be assigned to the variable (e.g., *one to two*, *three to four* or *more than four* for the variable *persons*). Furthermore, there are three different types of constraints:

- *COMP* represents incompatibility constraints of the form $\neg X \vee \neg Y$
- *PROD* the products with their attributes in disjunctive normal form (each product is described as a conjunction of individual product properties)
- *FILT* the given filter constraints of the form $X \rightarrow Y$

The knowledge base specified in Figure 7.2 can be transformed into a constraint satisfaction problem where &QUESTIONS represents U, &PRODUCTS represents P and &CONSTRAINTS represents *PROD*, *COMP*, and *FILT*. Based on this knowledge representation WEEVIS is able to determine recommendations that take into account a specified set of user requirements. The results collected are represented as unary constraints ($R = \{r_1, r_2, ..., r_k\}$). Finally the determined set of solutions (recommended items) is presented to the user.

7.3.4. Diagnosis and Repair of Requirements

In situations where requirements $r_i \in R$ (unary constraints defined on variables of U such as *emptying = monthly*) are inconsistent with the constraints in C, we are interested in a subset of these requirements that should be adapted to be able to restore consistency. On a formal level we define a *requirements diagnosis task* and a corresponding *diagnosis* (see Definition 1).

Definition 1 (Requirements Diagnosis Task). Given a set of requirements R and a set of constraints C (the recommendation knowledge base), the requirements diagnosis task is to identify a minimal set Δ of constraints (the diagnosis) that has to be removed from R such that $R - \Delta \cup C$ is consistent.

As an example $R = \{r_1 : persons = morethanfour,\ r_2 : maxprice = 600,\ r_3 : emptying = monthly, r_4 : containersize = 60\}$ is a set of requirements inconsistent with the defined recommendation knowledge. The recommendation knowledge base induces two minimal conflict sets (CS) (Junker (2004)) in R which are $CS_1 : \{r_1, r_4\}$ and $CS_2 : \{r_1, r_3\}$. For these requirements we can derive two diagnoses: $\Delta_1 : \{r_3, r_4\}$ and $\Delta_2 : \{r_1\}$. For example, to achieve consistency of Δ_1 at least r_3 and r_4 have to be adapted. Such diagnoses can be determined on the basis of a HSDAG (hitting set directed acyclic graph) (e.g. Felfernig et al. (2004)).

Determining conflict sets (Junker (2004)) at first and afterwards constructing a HSDAG (hitting set directed acyclic graph) to identify diagnoses tends to become inefficient especially in interactive settings. Direct diagnosis algorithms like FASTDIAG (Felfernig et al. (2012a)) reduce this two-step process to one step by calculating diagnoses directly without conflict determination. This was the major motivation for integrating FASTDIAG (Felfernig et al. (2012a)) into the WEEVIS environment. Like QUICKXPLAIN (Junker (2004)), FASTDIAG is based on a divide-and-conquer approach that enables the calculation of minimal diagnoses without the calculation of conflict sets. In WEEVIS the derived diagnosis are used as a basis for determining repair actions, which lead to the alternative solutions that are be presented to the user. A repair action is a concrete change of one or more user requirements in R on the basis of a diagnosis such that the resulting R' is consistent with C.

Definition 2 (Repair Task). Given a set of requirements $R = \{r_1, r_2, ..., r_k\}$ inconsistent with the constraints in C and a corresponding diagnosis $\Delta \subseteq R$ ($\Delta = \{r_l, ..., r_o\}$), the corresponding repair task is to determine an adaption $A = \{r'_l, ..., r'_o\}$ such that $R - \Delta \cup A$ is consistent with C.

In WEEVIS, repair actions are determined conform to Definition 2. For each diagnosis Δ determined by FASTDIAG, the corresponding solution search for $R - \Delta \cup C$ returns a set of alternative repair actions (represented as adaptation A). In the following, all solutions that satisfy $R - \Delta \cup A$ are shown to the user (see the right hand side of Figure 7.1).

Diagnosis determination in FASTDIAG is based on a total lexicographical ordering of the customer requirements (Felfernig et al. (2012a)). This ordering is derived from the sequence of the entered requirements. For example, if $r_1 : persons = morethanfour$ has been entered before $r_3 : emptying = monthly$ and $r_4 : containersize = 60$ then the underlying assumption is that r_3 and r_4

Knowledge base	**Number of solutions / requirements / constraints**
Small	5/5/5
Medium	20/10/15
Large	50/15/30

Table 7.1.: The different knowledge bases with sizes *Small*, *Medium* and *Large* used for the performance comparison.

are of lower importance for the user and thus have a higher probability of being part of a diagnosis. In our working example $\Delta_1 = \{r_3, r_4\}$. The corresponding repair actions (solutions for $R - \Delta_1 \cup C$) is $A = \{r_3' : emptying = weekly, r_4' : containersize = 120\}$, i.e., $\{r_1, r_2, r_3, r_4\} - \{r_3, r_4\} \cup \{r_3', r_4'\}$ is consistent. The item that satisfies $R - \Delta_1 \cup A$ is $\{LargePlan\}$ (see in Figure 7.2). The identified items (p) are ranked according to their support value (see Formula 7.1).

$$support(p) = \frac{\#adaptions\ in\ A}{\#requirements\ in\ R} \tag{7.1}$$

7.4. Performance Evaluation

7.4.1. Description of the evaluation

We have conducted a performance evaluation with the goal to highlight the ability of WEEVIS to calculate repair actions and if no solutions could be found. Therefore we set up an experiment with three WEEVIS recommenders based on the e-government example presented in Section 2. To illustrate the performance of WEEVIS, the knowledge base was extended and deployed with different complexity regarding the number of solutions (&PRODUCTS tag in WEEVIS), user requirements (&QUESTIONS tag WEEVIS), and constraints (&CONSTRAINTS tag WEEVIS) (see Table 7.1). According to these three attributes the knowledge bases were classified as *Small*, *Medium*, and *Large*. To fit the attributes of knowledge base *Small* from Table 7.1, the running example (see Figure 7.2) was adapted by adding one question, two products and removing the last two constraints. The *Medium* and *Large* knowledge base are extended versions of the running example.

7.4.2. Results of the evaluation

To provide an optimal user experience a focus of WEEVIS is to provide instant feedback after every interaction. Interacting with a WEEVIS recommender starts with the the entering of new requirements and the subsequent calculation of solutions for these requirements. If no solution could be found WEEVIS calculates one or more diagnoses and the complementing alternative products. With this

	time for identifying solutions
Small	$< 1ms$
Medium	$< 1ms$
Large	$< 2ms$

Table 7.2.: This table shows the time needed to come up with solutions for three knowledge bases. Even for the largest knowledge base the overall time is far below the limit of *an instantaneously reaction*(100ms).

	diagnosis calculation	**repair identification**	**overall time**
Small	$< 1ms$	$< 1ms$	$< 1ms$
Medium	$93ms$	$16ms$	$109ms$
Large	$499ms$	$19ms$	$518ms$

Table 7.3.: This table shows the time needed to come up with at least *one* alternative solution for each of the three knowledge bases. Even for the largest knowledge base the overall time is below the limit of *interrupt a users thought* (1,000ms).

performance evaluation we show that WEEVIS can identify at least one alternative solution even for large knowledge bases within recommended user interface response times (Nielsen (1994)):

- below 100ms, the user feels that the system reacts instantaneously
- 1,000ms is the upper limit for keeping the users thought uninterrupted
- 10,000ms is the upper limit for keeping the user's focus on the dialogue

For the first performance evaluation the goal was to measure the time needed for calculating the corresponding solutions to given requirements. After assigning answers to the questions for the three different knowledge bases, the resulting values are depicted in Table 7.2. The performance values in Table 7.2 show that for each of the knowledge bases WEEVIS identifies solutions fast enough to provide instantaneous feedback from the user interface. If no solution could be found due to inconsistencies between the requirements and the knowledge base, Table 7.3 shows the time needed to identify at least one alternative solution on the basis of one preferred diagnosis, Table 7.4 shows the time consumption of calculating all possible solutions. WEEVIS is able to calculate either one, two, three or all diagnoses and the corresponding alternative solutions. By taking the response time boundaries for user interfaces into account, the experiment shows that for *small* and *medium* knowledge bases it's possible to calculate all minimal diagnoses within acceptable response times (see Table 7.3). When it comes to *large* knowledge bases the presented alternative solutions can be reduced to increase the performance of the user interface instead (see Table 7.4).

	diagnosis calculation	**repair identification**	**overall time**
Small	$97ms$	$16ms$	$113ms$
Medium	$969ms$	$20ms$	$989ms$
Large	$3,028ms$	$60ms$	$3,088ms$

Table 7.4.: This table shows the time needed to come up with *all* possible alternative solutions for each of the three knowledge bases. For the largest knowledge base the overall time for repair calculation takes about three seconds which is above the recommended time boundaries for *interrupting the user's flow of thought*.

7.5. Conclusion

In this chapter we presented WEEVIS which is an open constraint-based recommendation environment. By exploiting the advantages of Mediawiki, WEEVIS provides an intuitive basis for the development and maintenance of constraint-based recommender applications. The results of our experiment show that due to the integrated direct diagnosis algorithms the WEEVIS user interface provides good the response times for common interactive settings.

Chapter 8

Conclusions

In this thesis we introduce the WEEVIS environment, a MediaWiki extension that supports the integration of knowledge-based recommenders into wiki pages. In it's current development status the WEEVIS environment supports the creation of recommendation knowledge bases that rely on an assortment of explicitly defined solutions (items) that can be enumerated. The recommendation approach was chosen because the number of items described in typical wiki pages can be represented with an enumerable amount of solutions. However, the introduced approaches for personalizing diagnoses and acquiring knowledge from the *wisdom of the crowd* are applicable in a recommendation as well as in a configuration scenario. A future version of WEEVIS may be extended to be able to deal with larger solution spaces that can't be represented explicitly.

For collaborative knowledge acquisition as well as during the interaction of users with recommenders or configurators, there are several challenges that can be tackled by applying diagnosis approaches: On the one hand the challenge of finding ways out of the *no solution could be found dilemma* and on the other hand the challenge to identify redundancies and inconsistencies in knowledge bases. Furthermore this work introduces and discusses the personalization of diagnoses. The dedication of WEEVIS is to be used by end users and experts from different domains who want to contribute to a knowledge platform by building knowledge bases in fields of their expertise. This chapter reflects on the research questions and the contributions that have been made to answer them.

Research Question Q1:

How to exploit the *wisdom of the crowds* for building knowledge bases?

In the line of *Wikipedia* and other wiki-based platforms that allow users to collaboratively create content, in this work we introduce WEEVIS. By exploiting the MediaWiki platform, WEEVIS supports the development and maintenance of recommender knowledge bases within wiki pages (see Chapter 5). For the definition of the WEEVIS syntax we oriented towards the MediaWiki syntax. This allows users who have already contributed to a

MediaWiki based knowledge platform an easy way to extend pages with recommendation knowledge bases. WEEVIS also takes advantage of MediaWiki features such as versioning, easy prototyping and mechanisms such as locking pages during editing, which are basic features for collaborative development. Furthermore, WEEVIS comes along with a minimalistic user interface that is seamlessly integrated with wiki pages. 2

Research Question Q2:

How to support users in managing anomalies (e.g., redundancies and inconsistencies) in knowledge bases during a collaborative knowledge engineering process?

In the context of knowledge-based recommendation and configuration there are several challenges to be tackled concerning the knowledge engineering process. Two major challenges are the detection of inconsistencies and redundancies within knowledge bases (see Chapter 6). Obviously, resolving these types of anomalies by hand comes along with a high cognitive effort (Reiterer (2015)). In this work we describe how to reduce these efforts by integrating algorithms for redundancy detection (Felfernig et al. (2011a)) and inconsistency detection (Reiterer (2015)). The integration of these algorithms enhances the value of WEEVIS as a knowledge acquisition tool for domain experts.

Research Question Q3:

How to optimize the performance of calculating diagnoses for interactive scenarios?

The WEEVIS environment (see Chapter 5) relies on a direct diagnosis approach (the FASTDIAG algorithm) to identify alternative solutions if a user taps into a *no solution could be found scenario*. Conflicts between user requirements (see Chapter 3) can occur at any time during the recommendation process and WEEVIS has to come up with alternative solutions immediately. Due to the high computational complexity involved in identifying diagnoses, there is a trade-off between performance and prediction quality. In Chapter 7 we present two knowledge bases from different domains implemented in WEEVIS and analyze algorithm performance for product assortments of different complexity.

Research Question Q4.1:

How to improve the prediction quality of personalized diagnoses by applying the ensemble-based approach?

In Chapter 4 we present an ensemble-based approach to calculate personalized diagnoses. Diagnosis approaches support users in different situations such as the *no solution could be found dilemma* and in finding constraints that violate test cases. In each of these situations, the prediction quality of the diagnosis approach plays a significant role for satisfying the user. The *ensemble-based* diagnosis approach introduced in Chapter 4 combines three personalization approaches: similarity-based diagnosis, utility-based diagnosis, and probability-based diagnosis. Based on the assumption that different diagnosis approaches often lead to a different diagnoses ranking, the individual prediction faults of each approach can be compensated by applying, for example, a majority-based approach to generate ensembles. In Chapter 4 we present a study which shows evidence of the increased prediction quality based on two data sets, one from the domain of *Computer Configuration* and another from the *Financial Services* domain.

Research Question Q4.2:

Is the performance of calculating personalized diagnoses applicable for interactive settings?

As mentioned in Chapter 7, performance of diagnosis algorithms is crucial for their applicability in interactive settings. The *ensemble-based diagnosis* algorithm introduced in Chapter 4 calculates personalized diagnoses with a high prediction quality and it is applicable in interactive settings. In addition to the *Computer Configuration* and the *Financial Services* datasets, the Renault benchmark knowledge base, which is part of the *configuration benchmark suite* was used for the performance evaluation. Considering the calculation of the first and the first five diagnoses, the *ensemble-based* approach performs nearly as well as the breadth-first approach.

Chapter 9

Future Work

This chapter focuses on ideas for future work that could be done in the context of WEEVIS. There are lots of data already available from interactions with WEEVIS that could be exploited and used for further development of the system. Psychological effects such as the *Serial Positioning Effect* (Murphy et al. (2006)) in the recommendation (solution) list should be taken into account in the next version of WEEVIS (see the following discussion).

9.1. Learning Requirements Preferences From Interaction Logs

The WEEVIS interaction log is a large resource of interaction data that can be exploited. Two aspects are especially interesting: On the one hand the order in which the proposed questions are answered and on the other hand solutions from the result list.

The order in which users answer questions could be used for a repositioning of the elements in the questions list. For example, questions that are more often answered by users might be interesting for other users as well. Thus the more interesting questions could be placed on top of the list of questions. As a consequence, the time effort of a user to identify a relevant question is reduced. In the current state of the WEEVIS development an interaction log is already collected but the exploitation of the data is subject of future work.

9.2. Considering Serial Positioning Effects in the List of Solutions

Knowledge-based recommender systems such as WEEVIS (see Chapter 5) or the CWADVISOR environment (see Felfernig et al. (2006a)) provide a list of solutions according to the requirements entered by the user. The goal of these environments is to identify the most relevant solutions. In the perception of solution lists users are biased by serial position effects (Murphy et al. (2006)): Users are more likely to recall items at the beginning and the end of the list than from the middle.

For knowledge-based systems in commercial settings, serial position effects can be exploited for promoting products from the solution list. Therefore the knowledge base description language must be extended by a syntax element that allows the creator of a recommender to highlight certain solutions for promotion. The system can then automatically order the solution list according to the serial position effect to promote the highlighted solutions. For example, if the solution list consists of 10 items from which 4 should be promoted, two can be set on top of the list and two at the end. We want to emphasize that, this creates partially ethical issues, that have to be taken into account.

9.3. Analyzing the Evolution of the Knowledge Base Construction Process

Since WEEVIS extends the MediaWiki platform, several functions such as versioning, user management, and rapid prototyping (*read/write* mode) are exploited for knowledge base creation and maintenance. Especially the integrated versioning of the wiki pages brings significant advantage for collaborative development. Users don't have to deal with the backtracking of faults they make during the recommender development process, versioning makes it easier to experiment with WEEVIS features. Anytime each page can be reverted to a version that was saved in the past. The collected data from the versioning can be exploited to reproduce and analyze each step of the recommender creation and maintenance process. By analyzing this data the knowledge acquisition process in WEEVIS could be improved and refined.

9.4. Data Extraction from Existing Wiki Pages

For the creation of a recommender knowledge base there are three different inputs necessary (see Section 5):

- *Solutions* aka *products*, the result of a recommendation process. Solutions are represented by a table of products with the corresponding attribute value definitions.
- *Questions*, they are used to collect the requirements from the user.
- *Constraints*, represent the ways customer requirements restrict the solution space.

These three aspects have to be defined by domain experts in the current version of the WEEVIS wiki extension. The *questions* aspect as well as the *constraints* aspect need some sort of creativity of a domain expert. The third aspect, *solutions*, is a list of items which is static and, for instance, provided by the producer of the products. Sometimes this information is already available in the form of a list within a plain text wiki page, for example, a list of cars, produced by a car manufacturer BMW. The community project *DBpedia*[*] extracts structured information from wiki pages and makes it available for automated processing. In a future version, WEEVIS could implement a *DBpedia* interface to include solution lists in an automated fashion.

9.5. Parallelized Direct Diagnosis

Diagnosis algorithms and conflict detection algorithms in knowledge-based systems support users in a variety of situations where inconsistencies occur. For example, for identifying alternative solutions if customer requirements are inconsistent with the knowledge base and for detecting constraints that are inconsistent with test cases. These scenarios are all interactive and with increased diagnosis calculation performance it would be possible to handle even larger knowledge bases (see Chapter 7). Since most of todays hardware devices such as smart phones or personal computers have multi core cpus integrated, the parallelization of direct diagnosis algorithms can help to further increase the performance of identifying diagnoses. Jannach et al. (2015) proposed techniques for the parallelization of the standard model-based diagnosis approach (conflict detection with subsequent diagnosis calculation by constructing a hitting set tree (see Reiter (1987))). Since direct diagnosis algorithms such as FASTDIAG (see Chapter 3) have a higher performance than the standard approach it can be expected that the exploitation of multi core resources brings further performance improvements.

[*]www.dbpedia.org

List of Figures

List of Tables

Bibliography

ADOMAVICIUS, G., BOCKSTEDT, J., CURLEY, S., AND ZHANG, J. 2011. Recommender systems, consumer preferences, and anchoring effects. In *RecSys 2011 Workshop on Human Decision Making in Recommender Systems*. 35–42. (Cited on page 25.)

ADOMAVICIUS, G. AND TUZHILIN, A. 2005. Toward the next generation of recommander systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering 17,* 6, 734–749. (Cited on pages 13 and 24.)

ALDANONDO, M. AND VAREILLES, E. 2008. Configuration for mass customization: how to extend product configuration towards requirements and process configuration. *Journal of Intelligent Manufacturing 19,* 5, 521–535. (Cited on page 2.)

ARAZY, O., MORGAN, W., AND PATTERSON, R. 2006. Wisdom of the crowds: Decentralized knowledge construction in wikipedia. In *16th Annual Workshop on Information Technologies & Systems (WITS) Paper*. (Cited on page 5.)

ARDISSONO, L., FELFERNIG, A., FRIEDRICH, G., GOY, A., JANNACH, D., PETRONE, G., SCHÄFER, R., AND ZANKER, M. 2003. A framework for the development of personalized, distributed web-based configuration systems. *AI Magazine 24,* 3, 93–108. (Cited on page 65.)

BAKKER, R., DIKKER, F., TEMPELMAN, F., AND WOGMIM, P. 1993. Diagnosing and solving over-determined constraint satisfaction problems. In *IJCAI 1993*. 276–281. (Cited on pages 75 and 79.)

BALLATORE, A., MCARDLE, G., KELLY, C., AND BERTOLOTTO, M. 2010. RecoMap: an interactive and adaptive map-based recommender. In *25th ACM Symposium on Applied Computing (ACM SAC 2010)*. Sierre, Switzerland, 887–891. (Cited on page 24.)

BARKER, V. E., O'CONNOR, D. E., BACHANT, J., AND SOLOWAY, E. 1989. Expert systems for configuration at digital: XCON and beyond. *Commun. ACM 32*, 298–318. (Cited on pages 18 and 28.)

BAUMEISTER, J., REUTELSHOEFER, J., AND PUPPE, F. 2011. KnowWE: A Semantic Wiki for Knowledge Engineering. *Applied Intelligence 35,* 3, 323–344. (Cited on pages 63, 64, and 79.)

BERKOVSKY, S., FREYNE, J., COOMBE, M., AND BHANDARI, D. 2010. Recommender algorithms in activity motivating games. *ACM Conference on Recommender Systems (RecSys'09)*, 175–182. (Cited on pages 15, 20, 21, and 25.)

BERKOVSKY, S., FREYNE, J., AND OINAS-KUKKONEN, H. 2012. Influencing Individually: Fusing Personalization and Persuasion. *ACM Transactions on Interactive Intelligent Systems 2,* 2, 1–8. (Cited on page 20.)

BLUMÖHR, U., MÜNCH, M., AND UKALOVIC, M. 2010. *Variant Configuration with SAP*. Galileo Press. (Cited on page 4.)

BROCCO, M. AND GROH, G. 2009. Team Recommendation in Open Innovation Networks. In *ACM Conference on Recommender Systems (RecSys'09)*. NY, USA, 365–368. (Cited on pages 15 and 23.)

BURKE, R. 2000. Knowledge-based recommender systems. *Encyclopedia of Library and Information Systems 69,* 32, 180–200. (Cited on pages 2, 3, 12, 14, 19, and 44.)

BURKE, R. AND RAMEZANI, M. 2010. Matching recommendation technologies and domains. *Recommender Systems Handbook*, 367–386. (Cited on page 14.)

BYUNG-KWAN, L. AND WEI-NA, L. 2004. The effect of information overload on consumer choice quality in an on-line environment. *Psychology & Marketing 21,* 3, 159. (Cited on page 5.)

CHANDOLA, V., BANERJEE, A., AND KUMAR, V. 2009. Anomaly Detection: A Survey. *ACM Computing Surveys 41,* 3, 1–58. (Cited on page 27.)

CHATZOPOULOU, G., EIRINAKI, M., AND POYZOTIS, N. 2009. Query Recommendations for Interactive Database Exploration. In *21st Intl. Conference on Scientific and Statistical Database Management*. 3–18. (Cited on pages 15, 17, and 18.)

CHESBROUGH, H. 2003. *Open Innovation: The New Imperative for Creating and Profiting from Technology*. Harvard Business School Press, Boston, MA. (Cited on page 23.)

CHUNG, R., SUNDARAM, D., AND SRINIVASAN, A. 2007. Integrated personal recommender systems. In *9th ACM Intl. Conference on Electronic Commerce*. Minneapolis, MN, USA, 65–74. (Cited on page 24.)

COSLEY, D., LAM, S., ALBERT, I., KONSTAN, J., AND RIEDL, J. 2003. Is seeing believing ,Äì how recommender system interfaces affect users,Äô opinions. In *CHI03*. 585–592. (Cited on page 25.)

CRESS, U. AND KIMMERLE, J. 2008. A systemic and cognitive view on collaborative knowledge building with wikis. *International Journal of Computer-Supported Collaborative Learning 3,* 2, 105–122. (Cited on page 4.)

CUBRANIC, D., MURPHY, G., SINGER, J., AND BOOTH, K. 2005. Hipikat: A Project Memory for Software Development. *IEEE Transactions of Software Engineering 31,* 6, 446–465. (Cited on page 14.)

DEKLEER, J. 1990. Using crude probability estimates to guide diagnosis. *Artificial Intelligence 45,* 3, 381–391. (Cited on page 51.)

DEKLEER, J., MACKWORTH, A., AND REITER, R. 1992. Characterizing diagnoses and systems. *Artificial Intelligence 56,* 2–3, 197–222. (Cited on pages 44, 46, 47, and 54.)

DIAS, M., LOCHER, D., LI, M., EL-DEREDY, W., AND LISBOA, P. 2008. The value of personalized recommender systems to e-business. In *2nd ACM Conference on Recommender Systems (RecSys'08)*. Lausanne, Switzerland, 291–294. (Cited on page 24.)

DUCHENEAUT, N., PATRIDGE, K., HUANG, Q., PRICE, B., AND ROBERTS, M. 2009. Collaborative Filtering Is Not Enough? Experiments with a Mixed-Model Recommender for Leisure Activities. In *17th Intl. Conference User Modeling, Adaptation, and Personalization (UMAP 2009)*. Trento, Italy, 295–306. (Cited on page 13.)

FAKHRAEE, S. AND FOTOUHI, F. 2011. TupleRecommender: A Recommender System for Relational Databases. In *22nd Intl. Workshop on Database and Expert Systems Applications (DEXA)*. Toulouse, France, 549–553. (Cited on pages 15 and 18.)

FALKNER, A., FELFERNIG, A., AND HAAG, A. 2011. Recommendation Technologies for Configurable Products. *AI Magazine 32,* 3, 99–108. (Cited on pages 2, 3, 19, and 20.)

FANO, A. AND KURTH, S. 2003. Personal choice point: helping users visualize what it means to buy a BMW. In *8th Intl. Conference on Intelligent User Interfaces (IUI 2003)*. Miami, FL, USA, 46–52. (Cited on pages 15 and 23.)

FAULRING, A., MOHNKERN, K., STEINFELD, A., AND MYERS, B. 2009. The Design and Evaluation of User Interfaces for the RADAR Learning Personal Assistant. *AI Magazine 30,* 4, 74–84. (Cited on page 24.)

FELFERNIG, A. 2007. Standardized configuration knowledge representations as technological foundation for mass customization. *IEEE Transactions on Engineering Management 54,* 1, 41–56. (Cited on pages 58 and 64.)

FELFERNIG, A., BENAVIDES, D., GALINDO, J., AND REINFRANK, F. 2013d. Towards Anomaly Explanation in Feature Models. In *Workshop on Configuration*. Vienna, Austria, 117–124. (Cited on page 71.)

FELFERNIG, A. AND BURKE, R. 2008. Constraint-based Recommender Systems: Technologies and Research Issues. In *10th ACM Intl. Conference on Electronic Commerce (ICEC'08)*. Innsbruck, Austria, 17–26. (Cited on pages 2, 15, 19, 67, 68, 77, and 78.)

FELFERNIG, A., FRIEDRICH, G., JANNACH, D., AND STUMPTNER, M. 2004. Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence 152,* 2 (February), 213–234. (Cited on pages 3, 4, 5, 28, 44, 54, 71, 74, 75, and 83.)

FELFERNIG, A., FRIEDRICH, G., JANNACH, D., STUMPTNER, M., AND ZANKER, M. 2003. Configuration knowledge representations for semantic web applications. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM) 17,* 1, 31–50. (Cited on page 64.)

FELFERNIG, A., FRIEDRICH, G., JANNACH, D., AND ZANKER, M. 2006a. An Integrated Environment for the Development of Knowledge-based Recommender Applications. *Intl. Journal of Electronic Commerce (IJEC) 11,* 2, 11–34. (Cited on pages 2, 4, 6, 12, 14, 17, 19, 59, 64, 67, 78, 79, and 91.)

FELFERNIG, A., FRIEDRICH, G., AND SCHMIDT-THIEME, L. 2007b. Introduction to the ieee intelligent systems special issue: Recommender systems. *IEEE Intelligent Systems 22,* 3, 18–21. (Cited on page 44.)

FELFERNIG, A., FRIEDRICH, G., SCHUBERT, M., MANDL, M., MAIRITSCH, M., AND TEPPAN, E. 2009a. Plausible Repairs for Inconsistent Requirements. In *IJCAI'09*. Pasadena, CA, 791–796. (Cited on pages 5, 6, 15, 17, 19, 20, 44, 54, 58, and 59.)

FELFERNIG, A., HOTZ, L., BAGLEY, C., AND TIIHONEN, J. 2014b. *Knowledge-based Configuration: From Research to Business Cases*. Elsevier/Morgan Kaufmann. (Cited on pages 2, 3, 8, and 75.)

FELFERNIG, A., ISAK, K., SZABO, K., AND ZACHAR, P. 2007a. The VITA Financial Services Sales Support Environment. In *AAAI/IAAI 2007*. Vancouver, Canada, 1692–1699. (Cited on pages 2, 6, and 67.)

FELFERNIG, A., JERAN, M., NINAUS, G., REINFRANK, F., AND REITERER, S. 2013b. Toward the next generation of recommender systems: applications and research challenges. In *Multimedia services in intelligent environments*. Springer, 81–98. (Cited on page 11.)

FELFERNIG, A., JERAN, M., NINAUS, G., REINFRANK, F., REITERER, S., AND STETTINGER, M. 2014a. Basic approaches in recommendation systems. In *Recommendation Systems in Software Engineering*. Springer, 15–37. (Cited on page 1.)

FELFERNIG, A., MAIRITSCH, M., MANDL, M., SCHUBERT, M., AND TEPPAN, E. 2009b. Utility-based repair of inconsistent requirements. In *Proceedings of IEA/AIE'09*. Springer Lecture Notes in Computer Science, vol. 5579. Tainan, Taiwan, 162–171. (Cited on pages 5 and 75.)

FELFERNIG, A., MANDL, M., PUM, A., AND SCHUBERT, M. 2010. Empirical Knowledge Engineering: Cognitive Aspects in the Development of Constraint-based Recommenders. In *23rd Intl. Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE 2010)*. Cordoba, Spain, 631–640. (Cited on pages 15, 17, and 18.)

FELFERNIG, A., REINFRANK, F., AND NINAUS, G. 2012b. Resolving Anomalies in Feature Models. In *20th Intl. Symposium on Methodologies for Intelligent Systems*. Macau, China, 1–10. (Cited on pages 5, 15, 17, 28, 37, and 64.)

FELFERNIG, A., REINFRANK, F., NINAUS, G., AND BLAZEK, P. 2014e. Redundancy Detection in Configuration Knowledge. In *Knowledge-based Configuration – From Research to Business Cases*, A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, Eds. Morgan Kaufmann Publishers, Chapter 12, 199–210. (Cited on page 27.)

FELFERNIG, A., REITERER, S., REINFRANK, F., NINAUS, G., AND JERAN, M. 2014d. Conflict Detection & Diagnosis in Configuration. In *Knowledge-based Configuration – From Research to Business Cases*, A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, Eds. Morgan Kaufmann Publishers, Chapter 7, 97–114. (Cited on pages 27 and 64.)

FELFERNIG, A., REITERER, S., STETTINGER, M., AND JERAN, M. 2014c. An overview of direct diagnosis and repair techniques in the weevis recommendation environment. In *25th Intl. Workshop on Principles of Diagnosis*. 1–6. (Cited on pages 8 and 67.)

FELFERNIG, A., REITERER, S., STETTINGER, M., REINFRANK, F., JERAN, M., AND NINAUS, G. 2013e. Recommender Systems for Configuration Knowledge Engineering. In *Workshop on Configuration*. 51–54. (Cited on page 74.)

FELFERNIG, A., SCHIPPEL, S., LEITNER, G., REINFRANK, F., ISAK, K., MANDL, M., BLAZEK, P., AND NINAUS, G. 2013c. Automated Repair of Scoring Rules in Constraint-based Recommender Systems. *AICom 26,* 2, 15–27. (Cited on pages 28 and 71.)

FELFERNIG, A. AND SCHUBERT, M. 2010. Fastdiag: A diagnosis algorithm for inconsistent constraint sets. In *DX 2010*. 31–38. (Cited on pages 71 and 75.)

FELFERNIG, A., SCHUBERT, M., AND REITERER, S. 2013a. Personalized Diagnosis for Over-Constrained Problems. In *IJCAI 2013*. 1990–1996. (Cited on pages 9, 43, and 75.)

FELFERNIG, A., SCHUBERT, M., AND ZEHENTNER, C. 2012a. An efficient diagnosis algorithm for inconsistent constraint sets. *AIEDAM 26,* 1, 53–62. (Cited on pages 3, 6, 8, 15, 17, 71, 72, 75, 79, and 83.)

FELFERNIG, A., TEPPAN, E., AND GULA, B. 2006b. Knowledge-based recommender technologies for marketing and sales. *International Journal of Pattern Recognition and Artificial Intelligence 21,* 2, 333–354. Special issue of Personalization Techniques for Recommender Systems and Intelligent User Interfaces. (Cited on page 67.)

FELFERNIG, A., ZEHENTNER, C., AND BLAZEK, P. 2011a. Corediag: Eliminating redundancy in constraint sets. In *22nd Intl. Workshop on Principles of Diagnosis. Munich, Germany*. Citeseer. (Cited on pages 6, 8, 15, 17, 18, 28, 65, 76, and 88.)

FELFERNIG, A., ZEHENTNER, C., NINAUS, G., GRABNER, H., MAALEJ, W., PAGANO, D., WENINGER, L., AND REINFRANK, F. 2011b. Group Decision Support for Requirements Negotiation. *Springer Lecture Notes in Computer Science* 7138, 1–12. (Cited on pages 14 and 15.)

FIJANY, A. AND VATAN, F. 2004. New approaches for efficient solution of hitting set problem. In *Winter International Symposium on Information and Communication Technologies*. Trinity College Dublin, Cancun, Mexico, 1–6. (Cited on pages 39 and 40.)

FLEISCHANDERL, G., FRIEDRICH, G. E., HASELBÖCK, A., SCHREINER, H., AND STUMPTNER, M. 1998. Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems 13,* 4, 59–68. (Cited on pages 28 and 58.)

FOGG, B. 2003. *Persuasive Technology – Using Computers to Change What We Think and Do*. Morgan Kaufmann Publishers. (Cited on page 20.)

FOSTER, M. AND OBERLANDER, J. 2010. User Preferences Can Drive Facial Expressions: Evaluating an Embodied Conversational Agent in a Recommender Dialog System. *User Modeling and User-Adapted Interaction (UMUAI) 20,* 4, 341–381. (Cited on page 25.)

FRIEDRICH, G. 2014. Interactive Debugging of Knowledge Bases. In *International Workshop on Principles of Diagnosis (DX'14)*. Graz, Austria, 1–4. (Cited on page 79.)

FRIEDRICH, G., GOTTLOB, G., AND NEJDL, W. 1990. Physical impossibility instead of fault models. In *AAAI*. Vol. 90. 331–336. (Cited on pages 6 and 9.)

FRIEDRICH, G., JANNACH, D., STUMPTNER, M., AND ZANKER, M. 2014. Knowledge Engineering for Configuration Systems. In *Knowledge-based Configuration – From Research to Business Cases*, A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, Eds. Morgan Kaufmann Publishers, Chapter 11, 177–198. (Cited on pages 27 and 28.)

FRIEDRICH, G. AND SHCHEKOTYKHIN, K. 2005. A general diagnosis method for ontologies. In *4th Intl. Semantic Web Conference (ISWC05)*. Number 3729 in Lecture Notes in Computer Science. Springer, Galway, Ireland, 232–246. (Cited on page 45.)

GARCIA-MOLINA, H., KOUTRIKA, G., AND PARAMESWARAN, A. 2011. Information seeking: convergence of search, recommendations, and advertising. *Communications of the ACM 54,* 11, 121–130. (Cited on page 19.)

GILES, J. 2005. Internet encyclopaedias go head to head. *Nature 438,* 7070, 900–901. (Cited on page 5.)

GODFREY, P. 1997. Minimization in cooperative response to failing database queries. *Intl. Journal of Cooperative Information Systems 6,* 2, 95–149. (Cited on pages 44 and 54.)

GOLBECK, J. 2009. *Computing with social trust.* Springer. (Cited on page 22.)

GOLDBERG, D., NICHOLS, D., OKI, B., AND TERRY, D. 1992. Using Collaborative Filtering to weave an information Tapestry. *Communications of the ACM 35,* 12, 61–70. (Cited on pages 1 and 12.)

HAMMER, S., KIM, J., AND ANDRE, E. 2010. MED-StyleR: METABO diabetes-lifestyle recommender. In *4th ACM Conference on Recommender Systems*. Barcelona, Spain, 285–288. (Cited on pages 15 and 23.)

HAPPEL, H. AND MAALEJ, W. 2008. Potentials and Challenges of Recommendation Systems for Software Engineering. In *Intl. Workshop on Recommendation Systems for Software Engineering*. Atlanta, GA, USA, 11–15. (Cited on page 14.)

HASELBÖCK, A. AND SCHENNER, G. 2014. S'UPREME. In *Knowledge-based Configuration – From Research to Business Cases*, A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, Eds. Morgan Kaufmann Publishers, Chapter 22, 327–336. (Cited on page 64.)

HOENS, T., BLANTON, M., AND CHAWLA, N. 2010. Reliable Medical Recommendation Systems with Patient Privacy. In *1st ACM Intl. Health Informatics Symposium (IHI 2010)*. Arlington, Virginia, USA, 173–182. (Cited on pages 15 and 22.)

HOFMANN, H. AND LEHNER, F. 2001. Requirements engineering as a success factor in software projects. *IEEE Software 18,* 4, 58–66. (Cited on page 15.)

HOLMES, R., WALKER, R., AND MURPHY, G. 2006. Approximate structural context matching: An approach to recommend relevant examples. *IEEE Transactions on Software Engineering 32,* 12, 952–970. (Cited on page 14.)

HOTZ, L., FELFERNIG, A., STUMPTNER, M., RYABOKON, A., BAGLEY, C., AND WOLTER, K. 2014. Configuration Knowledge Representation & Reasoning. In *Knowledge-based Configuration – From Research to Business Cases*, A. Felfernig,

L. Hotz, C. Bagley, and J. Tiihonen, Eds. Morgan Kaufmann Publishers, Chapter 6, 59–96. (Cited on pages 28, 59, 64, and 75.)

HOTZ, L. AND GÜNTER, A. 2014. KONWERK. In *Knowledge-based Configuration – From Research to Business Cases*, A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, Eds. Morgan Kaufmann Publishers, Chapter 24, 347–364. (Cited on page 64.)

HUANG, Y., CHANG, Y., AND SANDNES, F. 2010. Experiences with RFID-based interactive learning in museums. *Intl. Journal of Autonomous and Adaptive Communication Systems 3,* 1, 59–74. (Cited on pages 15 and 23.)

JANNACH, D. 2008. Finding Preferred Query Relaxations in Content-based Recommenders. In *Intelligent Techniques and Tools for Novel System Architectures*, P. Chountas, I. Petrounias, and J. Kacprzyk, Eds. Studies in Computational Intelligence, vol. 109. 81–97. (Cited on page 39.)

JANNACH, D. AND BUNDGAARD-JOERGENSEN, U. 2007. SAT: A Web-Based Interactive Advisor for Investor-Ready Business Plans. In *Intl. Conference on e-Business (ICE-B 2007)*. 99–106. (Cited on pages 15 and 23.)

JANNACH, D., SCHMITZ, T., AND SHCHEKOTYKHIN, K. 2015. Parallelized hitting set computation for model-based diagnosis. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*. (Cited on page 93.)

JANNACH, D., ZANKER, M., FELFERNIG, A., AND FRIEDRICH, G. 2010. *Recommender Systems – An Introduction.* Cambridge University Press. (Cited on pages 1 and 13.)

JANSSEN, J., BROEK, E., AND WESTERINK, J. 2011. Tune in to your emotions: a robust personalized affective music player. *User Modeling and User-Adapted Interaction (UMUAI) 22,* 3, 255–279. (Cited on page 25.)

JUNKER, U. 2004. QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In *19th Intl. Conference on Artifical Intelligence*. AAAI'04. AAAI Press, 167–172. (Cited on pages 6, 7, 28, 31, 33, 38, 47, 53, 71, 72, 75, 79, and 83.)

K. MCAREAVEY, W. L. AND MILLER, P. 2014. Computational approaches to finding and measuring inconsistency in arbitrary knowledge bases. *International Journal of Approximate Reasoning*, 1–35. (Cited on page 75.)

KAPOOR, N., CHEN, J., BUTLER, J., FOUTY, G., STEMPER, J., RIEDL, J., AND KONSTAN, J. 2007. Techlens: a researcher's desktop. In *1st Conference on Recommender Systems*. Minneapolis, Minnesota, USA, 183–184. (Cited on page 15.)

KERSTEN, M. AND MURPHY, G. 2010. Using task context to improve programmer productivity. In *14th ACM SIGSOFT Intl. Symposium on Foundations of Software Engineering*. 1–11. (Cited on page 14.)

KIMMERLE, J., MOSKALIUK, J., AND CRESS, U. 2009. Understanding learning: the wiki way. In *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*. ACM, 3. (Cited on page 2.)

KONSTAN, J., MILLER, B., MALTZ, D., HERLOCKER, J., GORDON, L., AND RIEDL, J. 1997. GroupLens: applying collaborative filtering to Usenet news. *Communications of the ACM 40,* 3, 77–87. (Cited on pages 12, 18, 19, 21, 67, and 78.)

KONSTAN, J. AND RIEDL, J. 2012. Recommender systems: from algorithms to user experience. *User Modeling and User-Adapted Interaction (UMUAI) 22,* 1, 101–123. (Cited on pages 12 and 13.)

KOREN, Y., BELL, R., AND VOLINSKY, C. 2009. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer 42,* 8, 30–37. (Cited on page 12.)

KREBS, T. 2014. EngCon. In *Knowledge-based Configuration – From Research to Business Cases*, A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, Eds. Morgan Kaufmann Publishers, Chapter 23, 337–346. (Cited on page 64.)

LEE, T., PARK, Y., AND PARK, Y. 2008. A time-based approach to effective recommender systems using implicit feedback. *Expert Systems with Applications 34,* 4, 3055–3062. (Cited on page 25.)

LEITNER, G., FERCHER, A., FELFERNIG, A., AND HITZ, M. 2012. Reducing the Entry Threshold of AAL Systems: Preliminary Results from Casa Vecchia. In *13th Intl. Conference on Computers Helping People with Special Needs*. Linz, Austria, 709–715. (Cited on pages 15 and 22.)

LEMAY, M., HAAS, J., AND GUNTER, C. 2009. Collaborative Recommender Systems for Building Automation. In *Hawaii Intl. Conference on System Sciences, Waikoloa, Hawaii, 2009*. Hawaii, USA, 1–10. (Cited on pages 15 and 22.)

LIGGESMEYER, P. 2009. *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. Springer Science & Business Media. (Cited on page 6.)

LINDEN, G., SMITH, B., AND YORK, J. 2003. Amazon.com Recommendations – Item-to-Item Collaborative Filtering. *IEEE Internet Computing 7,* 1, 76–80. (Cited on pages 12, 21, and 78.)

MACKWORTH, A. 1977. Consistency in Networks of Relations. *AI Journal 8,* 1, 99–118. (Cited on pages 59, 70, and 82.)

MANDL, M., FELFERNIG, A., TIIHONEN, J., AND ISAK, K. 2011. Status Quo Bias in Configuration Systems. In *24th Intl. Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE 2011)*. Syracuse, NY, USA, 105–114. (Cited on page 20.)

MARQUES-SILVA, J., HERAS, F., JANOTA, M., PREVITI, A., AND BELOV, A. 2013. On computing minimal correction subsets. In *IJCAI'2013*. 615–622. (Cited on page 75.)

MARQUES-SILVA, J. AND PREVITI, A. 2014. On Computing Preferred zehenMUSes and MCSes. In *SAT 2014*. 58–74. (Cited on page 75.)

MARQUES-SILVA, J. AND SAKALLAH, K. 1996. Grasp: A new search algorithm for satisfiability. In *Intl. Conference on Computer-Aided Design*. Santa Clara, CA, 220–227. (Cited on page 45.)

MARTIN, F., DONALDSON, J., ASHENFELTER, A., TORRENS, M., AND HANGARTNER, R. 2011. The Big Promise of Recommender Systems. *AI Magazine 32,* 3, 19–27. (Cited on page 24.)

MASTHOFF, J. 2011. Group recommender systems: Combining individual models. *Recommender Systems Handbook*, 677–702. (Cited on pages 12, 16, and 17.)

MAYER-SCHÖNBERGER, V. AND CUKIER, K. 2013. *Big data: A revolution that will transform how we live, work, and think.* Houghton Mifflin Harcourt. (Cited on page 1.)

MCCAREY, F., CINNEIDE, M., AND KUSHMERICK, N. 2005. Rascal – A Recommender Agent for Agile Reuse. *Artificial Intelligence Review 24,* 3–4, 253–273. (Cited on page 14.)

MCSHERRY, D. 2004. Maximally successful relaxations of unsuccessful queries. In *15th Conference on Artificial Intelligence and Cognitive Science*. Galway, Ireland, 127–136. (Cited on pages 44, 48, 49, and 54.)

MISIRLI, A., BENER, A., AND KALE, R. 2011. AI-Based Software Defect Predictors: Applications and Benefits in a Case Study. *AI Magazine 32,* 2, 57–68. (Cited on page 14.)

MITTAL, S. AND FALKENHAINER, B. 1990. Dynamic Constraint Satisfaction Problems. In *Proceedings of the eighth national conference on artificial intelligence (AAAI-90).* Boston, MA, USA, 25–32. (Cited on page 64.)

MITTAL, S. AND FRAYMAN, F. 1989. Towards a Generic Model of Configuration Tasks. In *11th International Joint Conference on Artificial Intelligence (IJCAI-89).* Vol. 2. Detroit, Michigan, USA, 1395–1401. (Cited on pages 58 and 64.)

MOBASHER, B. AND CLELAND-HUANG, J. 2011. Recommender Systems in Requirements Engineering. *AI Magazine 32,* 3, 81–89. (Cited on pages 14 and 15.)

MURPHY, J., HOFACKER, C., AND MIZERSKI, R. 2006. Primacy and recency effects on clicking behavior. *Journal of Computer-Mediated Communication 11,* 2, 522–535. (Cited on pages 91 and 92.)

NIELSEN, J. 1994. *Usability engineering*. Elsevier. (Cited on pages 6 and 85.)

NINAUS, G., FELFERNIG, A., STETTINGER, M., REITERER, S., LEITNER, G., WENINGER, L., AND SCHANIL, W. 2014. Intellireq: Intelligent techniques for software requirements engineering. In *21st European Conference on Artificial Intelligence/Prestigious Applications of Intelligent Systems (PAIS 2014), p. to appear, Prague, Czech Republic*. (Cited on page 15.)

O'SULLIVAN, B., PAPADOPOULOS, A., FALTINGS, B., AND PU, P. 2007. Representative explanations for over-constrained problems. In *AAAI'07*. 323–328. (Cited on pages 4, 44, 54, and 75.)

PAZZANI, M. AND BILLSUS, D. 1997. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning 27*, 313–331. (Cited on pages 12, 16, 67, and 78.)

PAZZANI, M. J. AND BILLSUS, D. 2007. Content-based recommendation systems. In *The adaptive web*. Springer, 325–341. (Cited on page 1.)

PEISCHL, B., ZANKER, M., NICA, M., AND SCHMID, W. 2010. Constraint-based Recommendation for Software Project Effort Estimation. *Journal of Emerging Technologies in Web Intelligence 2,* 4, 282–290. (Cited on page 14.)

PINXTEREN, Y., GELIJNSE, G., AND KAMSTEEG, P. 2011. Deriving a recipe similarity measure for recommending healthful meals. In *16th Intl. Conference on Intelligent User Interfaces*. Palo Alto, CA, USA, 105–114. (Cited on pages 15 and 23.)

PRIBIK, I. AND FELFERNIG, A. 2012. Towards Persuasive Technology for Software Development Environments: An Empirical Study. In *Persuasive Technology Conference (Persuasive 2012)*. 227–238. (Cited on pages 15, 20, 21, and 25.)

R. MOONEY, L. R. 2004. Content-based book recommending using learning for text categorization. *User Modeling and User-Adapted Interaction 14,* 1, 37–85. (Cited on page 16.)

RAMIEZ-GONZALES, G., MUNOZ-MERINO, P., AND DELGADO, K. 2010. A Collaborative Recommender System Based on Space-Time Similarities. *IEEE Pervasive Computing 9,* 3, 81–87. (Cited on page 24.)

RAMOS, C., AUGUSTO, J., AND SHAPIRO, D. 2008. Ambient Intelligence – the Next Step for Artificial Intelligence. *IEEE Intelligent Systems 23,* 2, 15–18. (Cited on page 24.)

REITER, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence 32,* 1 (April), 57–95. (Cited on pages 3, 4, 6, 7, 9, 17, 35, 44, 46, 47, 53, 54, 68, 75, 79, and 93.)

REITERER, S. 2015. An integrated knowledge engineering environment for constraint-based recommender systems. In *1st International Workshop on Personalization and Recommender Systems in Financial Services (FinRec'15)*. 11–18. (Cited on pages 6, 8, and 88.)

REITERER, S., FELFERNIG, A., BLAZEK, P., LEITNER, G., REINFRANK, F., AND NINAUS, G. 2014. WeeVis. In *Knowledge-based Configuration – From Research to Business Cases*, A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, Eds. Morgan Kaufmann Publishers, Chapter 24, 297–307. (Cited on page 57.)

REITERER, S., FELFERNIG, A., JERAN, M., STETTINGER, M., WUNDARA, M., AND EIXELSBERGER, W. 2015a. A wiki-based environment for constraint-based recommender systems applied in the e-government domain. In *3rd Workshop on PErsonalization in eGOVernment and Smart Cities: Smart Services for Smart Territories, UMAP 2015*. 1–10. (Cited on pages 8 and 77.)

REITERER, S., STETTINGER, M., JERAN, M., EIXELSBERGER, W., AND WUNDARA, M. 2015b. Advantages of extending wiki pages with knowledge-based recommendations. In *15rd International Conference on Knowledge Technologies and Data-Driven Business, i-KNOW 2015*. (Cited on page 2.)

ROBILLARD, M., WALKER, R., AND ZIMMERMANN, T. 2010. Recommendation Systems for Software Engineering. *IEEE Software 27,* 4, 80–86. (Cited on pages 13 and 14.)

SABIN, D. AND WEIGEL, R. 1998. Product Configuration Frameworks - A Survey. *IEEE Intelligent Systems 14,* 4, 42–49. (Cited on pages 19 and 75.)

SCARDAMALIA, M. AND BEREITER, C. 2003. Knowledge building. in encyclopedia of education (pp. 1370-1373). (Cited on pages 2 and 5.)

SCHAFER, J., KONSTAN, J., AND RIEDL, J. 2011. E-Commerce Recommendation Applications. *Journal of Data Mining and Knowledge Discovery 5,* 1–2, 115–153. (Cited on pages 13 and 21.)

SCHAFER, J. B., FRANKOWSKI, D., HERLOCKER, J., AND SEN, S. 2007. Collaborative filtering recommender systems. In *The adaptive web*. Springer, 291–324. (Cited on page 1.)

SCHUBERT, M. AND FELFERNIG, A. 2010. A Diagnosis Algorithm for Inconsistent Constraint Sets. In *21st Intl. Workshop on the Principles of Diagnosis*. (Cited on page 75.)

SCHUBERT, M. AND FELFERNIG, A. 2011. BFX: Diagnosing Conflicting Requirements in Constraint-Based Recommendation. *International Journal on Artificial Intelligence Tools 20,* 2, 297–312. (Cited on page 39.)

SCHUBERT, M., FELFERNIG, A., AND MANDL, M. 2010. FastXPlain: Conflict Detection for Constraint-Based Recommendation Problems. In *Trends in Applied Intelligent Systems (proc. of 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2010)*, N. García-Pedrajas, F. Herrera, C. Fyfe, J. Benítez, and M. Ali, Eds. Lecture Notes in Computer Science, vol. 6096. Springer, Cordoba, Spain, 621–630. (Cited on page 39.)

SHCHEKOTYKHIN, K. M., FRIEDRICH, G., FLEISS, P., AND RODLER, P. 2012. Interactive ontology debugging: Two query strategies for efficient fault localization. *Web Semantics: Science, Services and Agents on the World Wide Web 12–13*, 88–103. (Cited on page 65.)

SOININEN, T., TIIHONEN, J., MÄNNISTÖ, T., AND SULONEN, R. 1998. Towards a General Ontology of Configuration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM) 12,* 4, 357–372. (Cited on page 64.)

SOMMERVILLE, I. 2007. *Software Engineering*. Pearson. (Cited on page 15.)

STUMPTNER, M. 1997. An overview of knowledge-based configuration. *Aicommunications*, 111–125. (Cited on pages 2, 58, 64, and 75.)

STUMPTNER, M., FRIEDRICH, G., AND HASELBÖCK, A. 1998. Generative Constraint-based Configuration of Large Technical Systems. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM) 12,* 4, 307–320. (Cited on page 64.)

SUROWIECKI, J. 2005. The wisdom of crowds, anchor. (Cited on page 5.)

TAYEBI, M., JAMALI, M., ESTER, M., GLAESSER, U., AND FRANK, R. 2011. Crimewalker: A Recommender Model for Suspect Investigation. In *ACM Conference on Recommender Systems (RecSys'11).* Chicago, IL, 173–180. (Cited on pages 15 and 22.)

TERVEEN, L. AND HILL, W. 2001. Beyond Recommender Systems: Helping People Help Each Other. In *HCI in the New Millennium*. Addison-Wesley, 487–509. (Cited on page 23.)

THIESSE, F. AND MICHAHELLES, F. 2009. Building the Internet of Things Using RFID. *IEEE Internet Computing 13,* 3, 48–55. (Cited on pages 23 and 24.)

THORLEUCHTER, D., VANDENPOEL, D., AND PRINZIE, A. 2010. Mining ideas from textual information. *Expert Systems with Applications 37,* 10, 7182–7188. (Cited on pages 15 and 23.)

TIIHONEN, J. AND FELFERNIG, A. 2010. Towards recommending configurable offerings. *International Journal of Mass Customization 3,* 4, 389–406. (Cited on page 65.)

TIIHONEN, J., HEISKALA, M., ANDERSON, A., AND SOININEN, T. 2013. WeCoTin - A practical logic-based sales configurator. *AI Communications 26,* 1, 99–131. (Cited on pages 4, 58, and 64.)

TSANG, E. 1993. *Foundations of Constraint Satisfaction (Computation in Cognitive Science).* Academic Press. (Cited on pages 44, 45, and 59.)

TUZHILIN, A. AND KOREN, Y. 2008. 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Price Competition. 1–34. (Cited on page 12.)

WALTER, R., ZENGLER, C., AND KÜCHLIN, W. 2013. Applications of MaxSAT in Automotive Configuration. In *Workshop on Configuration*. Vienna, Austria, 21–28. (Cited on page 75.)

WIESNER, M. AND PFEIFER, D. 2010. Adapting Recommender Systems to the Requirements of Personal Health Record Systems. In *1st ACM Intl. Health Informatics Symposium (IHI 2010)*. Arlington, Virginia, USA, 410–414. (Cited on pages 15 and 23.)

WILSON, D., LELAND, S., GODWIN, K., BAXTER, A., LEVY, A., SMART, J., NAJJAR, N., AND ANDAPARAMBIL, J. 2009. SmartChoice: An Online Recommender System to Support Low-Income Families in Public School Choice. *AI Magazine 30,* 2, 46–58. (Cited on page 15.)

WINOTO, P. AND TANG, T. 2010. The role of user mood in movie recommendations. *Expert Systems with Applications 37,* 8, 6086–6092. (Cited on page 25.)

WINTERFELDT, D. AND EDWARDS, W. 1986. *Decision Analysis and Behavioral Research*. Cambridge University Press. (Cited on pages 49, 50, and 52.)

XU, S., JIANG, H., AND LAU, F. 2008. Personalized Online Document, Image and Video Recommendation via Commodity Eye-tracking. In *ACM Conference on Recommender Systems (RecSys'08)*. 83–90. (Cited on page 25.)

Y. MALITSKY, B. O'SULLIVAN, A. P. AND MARQUES-SILVA, J. 2014. A Portfolio Approach to Enumerating Minimal Correction Subsets for Satisfiability Problems. In *CPAIOR'2014*. 368–376. (Cited on page 75.)

YUAN, N., ZHENG, Y., ZHANG, L., AND XIE, X. 2012. T-Finder: A Recommender System for Finding Passengers and Vacant Taxis. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 1–14. (Cited on pages 15 and 22.)