Dissertation

Configuration and Recommendation Technologies for the Internet of Things

Seda Polat Erdeniz

Graz, 2019

Institute for Software Technology
Graz University of Technology

Supervisor/First reviewer: Univ.-Prof. Dipl.-Ing. Dr. techn. Alexander Felfernig
Second reviewer: Associate Prof. Dr. Ilias G. Maglogiannis

Abstract (English)

Artificial Intelligence (AI) refers to technologies which are capable of performing tasks normally requiring human intelligence. In recent history, AI has grown into a burgeoning, young sub-field of computer science and its ever-growing importance has made it a topic of much discussion in this community. Nowadays, the confluence of three powerful drivers (exponential data growth, more sophisticated distributed networks and smarter algorithms) has propelled Artificial Intelligence to the center stage for many applications. Simultaneously, the Internet of Things (IoT) is a term that has been introduced in recent years to define objects that are able to connect and transfer data via the Internet. 'Thing' refers to any device which is connected to the internet and transfers information to other devices. IoT is used to connect a wide range of things such as vehicles, mobile devices, sensors, industrial equipment's and manufacturing machines. This allows for the development of various smart systems including smart city, home, grid, industry, vehicle, health and environmental monitoring. There are numerous potential opportunities and benefits presented by the combination of AI and IoT.

This thesis consists of research in emerging artificial intelligence methods, especially focusing on configuration and recommendation technologies, supporting users of IoT applications. Configuration and recommendation technologies are successfully applied artificial intelligence methods. In order to utilize these technologies in IoT, new adaptations are required due to the limitations of IoT devices and its applications. Therefore, novel approaches and algorithms for configuration and recommendation technologies are introduced in this thesis. The proposed approaches have been designed to assist IoT gateway/sensor users starting from the beginning with the installation phase of the IoT infrastructure in a new environment. In order to support IoT infrastructure installation activities, new configuration approaches and heuristics have been proposed. Those heuristics help to improve performance (in terms of runtime and solution quality) of configuration systems in the cases of solving or diagnosing inconsistencies of huge sets of constraints. After installing a new IoT infrastructure successfully, supporting new users of the IoT environment is important because most users are not experienced in using IoT applications and devices. Therefore, in this thesis, new recommendation approaches are also proposed to assist IoT users.

All proposed approaches and algorithms have been tested and evaluated based on real-world datasets from pilot applications of an IoT gateway project which is called *AGILE IoT*. Experimental results show that the proposed approaches improve performance of configuration and recommendation technologies in IoT applications. Finally, all proposed configuration systems related heuristics were gathered in a open-source project, *CSPHeuristix Library*, and were introduced for further usage in teaching, research and industrial needs. The introduction of this resource offers limitless opportunities for continued innovation in this field. Some potential applications thereof are detailed at the end of this thesis.

Abstract (German)

Künstliche Intelligenz (Artificial Intelligence, AI) bezieht sich auf Technologien und Aufgaben, die normalerweise menschliche Intelligenz erfordern. Heutzutage hat die Verschmelzung von drei leistungsstarken Treibern mit exponentiellem Datenwachstum, ausgefeilteren verteilten Netzwerken und Algorithmen künstliche Intelligenz für viele Anwendungen in den Mittelpunkt gerückt. Andererseits ist das Internet der Dinge (Internet of Things, IoT) ein Begriff, der in den letzten Jahren eingeführt wurde, um Objekte zu definieren, die in der Lage sind, Daten zu verbinden und Daten über das Internet zu übertragen. „Thing" bezieht sich auf ein Gerät, das mit dem Internet verbunden ist und die Geräteinformationen an andere Geräte überträgt. IoT wird verwendet, um eine Vielzahl von Dingen zu verbinden, z. B. Fahrzeuge, mobile Geräte, Sensoren, Industrieanlagen und Fertigungsmaschinen, um verschiedene intelligente Systeme zu entwickeln. Dazu gehören Smart City, Home, Grid, Industry, Vehicle, Health und Umweltüberwachung.

Diese Arbeit besteht aus Forschungsergebnissen in aufkommenden Methoden der künstlichen Intelligenz, insbesondere der Fokussierung Konfigurations- und Empfehlungstechnologien zur Unterstützung der Benutzer von IoT-Anwendungen. Konfigurations- und Empfehlungstechnologien sind erfolgreich angewandte Methoden der künstlichen Intelligenz. Um Konfigurations- und Empfehlungstechnologien im IoT zu nutzen, werden in dieser Arbeit neuartige Ansätze und Algorithmen eingeführt. Vorgeschlagene Ansätze wurden entwickelt, um Benutzern von IoT-Gateways/Sensoren dabei zu helfen, von der Installationsphase der IoT-Infrastruktur in eine neue Umgebung zu starten. Zur Unterstützung der IoT-Infrastrukturinstallationsaktivitäten wurden neue Konfigurationsansätze und Heuristiken vorgeschlagen.

Diese Heuristiken helfen dabei, die Leistung (in Bezug auf Laufzeit und Lösungsqualität) von Konfigurationssystemen zu verbessern. Im Falle von großen Datenmengen und von Einschränkungen sollen die Heuristiken an der Diagnose und an der Leistungsverbesserungen Abhilfe schaffen. Nach der erfolgreichen Installation einer neuen IoT-Infrastruktur ist die Unterstützung neuer Benutzer der IoT-Umgebung erforderlich, da die meisten von ihnen keine Erfahrung mit der Verwendung von IoT-Anwendungen und -Geräten haben. Daher werden in dieser Arbeit auch neue empfehlungsansätze vorgeschlagen, um IoT-Benutzer zu unterstützen.

Alle vorgeschlagenen Ansätze und Algorithmen wurden basierend auf realen Daten aus Pilotanwendungen eines IoT-Gateway-Projekts under dem Namen *AGILE IoT* getestet. Die experimentellen Ergebnisse zeigen, dass die vorgeschlagenen Ansätze die Leistung von Konfigurations- und Empfehlungstechnologien in IoT-Anwendungen verbessern. Schließlich wurden alle vorgeschlagenen Heuristiken für Konfigurationssysteme in einem Open-Source-Projekt zusammengetragen, und die CSPHeuristix Library wurde für die weitere Verwendung in Lehre, Forschung und Industrie entwickelt.

Acknowledgement

First of all, I would like to thank my advisor, Prof. Alexander Felfernig for guiding and supporting me over the years.

Then, I'd like to thank my fellow graduate students, research technicians, collaborators who contributed to this research. I am very grateful to all of you. I would like to thank my thesis committee members for all of their guidance through this process.

Finally, I would like to thank my family for the love, support, and constant encouragement I have gotten over the years. In particular, I would like to thank my husband Pamir, my human-son Paul Attila, my dog-son Büdü, my parents and my sister.

Seda Polat Erdeniz
Graz, 2019

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

List of Figures

List of Tables

Chapter 1

Introduction

The Internet of Things (IoT) is an emerging paradigm that envisions a networked infrastructure enabling different types of devices to be interconnected. It creates different kinds of artifacts (e.g., services and applications) in various application domains such as health monitoring, sports monitoring, animal monitoring, enhanced retail services, and smart homes. Additionally, in recent years, artificial intelligence (AI) with AI-related research and applications have become an important topic. Industry is interested in potential uses of AI. However, with the popularization of IoT, permeation of sensor networks, emergence of big data, increase in size of the information community, and interlinking information throughout human society, physical space, and cyberspace, the information environment related to the current development of AI has profoundly changed, especially in IoT. Thus, AI faces important adjustments with new breakthroughs in IoT (Pan, 2016).

This thesis deals with IoT-based challenges of two important AI technologies: configuration and recommendation systems. *Configuration systems* have a long history as a successful AI application area. These frameworks have advanced from their rule-based origins to the utilization of higher level representations, for example, different types of constraint satisfaction, description logic, or functional reasoning (Felfernig et al., 2004). Configuration (also mentioned as product configuration or product customization) is an activity of customising a product that allows the utilisation of AI techniques for searching for a valid configuration to meet the needs of a particular customer. Some common application domains are the automotive industry, the telecommunication industry, the computer industry, and power electric transformers. *Recommender systems* are software tools and techniques providing suggestions to users. The suggestions provided are aimed at supporting their users in various decision-making processes, such as what items to buy, what music to listen, or what news to read. *Item* is the general term used to denote what the system recommends to users. An RS normally focuses on specific *items* (e.g., CDs, or news) and accordingly its design, its graphical user interface, and the core recommendation technique used to generate the recommendations are all customized to provide useful and effective suggestions for these items. Recommender systems are progressively attracting the interest of new application domains as a valuable solution to increase system autonomy and efficiency (Ricci et al., 2015).

Modern embedded systems engaged with IoT scenarios can utilize the advantages offered by context-aware and profile-driven recommendations and configurations. Therefore, in this thesis, configuration and recommendation system requirements and challenges in IoT (especially based on the IoT gateway project *AGILE IoT* – see Section 2.4) will be analyzed and novel methods to improve performance and benefits to support IoT environment stakeholders will be proposed.

1.1. Motivation

This thesis focuses on algorithms and techniques of configuration and recommendation technologies in IoT (especially scenarios in the IoT gateway project *AGILE IoT* (see Section 2.4). As aforementioned, state-of-the-art in these technologies need improvements to be successfully employed in IoT scenarios. In this context, this thesis covers the following challenges:

- **(1) Ramp-up configuration:** supporting inexperienced end-users during installation of a new IoT environment,
- **(2) Runtime configuration:** providing effective solutions for optimized configuration of IoT protocols (e.g. communication protocols),
- **(3) Recommendation:** supporting inexperienced end-users during their usages of their IoT environments,
- **(4) Improving *runtime performance*:** providing quick configurations/recommendations in real-time scenarios,
- **(5) Improving *prediction quality*:** providing configurations/recommendations which are acceptable by end-users.

(1) Ramp-up configuration: In contrast with the developments in (Falkner and Schreiner, 2014; Krebs et al., 2002; Perera et al., 2013), the "ramp-up" configuration approach developed for AGILE IoT focuses on advanced testing methods for supporting configuration knowledge engineering. This configuration also focuses on approaches to improve configurator usability by the inclusion of different types of personalized consistency restoration methods. Initial approaches to include recommendation methods into configuration problem solving are documented, for example, in (Tiihonen and Felfernig, 2010).

These approaches do not adequately take into account the issue of consistency management. Hence, this is a major focus of our work in the AGILE project. IoT gateways can be deployed in different domains which require pre-configuration that estimates the needed hardware and software components/devices to be deployed in the ramp-up phase of the system. We denote this type of configuration as *ramp-up configuration* because each scenario requires a specific set of hardware components and software components (including apps) to "ramp-up" the system.

(2) Runtime configuration: Although different from basic IoT scenarios (Atzori et al., 2010), there exist applications that support the configuration of systems including hardware and software components. Falkner and Schreiner (Falkner and Schreiner, 2014) introduce approaches to the configuration of railway interlocking systems as examples of complex industrial systems designed on the basis of constraint-based configuration technologies. Krebs et al. (Krebs et al., 2002) show the application of configuration technologies in the area of car periphery supervision (including detection of the car environment, the recognition of hazardous situations, and the handling of difficult traffic situations). Related applications are pre-crash detection, the detection of obstacles, and parking assistance. Car configuration processes have to take into account existing hardware components and to combine these with the corresponding software units. Finally, Perera et al. (Perera et al., 2013) introduce an approach to the end-user-oriented configuration of IoT middleware components.

Specifically in IoT scenarios, modern embedded systems support a rich set of connectivity solutions (e.g., 3G, LTE, TD-LTE, FDD-LTD, WIMAX, and Lora). In this context, configuration technologies play an important role in terms of suggesting optimal connectivity configurations. Such configurations include a collection of connectivity solutions that are needed to support a set of active applications (apps). Criteria that must be taken into account are, for example, location information, available connectivity, performance and reliability requirements, contractual aspects, and cost.

(3) Recommendation: Compared to other recommendation scenarios, IoT-based applications enable a deeper understanding of user preferences and behaviors which can primarily be explained by the availability of heterogeneous information sources (Frey et al., 2015; Yao et al., 2016). *Personalized shopping*, for example, is a core element of IoT technology based retail environments (Magerkurth et al., 2011). Customers entering a store receive recommendations regarding items and corresponding price offers – these recommendations depend on the condition of the offered items. For example, if the durability of some of the offered items decreases, corresponding special offers could be announced to the customer. Important IoT related aspects of recommendation are *automated quality control of items*, *context-dependent pricing*, and *targeted product information*. The recommendation approach presented by (Magerkurth et al., 2011) follows a knowledge-based (rule-based) approach. Such rules can be generated on the basis of techniques such as sequence mining, i.e., what the next item a customer is interested in will be. Valtolina et al. (Valtolina et al., 2014) introduce a *household scenario* where users ask for recommendations regarding recipes. In the context of an IoT infrastructure, a recommender system must not solely rely upon the preferences of the user but can take into account information from further sources.

The availability of orthogonal data sources provided by different IoT devices will help to increase the prediction quality of recommendation algorithms, however, visualization techniques and persuasive interfaces are an important additional means of making recommendations acceptable for users (Jannach et al., 2010). Munoz-Organero et al. (Munoz-Organero et al., 2010) introduces *technology fairs* as a scenario in which context-aware recommender systems can be applied. In such a scenario, users can receive information about exhibits of relevance and also be informed about lectures to attend depending on their personal preferences. Similar to knowledge-based recommender systems (Burke et al., 2011; Felfernig and Burke, 2008), knowledge-based configuration (Felfernig et al., 2014a) is a process in which users specify their requirements and the configuration system (often denoted as *configurator*) provides feedback.

There are many different scenarios in which recommendation technologies can play a role in IoT. For example, a user installs an IoT gateway to make his/her home smarter. The user is already using temperature and pressure sensors and alarm applications. He/she buys a gas sensor, plugs it onto the gateway and opens the management user interface of the gateway in the web browser. The user wants to receive some app recommendations according to the overall setting on the gateway. He or she presses the *get recommendation* button on the management user interface. Based on the gateway profile, the recommender applies collaborative filtering Koren (2010) on the profiles knowledge base. The recommender returns the recommended applications for the gateway. The user selects the *fire alarm app* from the recommendation list and installs the app. He/she also wants to implement their own smart home application on the IoT gateway. Therefore the user starts to build a workflow with the IoT gateway's development environment by adding a temperature node which collects data from the installed temperature sensor. The user then activates the recommender to determine how to best extend the current workflow. On the basis of the current workflow content (the temperature node), a collaborative filtering recommender can suggest possible extensions (e.g., a pressure node).

(4) Improving *runtime performance*: Configuration systems must be able to solve complex constraints and deal with inconsistencies occurring in different contexts. This is especially true in interactive settings, in which users specify requirements and a constraint solver must identify solutions. In this case, inconsistencies may arise more often. Therefore, diagnosis algorithms are required to find repairs for these *unsolvable problems*. Runtime efficiency of configuration system is especially crucial in real-time scenarios such as production scheduling, robot control, and communication networks. For such scenarios, diagnosis algorithms should determine solutions within predefined time limits. Furthermore, an online recommendation system should also be able to calculate recommendations as fast as possible in order to catch the users on the system. Knowledge-based recommendation (Burke, 2002) attempts to suggest objects based on inferences about a user's needs and preferences. One type of knowledge-based recommender systems is *constraint-based recommendation* (Felfernig and Burke, 2008) in which the recommendation tasks can be composed of many variables and constraints. In these settings, achieving an acceptable runtime performance can quickly become very challenging.

Jannach (Jannach, 2013) proposes a learning solution for domain specific heuristics. In this work, it is mentioned that solving complex configuration problems often requires the usage of domain-specific search heuristics which have to be explicitly modeled by domain experts and knowledge engineers. Li et al. (Li and Epstein, 2010) apply a clustering approach to divide a search problem into sub-problems. The authors apply a variable and value ordering heuristic over these clusters. In our approach, similar problems are clustered to be able to determine cluster-specific search heuristics. We do not divide a problem into sub-problems. O'Sullivan et al.(O'Sullivan et al., 2004) use a constraint solver and decision tree learning to solve a CSP query of a user. The ultimate goal of this work is also to improve the overall efficiency of search. Balduccini et al. (Balduccini, 2011) implemented domain specific heuristics for Answer Set Programming Solvers. Heuristics are learned by the proposed platform. This work is similar to ours but does not take into account clustering mechanisms. It learns heuristics for domains as *Domain-Specific Heuristics* but does not cluster the underlying problems. Epstein et al. (Epstein and Wallace, 2006) postulate several types of crucial sub-problems and show how local search can be harnessed to solve them before global search is triggered. A variety of heuristics and metrics are then used to guide (global) solution search. Liu et al. (Liu et al., 2008) generate a variable ordering strategy for solving Disjunctive Temporal Problems (DTPs) which are an essential aspect for building systems that reason about time and actions. DTP model events and their relationships (as distances between events) and provide the means to specify the temporal elements of an episode with a temporal extent. Ciccio et al. (Di Ciccio et al., 2017) introduce techniques which guarantee the consistency of the discovered models and keep the most interesting constraints in the pruned set. The level of interest to the user is dictated by user-specified prioritization criteria. Merhej et al. (Merhej et al., 2017) introduce an approach to assign weights to rules of thumb by sampling from a pool of possible repairs.

Improving the runtime performance of configuration and recommendation algorithms is beneficial especially when these improvements are applied in IoT applications (limited resources, real-time interactions, huge set of variables and constraints, etc.). Therefore, search should be guided by intelligent search strategies called *heuristics* (Groër et al., 2010). The existing heuristics can improve the runtime performance of configuration and recommendation in IoT. However, more advanced heuristics can be identified when they are learned based on the past transactions of the problem domain.

(5) Improving *prediction quality*: In recommender systems, prediction *quality* is by far the most discussed property in the recommendation system literature. At the base of the vast majority of recommender systems is something of a prediction engine. This engine may predict user opinions over items (e.g. ratings of movies) or the probability of usage (e.g. purchase). A basic assumption in a recommender system is that a system capable of providing more accurate predictions will be preferred by the user. Thus, many researchers set out to find algorithms that provide better predictions (Ricci et al., 2011). Besides, configuration systems use constraint solving algorithms and heuristics to improve the overall runtime performance. However, these methods do not take the performance criterion *quality* into account. There exist some approaches which are based on collaborative filtering that aim to improve the accuracy of solutions but these approaches lack improvements to runtime.

Ardissono et al. (Ardissono et al., 2002) presented a framework for the business-oriented personalized configuration. They classified the system users according to their expertise in the domain. The personalization of the interaction and dialogues are adapted according to this user modeling. They have applied this framework to a telecommunication switches domain. They experienced successful results (increased speed in configuration process) in user configurations. However, this approach uses explicit personalization rules whereas, in our model, search heuristics are learned from historical transactions. Felfernig et al. (Felfernig and Burke, 2008) presented an overview of several techniques to solve constraint based recommendation tasks. An open research issue in this particular context the question of how to efficiently guide the solution search towards the relevant items. Sandvig et. al (Sandvig et al., 2007) adapted association rule mining to collaborative filtering in order to discover valuable patterns between items that have similar ratings. Using item ratings such as *like* or *dislike*, the authors could achieve a good level of accuracy in the results. Although the achieved accuracy comes very close to the compared *k-nearest neighbor* method, the paper does not cover the performance aspect in more detail. Zanker (Zanker, 2008) proposed a system

that learns rule-based preferences from successful interactions in historic transaction data. The author uses collaborative filtering to derive preferences from a user's nearest neighbors. The paper demonstrates that this approach can improve the overall prediction quality of the recommender system. However, this work does not focus on runtime performance any more. Another work by Zanker (Zanker et al., 2010) presented an approach that ranks the recommended items according to their degree of constraint fulfillment. They used historical transactions to evaluate the prediction quality of the recommendation results. This work focuses on relaxation of low weighted constraints whereas our approach satisfies all user constraints. While constraint relaxation may help to reduce the runtime, there is no explicit runtime performance evaluation of the proposed approach in (Zanker et al., 2010).

Improving the configuration/recommendation *quality* is beneficial in IoT applications since these systems are supporting end-users who have different preferences. Configurations/recommendations that do not satisfy user's needs and preferences should be ignored.

1.2. Research Objectives

In the previous section we have discussed challenges in the fields of configuration and recommendation systems in the context of this thesis. These challenges raised the following research questions:

1. **How can we address the configuration and recommendation needs in IoT?**
   Compared to other configuration/recommendation scenarios, IoT-based applications enable a deeper understanding of user preferences and behaviors which can primarily be explained by the availability of heterogeneous information sources (Amato et al., 2013). For instance, *personalized shopping* is a core element of IoT technology based retail environments (Magerkurth et al., 2011). Customers entering a store receive recommendations regarding items and corresponding price offers – these recommendations depend on the condition of the offered items. For example, if the expiration date of some of the offered items is approaching, and this information is detected via their RFID (radio frequency identification) tags (Finkenzeller, 2010), corresponding special offers can be announced to the customer.

   In AGILE IoT, configuration and recommendation systems play critical roles such as: supporting inexperienced end-users during installation of a new IoT environment, providing effective solutions for optimized configuration of IoT protocols (e.g. communication protocols), and supporting inexperienced end-users during their use of their new IoT environments. In this context, we focus specifically on providing solutions for configuration and recommendation requirements of AGILE IoT and its pilot applications. This leads to our first research question:

   (Q1) How can we adapt state-of-the-art configuration and recommendation systems to IoT applications in order to improve experiences of IoT environment stakeholders?

2. **How can we improve runtime performance of configuration systems for IoT applications?**
   Various search techniques can be applied in order to improve the performance of CSP solvers. Searching for a solution of a given CSP consists of techniques for systematic exploration of the space of all solutions. The basic brute force search algorithm, *generate and test* (*trial and error*), is based on the idea of testing every possible combination of values to obtain a solution of a CSP. Variable and value ordering heuristics are common intelligent search techniques used for solving many kinds of problems such as *configuration*, *job shop scheduling*, and *integrated circuit design* (Jannach, 2013; Li and Epstein, 2010; Liu et al., 2008; O'Sullivan et al., 2004; Pearl, 1984; Sadeh and Fox, 1996). To avoid poor performance of search, the algorithm commonly used for solving CSPs is *backtracking* (or *depth first search with chronological backtracking*) which is a general purpose search strategy (Castro, 1996). Thanks to the time complexity analysis of the backtracking algorithm, it is well known that search efficiency could be improved if the possible values that the variables can take

are reduced as much as possible (Smith, 1996). The variable and value orders can be determined either in advance (a static ordering) or dynamically, using information available at the time that the choice is made. These choices can have a dramatic effect on the time taken to find a solution to a CSP.

The design of problem solving systems has received much attention within AI. Such learning systems have appeared in many problem solving contexts and several have been quite successful (Smith, 1983). In unsupervised learning (Alpaydin, 2016), "the outcome or output for the given inputs is unknown", so the model is run on input data. The inputs given are grouped together to derive insight about a particular data-set. The main algorithms used include clustering and learning algorithms. In this context, to learn heuristics, we proposed utilizing two different supervised learning techniques: *genetic algorithms* (Davis, 1991) and *matrix factorization* (Koren et al., 2009).

Most IoT applications require real-time configuration support during which configurations must be calculated in a short time frame (in milliseconds or seconds) for awaiting users or software. These configuration systems are generally based on interactive settings where user requirements are taken as constraints into consideration as constraints. Therefore, there is a huge availability of user interaction data that can be used during supervised learning. This triggers the research questions below:

(Q2.1) How to learn heuristics using genetic algorithms in order to improve runtime performance of configuration systems?

(Q2.2) How to learn heuristics using matrix factorization techniques in order to improve runtime performance of configuration systems?

3. **How to improve prediction quality of configuration systems for IoT applications?**
A configuration is *accurate* if it is accepted by the configuration system user. Configuration systems use constraint solving algorithms and heuristics to improve the overall runtime performance. However, these methods do not take the performance criterion *quality* into account. There exists some approaches which are based on collaborative filtering that aiming to improve the accuracy of solutions/diagnoses but lack improvement to runtime performance. As proposed in the previous research question, regarding the learning of quality-improving heuristics, we can utilize the following supervised learning techniques: *genetic algorithms* (Davis, 1991) and *matrix factorization* (Koren et al., 2009).

   In this thesis, we aim to improve prediction quality of configuration/recommendation systems because most of IoT applications are based on interactive scenarios in which end-users are assisted with recommended configurations. Since state-of-the-art techniques lack of improvements to *runtime performance* and *prediction quality*, we introduce approaches to improve performance of configuration systems in terms of both *runtime performance* as well as *prediction quality*. For that purpose, we introduce two additional research questions:

   (Q3.1) How to learn heuristics using genetic algorithms in order to improve prediction quality of configuration systems?

   (Q3.2) How to learn heuristics using matrix factorization techniques in order to improve prediction quality of configuration systems?

1.3. Contributions

An overview of the research questions and contributions of this thesis are provided in Table 1.1.

Table 1.1.: Overview of the research questions and the corresponding contributions.

Research Questions	**Contributions**
(Q1) How to adapt state-of-the-art configuration and recommendation systems to IoT applications in order to improve experiences of IoT environment stakeholders?	In this work, we first present how to apply conventional configuration and recommendation technologies to IoT applications. Furthermore, we introduce novel recommendation and configuration approaches in IoT. To demonstrate the applicability of these novel approaches, we used real-world data sets based evaluations thanks to AGILE IoT pilot applications. The results of our real-world data sets based evaluations indicate that our novel configuration and recommendation approaches in IoT domain decrease costs and time to use the systems (Felfernig et al., 2016, 2018b; Erdeniz et al., 2018b) (also see Section 3.1, Section 4.1 and Section 4.2).
(Q2.1) How to learn heuristics using genetic algorithms in order to improve runtime performance of configuration systems?	We conducted experimental evaluations based where our proposed novel heuristics are used during constraint solving. These heuristics are learned based on historical data sets. In order to learn heuristics in an offline phase, we first clustered the data sets according to their similarities, then employed a genetic algorithm on each. After that, in the online phase we utilized these heuristics for solving constraint satisfaction problems. The results of our studies indicate that this approach improves the runtime performance of configuration systems (Erdeniz et al., 2017; Erdeniz and Felfernig, 2018a,b) (also see Section 3.2 and Section 3.3).
(Q2.2) How to learn heuristics using matrix factorization techniques in order to improve runtime performance of configuration systems?	We proposed another heuristics learning method based on matrix factorization techniques. These heuristics are also learned based on historical data sets. In order to learn heuristics in an offline phase, first we generated a sparse matrix using the data sets, then employed a matrix factorization. After that, in the online phase we utilized these heuristics for solving constraint satisfaction problems. The results of our studies indicate that this approach also improves the runtime performance of configuration systems (Erdeniz et al., 2019a,b) (also see Section 3.4 and Section 3.5).
(Q3.1) How to learn heuristics using genetic algorithms in order to improve prediction quality of configuration systems?	To improve prediction quality in configuration systems using the same clustering and learning techniques with Q2.1, we used historical data sets which include real user interactions with a decided configuration (e.g. a purchase of a configurable product). The experimental evaluations indicated that this approach improves the prediction quality of configurations (Erdeniz et al., 2017; Erdeniz and Felfernig, 2018b,a) (also see Section 3.2 and Section 3.3).
(Q3.2) How to learn heuristics using matrix factorization techniques in order to improve prediction quality of configuration systems?	Similarly, we used the matrix factorization techniques (as in Q2.2) with data sets which include real user interactions with a decided configuration (e.g. a purchase of a configurable product). The experimental evaluations indicated that this approach improves the prediction quality of configurations (Erdeniz et al., 2019a,b) (also see Section 3.4 and Section 3.5).

1.4. Thesis Outline

This thesis consists of six chapters, which are organized as follows:

- **Chapter 1** introduces the motivation and research objectives of our work. Our research questions regarding recommendation and configuration technologies in IoT are discussed. Finally, an overview of the structure of this thesis concludes the chapter.
- **Chapter 2** gives an introduction to the research field of recommender systems, configuration systems and IoT.
- In **Chapter 3** presents the results of our research aiming to improve configuration approaches in IoT.
- In **Chapter 4** the results of our research presented aiming to improve recommendation approaches in IoT.
- In **Chapter 4** we present our Eclipse-licensed CSP Heuristix Library which is publicly available on the corresponding GitHub repository [1] including the java implementation of our proposed heuristics.
- **Chapter 6** concludes this thesis. We reflect on our research questions and contributions, and offer an outlook on future research issues.

[1] https://github.com/CSPHeuristix

Chapter 2

Preliminaries

This chapter introduces the background of three main research areas of this thesis: *configuration systems*, *recommender systems*, and *the internet of things*. In Section 2.1, configuration systems are explained with fundamental definitions. Section 2.2 gives an overview of the main technologies in recommender systems. In Section 2.3, an introduction to IoT environment is provided. Finally, Section 2.4 introduces the IoT project *AGILE IoT* in which the research outputs of this thesis are employed.

2.1. Configuration Systems

The mass production of identical products is now a business model of the past as buyer markets continue to dominate. This situation introduces new challenges with production and sales processes because nowadays companies are now forced to provide products that meet the individual needs of their customers. As a consequence, a mass customization paradigm has been established. This paradigm is based on the idea of the customer-individual production of highly variant products under near mass production pricing conditions. This means that the major goal was not only to perform a paradigm shift toward customer preferences but to also achieve this goal under mass-production-level time and pricing conditions. The era of mass customization rang a bell for technological developments urgently needing to effectively implement this paradigm. The resulting technologies have evolved into a leading field to support mass customization business scenarios. Configuration technologies significantly reduce development and maintenance costs of key functionalities needed for the implementation of the mass customization paradigm. These functionalities encompass the efficient development and maintenance of constraint sets (configuration knowledge bases), the development and maintenance of configurator user interfaces, and the integration of configurators into existing software environments (enterprise resource planning systems and systems supporting product data management) (Felfernig et al., 2014b).

One definition of configuration (as an activity) has been given by Sabin and Weigel (Sabin and Weigel, 1998), who define configuration as “a special case of design activity where the artifact being configured is assembled from instances of a fixed set of well defined component types which can be composed conforming to a set of constraints". In Sabin and Weigel’s definition, configuration is typically knowledge-based (knowledge-based configuration) since it relies on product domain and problem-solving knowledge.

One example of a configuration problem is *the customization of cars* scenario in which many hardware and software modules exist and all of them must work together without any conflicts. As an example, we introduce a small problem of configuring an automobile. A car in this example is modelled using five

variables; in this case, package, frame, engine, transmission and type. The domain of a variable is the set of the possible values for this variable and specific customer requirements are simply values for the variables type and package. The five variables and their possible values are described in more detail in Table 2.1 (Faltings and Weigel, 1994).

Table 2.1.: A Car Configuration Example

1. **Package** with values: deluxe, luxury, and standard:
   The package variable describes the equipment of the car. The deluxe package will include both an airconditioner and a central locking system. The luxury package includes only the central locking system. None of the equipment items is included within the standard package.
2. **Type** with values: sportscar and familycar:
   The marketing division decided to sell only in two car types. No pickups or other kind of cars can be sold except of sportscars and familycars.
3. **Engine** with values: A, B, and C:
   Three different engine types are available. Engine A is a sport version, might therefore be used for the sportscar. Engine B is a smooth-running version for limousines.
4. **Transmission** with values: manual, automatic, and half-automatic:
   Automatic and half-automatic transmissions are normally found in limousines. Manual transmissions are used for off-road cars and sportscars.
5. **Frame** with values: convertible, hatchback, and sedan:
   Cars can be produced with three different frametypes. Hatchback and sedan frametypes are well suited for families. Convertible frames on the other hand are generally used for sportcars.

Configuration Knowledge Representation

There is quite a long history of research dedicated to the development of configuration knowledge representation languages. We can group these representations into two groups: *rule-based* and *model-based*.

Rule-based configuration systems. *Rule-based* systems operate on a working memory that stores assertions made by rules. Rules are represented in an if-then style. If a rule is activated (given that the if condition is fulfilled) it can modify the contents of the working memory. Past research showed that a major problem with rule-based systems is the intermingling of (product) domain knowledge and problem solving knowledge. A consequence of this mingling is that, if product knowledge is changed, rules related to the sequence in which components are integrated in a configuration must be adapted as well, triggering enormous maintenance overheads (Faltings and Weigel, 1994).

When we need to prepare the rule-based knowledge-base of the car configuration example in Table 2.1, all the rules are of the form *"IF <premise> THEN <conclusion>"*. The forward chaining algorithm selects the rules where the premises are true and triggers them to conclude new facts. Knowing for example, that the customer wishes a sportscar the algorithm concludes immediately by Rule 8 (see Table 2.2 (Faltings and Weigel, 1994)), that the car will have a manual transmission.

Table 2.2.: Rule-based Representation of Table 2.1

- **Rule1:** IF Package = Deluxe and Frame = convertible THEN Engine A
- **Rule2:** IF Package = Deluxe and Frame = hatchback THEN Engine B
- **Rule3:** IF Package = Standard and Frame = convertible THEN Engine A
- **Rule4:** IF Engine = A THEN Transmission = manual
- **Rule5:** IF Engine = B THEN Transmission = automatic
- **Rule6:** IF Type = Sportscar THEN Frame = convertible
- **Rule7:** IF Type = Familycar THEN Frame = sedan
- **Rule8:** IF Type = Sportscar THEN Transmission = manual

Model-based configuration systems. The shortcomings of rule-based approaches were the motivation for the development of *model-based* knowledge representations that support a clear separation of domain and problem solving knowledge. Mittal and Frayman (Frey et al., 2015) introduced a model-based definition of a configuration task that is characterized by: (1) a description of generic components in terms of their properties and relationships and (2) user requirements and preferences regarding functional properties of the solution (configuration). This characterization is a major foundation for configuration research and industry. It also includes the idea of separating functional requirements (customer requirements) from the technical properties of a configurable product. A major representative of model-based approaches are *constraint-based* systems. A finite, discrete constraint-based knowledge representation consists of:

- variables
- domains
- constraints, expressing relations and possible variable-value combinations

In Table 2.3 (Faltings and Weigel, 1994), the configuration knowledge is described by constraints, whereas rules were used in the previous example.

Table 2.3.: Constraint-based Representation of Table 2.1

- **C1:** Package Frame Engine allowed value-combinations:
  (Deluxe convertible A)(Deluxe hatchback B)(Deluxe convertible C)...
- **C2:** Engine Transmission allowed value-combinations:
  (A manual)(B automatic)(C half-automatic)(A half-automatic)...
- **C3:** Type Frame allowed value-combinations:
  (Sportscar convertible)(Familycar sedan)...
- **C4:** Type Transmission allowed value-combinations:
  (Sportscar manual)(Familycar half-automatic)...

In systems built using deductive rules, particularly expert systems, the context-dependence results in severe problems of maintaining knowledge in a dynamic world. Even minute changes of technology or marketing policy require revision of the entire rule set, and this can be very costly. In the rule-based approach, adding a new car-type forces us to create a new relation between engine, type and transmission. Expressing knowledge without looking at the context is one of the major advantages of constraint based reasoning. No relation between engine, type and transmission is necessary in this context (Faltings and Weigel, 1994). Therefore, in this thesis, we focus on improvements in a major representation of constraint-based configuration systems called constraint satisfaction problems.

Constraint Satisfaction Problems

Many configuration problems can be defined as constraint satisfaction problems (*CSPs*) (Tsang, 1993). These problems include a set of variables, domains (a set of possible values of each variable), and a set of constraints (see Definition 2.1.1 (Felfernig et al., 2014c)). A solution of a CSP is a complete set of variable assignments where all constraints are satisfied.

For example, the map-coloring problem can be cast as CSP. In this problem, we need to color (from a set of colors) each region of the map such that no two adjacent regions have the same color. Figure 2.1 (Kumar, 1992) shows an example map-coloring problem and its equivalent CSP. The map has four regions that are to be colored red, blue, or green. The equivalent CSP has a variable for each of the four regions of the map. The domain of each variable is the given set of colors. For each pair of regions that are adjacent on the map, there is a binary constraint between the corresponding variables that disallows identical assignments to these two variables (Kumar, 1992).

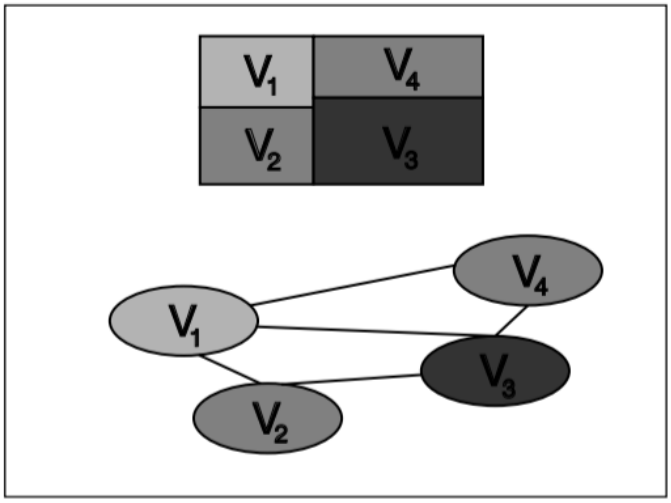


Figure 2.1.: An Example Map-Coloring Problem and Its Equivalent Constraint-Satisfaction Problem.

Definition 2.1.1. (Configuration Task and Configuration). 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. $D = \{dom(v_1), dom(v_2), \ldots, dom(v_n)\}$ represents a set of variable domains $dom(v_n)$ where $dom(v_n)$ represents the domain of variable v_n. $C = (C_{KB} \cup REQ)$ where $C_{KB}=\{c_1, c_2, .., c_q\}$ is a set of domain specific constraints (the configuration knowledge base) that restricts the possible combinations of values assigned to the variables in V. $REQ=\{c_{q+1}, c_{q+2}, .., c_t\}$ is a set of user requirements, which is also represented as constraints. A configuration (S) for a configuration task is a set of assignments $S= \{v_1 = a_1, v_2 = a_2, ..., v_n = a_n\}$ where $a_i \in \text{dom}(v_i)$ which is consistent with C.

A CSP can be solved using the *generate-and-test* paradigm. In this paradigm, each possible combination of the variables is systematically generated and then tested to see if it satisfies all the constraints. The first combination that satisfies all the constraints is the solution. The number of combinations considered by this method is the size of the Cartesian product of all the variable domains. A more efficient method uses the *backtracking* paradigm. In this method, variables are instantiated sequentially. As soon as all the variables relevant to a constraint are instantiated, the validity of the constraint is checked. If a partial instantiation violates any of the constraints, backtracking is performed to the most recently instantiated variable that still has alternatives available. Clearly, whenever a partial instantiation violates a constraint, backtracking is able to eliminate a subspace from the Cartesian product of all variable domains. The backtracking method essentially performs a depth-first search of the space of potential CSP solutions (Kumar, 1992).

State-of-the-art CSP solvers (ILOG Solver (Solver, 2003), Choco Solver (Prud'homme et al., 2016)) perform backtrack searches and enforce *arc consistency* (see Definition 2.1.2) at every visited node. Backtracking occurs each time AC yields an empty domain. It is well recognized that such features are fundamental to efficiency of this process. Consider the constraint graph of another map-coloring problem given in Figure 2.2 (Kumar, 1992). In this constraint graph, arc (V3,V2) is consistent because green is the only value in the domain of V3, and for V3 = green, at least one assign- ment for V2 exists that satisfies the constraint between V2 and V3. However, arc (V2, V3) is not consistent because for V2 = green, there is no value in the domain of V3 that is permitted by the constraint between V2 and V3 (see Table 2.4).

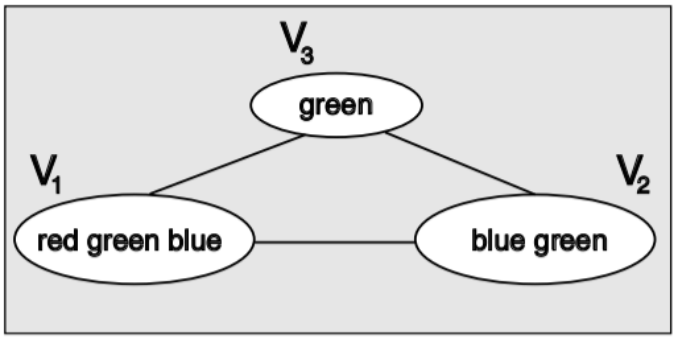


Figure 2.2.: A constraint graph for a map-coloring problem.

Table 2.4.: The CSP Representation of Figure 2.2

- **V:** $\{V_1,V_2,V_3\}$
- **D:** $\{dom(V_1),dom(V_2),dom(V_3)\}$
  $dom(V_1)$: {red,green,blue}
  $dom(V_2)$: {blue,green}
  $dom(V_3)$: {green}
- **C:** { c_1: $V1 \neq V2$, c_2: $V2 \neq V3$, c_3: $V1 \neq V3$ }

Definition 2.1.2. (Arc consistency). (i, a) is arc consistent with respect to constraint C_{ij} if it is node consistent and there is a value $b \in D_j$ such that (a, b) $\in C_{ij}$. Such a value b is called a support of a. Variable i is arc consistent if all its values are arc consistent with respect to every binary constraint involving i. A CSP is arc consistent (AC) if every variable is arc consistent.

The constraint graph in Figure 2.3-a (Kumar, 1992) is arc consistent, but none of the possible instantiations of the variables are solutions to CSP. In general, even after achieving arc consistency, a network can have (1) no solutions (Figure 2.3-a), (2) more than one solution (Figure 2.3-b (Kumar, 1992)), or (3) exactly one solution (Figure 2.3-c (Kumar, 1992)). In each case, search might be needed to find the solution(s) or discover that there is no solution. Nevertheless, by making the constraint graph arc consistent, it is often possible to reduce the search done by the backtracking procedure.

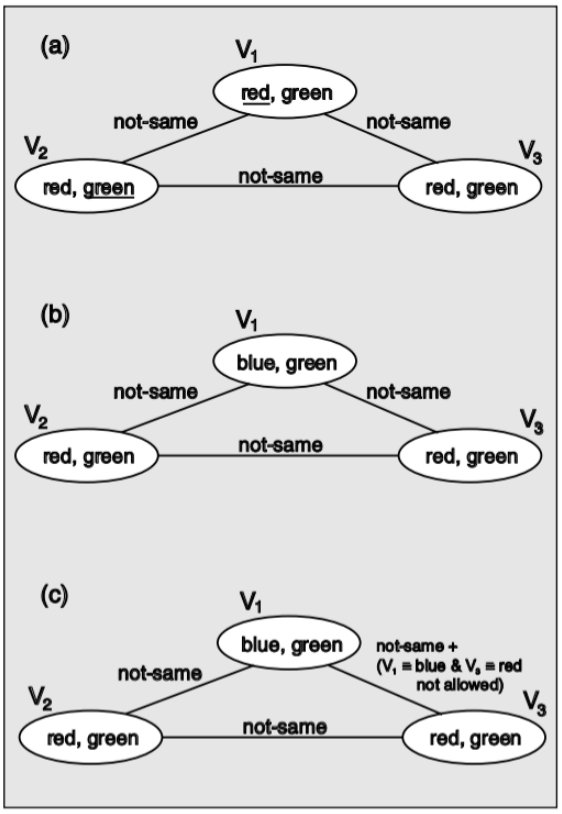


Figure 2.3.: Examples of Arc-Consistent Constraint Graphs. (a) The graph has no solutions. (b) The graph has two solutions ((blue,red,green), (blue,green,red)). (c) The graph has exactly one solution (blue,red,green).

Constraint satisfaction problems on finite domains are typically solved using a form of search. The most used search techniques are variants of *search-based*, *consistency-based*, and their hybrid methods. However, solving a constraint satisfaction problem on a finite domain is an NP-complete problem with respect to the domain size and the complexity of the given constraints. Consequently, the *direct* use of one of these most common search techniques can not help to increase the runtime performance. Therefore, more suitable heuristics are needed in order to take full advantage of power of search algorithms (Kumar, 1992).

Heuristics

Heuristics are strategies using readily accessible information to control problem-solving processes. The constraint satisfaction community has developed a number of different heuristics. Several search heuristics exist that are applicable in the context of constraint solving.

Searching for a solution of a given CSP consists of techniques for the systematic exploration of the space of all solutions. The basic brute force algorithm *generate and test* (also referred to as *trial and error*) search, is based solely on the idea of testing every possible combination of values in order to obtain a solution of a CSP. If backtracking is used to solve CSP, then another issue is the order in which variables are considered for instantiating. Experiments and analysis by several researchers show that the ordering in which variables are chosen for instantiating can have substantial impact on the complexity of backtrack search. The most popular heuristics for constraint solving are variable and value ordering heuristics (Narodytska and Walsh,

2007; Jannach, 2013; Sadeh and Fox, 1996).

Variable and Value Ordering Heuristics. A general paradigm for solving CSPs relies on the use of depth-first backtrack search. Variables are successively instantiated. Each time a new variable is instantiated, a new search state is created that corresponds to a new, more complete, partial solution. This process goes on until either a complete solution is obtained (*success* state) or until a *fail* state is reached. A *fail* state is one whose partial solution cannot be completed without violating one or several problem constraints. When in a *fail* state, the procedure has to undo one or several assignments and try alternative ones, if there are any left. Otherwise the problem is infeasible. This process of undoing earlier assignments is known as *backtracking* (Sadeh and Fox, 1996).

In practice, the average complexity of the procedure can be improved by employing search with the application of consistency enforcing mechanisms and variable and value ordering heuristics:

- Consistency checking techniques: These techniques prune the search space by eliminating local inconsistencies that cannot participate in a global solution. This is done by inferring new constraints and adding them to the current problem formulation. If, during this process, the domain of a variable becomes empty, a *fail* situation has been identified (e.g. forward checking).

- Variable and value ordering heuristics: These heuristics are concerned with the order in which variables are instantiated and values assigned to each variable. These heuristics can have a great impact on search efficiency.

A powerful way of reducing the average complexity of backtrack search is to judiciously select the order in which variables are instantiated. The assumption is that, by instantiating difficult variables first, backtrack search will generally avoid building partial solutions that it will not be able to complete later on. This reduces the chances (i.e. the frequency) of backtracking. Instantiating difficult variables first can also help reduce the amount of backtracking when the system is in a *fail* state that is not immediately detected by its consistency checking mechanism. Indeed, by instantiating difficult variables, the system moves to more constrained *fail* states that are easier to detect. This reduces the time the system wastes attempting to complete partial solutions that cannot be completed (Sadeh and Fox, 1996). For example, when we start instantiating variables of the configuration problem in Figure 2.3 according to the variable ordering in Figure 2.5, we find the solution after maximum three failing states which are {V_3= green, V_2=green}, {V_3= green, V_2=blue,V_1=green}, {V_3= green, V_2=blue,V_1=blue}).

Table 2.5.: An example variable ordering for solving the CSP in Table 2.4

{V_3,V_2,V_1}

Another powerful way of reducing the average complexity of backtrack search relies on judiciously selecting the order in which possible values are tried for each variable. A good value ordering heuristic is one that assigns least constraining values. A least constraining value is one that is expected to participate in many solutions to the overall problem or, better, one expected to participate in a large number of solutions compatible with the current search state. By first trying least constraining values, the system will generally maximize the number of values left to variables that still need to be instantiated, and hence it will avoid building partial solutions that cannot be completed (Sadeh and Fox, 1996). For example, when we start instantiating variables of the configuration problem in Figure 2.3 according to the variable ordering in Figure 2.5 and the value orderings in Figure 2.5, in the first trial we directly find the solution {V_3= green, V_2=blue, V_1=red}.

Table 2.6.: Example value orderings for solving the CSP in Table 2.4

V_1: {red,green,blue}
V_2: {blue,green}
V_3: {green}

Consistency Based Diagnosis

In consistency-based configuration systems, the system can sometimes not find a configuration for a given task because it is not possible to satisfy all constraints. Such a "no solution" dilemma is caused by at least one conflict between a) the constraints in the knowledge base C_{KB} and the user requirements REQ, or b) within the set C_{KB} itself. In such situations, consistency-based diagnosis algorithms can be used to diagnose the conflicting constraints.

To demonstrate consistency-based diagnoses, we use the following car configuration task with user requirements (Felfernig et al., 2014c). The variable type represents the type of the car, pdc is the park distance control feature, fuel represents the average fuel consumption per 100 kilometers, a skibag allows convenient ski stowage inside the car, and 4-wheel represents the gear type (4-wheel supported or not supported). These variables represent the possible combinations of customer requirements. The configuration knowledge base C_{KB} = c_1, c_2, c_3, c_4, c_5 defines additional restrictions on the set of possible customer requirements REQ = c_6, c_7, c_8, c_9, c_{10}.

Table 2.7.: An inconsistent car configuration task with user requirements

- Variables (V) = type, fuel, skibag, 4-wheel, pdc
- Domains (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}}
- Knowledge Base (C_{KB}) = {
  c_1 : 4-wheel = yes $\rightarrow$ type = xdrive,
  c_2 : skibag = yes $\rightarrow$ type = city,
  c_3 : fuel = 4l $\rightarrow$ type = city,
  c_4 : fuel = 6l $\rightarrow$ type = xdrive,
  c_5 : type = city $\rightarrow$ fuel = 10l}
- Customer Requirements (REQ) = {
  c_6 : 4-wheel = no,
  c_7 : fuel = 4l,
  c_8 : type = xdrive,
  c_9 : skibag = no,
  c_{10} : pdc = yes }

The example in Table 2.7 (Felfernig et al., 2014c) is inconsistent. This means it has a conflict, so there is no solution this problem. If we have an inconsistency in our knowledge base, we can say that $C_{\mathrm{KB}} \cup REQ$ is always a conflict set. Therefore, in our example the most trivial conflict set is CS1 = {c_1,c_2,c_3,c_4,c_5,c_6,c_7,c_8,c_9,c_{10}}. Definition 2.1.3 introduces a formal representation of a conflict.

Definition 2.1.3. (Conflict). A conflict is a set of constraints $CS \subseteq C_{KB} \cup REQ$ which can not be fulfilled by the CSP, s.t. *CS* is inconsistent.

In the context of an inconsistent configuration, we introduce the term "minimal conflict" (see Definition 2.1.4). In our example, we are able to identify the following minimal conflict set: CS2 = {c_3, c_7, c_8}.

Definition 2.1.4. (Minimal Conflict). A minimal conflict *CS* is a conflict (see Definition 2.1.3) and the set *CS* only contains constraints which are responsible for the conflict, s.t. $c \in CS$, CS - {c} is inconsistent.

Assuming that C_{KB} is consistent, in inconsistency cases we can help users to resolve the conflicts with a diagnosis Δ, a set of constraints. The removal of Δ from *REQ* leads to a consistent knowledge base (see Definition 2.1.5). In our example, a diagnosis can be Δ_1 = {c_7, c_8}. That means, when we remove c_7 and c_8 from the user requirements, we can find a solution for this car configuration example.

Definition 2.1.5. (REQ Diagnosis Task and Diagnosis). A user requirements diagnosis task (*RDT*) is defined as a tuple (C_{KB}, *REQ*) where *REQ* is the set of given user requirements and C_{KB} represents the constraints part of the configuration knowledge base. A diagnosis for a REQ diagnosis task (C_{KB}, *REQ*) is a set $\Delta \subseteq REQ$, s.t. $C_{KB} \cup (REQ - \Delta)$ is consistent. Δ = {c_1,c_2,..,c_n} is minimal if there does not exist a diagnosis $\Delta' \subset \Delta$, s.t. $C_{KB} \cup (REQ - \Delta')$ is consistent.

Assuming that C_{KB} is consistent, we can also say that the knowledge base always will be consistent if we remove *REQ*. However, the determination of minimal cardinality diagnoses (i.e., minimal diagnoses with the lowest possible number of included constraints) is important since users do not want to remove many requirements during diagnosis. Definition 2.1.6 introduces the term *minimal diagnosis* which helps to reduce the number of constraints within a diagnosis. In our example, minimal diagnoses are Δ_2 = {c_7} and Δ_3 = {c_8}. That means, when we remove only c_7 or only c_8 from the user requirements, we can find a solution for this car configuration example.

Definition 2.1.6. (Minimal Diagnosis). A minimal diagnosis Δ is a diagnosis (see Definition 2.1.5) and there doesn't exist a subset $\Delta' \subset \Delta$ which has the same property of being a diagnosis.

2.2. Recommender Systems

Recommendation technologies (Jannach et al., 2010; Felfernig et al., 2018a) can support the efficient identification of relevant artifacts. Recommender systems suggest items (alternatives, solutions) that are of potential interest for a user. Examples of related questions are: *which book should be purchased?*, *which test method should be applied*?, *which method calls are useful in a certain development context?* or *which apps (applications) are of potential interest for the current user?* A recommender system can be defined as *any system that guides a user in a personalized way to interesting or useful objects in a large space of possible options or that produces such objects as output* (Felfernig and Burke, 2008).

Recommender technologies are mainly based on two fundamental approaches; collaborative filtering and content-based filtering. *Collaborative Filtering* (Konstan et al., 1997) refers to the process of using the opinion of users with similar preferences whereas *Content-based Filtering* (Pazzani and Billsus, 1997) is based on comparing the content of already consumed items with new items that can potentially be recommended to the user. Other basic recommendation approaches are *knowledge-based recommendation*, *group recommender systems*, and *hybrid recommendation*. *Knowledge-based recommender systems* (Felfernig et al., 2015a) are based on explicit knowledge, rules or constraints about the item assortment, user preferences, and recommendation criteria (i.e., which item should be recommended in which context). *Group recommender systems* (Felfernig et al., 2018a; Masthoff, 2011) calculate recommendations in which the whole group must be satisfied with the given recommendation. *Hybrid recommendation* (Burke, 2002) combines basic recommendation approaches to compensate the weaknesses of individual ones.

As an example, we introduce a movie recommendation dataset. Table 2.8 contains the userId, movieId, ratings, and timestamp attributes. Each row in the dataset corresponds to one rating. The userId column contains the ID of the user who left the rating. The movieId column contains the Id of the movie, the rating column contains the rating left by the user. Ratings can have values between 1 and 5. And finally, the timestamp refers to the time at which the user left the rating. In Table 2.11, the movies dataset contains movieId, the title of the movie, and its genre.

Table 2.8.: User ratings for movies

userId	movieId	rating	timestamp
1	1	2.5	1260759144
1	3	3.0	1260759179
1	4	4.0	1260759185
2	1	3.0	1260759182
2	4	2.0	1260759185
2	5	5.0	1260759189
3	1	4.0	1260759205

Table 2.9.: Features of movies

movieId	title	genres
1	Toy Story (1995)	Adventure, Animation, Children, Comedy, Fantasy
2	Jumanji (1995)	Adventure, Drama, Children, Fantasy
3	Grumpier Old Men (1995)	Comedy, Romance
4	Waiting to Exhale (1995)	Comedy, Drama, Romance
5	Father of the Bride Part II (1995)	Comedy

Collaborative Filtering

Collaborative filtering (Konstan et al., 1997) is based on the idea of word of mouth promotion, i.e., the opinion of users with similar preferences plays a major role in a decision. These users are also denoted as *nearest neighbors*, i.e., users with similar preferences compared to the current user. There are various similarity metrics and approaches in collaborative filtering (Ricci et al., 2015).

For example, the first step of a collaborative filtering recommender can be to identify the *k-nearest neighbors* (*k* represents the number of users with similar ratings compared to the current user) and to extrapolate from the ratings of these users the preferences of the current user using Formula 2.1.

$$similarity(item_a, item_b) = \frac{1}{1 + \Sigma_{i=1, i \in users}^{n} |eval(item_a) - eval(item_b)|} \tag{2.1}$$

In our movie example, when a movie recommendation for the user with userId=3 is needed, it can be calculated using Formula 2.1 on Tables 2.8 and 2.11, for the user with userId=3, the movie with movieId=5 "Father of the Bride Part II (1995)" is recommended.

Content-based Filtering

Content-based filtering (Pazzani and Billsus, 1997) is based on the assumption of monotonic personal interests. For example, users interested in cars are typically not changing their interest profile from one

day to another but can also be expected to be interested in the topic in the (near) future. There are various techniques in content-based filtering (Ricci et al., 2015)

For example, the basic approach of content-based filtering is to compare the content of already consumed items with new items that can potentially be recommended to the user. The goal is to find items that are similar to those already consumed (and positively rated) by the user using Formula 2.2.

$$similarity(user, item) = \frac{features(user) \cap features(item)}{features(user) \cup features(item)} \tag{2.2}$$

In our movie example, when a movie recommendation for the user with userId=3 is needed, it can be calculated using Formula 2.2 on Tables 2.8 and 2.11. In this case, for the user with userId=3, the movie with movieId=2 "Jumanji (1995)" is recommended.

Knowledge-based Recommendation

Knowledge-based recommendation (Burke, 2000; Felfernig and Burke, 2008) does not rely on item ratings and textual item descriptions but on deep knowledge about the offered items represented in terms of constraints, rules, or similarity metrics. Such deep knowledge (semantic knowledge) describes an item in more detail and thus allows for a different recommendation approach. The current user articulates his/her requirements in terms of item property specifications and these requirements are internally as well represented as rules (constraints). Constraints are interpreted and the resulting items are presented to the user. Such items can also be interpreted as cases (consistent with the constraints) which are recommended to the current user as solutions for his/her current requirements (problem setting).

There are two well-known approaches to knowledge-based recommendation: case-based recommendation and constraint-based recommendation. Case-based recommendation treats recommendation primarily as a similarity-assessment problem. The mechanism for finding a product most similar to what the user has in mind involves domain-specific knowledge and considerations. Constraint-based recommendation takes into account explicitly defined constraints satisfies the requirements of a customer (the calculated similarity value exceeds a certain threshold for all relevant products or the set of constraints is inconsistent with the given set of customer requirements) both knowledge-based approaches exploit mechanisms supporting the determination of minimal set of changes to the given set of customer requirements such that a solution can be found. Constraint-based recommenders exploit explicit user requirements as well as deep knowledge about the underlying product domain for the computation of recommendations. A constraint-based recommendation task can be defined as a constraint satisfaction problem (see Definition 2.2.1) and can be solved using so-called constraint satisfaction algorithms and heuristics.

Definition 2.2.1. (Constraint-based Recommendation Task). In general, a recommendation task (RT) can be defined as a constraint satisfaction problem (V, C) where V is a set of variables, C is a set of system constraints which may also include a set of unary constraints representing concrete customer requirements REQ. An assignment of the variables in V is denoted as consistent recommendation result (REC) for a recommendation task (V, C) if REC does not violate any of the constraints in C.

Table 2.10.: A constraint-based movie recommendation task

- Variables (V) = {movieId,genre}
- Domains (D) = {
  dom(movieId) = {1,2,3,4,5},
  dom(genre) = {Adventure, Animation, Children, Comedy, Fantasy, Romance, Drama},
- Knowledge Base (C_{KB}) = {
  c_1 : movieId = 1 $\rightarrow$ genre = Adventure $\wedge$ Animation $\wedge$ Children $\wedge$ Comedy $\wedge$ Fantasy
  c_2 : movieId = 2 $\rightarrow$ genre = Adventure $\wedge$ Drama $\wedge$ Children $\wedge$ Fantasy
  c_3 : movieId = 3 $\rightarrow$ genre = Comedy $\wedge$ Romance
  c_4 : movieId = 4 $\rightarrow$ genre = Comedy $\wedge$ Drama $\wedge$ Romance
  c_5 : movieId = 5 $\rightarrow$ genre = Comedy }
- Customer Requirements (REQ) = {
  c_6 : genre = Comedy $\wedge$ Drama }

In our movie example, the user with userId=3 specifies a preference such as "genres should include Comedy and Drama". This constraint-based recommendation task is defined as in Table 2.10. Based on the table, the movie with movieId=4 "Waiting to Exhale (1995)" is recommended using a constraint-based recommender based on Definition 2.2.1.

Utility-based Recommendation

Utility-based recommendation (Felfernig and Burke, 2008) is based on the idea that – given a set of items, item rank is determined on the basis of multi-attribute utility theory (MAUT) (Winterfeldt and Edwards, 1986). In this case, each item is evaluated with regard to a set of interest dimensions as shown in Formula 2.3. In the context of optimizing the used data transfer protocols, example dimensions could be efficiency (measured in terms of transfer rates) or economy (measured in terms of costs for data connections). Utility-based recommendation is often combined with knowledge-based recommendation since item ranking is needed after constraints (rules) have pre-selected the items of potential relevance for the user. In this context, customer-individual preferences can also be learned by analyzing existing user interaction data (Jannach et al., 2010).

For example, a utility-based recommender for web hosting services has knowledge about the major strengths and weaknesses of the offered products. Those can be specified in terms of their contribution to the interest dimensions reliability, economy, and performance.

$$utility(item, user) = \Sigma_{d \in dim} interest(user, d) \times value(item, d) \tag{2.3}$$

Table 2.11.: A utility knowledge base of web services

web service	reliability	economy	performance
service1	7	7	7
service2	6	7	9

A user for example, has the following interests: reliability=7, economy=7, and performance=8. The utility-based recommendation for the user is calculated based on Formula 2.3 and service2 is recommended.

Hybrid Recommendation

Hybrid recommendation (Burke, 2002) is based on the idea of combining basic recommendation approaches in such a way that each method helps to compensate the weaknesses of the others. For example, when combining content-based filtering with collaborative recommendation, content-based recommendation helps to recommend items which were not rated up-to-now. If a user has already consumed some items, the content description of a new item can be compared with the descriptions of items already purchased by the user. If the new item is similar to some of the already consumed ones, it can be recommended to the user. Hybrid recommendation can also combine recommendation approaches to increase prediction quality. Combining the recommendations of different algorithms, for example, on the basis of a voting mechanism, can help to significantly increase prediction quality (Jannach et al., 2010).

In our movie example, when a movie recommendation for the user with userId=3 is needed, it can be first calculated using Formula 2.2 on Tables 2.8 and 2.11 as the movie with movieId=2 "Jumanji (1995)". Then using Formula 2.1 the movie the most similar to movieId=2 is found as movieId=4 "Waiting to Exhale (1995)". Thus, movieId=4 is recommended by a hybrid recommender.

Group Recommender Systems

Recommender systems have become a fundamental means for providing personalized guidance to users in their searches for interesting or useful products such as movies, songs, restaurants, software requirements, or digital cameras. Although most existing recommender systems support single users, there are many scenarios where items may be used by groups. In these scenarios, the presentation of recommendations to groups is a more natural approach than trying to address individual users. For example, music recommendations in fitness studios must to take into account the preferences of all individuals currently present in the studio. Stakeholders in a software project have to establish agreement regarding the requirements/features that have to be developed within the scope of the next release. Personnel decisions are often taken in groups, i.e., a group has to decide which job applicant will be hired. Groups of friends have to decide about the hotel for the next summer holidays or a skiing resort for the next winter holidays. A public display should be personalized in order to be able to display information to persons currently in the surrounding. Finally, travel groups should receive a personalized museum guidance in such a way that the personal preferences of group members are fulfilled (Felfernig et al., 2018a).

Recommendations in this context are often determined on the basis of group decision heuristics (Masthoff, 2011). For example, *least misery* is a heuristic that recommends items which minimize the misery of all group members as shown in Formula 2.4. In contrast, *most pleasure* tries to maximize the pleasure of individual group members. Also in the context of group recommender systems, hybrid approaches can be developed, i.e., individual group recommendation heuristics can be combined with each other.

$$LM = \max_{(t \in I)}(min(t)) \tag{2.4}$$

In our movie example, if the users with userId=1 and userId=2 wish watch a movie by selecting between movies with movieId=1 and movieId=4, the least misery recommendation for the group is calculated according to Formula 2.4 is movieId=1 "Toy Story (1995)".

2.3. The Internet of Things

As an emerging paradigm, the Internet Of Things (IoT) (Atzori et al., 2010; Greengard, 2015) represents a networked infrastructure connecting different types of devices in any place and anytime. As shown in Figure 2.4, IoT sensors can be connected to an IoT gateway using various connection protocols: for example, 5G, BLE, LORA, and ZigBee. Users of the gateway can connect via WAN/LAN to manage/monitor their data and services. They can also manage/monitor the collected data by linking a cloud application with their IoT gateway.

An IoT gateway is a hardware and software-based solution, which primarily, enables device-to-device and/or device-to-cloud communication. It is a platform to support connections between different data sources (sensors with various communication protocols) and destinations (local or remote data management entities, as well as various actuators). IoT gateways, positioned at the edge of or near the devices, could also play a crucial role in the execution of services. A typical IoT gateway platform is composed of a device manager, a communication/data protocols manager, an application manager, and a data manager (see Figure 2.4). Advanced IoT gateways contain additional functionalities among which a configurator and a recommender engine can be included to assist users in the configuration of the gateway or in recommending useful applications based on given gateway settings and user interaction protocols.

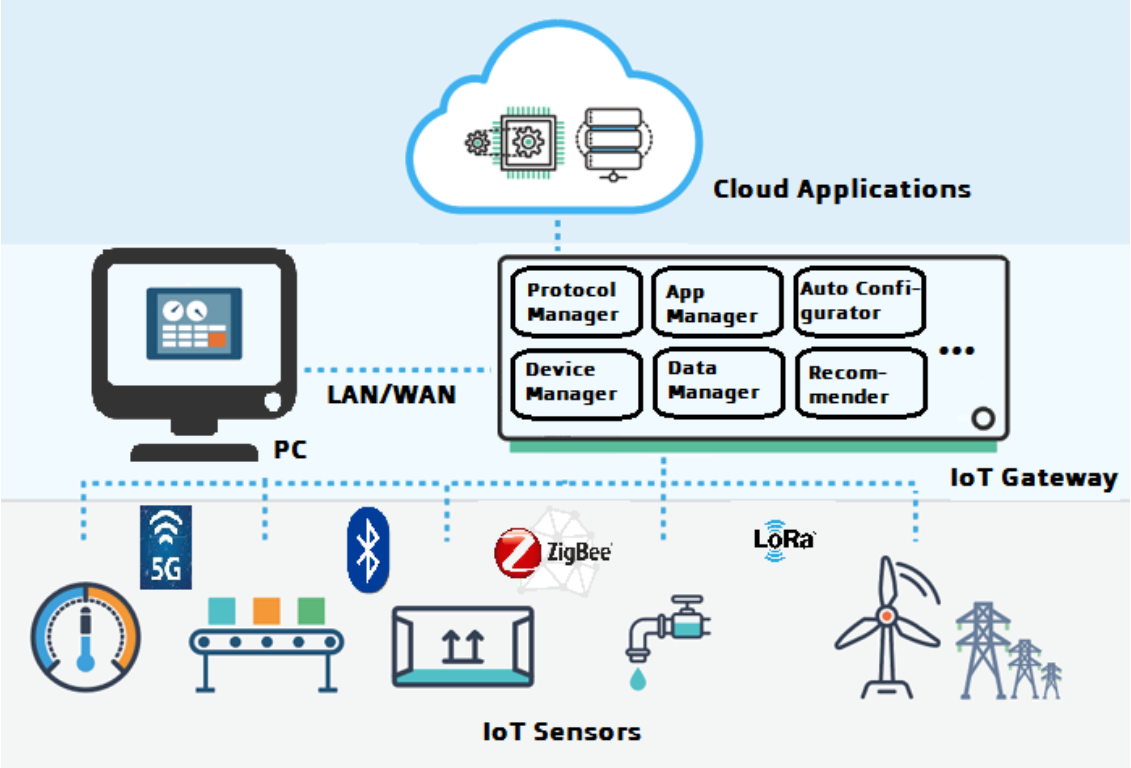


Figure 2.4.: Software and hardware architecture of an IoT environment.

An Example: Monitoring the Supply Chain of Bottled Wine

Maintaining appropriate conditions in the storage and transport of sensitive products is an age-old problem for food supply chains. Many types of goods can be easily damaged by improper variations in temperature or excessive duration of transport. The impact of unsuitable transport and storage for wine is far greater than the mere cost of damaged goods, as it also includes brand damage for companies such as wine producers, wholesalers, retailers and logistics companies. The size of the problem is far-reaching as nearly 5% of bottled wine worldwide is damaged, up to 25% is negatively affected, and the uprise in quality problems due to improper storage settings, and 20% of the quality wine consumed worldwide is counterfeit.

ISTMOS monitoring platform[1] (see Figure 2.5 and Figure 2.6) allows the supply chain stakeholders of bottled wine to monitor the most critical parameters affecting wine quality during its storage and transportation. Delivering traceability information down to individual bottle, the platform includes complex-event processing, rule and alert management and an analytics engine. We employ monitoring stations (fixed for storage and mobile for transportation) and collect and analyze the information on our own IoT platform relying on FI-WARE enablers, particularly PERSEO complex event processing and ORION context broker.

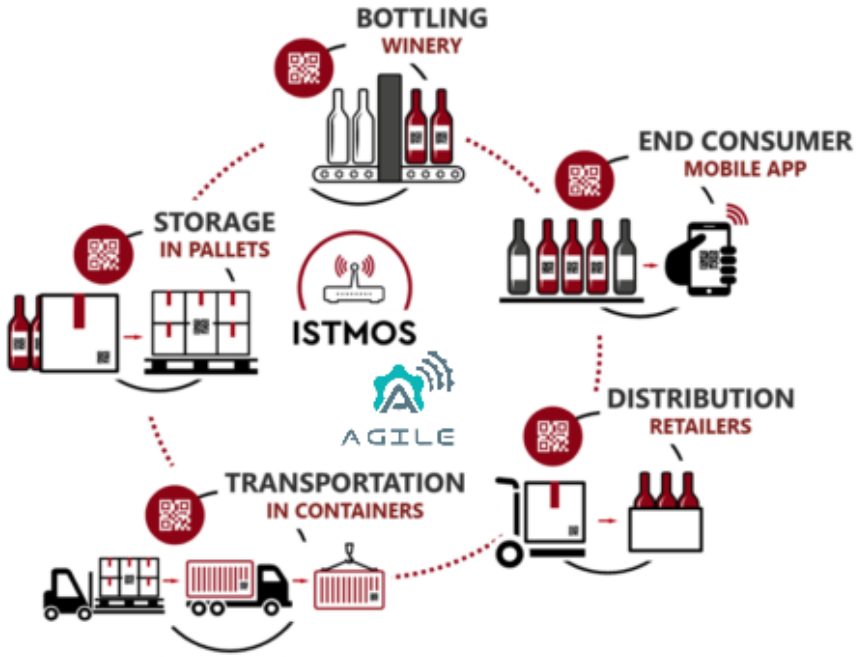


Figure 2.5.: The supply chain of bottled wine.

Wineries and wine retailers already have various types of measurement instruments in place to control temperature and humidity in storage spaces. Soon, an increasing number of stakeholders in the wine supply chain are becoming equipped with a plethora of IoT devices from different hardware vendors to measure wine storage environments. These sensors will operate across a wide range of wireless communication protocols, notably BLE and LoRaWAN, where low-power consumption is important, but also WiFi or NB-IoT.

[1] http://agile-iot.eu/2018/12/08/monitoring-the-supply-chain-of-bottled-wine-with-agile/

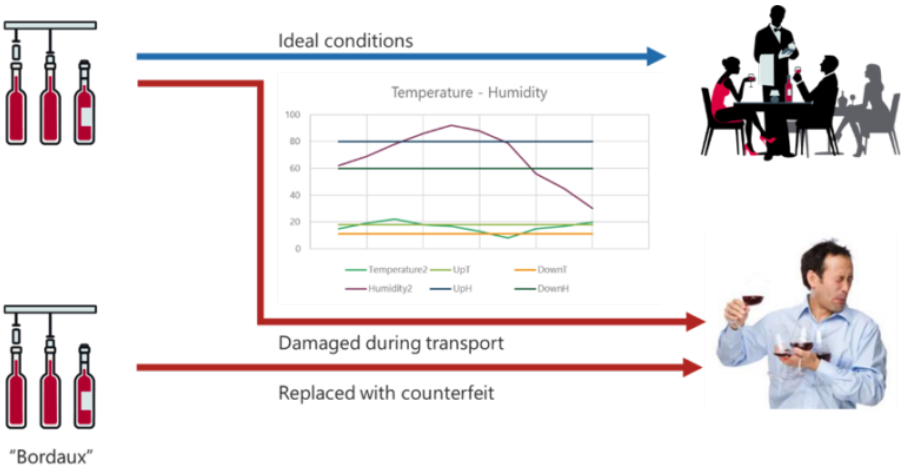


Figure 2.6.: Monitoring temperature and humidity of wine during transportation.

2.4. The AGILE IoT Project

Funding for this research from 2016 to 2018 was provided by HORIZON 2020 EU Framework Programme for Research and Innovation[1] was inspired by an IoT project called: Adaptive Gateways for dIverse muLtiple Environments (AGILE) project[2]. AGILE IoT builds a modular hardware and software gateway for the Internet of Things with support for protocol interoperability, device/data management, IoT apps execution, and external Cloud communication, featuring diverse pilot activities, Open Calls & Community building. It is developed by a consortium of 16 prominent European partners as in Figure 2.7.

AGILE run five pilots by Quantified Self covering everything from wearables for self-tracking to open-air crop and livestock monitoring using drones to smart retail solutions for enhanced shopping experiences. These pilots demonstrate both the applicability of the hardware and software in managing IoT devices and creating applications but also the importance of application design and data-sharing. AGILE's pilots set a foundation for further commercial exploitation of the Project and its innovations.

AGILE has become a part of the existing IoT-Lab infrastructure in France managed by INRIA. With more than 2500 sensors deployed in 5 locations, AGILE users have the opportunity to evaluate their IoT applications in real-world environments, collect and store sensor data, and interact with real devices.

AGILE also builds a modular and adaptive gateways for IoT devices. Modularity at the hardware level provides support for various wireless and wired IoT networking technologies (e.g. KNX, ZWave, ZigBee, Bluetooth Low Energy, etc.) and allows fast prototyping of IoT solutions for various domains (e.g. home automation, environment monitoring, wearables, etc.). At the software level, different components enable new features: data collection and management on the gateway, intuitive interface for device management, visual workflow editor for creating IoT apps with less coding, and an IoT marketplace for installing IoT apps locally. The AGILE software can auto-configure and adapt based on the hardware configuration so that driver installation and configuration are performed automatically. IoT apps are recommended based on hardware setup, reducing the gateway setup and development time significantly.

[1] https://ec.europa.eu/programmes/horizon2020/what-horizon-2020
[2] http://agile-iot.eu/

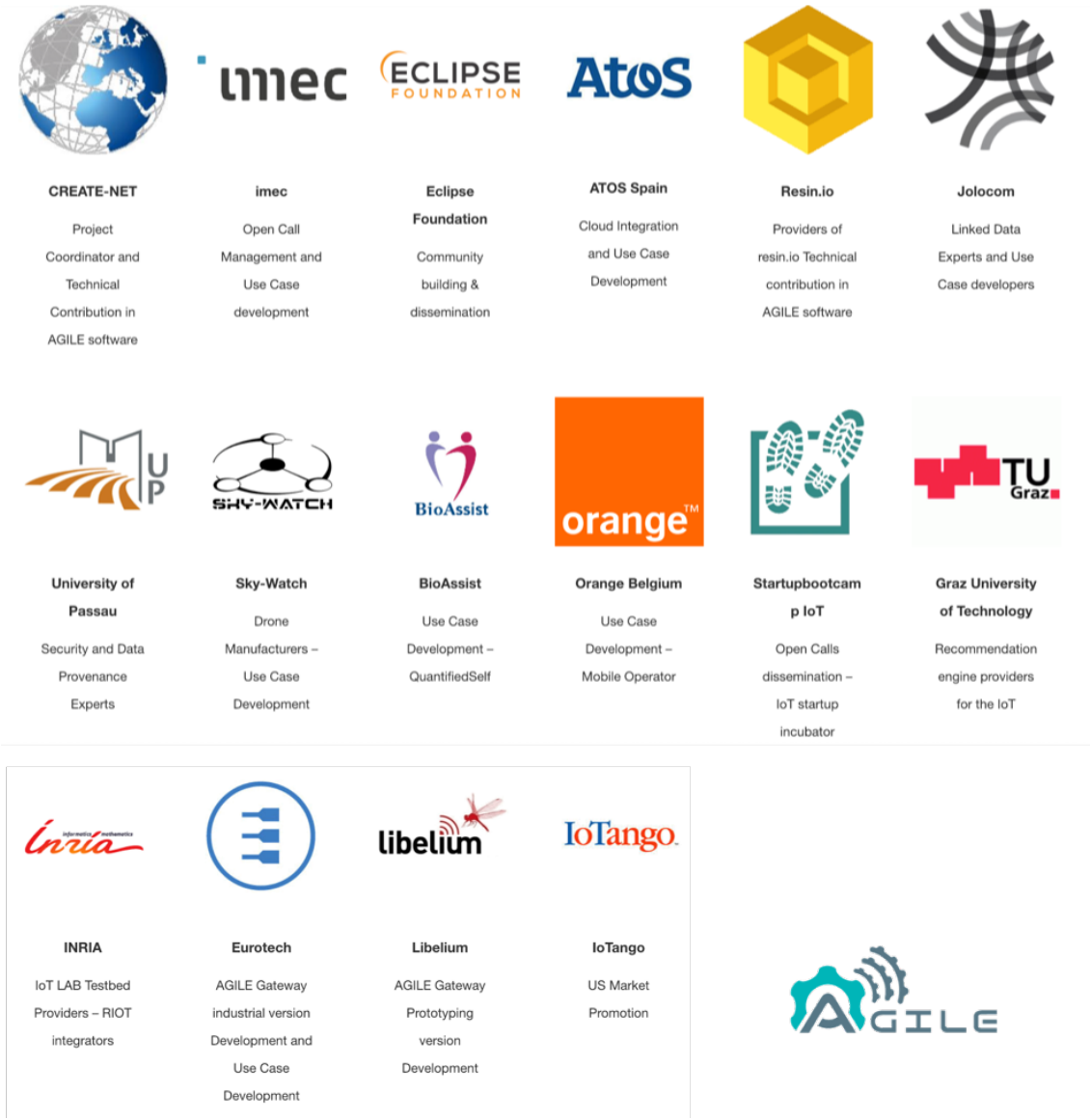


Figure 2.7.: Project consortium of the AGILE IoT Project.

All AGILE software modules (see Figure 2.8) will be delivered as 100% Open Source, with the majority of them becoming part of a new Eclipse Foundation IoT Project. The objective is to provide IoT developers and communities with software components for effective and agile IoT prototyping while simultaneously establishing a community of users and developers, maximizing the adoption of the AGILE Project.

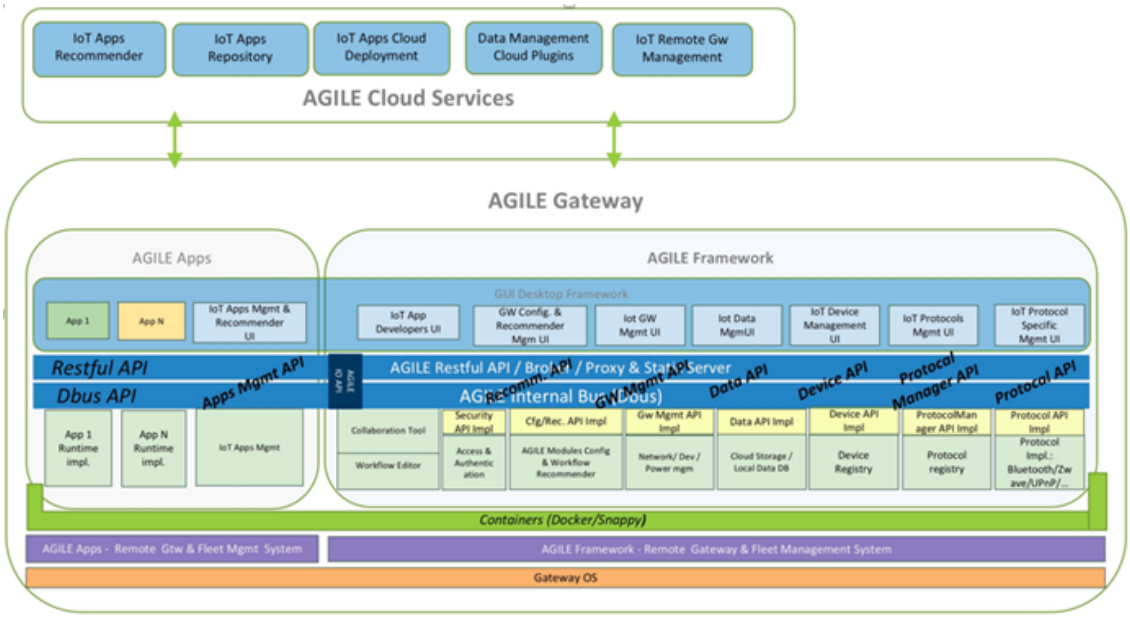


Figure 2.8.: Software Architecture of AGILE IoT.

In the context of this project, there are various scenarios that require the support of configuration and recommendation technologies. As a university partner of AGILE IoT consortium, we have developed several recommendation and configuration technologies for IoT scenarios. This thesis abbreviate this project shortly as *"AGILE"* or *"AGILE IoT"*.

Chapter 3

Improving Configuration Technologies for IoT Scenarios

Parts of the contents of this chapter have been published in the proceedings of ConfWS'16, IEA/AIE'17, IIST'18, KI'18, PATAT'18, IEA/AIE'19, *and* SAC'19 *(Felfernig et al., 2016; Erdeniz et al., 2017; Erdeniz and Felfernig, 2018b; Erdeniz et al., 2018a; Erdeniz and Felfernig, 2018a; Erdeniz et al., 2019a,b).*

Configuration systems (Sabin and Weigel, 1998) are used to find solutions for problems which have many variables and constraints. A configuration problem corresponds to a complex design activity, which can be defined as follows: given a catalog of components, compose an artifact such that the complex requirements reflecting the individual needs of a customer/user and the compatibility of a system's structures are satisfied.

A configuration problem can be defined as a constraint satisfaction problem (*CSP*) (Tsang, 1993). If constraints of a *CSP* are inconsistent, no solution can be found. In this context, diagnosis (Bakker et al., 1993) is required to find at least one solution for an inconsistent *CSP*. Configuration systems must be able to deal with inconsistencies which can occur in different contexts. Especially in interactive settings, where users specify requirements and a constraint solver has to identify solutions, inconsistencies may more often arise. In inconsistency situations, there is a need of diagnosis methods that support the identification of minimal sets of constraints that have to be adapted or deleted in order to restore consistency.

Performance of a configuration system can be evaluated in terms of time to find a solution/diagnosis (the runtime) and solution/diagnosis quality. Runtime efficiency is especially crucial in real-time scenarios such as production scheduling, robot control, and communication networks. However, there is a trade off between solution/diagnosis quality and the runtime efficiency of a configuration system. In this chapter, we deal with solving *the quality-runtime performance trade off problem* of configuration systems. In this context, we propose *novel learning approaches for variable and value ordering or constraint ordering*. We show that our approaches improve the runtime performance and solution/diagnosis quality of configuration systems.

In this chapter, we present our proposed heuristics learning approaches for configuration systems. First in Section 3.1 (Felfernig et al., 2016), we provide an overview of configuration technologies in IoT. In Section 3.2 (Erdeniz et al., 2017; Erdeniz and Felfernig, 2018a,b), our approach employs a learning algorithm based on clustered historical transactions. In Section 3.3 (Erdeniz et al., 2018a), we adapt the similar approach given in the previous section for diagnosis. Then we introduce another novel learning algorithm for configuration systems in Section 3.4 (Erdeniz et al., 2019b) and finally adapt this method for diagnosing in Section 3.5 (Erdeniz et al., 2019a).

3.1. An Overview of Configuration Systems in IoT

Configuration is a process in which agents (users or external systems) can specify requirements and the configuration system (often denoted as the *configurator*) provides feedback in terms of solutions and/or explanations (Felfernig et al., 2014b; Sabin and Weigel, 1998; Stumptner, 1997). Configuration can be interpreted as a type of design activity where a product is composed (configured) from a set of instances corresponding to predefined component types such that the resulting configuration (solution) is consistent with a given set of constraints (Sabin and Weigel, 1998; Stumptner, 1997). Requirements can be regarded as specifications of intended properties of the product (i.e., specific constraints), for example, a specific *application* or *mail server version* should be included in the *system configuration*. In such contexts, system feedback for a user is provided in terms of configurations, reconfigurations, and explanations for situations in which no solution could be found. There is a multitude of application examples of knowledge-based configuration, for example, in the automotive domain, railway interlocking systems, financial services, operating systems, and software product lines (Felfernig et al., 2014b).

Configuration services for the Internet of Things (IoT) domain (Atzori et al., 2010) is a new application area. The IoT is an emerging paradigm that envisions a networked infrastructure enabling different devices (things) to be interconnected at anyplace and anytime. In this section, we discuss two basic scenarios that will be supported by software components to be developed in the AGILE research project. First, *ramp-up configuration* services will be developed that help to determine an initial configuration for the whole IoT gateway infrastructure. One of the major tasks of such gateways[1] is to bridge devices to corresponding applications on the basis of different communication protocols such as Hue and Zigbee. For example, in the smarthome domain, a configuration would determine the set of sensors, connection protocols, and apps needed to make a gateway operable for the user. Second, we will develop technologies that help to *optimize configuration and reconfiguration* of communication protocols in such a way that user requirements (e.g., performance requirements) and side conditions (e.g., available bandwidth) can be taken into account.

3.1.1. Ramp-Up Configuration in IoT

AGILE gateways will be deployed in different domains such as *health monitoring*, *animal monitoring in wildlife areas*, *air quality and pollution monitoring*, *enhanced retail services*, *smart homes*, and *port area monitoring*. Each application scenario requires a pre-configuration which estimates the needed hardware and software components / devices to be deployed in the ramp-up phase of the system. We denote this type of configuration *ramp-up configuration* since each scenario requires a specific set of hardware components and software components (including apps) to "ramp-up" the system.

AGILE Air Pollution Monitoring

Environmental pollution has become an issue of serious international concern and is increasingly stimulating the development and adoption of solutions to monitor and reduce the effects of pollution. This is an interesting and challenging market, with both potential economical outcomes and a strong societal impact. The convergence of hardware integration, reduction of sensor costs, IoT and M2M technologies introduces a new panorama where it is really possible to deliver low cost, high quality monitoring systems with a capillary coverage of the territory. This convergence leads to a new era of solution for environmental pollution monitoring. Air quality and pollution monitoring stations are complex systems that, depending on the application context, deliver added value pollution monitoring services based on a delicate equilibrium between the adoption of the most appropriate sensors, their correlation, the selection of the correct algorithms and the configuration of their hardware and software parameters. A wrong or imprecise selection

[1] *Raspberry Pi* is one of the hardware platforms used in AGILE – see www.raspberrypi.org

and configuration of these elements leads to misleading, wrong, and completely useless results and services. *Air Pollution Monitoring* is in the need of configuration support since the measuring equipment has to be pre-selected and parametrized in the line of the environmental conditions, for example, in a city.

Further AGILE Scenarios

The basic task of a configurator in *health monitoring* is to figure out which measuring devices are needed (including their parametrization) to be able to monitor and analyze specific body functions. In the *animal monitoring* scenario it is important to figure out which infrastructure can be used to complete predefined data collection tasks. In such scenarios, *reachability* of animals (and corresponding sensors) plays a major role in order to be able to complete data collection. Reachability depends on the selected drone types but also on the selected communication protocols which have different degrees of power consumption. *Enhanced retail services* that allow a personalized shopping experience in physical stores are in the need of configuration functionalities that indicate the amount and positioning of sensors (e.g., for indoor position detection) and displays that are needed to successfully support customers in their shopping experiences. In the *port area monitoring* scenario, configuration technologies are needed that help to select relevant sensors (e.g., gas, radioactivity, and water quality sensors) that are able to provide the needed data. In *smarthome* scenarios, the task of the configurator is to identify sensors, communication components, and protocols that are needed to provide the smarthome functionalities required by a customer. In this context, examples of customer requirements are rooms in the house and their type, maximum accepted price, and needed functionalities (e.g., presence monitoring and simulation, and video surveillance).

Knowledge Acquisition & Representation

When configuring, for example, *smart homes*, the configuration model includes information about the relationships between building properties and corresponding sensors (e.g., *if a room is a kitchen and includes an oven, then a corresponding temperature sensor has to be included for the room*) or between user preferences and the corresponding technical infrastructure (e.g., *if a user wants to save money, wireless communication is preferred*). A configuration for a given configuration task includes information about which components, devices, and drivers are part of the initial gateway installation.

In AGILE, we will evaluate the applicability of different types of configuration knowledge representations such as answer set programs (ASP) (Myllärniemi et al., 2014) and constraint-based representations (Felfernig et al., 2014b; Fleischanderl et al., 1998). Our aim is to *identify a knowledge representation language* that can be applied for each of the different application scenarios in order to provide a basic technology for supporting IoT ramp-up configuration tasks. The applicability of these languages will be primarily evaluated with regard to expressiveness and reasoning efficiency. Especially, ASP-based configuration approaches will be evaluated with regard to their applicability in typical gateway ramp-up scenarios.

Consistency Management of Knowledge Bases

Configuration knowledge bases can become inconsistent, i.e., the defined component types and constraints lead to the problem that no solution can be identified. Such a situation can occur in the context of regression testing (Felfernig et al., 2004) but also in situations where the conflict is induced by the configuration knowledge base itself. In such scenarios, configuration technologies in combination with model-based diagnosis (Reiter, 1987) can be exploited to automatically identify the sources (e.g., constraints) of a given inconsistency (Felfernig et al., 2004). Such functionalities will be included in a development environment for IoT configuration knowledge bases.

In the context of AGILE, we focus on the development of techniques that help to improve the efficiency

user	req_1	req_2	x_1	x_2	x_3	x_4
u_1	1	2	3	4	4	2
u_2	2	2	8	3	4	2
u_3	1	2	3	4	5	2
current	1	1	?	?	?	?

Table 3.1.: Example configuration log.

of configuration knowledge engineering processes. Although automated debugging (Felfernig et al., 2004) is a useful means to reduce time efforts related knowledge base development and maintenance, the development and maintenance of related test cases (also denotes as examples (Felfernig et al., 2004)) is still costly. We will analyze the applicability of different testing approaches from software engineering and will especially focus on the the development of *mutation testing* approaches for knowledge bases (Jia and Harman, 2011). In this context, a mutation will serve as a basis for generating tests that are, for example, accepted by the original knowledge base but should not.

Consistency Management of User Requirements

Consistency management not only plays a role in the context of knowledge base development and maintenance but also within the scope of a configuration process. A user of an AGILE configurator could articulate a set of requirements in such a way that no solution can be identified. Also in such a situation, model-based diagnosis approaches can be exploited to indicate sets of user requirements that have to be adapted such that at least one solution can be identified (Felfernig et al., 2004; Marques-Silva et al., 2013; Walter et al., 2016). A similar situation occurs in the context of reconfiguration, i.e., in a situation where hardware and software components of an IoT gateway have to be adapted. In this context, minimal changes have to be proposed that indicate how the existing configuration has to be adapted such that a consistent configuration can be determined that takes into account all requirements (Felfernig et al., 2015b).

In AGILE, we focus on the development of personalization techniques that help to improve the diagnosis prediction quality, i.e., to identify those diagnoses that will be accepted by the user. Such personalized diagnoses will be determined on the basis of an analysis of the interaction behavior of users of similar gateway installations (available in gateway profile repositories). In this context we will develop learning-based approaches that help to calibrate search heuristics in order to improve efficiency and prediction quality of configuration and reconfiguration.

A simple example of our envisioned approach is the following. Let us assume the existence of a *configuration log* as the one shown in Table 3.1 as a basis for optimizing the prediction quality of configuration parameters. The parameters req_i indicate user requirements and x_i indicate technical product parameter settings (consistent with the user requirements) accepted by the user u_i. The overall goal is to optimize the configurator search heuristics (e.g., variable and domain orderings) in such a way, that the prediction quality for the technical parameter settings is maximized. More precisely, we want to identify search heuristics that guide to solutions (configurations) that will be accepted by the *current* user. User interactions (see, e.g., Table 3.1) serve as a basis for learning. Prediction quality can be measured, for example, in terms of the user acceptance degree of parameter settings (configurations) proposed by the configurator. In this context we will evaluate different clustering techniques, i.e., to learn heuristics not on a global level, but depending on a specific cluster derived, for example, from the user requirements.

configuration	performance	reliability	costs
$conf_a$	9	5	2
$conf_b$	5	8	3

Table 3.2.: Configuration utilities w.r.t. interest dimensions *performance*, *reliability*, and *costs*.

user	performance	reliability	costs
u_1	10	3	1
u_2	5	7	10

Table 3.3.: User preferences w.r.t. interest dimensions *performance*, *reliability*, and *costs*.

3.1.2. Runtime Configuration in IoT

Modern embedded systems included in IoT scenarios support a rich set of connectivity solutions (e.g., 3G, LTE, TD-LTE, FDD-LTD, WIMAX, and Lora). In this context, configuration technologies play an important role in terms of suggesting optimal connectivity configurations. Such configurations include a collection of connectivity solutions that are needed to support a set of active applications (apps). Criteria that have to be taken into account are, for example, location information, available connectivity, performance and reliability requirements, contractual aspects, and costs.

In AGILE, runtime configuration must be performed on the gateway – in contrast, ramp-up configuration can also take place in the cloud. On the one hand we will evaluate different types of reasoning engines, for example, the CHOCO constraint solver (choco-solver.org) and the Sat4j boolean satisfaction library (sat4j.org). We will also take into account the application of rule engines (java-source.net/open-source/rule-engines), optimization libraries, and knowledge compression techniques (Andersen, 1999) to assure efficiency of problem solving on the gateway level.

configuration topic	research objective
appropriate knowledge representations	knowledge representations for easy modeling and efficient configuration search
efficiency of knowledge base development and maintenance	automated test case generation and mutation testing
personalized consistency management	personalized configuration based on learning search heuristics and knowledge compression techniques

Table 3.4.: Overview of AGILE research objectives.

In AGILE, gateway configurations can be manually defined by users but also be determined on the basis of a configurator that is in charge of keeping the overall system installations consistent. A configurator (e.g., a constraint solver) can determine alternative configurations which have to be ranked. In order to determine a ranking for alternative configurations, a MAUT (Multi-attribute utility theory) approach can be used (Winterfeldt and Edwards, 1986). Examples of evaluation *dimensions* (dim) used in MAUT could be *performance*, *reliability*, and *costs*. Depending on the current gateway configuration and the usage context, a configurator can determine alternative (re-)configurations and rank them accordingly.

A simplified example of the application of a utility-based approach is the following. Table 4.4 includes an evaluation of connectivity protocol configurations $conf$ ($conf_a$ and $conf_b$) to be used on the gateway, for example, for different types of data exchange. The three evaluation dimensions used in this example are performance, reliability, and costs. Furthermore, Table 4.5 includes the personal preferences of two different gateway users (u_1 and u_2).

In order to determine the configuration that should be chosen for a specific user, we can apply the utility function (see, e.g., Formula 2.3) to configurable items as shown in Formula 3.1.

$$utility(conf,u) = \Sigma_{d \in dim} interest(u,d) \times value(conf,d) \tag{3.1}$$

In this context, *utility(conf,u)* denotes the utility of the configuration $conf$ for the user u, *interest(u,d)* denotes the interest of user u in evaluation dimension d, and *value(conf,u)* denotes the contribution of configuration $conf$ to the interest dimension d. In the example, configuration $conf_a$ has a higher utility for user u_1 (107.0) whereas configuration $conf_b$ has a higher utility for u_2 (111.0). Note that for simplicity we omitted to sketch the determination of the evaluations depicted in Table 4.4 – for details see (Felfernig et al., 2006). In order to increase the efficiency of runtime configuration, we will evaluate knowledge compression techniques that help to reduce search efforts as much as possible. For example, we will apply decision diagram techniques (Andersen, 1999) to pre-calculate possible configurations and re-configurations.

Table 3.4 provides a summary of the configuration-related research objectives in AGILE. Within the context of ramp-up configuration scenarios we will identify knowledge representation mechanisms that allow an easy representation of the AGILE IoT domains introduced in Section 3.1.1. Furthermore, we will develop test case generation techniques that will help to make the development and management of test cases more efficient. For AGILE scenarios, we will develop concepts that support the learning of search heuristics to optimize configuration and reconfiguration processes. Furthermore, we will work on knowledge compression techniques (Andersen, 1999) that help to make solution search on the gateway level as efficient as possible.

Although different from basic IoT scenarios (Atzori et al., 2010), there exist applications that support the configuration of systems including hardware and software components. Falkner and Schreiner (Falkner and Schreiner, 2014) introduce approaches to the configuration of railway interlocking systems as examples of complex industrial systems designed on the basis of constraint-based configuration technologies. Krebs et al. (Krebs et al., 2002) show the application of configuration technologies in the area of car periphery supervision that includes detection of the car environment, the recognition of hazardous situations, and the handling of difficult traffic situations. Related applications are pre-crash detection, the detection of obstacles, and parking assistance. Related car configuration processes have to take into account existing hardware components and to combine these with the corresponding software units. Finally, Perera et al. (Perera et al., 2013) introduce an approach to the end-user-oriented configuration of IoT middleware components.

The afore mentioned approaches are in the line of the mentioned "ramp-up" scenario, i.e., infrastructures are configured before the system is operable. In contrast to the developments in (Falkner and Schreiner, 2014; Krebs et al., 2002; Perera et al., 2013), the "ramp-up" configuration approach that is currently developed in AGILE focuses on advanced testing methods for supporting configuration knowledge engineering and also on approaches to improve configurator usability by the inclusion of different types of personalized consistency restoration methods. Initial approaches to include recommendation methods into configuration problem solving are documented, for example, in (Tiihonen and Felfernig, 2010). These approaches do not take into account the issue of consistency management in a satisfactory fashion which will be a major focus of our work in the AGILE project. Finally, for an overview of different IoT smart solutions available on the market we refer to (Perera et al., 2015).

3.1.3. Conclusions

In this section, we provided an introduction to basic configuration scenarios of the AGILE project. We discussed the two scenarios of "ramp-up configuration" and "runtime optimization". Major challenges in this context are approaches to automated test case generation for configuration knowledge bases, efficient techniques to solve the "no solution can be found" problem in interactive configuration settings, and the personalization of related repair approaches.

3.2. Cluster-Specific Heuristics for Constraint Solving

In complex configuration problems, *configuration systems* (Benavides et al., 2013; Erdeniz et al., 2017; Felfernig et al., 2001, 2018c, 2004) are used to find solutions. *Graph coloring problems* can be formulated in the form of constraint satisfaction problems (*CSP*) (Felfernig et al., 2000; Tsang, 1993; Yang et al., 2012) which is defined as a triple *(V,D,C)* where V is a set of variables, D is a set of domains for each variable, and C is a set of constraints. The constraint set may also include the additional (dynamic) constraints C_{REQ} if available. In this case, set of constraints have two components as $C = C_{KB} \cup C_{\text{REQ}}$, where C_{KB} defines knowledge base constraints and C_{REQ} defines dynamic constraints.

Various search techniques can be applied in order to improve the performance of CSP solvers. Variable and value ordering heuristics are common intelligent search techniques which are used for solving many kinds of problems such as *configuration*, *job shop scheduling*, and *integrated circuit design* (Jannach, 2013; Li and Epstein, 2010; Liu et al., 2008; O'Sullivan et al., 2004; Pearl, 1984; Sadeh and Fox, 1996). Searching for a solution of a given CSP consists of techniques for systematic exploration of the space of all solutions. The basic brute force algorithm *generate and test* (*trial and error*) search is based on the idea of testing every possible combination of values to obtain a solution of a CSP.

Variable and value ordering heuristics are used to determine which variable and value to assign next on the search tree. The heuristics can vary from branch to branch which is called *dynamically* chosen variable ordering and it has a tremendous impact on performance. The variable and value orders can be figured out in advance (a static ordering) or dynamically, using information available at the time that the choice is made. These choices can have enormous effects (Beck et al., 2004; Dechter and Pearl, 1988; Johnston and Minton, 1994; Kumar, 1992; Mouhoub and Jafari, 2011). An example of a variable ordering heuristic could be the *fail-first* principle which means; *choose next the variable with the smallest remaining domain*. On the other hand, an example of value ordering heuristics could be the *max-domain* principle in which search starts with the highest domain values of the variables.

The variable and value orders can be determined in advance (a static ordering) or dynamically, using information available at the time that the choice is made. These choices can have a dramatic effect on the time taken to and a solution to a CSP. In order to increase the runtime performance of CSP solvers, in paper (Da Col and Teppan, 2017), authors propose a genetic algorithm based learning approach that automatically derives heuristics for the exploitation by constraint solvers. In another similar approach, the authors of paper (Erdeniz et al., 2017) also uses clustering techniques but only for learning variable ordering heuristics.

We propose a novel approach for learning both variable and value ordering heuristics to improve runtime performance of CSP solvers. We demonstrate how the proposed approach is applied and present the experimental evaluation based on a well-known and real-life configuration problem from *graph coloring* which is called *"Precoloring Extension Problems"*. We have chosen precoloring extension problems because by default these problems include the additional (dynamic) constraints C_{REQ} and in our approach we exploit those additional constraints to learn heuristics. The research results are applied in the internet of things domain (AGILE).

Precoloring Extension Problems

Configuration systems are mostly exploited in scheduling systems. There are numerous examples in scheduling theory, where the assignment of resources can be reduced to a *problem of assigning colors in a graph*, such as; scheduling committees, aircraft assignment, optical networks, register allocations, timetabling, task (job) scheduling (Barba et al., 2017; Yuan et al., 2017).

In most cases, however, the real-life problem does not appear in such a pure form as in the examples above, there are additional (dynamic) constraints that have to be satisfied. For example, a flight can be performed only by certain aircrafts, or a teacher is not available on certain days. Such problems can be defined as *precoloring extension problems*.

Graph Theory. Graph coloring is certainly among the most studied questions in computer science and combinatorics, and countless applications and variants have been tackled since it was first posed for the special case of maps in the mid-nineteenth century. Graph coloring was motivated by map makers that apparently every planar map can be colored using four colors in such a way that countries sharing a boundary have different colors.

As with other parts of graph theory (and with mathematics in general), the new directions were motivated both by pure theoretical interest and by possible practical applications. It turned out that besides coloring maps, there are several other situations that can be modeled by graph coloring and its variants. There are numerous examples in *scheduling theory*, where the assignment of resources can be reduced to a problem of assigning colors in a graph (Marx, 2004).

Graph theory (Kumar, 1992) has considerable application to a large variety of complex problems involving optimization. In particular conflict resolution can often be accomplished by means of graph coloring. The constraints are usually expressible in the form of pairs of incompatible objects (e.g., pairs of chemicals that cannot be stored on the same shelf). Such incompatibilities are usefully embodied through the structure of a graph. Examples of such problems include: the scheduling of exams in the smallest number of time periods such that no individual is required to participate in two exams simultaneously, the storage of chemicals on the minimum number of shelves such that no two mutually dangerous chemicals (i.e., dangerous when one is in the presence of the other) are stored on the same shelf, and the pairing of individuals (as in a computer dating agency) such that the maximal number of compatible persons are paired together (Leighton, 1979).

In order to convert such problems into a graph coloring problem (see Definition 3.2.1), each object is represented by a node and each in compatibility is represented by an edge joining the two nodes. A coloring of this graph is then simply a partitioning of the objects into blocks (or colors) such that no two incompatible objects end up in the same block.

Definition 3.2.1. (Vertex Coloring). A vertex coloring can be defined as G = (V, E, C) where V = (v_1,..,v_n) is the set of vertices, E = (e_1,..,e_n) is the set of edges between two vertices, and C = (c_1,..,c_k) is the set of available colors. G is a function f : V $\Longrightarrow$ C from the set V of vertices to a set C of colors such that any two incident vertices are assigned different colors.

Precoloring Extension. In graph theory, precoloring extension is the problem of extending a graph coloring of a subset of the vertices of a graph, with a given set of colors, to a coloring of the whole graph that does not assign the same color to any two adjacent vertices. It has the usual graph coloring problem as a special case, in which the initially colored subset of vertices is empty; therefore, it is NP-complete. In the precoloring extension problem some vertices of a graph have preassigned colors, and this precoloring has to be extended to the whole graph using the given number of colors.

The problem is not equivalent to vertex coloring: it is possible that a graph is k-colorable, but there is an unfortunate precoloring that cannot be finished using k-colors. Precoloring extension can be viewed as

a special case of vertex coloring: a precolored vertex contains only a single color, while a not precolored vertex contains all the available colors. Thus, we can expect that in certain situations, this special case of list coloring is easier to solve than the general problem.

Definition 3.2.2. (Precoloring Extension). A precoloring extension of graph coloring can be defined as G = (V, E, C, P) where V = (v_1,..,v_n) is the set of vertices, E = (e_1,..,e_n) is the set of edges between two vertices, C = (c_1,..,c_k) is the set of available colors, and P = (v_k = c_x,..,v_m = c_y) is the set of precolored vertices. G is a function f : V $\Longrightarrow$ C from the set V of vertices to a set C of colors such that any two incident vertices are assigned different colors in consistency with the precoloring set P.

State-of-the-art

Our approach deals with *precoloring extension problems* rather than pure graph coloring problems. However, we analyzed several state-of-the-art approaches for graph coloring problems since it is a well-studied and active research topic. Because the graph coloring problem is NP-hard, most of them are heuristic methods like greedy algorithms (DSATUR (Brélaz, 1979), RLF (Leighton, 1979)), local search (tabu (Hertz and de Werra, 1987), simulated annealing (Morgenstern and Shapiro, 1986), genetic algorithm (Fleurent and Ferland, 1996)), possibly featuring a certain amount of enumeration. Most techniques generate the coloring either by sequentially assigning a color to all the vertices, or by partitioning the vertices into independent sets mapped to color classes (Barnier and Brisset, 2004).

Constraint programming has already been used as well to color graphs of *reasonable* size. They are most often implicitly represented, mapping nodes on variables and edges on disequality constraints. The most popular variable ordering strategies were inspired by heuristics developed for pure coloring problems (Brélaz, 1979; Dincbas et al., 1990). However, this early approach did not use clusters or learning techniques to decide on heuristics. A very different CSP approach is presented in (Caramia and Dell'Olmo, 2002) to obtain improved runtime performance. However, this technique would be hard to implement using the high level primitives of a standard CP system only (Barnier and Brisset, 2004).

Unlike previously mentioned techniques, the Constraint Programming (especially the constraint satisfaction problems) provides a high level of abstraction which allows to easily refine the model by adding other various constraints. This is a highly suitable feature for our application since it is based on dynamic constraints on top of a graph coloring problem. One of its important limitations is the computational cost of standard backtrack search on large instances. Various attempts can be found to overcome this issue like the Incomplete Dynamic Backtracking algorithm presented in (Prestwich, 2001), but with the cost of losing completeness. However, our approach offers a complete solution with a good runtime performance specifically for precoloring extension problems.

3.2.1. Problem Definition

As described above, a real-life problem (which can be converted into a graph coloring problem) does not appear as a pure graph coloring problem, there are additional constraints that have to be satisfied. Therefore, in this work we focus on solving *precoloring extension problems* efficiently.

The *precoloring extension problem*, which is shown in Fig. 3.1, will serve as a working example throughout this section. In this problem, we need to color each of four vertices of the graph, from the domain of colors: red, blue, green, such that no two adjacent vertices have the same color and the additional constraint (**V1**=blue) should be also satisfied.

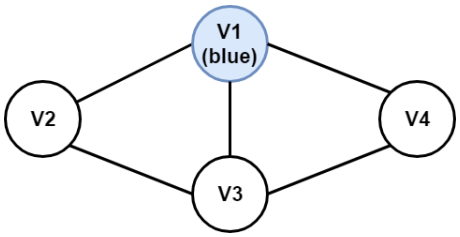


Figure 3.1.: A precoloring extension problem

Table 3.5.: The CSP definition of the working example: CSP_{pre}

V	$\mathbf{V1}, \mathbf{V2}, \mathbf{V3}, \mathbf{V4}$
D	$dom(\mathbf{V1}) = \{red, blue, green\}$, $dom(\mathbf{V2}) = \{red, blue, green\}$, $dom(\mathbf{V3}) = \{red, blue, green\}$, $dom(\mathbf{V4}) = \{red, blue, green\}$
C_{KB}	$c_1 : \mathbf{V1} \neq \mathbf{V2}$, $c_2 : \mathbf{V2} \neq \mathbf{V3}$, $c_3 : \mathbf{V3} \neq \mathbf{V4}$, $c_4 : \mathbf{V4} \neq \mathbf{V1}$, $c_5 : \mathbf{V1} \neq \mathbf{V3}$
C_{REQ}	$c_6 : \mathbf{V1} = blue$

In Table 3.5, we formulate our working example in the form of constraint satisfaction problem (see Definition 2.1.1). As observed, each edge in the graph is defined as a constraint in C_{KB} (five edges make five constraints) and the additional precoloring constraint is defined as C_{REQ}.

The following depth-first backtrack search tree in Fig. 3.2 demonstrates how the solution search (without employing any ordering heuristics) is applied on CSP_{pre} on the basis of depth-first backtrack search. The search stops at a node, when the assignments on the *arc consistency* is violated. In this example (no heuristics applied) the search tree is generated according to the given orders of variables and values in the CSP definition; variable order from top to down is **V1**, **V2**, **V3**, **V4**; and value order is respectively *red*, *blue*, and *green*. Based on the default variable and value orders, we find a solution: {**V1**=blue, **V2**=red, **V3**=green, **V4**=red}.

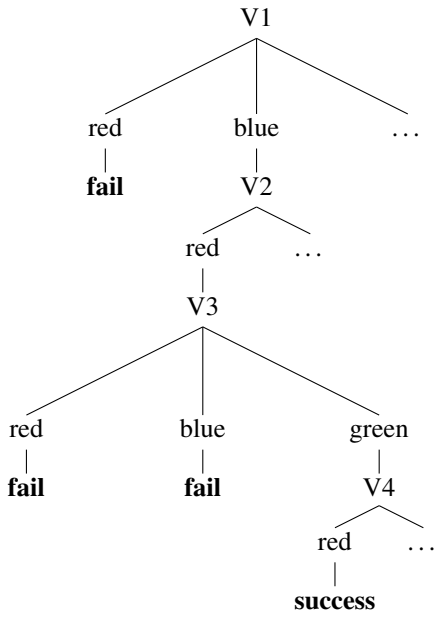


Figure 3.2.: The depth-first backtrack search tree to solve CSP_{pre} without heuristics

However, in our proposed approach we aim to learn variable and value ordering heuristics based on *historical dynamic constraints*. These are the dynamic constraints which are added on top of the same graph coloring problem in the past. For example, a historical dynamic constraint set (C_{REQ}) can hold a former teacher's available days and hours for teaching a course.

In Table 3.6, we have six sets of historical dynamic constraints ($C_{\text{REQ}_1}..C_{\text{REQ}_6}$) which are used to learn variable and value ordering heuristics to solve the working example. Each set of historical dynamic constraints contains several constraints. For example, C_{REQ_1} consists of four dynamic constraints as $c_6 : \mathbf{V1} = blue$, $c_7 : \mathbf{V2} = \{red, blue, green\}$, $c_8 : \mathbf{V3} = \{red, blue, green\}$, and $c_9 : \mathbf{V4} = green$.

Table 3.6.: Six Sets of Historical Dynamic Constraints

		C_{REQ_1}	C_{REQ_2}	C_{REQ_3}	C_{REQ_4}	C_{REQ_5}	C_{REQ_6}
c6	**V1**	blue	red	blue	red	red	red
c7	**V1**	-	blue	-	blue	-	green
c8	**V1**	-	-	-	-	-	-
c9	**V1**	green	-	red	-	blue	-

3.2.2. The Proposed Method

Our proposed method CLUSTER AND LEARN improves runtime efficiency of solution searching for *pre-coloring extension problems*. It is a novel variable and value ordering heuristics learning method based on historical dynamic constraints based on the same graph coloring problem.

In an offline phase, it uses a k-means clustering algorithm to cluster sets of historical dynamic constraints. After clustering, again in an offline phase, it applies supervised learning based on a genetic algorithm (Whitley, 1994) to learn the variable and value ordering heuristics for each cluster. After completing

learning in an offline phase, it starts working in an online phase. When a *precoloring extension problem* (based on the same graph coloring problem of sets of historical dynamic constraints) is needed to be solved, CLUSTER AND LEARN finds the closest cluster to the new set of dynamic constraints and employs the corresponding learned heuristics of this cluster. These steps are described based on the working example in the following subsections.

Clustering Dynamic Constraints

In an offline phase, we cluster sets of historical dynamic constraints. We have employed clustering to quickly estimate accurate heuristics in the online phase during solving a precoloring extension problem. Otherwise, we would have to compare the new precoloring extension problem with all historical precoloring extension problems to find the most similar one. This would take a very long time when the historical problems dataset gets bigger. Moreover, by using cluster-specific heuristics, we also optimize the performance of heuristics by finding a common heuristic which works well for each cluster member (each precoloring extension problem in a cluster). Therefore, we expect that a cluster-specific heuristic works also well for a similar precoloring extension problem to this cluster.

We have preferred to use a well known clustering algorithm *k-means clustering* (see Formula 4.2) (Lloyd, 1982; MacQueen et al., 1967) because of the structure of our dataset. We have colors as values and decided to take the distances of each different colors as 1. This means, only the difference is meaningful, rather than the values.

In our working example, we have six past dynamic constraints as shown in Table 3.6. Constraints start with c_6 because there are already defined five knowledge base constraints in the precoloring extension problem in Table 3.5. Distances between two sets of historical dynamic constraints are calculated based on euclidean distance as shown in Formula 4.3.

Since we have colors as values in graph coloring problems, we assume the distance between two different precolored variables is 1, the distance between one precolored, one non-precolored variable is 2, and rest is 0. For example, the distance between the two sets of historical dynamic constraints C_{REQ_1} and C_{REQ_2} is calculated according to Formula 4.3 as follows:

$$\sqrt{\sum_{i=6}^{9}(C_{\text{REQ}_{1\_c_i}} - C_{\text{REQ}_{2\_c_i}})^2} = \sqrt{1^2+2^2+0^2+2^2} = 3.$$

We cluster these past six dynamic constraints, by applying k-means clustering which minimizes the total distances between $C_{REQ}s$ (see Formula 4.2). In this working example, we set the number of clusters (k) to 2 (to keep the example small and understandable). After applying k-means clustering (k=2 and distance measure is euclidean distance), we obtain the following two clusters; *Cluster-1: {C_{REQ_1}, C_{REQ_3}, C_{REQ_5}}* and *Cluster-2: {C_{REQ_2}, C_{REQ_4}, C_{REQ_6}}*.

Learning *CLH*

After clustering past dynamic constraint sets, again in an offline phase, we run a genetic algorithm separately on each cluster to learn best fitting variable and value ordering heuristics for solving precoloring problems in a cluster. *Runtime* is our performance indicator which represents the time spent by the CSP solver to find a configuration. To improve the runtime performance of CSP solving, we calculate CLUSTER AND LEARN heuristics (*CLH*) which minimize the total runtime of CSP solving of problems in a cluster.

As observed in the nature, genetic algorithms use *fitness functions* to find a best fitting individual out of

a population (*survival of the fittest*). In our case, a population consists of generated individuals which are candidate heuristics. The runtime performances of candidate heuristics are tested on past sets of historical dynamic constraints on top of a graph coloring problem. Our fitness function in Formula 3.2 minimizes the total runtime to find solutions for all CSPs (precoloring extension problems based on same graph coloring problem and different dynamic constraint sets) in a cluster.

$$min(\sum_{i=1}^{n} runtime(Solve(CSP_{\mathrm{i}})) \tag{3.2}$$

In our working example, learned heuristics combinations for cluster-1 are CLH_{var1} and CLH_{val1} whereas for cluster-2 they are CLH_{var2} and CLH_{val2} as shown in Table 3.7.

Table 3.7.: CLUSTER AND LEARN heuristics (*CLH*)

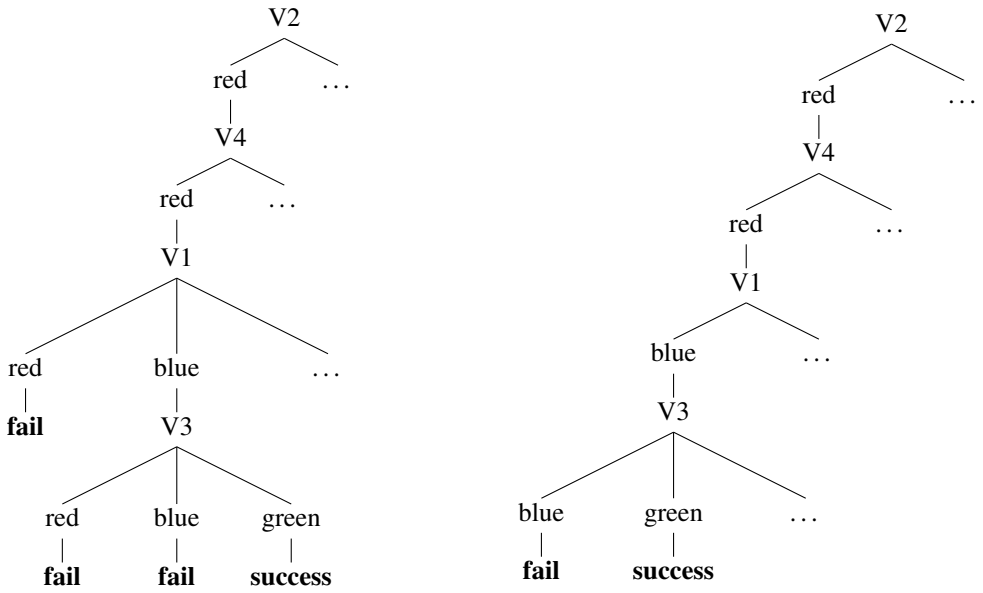
Cluster-1: $\{C_{\mathrm{REQ}_1}, C_{\mathrm{REQ}_3}, C_{\mathrm{REQ}_5}\}$	Cluster-2: $\{C_{\mathrm{REQ}_2}, C_{\mathrm{REQ}_4}, C_{\mathrm{REQ}_6}\}$
CLH_{var1}: $\mathbf{V2}, \mathbf{V4}, \mathbf{V1}, \mathbf{V3}$	CLH_{var2}: $\mathbf{V2}, \mathbf{V1}, \mathbf{V4}, \mathbf{V3}$
CLH_{val1}: $\mathbf{V1} : \{blue, red, green\}$, $\mathbf{V2} : \{red, blue, green\}$, $\mathbf{V3} : \{blue, green, red\}$, $\mathbf{V4} : \{red, blue, green\}$	CLH_{val2}: $\mathbf{V1} : \{red, blue, green\}$, $\mathbf{V2} : \{red, blue, green\}$, $\mathbf{V3} : \{blue, green, red\}$, $\mathbf{V4} : \{blue, green, red\}$

Solving with *CLH*

CLH heuristics are learned for each cluster in the offline phase using sets of historical dynamic constraints (past C_{REQ}s). After this offline phase, a CSP solver can solve new precoloring extension problems (based on the same graph coloring problem with historical problems) in the online phase with the help of these learned heuristics. First, we find the most similar cluster to the new precoloring extension problem's dynamic constraints. Then we directly employ the heuristics of the most similar cluster for solving the new precoloring extension problem.

In our working example, CSP_{pre} has the dynamic constraints set which holds one constraint $C_{\mathrm{REQ}} = \{c_6 : \mathbf{V1} = blue\}$. First, we find the closest cluster to CSP_{pre} using the euclidean distance Formula 4.3. The closest cluster that we find is $Cluster-1$ and we apply the $Cluster-1$'s variable and value ordering heuristics to solve CSP_{pre}.

In the following two backtrack search trees (see Fig. 3.3), we show the CSP solver's search space (so the number of consistency checks which increases parallel with the runtime), first using only CLH_{var1} and then combining CLH_{var1} with CLH_{val1}. As shown in the first backtracking search tree in Fig. 3.3-(a), when CLH_{var1} is applied to solve the CSP_{we}, map coloring problem's search is completed with seven consistency checks. The found solution is {**V1**=blue, **V2**=red, **V3**=green, **V4**=red}. The number of applied consistency checks is same with the search without heuristics (see Fig. 3.2). Thus, for CSP_{pre}, variable ordering heuristics did not help to decrease the search space. However, it decreases the search space of $CSP_{\mathrm{C_{REQ2}}}$.



(a) Only Variable Ordering: CLH_{var1}

(b) Variable and Value Ordering: $CLH_{var1\&val1}$

Figure 3.3.: Depth-first backtrack search trees to solve CSP_{pre} with heuristics

As shown in the second backtracking search tree in Fig. 3.3-(b), when CLH_{var1} is applied together with and CLH_{val1}, search results in five nodes (consistency checks) which is better than no heuristics and also using only CLH_{var1}. Due to the applied variable ordering, the first row of the tree is composed of the possibilities of **V2**, then respectively **V4**, **V1** and **V3**. The order of values tried for each variable is based on the value ordering heuristics. For example, for **V3** the first tried value is *blue* whereas for **V2** it is *red*. Since the search space is the smallest, the execution time of the CSP solver is also the shortest when cluster specific variable and value ordering heuristics are applied together.

3.2.3. Experimental Evaluation

In this section we perform an empirical evaluation of the effect of variable and value ordering heuristics on runtime performance of CSP solvers during solving precoloring extension problems. All cases include heuristics combinations (variable and value ordering) which are used in depth-first search and at every node they enforce AC. Our experiments have been conducted on an Intel Core i5-5200U PC, 2.20 GHz processor, 8 GB RAM, and 64 bit Windows 7 Operating System and Java Run-time Environment 1.8.0.

Datasets. We have tested our proposed approach CLUSTER AND LEARN using *Precoloring Extension Problems Benchmark in Choco (PEPBC)*[1] which consists of 150 precoloring extension problems (150 sets of dynamic constraints based on the same graph-coloring problem). 100 of them are used in heuristics learning and the rest 50 are used in testing. In this benchmark, precoloring extension problems consist of 100 vertices and 5 dynamic constraints.

Settings. In our CLUSTER AND LEARN implementation, we have used k-means clustering algorithm with its default parameters (100 iterations, 4 clusters, a default random generator and using the Euclidean distance). For the learning phase, we have implemented a genetic algorithm with the population size:

[1] https://github.com/CSPHeuristix/PEPBC

Table 3.8.: Runtime (in *millisecond*) of choco-solver with variable and value ordering heuristics.

TEST RESULTS - 1st Part

	Value Ordering Heuristics			
Variable Ordering Heuristics	Int-Domain-Min	Int-Domain-Max	Int Domain-Median	$\boldsymbol{CLH}_{\text{val}}$
Smallest	17.506	18.071	18.222	17.730
Largest	16.620	18.050	17.415	18.764
ActivityBased	25.700	28.348	27.189	29.410
FirstFail	5.253	5.447	5.531	5.903
AntiFirstFail	10.791	11.207	11.162	12.153
Cyclic	16.389	17.806	17.327	0.251
MaxRegret	15.265	18.553	17.148	24.068
Occurance	15.549	17.515	17.563	16.384
Input Order	0.188	0.198	0.383	0.203
DomOverDweg	12.393	13.680	13.026	8.163
ImpactBased	2,499.707	2,853.424	2,813.963	3,086.336
GeneralizedMinDomain	16.656	17.203	17.516	12.179
Random	16.401	16.930	17.343	9.666
$\boldsymbol{CLH}_{\text{var}}$	0.013	0.016	0.015	**0.008**

TEST RESULTS - 2nd Part

	Value Ordering Heuristics			
Variable Ordering Heuristics	Int Domain-Middle	Int Domain-Random	Int Domain-RandomBound	$\boldsymbol{CLH}_{\text{val}}$
Smallest	19.040	18.136	18.643	17.730
Largest	17.096	18.356	18.795	18.764
ActivityBased	26.543	28.189	27.367	29.410
FirstFail	5.750	5.592	5.490	5.903
AntiFirstFail	11.905	11.593	11.288	12.153
Cyclic	17.142	18.382	18.153	0.251
MaxRegret	19.044	18.874	19.252	24.068
Occurance	17.380	18.477	18.003	16.384
InputOrder	0.186	0.419	0.229	0.203
DomOverDweg	13.180	13.737	12.881	8.163
ImpactBased	2,859.879	2,906.035	2,888.508	3,086.336
GeneralizedMinDomain	18.288	17.898	17.535	12.179
Random	17.697	17.718	17.169	9.666
$\boldsymbol{CLH}_{\text{var}}$	0.015	0.010	0.013	**0.008**

100, gene size: 100 (domain size of the variables) and target fitness value: 0.050*milliseconds* (and other parameters are set to default as uniform rate: 0.5, mutation rate: 0.015, tournament size: 5, elitism: *true*).
Compared Methods. We have compared our proposed variable and value ordering heuristics with the built-in variable and value ordering heuristics in Choco-solver (Prud'homme et al., 2016; Jussien et al., 2008). Choco is a problem modeler and a constraint programming solver available as a Java library. Moreover, its architecture allows the plugin of other (non CP based) solvers.

In Choco-solver, there are 13 built-in variable ordering heuristics and six built-in value ordering heuristics for integer domains. These variable ordering heuristics are the following; *ActivityBased* (Michel and Van Hentenryck, 2012), *AntiFirstFail*, *Cyclic* (a cyclic variable selector which iterates over variables according to lexicographic ordering in a cyclic manner, loop back to the first variable), *DomOverWDeg* (Boussemart et al., 2004), *FirstFail*, *GeneralizedMinDomVarSelector* (first fail variable selector generalized to all variables), *ImpactBased* (Refalo, 2004), *InputOrder*, *Largest*, *MaxRegret*, *Occurrence*, *Random*, *Smallest*. The value ordering heuristics are the following; *IntDomainMax* (selects the variable upper bound),, *IntDomainMedian* (selects the median value in the variable domain), *IntDomainMiddle* (selects the value in the variable domain closest to the mean of its current bounds), *IntDomainMin* (selects the variable lower bound), *IntDomainRandom* (selects randomly a value in the variable domain), *IntDomainRandomBound* (selects randomly between the lower and the upper bound of the variable).

Results. In our tests, we have observed that our approach (when we combine CLH_{var} and CLH_{val}) finds the first solution in the shortest runtime compared to other combinations of variable and value ordering heuristics. For example, when we combine the variable ordering heuristic *AntiFirstFail* with the value ordering heuristic *IntDomainMiddle*, the runtime of the CSP solver to find the first solution is 11.905 milliseconds which is around 1500 times slower than $CLH_{var\&val}$ (0.008 milliseconds).

3.2.4. Conclusions

A graph coloring is an assignment of colors, to the vertices of a graph such that no two adjacent vertices share the same color. However, the real-life problem does not appear in such a pure assignment form, there are additional constraints that have to be satisfied. Such problems can be defined as *precoloring extension problems*.

In this section, our motivation was to improve runtime performance of CSP solvers to solve *precoloring extension problems*. To that aim, we have proposed a novel variable and value ordering heuristics learning approach CLUSTER AND LEARN. CLUSTER AND LEARN clusters sets of historical dynamic constraints and applies a supervised learning to learn cluster specific variable and value ordering heuristics in an offline phase. These learned heuristics are used in an online phase for solving new precoloring extension problems. The most similar cluster's heuristics to a new precoloring extension problem are used to solve it. We have employed clustering to learn an optimized heuristic for all precoloring extension problem in a cluster. Therefore, for another similar precoloring extension problem, it is expected to work well.

The key point is the new precoloring problems and the ones in historical sets should be based on same graph coloring problem. The only difference between them are the dynamic constraints on top of the same graph coloring problem. Based on our experiments, we observed that CLUSTER AND LEARN is outperforming all compared variable and value ordering heuristics combinations in terms of runtime performance.

However, this model is over-simplified and only aims at improving runtime performance of solving precoloring extension problems based on learned heuristics from similar historical precoloring extension problems. To be more realistic, some additional situations may be taken into account. For example, dynamic constraints (of a precoloring extension problem) can be inconsistent with constraints of its graph coloring problem. This could be handled by resolving inconsistencies automatically by employing diagnosis algorithms. There are consistency-based direct diagnosis algorithms (Felfernig et al., 2018c) available which specifically focus on the dynamic constraints to resolve the inconsistency. Note that these methods

diagnose the inconsistencies in dynamic constraints, so that knowledge base constraints are assumed to be consistent. If knowledge base constraints also include inconsistencies, the inconsistency problem should be globally diagnosed but optimality may be inefficient because of the huge size of constraints, so alternative local diagnosis approaches (Erdeniz et al., 2017) could be more efficient.

Future work includes the use of more efficient clustering and learning techniques to learn better heuristics, the design of criteria to dynamically select the closest clusters during solving with heuristics (the online phase), the addition of more flexibility to the model by taking into account features of sets of historical dynamic constraints. A further considered refinement of the model would be to allow the adaptation of learned heuristics according to the active precoloring extension problem.

3.3. Cluster-Specific Heuristics for Direct Diagnosis

When constraints of a *CSP* are inconsistent, no solution can be found. Therefore, diagnosis (Bakker et al., 1993) is required to find at least one solution for this inconsistent *CSP*. A diagnosis algorithm can be evaluated in terms of runtime performance and diagnosis quality. Runtime performance is very crucial for the real-time scenarios, in order to provide a solution in an acceptable time period. Diagnosis quality can be determined in terms of precision and minimality. A minimal diagnosis includes the minimum number of constraints, so the originality of the problem is kept in the diagnosed problem. The minimality of a diagnosis algorithm can be represented in the [0..1] interval. When the diagnosis algorithm provides always the minimal diagnosis, its *minimality=1*. The precision of a diagnosis states its acceptance rate by users. This acceptance rate is mapped to the [0..1] interval. Therefore, *precision=1* means the diagnosis algorithm provides fully acceptable results by users.

Many different approaches to provide efficient solutions for diagnosis problems are proposed (Nica et al., 2013). One approach (Wotawa, 2001) focuses on improvements of HSDAG. Another approach (Wang et al., 2009) uses pre-determined set of conflicts based on binary decision diagrams. In diagnosis problem instances where the number of minimal diagnoses and their cardinality is high, the generation of a set of minimum cardinality diagnoses is unfeasible with the standard conflict-based approach. An alternative approach to solve this issue is *direct (sequential) diagnosis* (Shchekotykhin et al., 2014) which determines diagnoses by executing a series of queries. These queries check the consistency of the constraint set without the need to identify the corresponding conflict sets.

When diagnoses have to be provided in real-time, response times should be less than a few seconds (Card et al., 1991). For example, in communication networks, efficient diagnosis is crucial to retain the quality of service (Nica et al., 2014). To satisfy these real-time diagnosis requirements, a direct diagnosis algorithm FLEXDIAG (Felfernig et al., 2018c) uses a parameter (m) that helps to systematically reduce the number of consistency checks (so the runtime) but in the same time the minimality of diagnoses is deteriorated. Therefore, in FLEXDIAG, there is a tradeoff between the diagnosis quality (in terms of minimality and precision) and the runtime performance. When the runtime performance (# of diagnoses per second) increases, the quality of diagnosis may decrease.

This section introduces *a novel heuristics learning method* for solving *the quality-runtime performance trade off problem* of direct diagnosis algorithms. Our approach (LEARNORDER) learns constraint ordering heuristics (search strategies) (Khalil et al., 2017) to improve runtime performance and quality of diagnosis at the same time. It is used in combination with a direct diagnosis algorithm. For evaluations, we used a real dataset collected in one of our user studies and compared performance of a direct diagnosis algorithm (FLEXDIAG) with and without LEARNORDER heuristics. Our experiments show that, LEARNORDER significantly improves the performance of the direct diagnosis algorithm in terms of precision, runtime, and minimality.

3.3.1. Problem Definition

The following (simplified) assortment of digital cameras and the user requirements of *Lisa* (see Table 3.24) will serve as a working example to demonstrate how LEARNORDER works. It is formed as a configuration task (Tsang, 1993) on the basis of Definition 2.1.1.

The example shows that a knowledge base can have more than one conflict. In such cases we can help users to resolve the conflicts with diagnosis. A diagnosis Δ is a set of constraints. The removal of the set Δ from C_R leads to a consistent knowledge base, formally described in Definition 2.1.5. The example configuration knowledge base contains two minimal diagnoses. The removal of the set $\Delta_1 = \{c2, c9\}$ or $\Delta_2 = \{c3, c9\}$ leads to a consistent configuration knowledge base.

3.3.2. Proposed Method

The order of constraints effect the size of the search tree and the search result of a direct diagnosis algorithm. The main idea of using LEARNORDER heuristics is reordering constraints before executing the direct diagnosis algorithm in order to improve the search performance.

Calculating LEARNORDER heuristics are all offline (clustering and learning) steps which are applied on past inconsistent *REQs* (see Table 3.25) in order to obtain LEARNORDER heuristics. After learning LEARNORDER heuristics based on past inconsistent *REQs*, any new inconsistent *REQ*s can be diagnosed by a direct diagnosis algorithm with the guidance of LEARNORDER heuristics.

Historical transactions, which contain inconsistent user requirement sets and also the corresponding selected products, are appropriate to be used in the LEARNORDER heuristics learning.

Table 3.10.: Inconsistent Historical Transactions

	Alice	Bob	Tom	Ray	Joe	Eva
eff. res.	20.9	6.1	20.9	20.9	6.2	6.2
display	3.5	2.2	2.5	2.5	1.8	1.8
touch	yes	no	yes	yes	no	no
wifi	yes	no	yes	yes	no	no
nfc	no	yes	no	no	yes	yes
gps	yes	yes	yes	yes	yes	no
vid. res.	4K-UHD	No-Video	4K-UHD	4K-UHD	4K-UHD	Full-HD
zoom	3.0	5.8	3.0	5.8	5.8	3.0
weight	560	700	475	475	860	860
price	469	189	469	659	189	469
P	P1	P3	P1	P4	P5	P4

Based on the historical transactions in Table 3.10, we know that, the product $P1$ is selected (purchased) by *Alice* even though his initial requirement set is inconsistent with the product table (there is no solution available based on Alice's initial requirements). This means, Alice has renounced her last two requirements $weight$=560g and $price$=469 and purchased the product $P1$ which has $weight$=475g and $price$=659. Therefore, $\Delta_{\text{Alice}} = \{weight, price\}$ is a diagnosis for REQ_{Alice}. When we remove the constraints of the diagnosis from the inconsistent requirement set, the diagnosed requirement set becomes $REQ'_{\text{Alice}} = \{effectiveResolution = 20.9Megapixel, display = 3.5inches, touch = yes, wifi = yes, nfc = no, gps = yes,$

Table 3.9.: An example for a camera configuration problem

V	$\{effectiveResolution, display, touch, wifi, nfc, gps, videoResolution, zoom, weight, price\}$
D	• $dom(effectiveResolution)$= {6.1Megapixel, 6.2Megapixel, 20.9Megapixel}, • $dom(display)$= {1.8inches, 2.2inches, 2.5inches, 3.5inches}, • $dom(touch)$={yes, no}, • $dom(wifi)$={yes, no}, • $dom(nfc)$={yes, no}, • $dom(gps)$={yes, no}, • $dom(videoResolution)$= {4K-UHD, Full-HD, No-Video}, • $dom(zoom)$={3.0x, 5.8x, 7.8x}, • $dom(weight)$= {475g, 560g, 700g, 860g, 1405g}, • $dom(price)$={€189, €469, €659, €2329, €5219}
C_{KB}	$c1$:{P1∨ P2∨ P3∨ P4∨ P5} where; • P1: { effectiveResolution=20.9Megapixel∧ display=3.5inches∧ touch=yes ∧ wifi=yes ∧ nfc=no ∧ gps=yes ∧ videoResolution=4K-UHD ∧ zoom=3.0x∧ weight=475g ∧ price=€659}, • P2: { effectiveResolution=6.1Megapixel∧ display=2.5inches∧ touch=yes ∧ wifi=yes ∧ nfc=no ∧ gps=yes ∧ videoResolution=4K-UHD ∧ zoom=3.0x∧ weight=475g ∧ price=€659}, • P3: { effectiveResolution=6.1Megapixel∧ display=2.2inches∧ touch=no ∧ wifi=no ∧ nfc=no ∧ gps=no ∧ videoResolution=no-video ∧ zoom=7.8x∧ weight=700g∧ price=€189}, • P4: { effectiveResolution=6.2Megapixel∧ display=1.8inches∧ touch=no ∧ wifi=no ∧ nfc=no ∧ gps=no ∧ videoResolution=4K-UHD ∧ zoom=5.8x∧ weight=860g∧ price=€2329}, • P5: { effectiveResolution=6.2Megapixel∧ display=1.8inches∧ touch=no ∧ wifi=no ∧ nfc=no ∧ gps=yes ∧ videoResolution=Full-HD ∧ zoom=3.0x∧ weight=560g ∧ price=€469}
REQ_{Lisa}	$c2$: effectiveResolution=20.9Megapixel $c3$: display=2.5inches $c4$: touch=yes $c5$: wifi=yes $c6$: nfc=no $c7$: gps=yes $c8$: videoResolution=4K-UHD $c9$: zoom=5.8x $c10$: weight=475g $c11$: price=€659

videoResolution=$4K-UHD$, *zoom*=3.0x}. Based on the diagnosed requirement set, the solution set is {$P1$} which includes the purchased products by Alice ($P1$). This means, the diagnosis is correct.

Clustering

We cluster past inconsistent *REQs* using *k-means* clustering (Jain, 2010). *K-means clustering* generates k clusters where it minimizes the sum of squares of distances between cluster elements and the centroids (mean value of cluster elements) of their corresponding clusters. In Formula 3.3, k is the number of clusters, κ is the set of k clusters, μ_i is the centroid of a cluster $\kappa_i \in \kappa$, and x is a cluster element in κ_i.

$$min \sum_{i=1}^{k} \sum_{x \in \kappa_i} \|Distance(x, \mu_i)\|^2 \tag{3.3}$$

To equalize the effects (Visalakshi and Thangavel, 2009) of all variables in a *REQ* during k-means clustering, we have employed *Min-Max Normalization* before clustering according to Formula 3.4.

$$v_{i\_norm} = \frac{v_i - dom(v_i)_{min}}{dom(v_i)_{max} - dom(v_i)_{min}} \tag{3.4}$$

After k-means clustering is applied with the parameter *number of clusters (k)* = 2, two clusters (κ_1 and κ_2) of *REQs* are obtained as shown in Table 3.11. We used k = 2 (not a higher value) to demonstrate our example in an understandable way.

Table 3.11.: Clusters of User Requirements

	cluster elements	centroid (μ)
κ_1	REQ_{Alice}, REQ_{Tom}, REQ_{Ray}	μ_1: {1, 0.60, 1, 1, 0, 1, 1, 0.19, 0.03, 0.63}
κ_2	REQ_{Bob}, REQ_{Joe}, REQ_{Eva}	μ_2: {0, 0.07, 0, 0, 1, 0.66, 0.5, 0.38, 0.35, 0.01}

Learning

After clustering is completed, a genetic algorithm based supervised learning (Venturini, 1993) is employed to select the best performing individual (the generated constraint orderings) by testing the performances. For each cluster (κ_i) four different constraint ordering heuristics are calculated for runtime (τ), precision (π), minimality (Φ) and the combination of them (α) (Table 3.12).

Table 3.12.: Constraint ordering heuristics

κ_1	LEARNORDER$_1\tau$:	{c9, c3, c2, c11, c4, c5, c7, c8, c6, c10}
	LEARNORDER$_1\pi$:	{c2, c9, c3, c10, c11, c7, c8, c4, c6, c5}
	LEARNORDER$_1\Phi$:	{c2, c3, c9, c11, c4, c5, c7, c8, c6, c10}
	LEARNORDER$_1\alpha$:	{c9, c2, c3, c11, c4, c5, c7, c8, c6, c10}
κ_2	LEARNORDER$_2\tau$:	{c6, c9, c7, c11, c10, c5, c2, c8, c4, c3}
	LEARNORDER$_2\pi$:	{c9, c11, c10, c6, c7, c5, c3, c2, c4, c8}
	LEARNORDER$_2\Phi$:	{c6, c7, c9, c11, c10, c5, c2, c8, c4, c3}
	LEARNORDER$_2\alpha$:	{c11, c9, c6, c7, c10, c5, c2, c8, c4, c3}

Runtime (τ). Runtime, as one of our performance indicators, represents the time spent by the direct diagnosis algorithm to calculate a diagnosis. To improve the runtime performance of direct diagnosis, we calculate LEARNORDER heuristics which minimize the total runtime of the diagnosis algorithm. The runtime performances of candidate heuristics are tested on past n inconsistent REQs in cluster κ_x as shown in Formula 3.5.

$$\tau = \frac{1}{n} \times \sum_{i=1}^{n} runtime(Diagnose(REQ_i)) \tag{3.5}$$

The runtime of our approach is calculated based on the steps ın the online phase.

Precision (π). Based on the diagnosed REQ, a CSP solver can find at least one solution. If the solution set includes the product which is purchased by the user (see last row of Table 3.25), then we say that the prediction is correct (precision is 1), otherwise it is not a correct prediction (precision is 0). The average precisions of the diagnoses in κ_x is in between 0..1. To improve the diagnosis precision of direct diagnosis, we calculate LEARNORDER heuristics which maximize the precision of past inconsistent REQs in cluster κ_x as shown in Formula 3.6.

$$\pi = \frac{\#(\text{correct predictions})}{\#(\text{predictions})} \tag{3.6}$$

The precision of our approach is calculated by comparing the solution set after diagnosis with the selected product information by the corresponding user (see Table 3.10).

Minimality (Φ). Diagnosis quality can be measured in terms of the degree of minimality of a diagnosis. To improve the minimality of direct diagnosis, we calculate LEARNORDER heuristics which maximize the diagnosis minimality of n past inconsistent REQs in cluster κ_x as shown in Formula 3.7 where $|\Delta|$ represents the cardinality (number of constraints) of Δ. Φ is tried to be maximized by the learning algorithm.

$$\Phi = \frac{1}{n} \times \sum_{i=1}^{n} \frac{|\Delta_{\min}|}{|\Delta_i|} \tag{3.7}$$

The minimality of our approach is calculated by comparing the cardinality of the calculated diagnosis by our approach and FLEXDIAG-m=1 since it is guaranteed that FLEXDIAG-m=1 finds always the minimal diagnosis (Felfernig et al., 2018c).

Combined-Performance (α). A combination of all three performance indicators (π, τ, and Φ) are used to calculate another constraint ordering heuristic based on combined-performance (α). The learning

algorithm tries to maximize the fitness value (α) to improve all three performances of a cluster κ_x (of past inconsistent *REQ*s) at the same time as shown in Formula 3.8.

$$\alpha = \frac{1}{\tau} \times \pi \times \Phi \tag{3.8}$$

Direct Diagnosis with LEARNORDER

LEARNORDER heuristics are pre-calculated in the offline phase using historical transactions (past inconsistent *REQ*s). After this offline phase, the direct diagnosis algorithm can solve the diagnosis tasks in the online phase with the help of the pre-calculated LEARNORDER heuristics.

In order to select the most appropriate LEARNORDER heuristics to solve a new diagnosis task, first we *find the closest cluster* to the new inconsistent *REQ* and then *reorder its constraints* according to the LEARNORDER heuristics. After that, the direct diagnosis algorithm searches for a diagnosis based on the *reordered constraints* of the new inconsistent *REQ*.

Finding the closest cluster. When a new inconsistent *REQ* is needed to be diagnosed, first we find the distances between clusters and the new *REQ* using Euclidean Distance Formula 3.15.

$$\sqrt{\sum_{j=1}^{n} \|REQ_{\text{Lisa}}\_norm_{\text{j}} - \mu_{\text{ij}}\|^2} \tag{3.9}$$

In our working example, where the normalized values of REQ_{Lisa} is *REQ$_{Lisa}$_norm={1, 0.41, 1, 1, 0, 1, 0, 0.58, 0, 0, 0.09}*, the closest cluster (the shortest distance) to REQ_{Lisa}_norm is identified as κ_1 (see Table 3.13).

Table 3.13.: Calculated euclidean distances

Distance(μ_1,REQ_{Lisa}_norm)	**1.21**
Distance(μ_2,REQ_{Lisa}_norm)	**2.15**

Reordering constraints. Learned heuristics (see Table 3.12) of the closest cluster (shortest distance in Table 3.13) is used to reorder the new inconsistent *REQ*.

To demonstrate on the working example, we use the combined-performance heuristic (LEARNORDER$_1\alpha$ from Table 3.12). Using the constraint ordering heuristic LEARNORDER$_1\alpha$, constraints of REQ_{Lisa} are ordered as in Table 3.14.

Table 3.14.: Reordered user requirements

$REQ_{\text{Lisa}}\_reordered$: {$c9$, $c2$, $c3$, $c11$, $c4$, $c5$, $c7$, $c8$, $c6$, $c10$}

Direct Diagnosis. Using the reordered constraints of the inconsistent *REQ*, the direct diagnosis algorithm searches for a diagnosis. The learned constraint ordering heuristic helps the direct diagnosis algorithm in terms of the target performance criterion. LEARNORDER-τ guides the diagnostic search to decrease the runtime to find a diagnosis. LEARNORDER-π guides the diagnostic search to find a diagnosis

with a high precision. LEARNORDER-Φ guides the diagnostic search to find a diagnosis with a high minimality. Finally, LEARNORDER-α guides the diagnostic search to find a diagnosis in a short runtime with a high precision and minimality.

FLEXDIAG, as a direct diagnosis algorithm, searches directly for a diagnosis out of a given inconsistent constraints set. Its search idea is based on a recursively applied *divide-and-conquer* mechanism, so the order of the input constraints has an high impact on searching. FLEXDIAG starts searching for a diagnosis from the left half to right half by recursively dividing the set of constraints into two.

When the input of FLEXDIAG is the set of inconsistent requirements REQ_{Lisa}: {$c2$, $c3$, $c4$, $c5$, $c6$, $c7$, $c8$, $c9$, $c10$, $c11$}, the first consistency check is applied to the first half $REQ_{\text{Lisa\_leftHalf}}$: {$c2$, $c3$, $c4$, $c5$, $c6$} according to the the product table (C_{KB}).

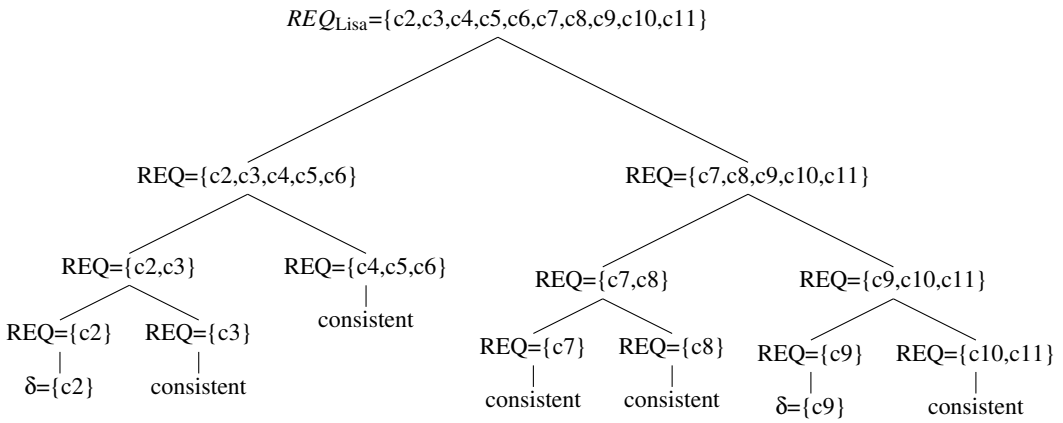
If the input constraint set is consistent with the product table (C_{KB}), the search stops for this recursion and continues from the next recursion. If not, the search continues till finding a *conflicting* subset of REQ with cardinality m. The algorithm continues dividing the inconsistent constraint set into two, till the *conflicting* subset of REQ has maximum m constraints.

When the input of FLEXDIAG is the reordered set of inconsistent requirements $REQ_{\text{Lisa\_}reordered}$: {$c9$, $c2$, $c3$, $c11$, $c4$, $c5$, $c7$, $c8$, $c6$, $c10$} (according to LEARNORDER$_1$α, see Table 3.14), the number of consistency checks and the diagnosis result may vary.

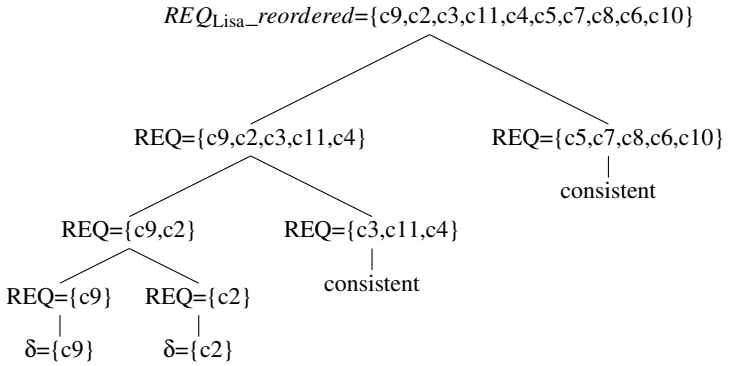
We can not measure the precision of a diagnosis if we do not know the user's decision. Therefore, in our working example, we can only measure the minimality as a diagnosis quality criterion. On the other hand, to measure the runtime performance we can either use the time spent or the number of consistency checks to find a diagnosis. In our working example, we can demonstrate the number of consistency checks. Consequently, to evaluate the performances on the demonstrated working example we know that, for the number of consistency check *the less is better*, and for the minimality *the higher is better*.

We demonstrate the quality (in terms of minimality) and the runtime performance (in terms of the number of consistency checks) effects of using LEARNORDER heuristics for diagnosing the inconsistent user requirement set REQ_{Lisa} based on the direct diagnosis algorithm FLEXDIAG (m=1 and m=2) as follows:

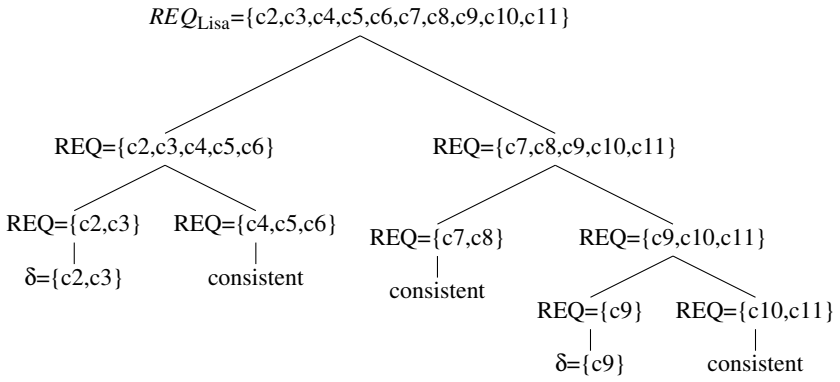
(1) FLEXDIAG (m=1) without heuristics: Δ = {$c2, c9$} (**# of consistency checks = 12**, **minimality = 1**).



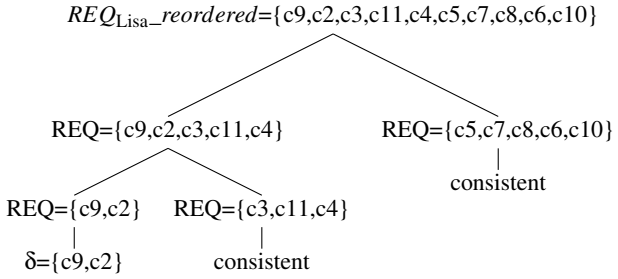
(2) FLEXDIAG (m=1) with LEARNORDER$_1\alpha$: Δ = {$c2, c9$} (**# of consistency checks = 6**, **minimality = 1**).



(3) FLEXDIAG (m=2) without heuristics: Δ = {$c2, c3, c9$} (**# of consistency checks = 8**, **minimality = 0.66**).



(4) FLEXDIAG (m=2) with LEARNORDER$_1\alpha$: Δ = {$c2, c9$} (**# of consistency checks = 4**, **minimality = 1**).



3.3.3. Experimental Evaluation

Especially in order to evaluate the precision quality of our approach, we have required a real-world dataset which contains inconsistent user requirements and the purchased products by the corresponding users. We have used a publicly available real-world purchase data (*CameraKB_ConfigurationDataset*) from *Configuration/Diagnosis Benchmarks in Choco (CDBC)* [1] which includes historical transactions based on the real product catalog from a digital camera online shop.

This dataset contains 264 transactions which contains a list of user requirements and corresponding

[1] https://github.com/CSPHeuristix/CDBC

user's preferred product IDs from the product catalog. Out of these 264 inconsistent *REQ*s, we have used 200 inconsistent *REQ*s as historical transactions for learning heuristics and the rest 64 inconsistent *REQ*s for evaluations.

In our experiments[2], for clustering *REQ*s, we used the *k-means clustering* algorithm of the *Java Machine Learning Library*[3] with k=50 (number of clusters), 100 iterations, a default random generator and using the Euclidean distance. For the supervised learning, we used a genetic algorithm[4] with the population size:100, gene size:10 (size of a *REQ*)[5]. Consistency checks in our implementation are done by a CSP solver (choco-solver[6]) executions.

We have analyzed the major aspects *runtime performance* and *diagnosis quality* (minimality and precision) on the basis of the m parameter of FLEXDIAG. As mentioned, FLEXDIAG-m=1 gives always the minimal diagnosis whereas FLEXDIAG-m>1 does not guarantee to find always a minimal diagnosis.

We have compared the performances of our four type LEARNORDER heuristics when combined with FLEXDIAG and FLEXDIAG without any heuristics. As observed, FLEXDIAG performs best in terms of precision when it is combined with LEARNORDER-π heuristics, in terms of runtime performance when it is combined with LEARNORDER-τ and in terms of minimality when it is combined with LEARNORDER-Φ.

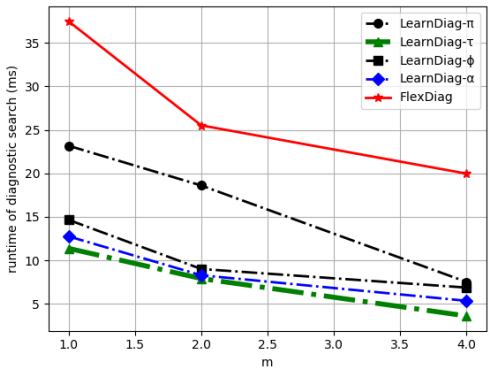
[2]Our experiments are executed on a computer with an Intel Core i5-5200U, 2.20 GHz processor, 8 GB RAM and 64 bit Windows 7 Operating System and Java Run-time Environment 1.8.0.

[3]http://java-ml.sourceforge.net/

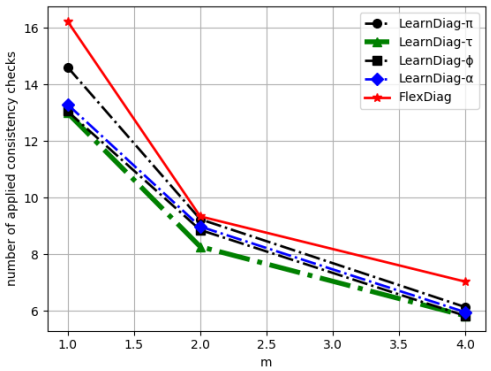
[4]http://www.theprojectspot.com/

[5]Other parameters are set to default as uniformRate: 0.5, mutationRate: 0.015, tournamentSize: 5, elitism: true

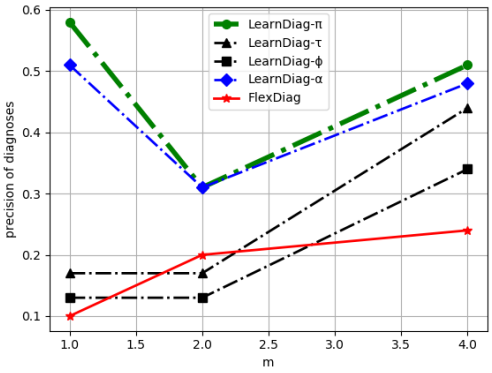
[6]http://www.choco-solver.org/



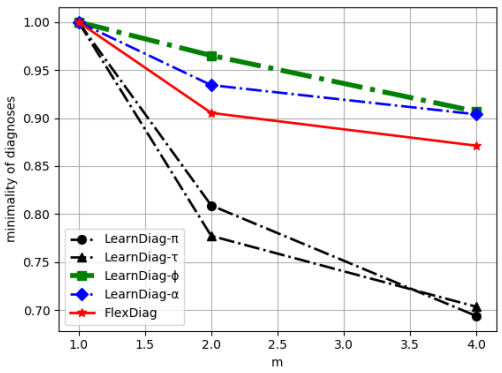
(b) Runtime



(b) Consistency Checks



(c) Precision



(d) Minimality

Figure 3.4.: Performance results of LEARNORDER.

All the best performing heuristics are shown with green lines in Fig. 3.4 whereas the second best performing heuristics are shown with blue lines. According to each performance criterion, the type of best performing heuristics vary but the second best performing heuristics are always LEARNORDER-α. Consequently, we have observed that using LEARNORDER-α heuristics with FLEXDIAG is a solution for solving *the quality-runtime performance tradeoff problem* of FLEXDIAG. As shown in Fig. 3.4, FLEXDIAG with LEARNORDER-α always gives better performance results compared to FLEXDIAG with no heuristics.

3.3.4. Conclusions

We have proposed an out-performing constraint ordering heuristics LEARNORDER for direct diagnosis algorithms to solve *the quality-runtime performance tradeoff problem*. In order to evaluate the performance effects our approach, we have tested each LEARNORDER heuristic in combination with the direct diagnosis algorithm FLEXDIAG. According to our experimental results, FLEXDIAG with LEARNORDER-α solves *the quality-runtime performance tradeoff problem* by improving runtime performance and quality (minimality, precision) of diagnosis at the same time. Besides, FLEXDIAG performs best in terms of precision using LEARNORDER-π, in terms of runtime performance using LEARNORDER-τ and in terms of minimality using LEARNORDER-Φ. Future work will include the evaluation of different alternative datasets (e.g. more complex, industrial benchmarks) w.r.t. their impact on performance indicators.

3.4. Matrix Factorization based Heuristics for Constraint Solving

Due to the new century's information overload problem, recommender systems (Ricci et al., 2011) are becoming more important in various domains. Knowledge-based recommendation (Burke, 2002) attempts to suggest objects based on inferences about a user's needs and preferences.

One type of knowledge-based recommender systems are *constraint-based recommenders* (Zanker et al., 2007) in which the recommendation tasks can be composed of many variables and constraints. In these settings, achieving an acceptable runtime performance of recommending suitable items to users can quickly become a very challenging. Therefore, search should be guided by intelligent search strategies, so-called *heuristics* (Groër et al., 2010). Among the existing heuristics, variable and value ordering heuristics are widely adopted in the context of constraint-based recommenders.

Determining accurate heuristics for constraint-based recommenders is challenging due to two reasons:

- *Complexity.* Solving a constraint-based recommendation task on a finite domain is an NP-complete problem with respect to the domain size and the complexity of the given constraints.
- *Accuracy.* A recommendation result should have a high probability to be *accepted by the user*.

We propose a novel search heuristic-based approach for constraint-based recommenders to overcome the aforementioned two challenges. Our approach uses (historical) transactions of the configuration system to calculate value ordering heuristics which are specific to the requirements of a user. Thereafter, these heuristics are used to solve the recommendation task with a high *prediction accuracy* in a *short runtime*.

State-of-the-art

Constraint based recommenders use constraint solving algorithms and heuristics to improve the overall runtime performance. However, these methods do not take the performance criterion *accuracy* into account. There exist some approaches which are based on collaborative filtering that aim to improve the accuracy of recommenders but lacks of runtime performance improvement at the same time.

Ardissono et al. (Ardissono et al., 2002) presented a framework for the personalized configuration in business-oriented domain. They classified the system users according to their expertise in the domain. The personalization of the interaction and dialogues are adapted according to this user modeling. They have applied this framework to a telecommunication switches domain. They experienced successful results (speed up in configuration process) in user configurations. However, this approach uses explicit personalization rules whereas we learn search heuristics from historical transactions.

Felfernig et al. (Felfernig and Burke, 2008) presented an overview of several techniques to solve constraint based recommendation tasks. An open research issue in this particular context is to efficiently guide the solution search towards the relevant items.

Sandvig et. al (Sandvig et al., 2007) adapted association rule mining to collaborative filtering in order to discover valuable patterns between items that have similar ratings. Using item ratings such as *like* or *dislike*, the authors could achieve a good level of accuracy in the results. Although the achieved accuracy comes very close to the compared method *k-nearest neighbor*, the paper does not cover the performance aspect in more detail.

Zanker (Zanker, 2008) proposed a system that learns rule-based preferences from successful interactions in historic transaction data. The author uses collaborative filtering to derive preferences from a user's nearest neighbors. The paper demonstrates that this approach can improve the overall prediction accuracy of the recommender system. However, no focus is put on the runtime performance aspect in this work.

Another work of Zanker (Zanker et al., 2010) presented an approach that ranks the recommended items according to their degree of constraint fulfillment. They used historical transactions to evaluate the prediction quality of the recommendation results. This work focuses on relaxation of low weighted constraints whereas we satisfy all the user constraints in our approach. Even the constraint relaxation may help to reduce the runtime, there is no explicit runtime performance evaluation of the proposed approach in (Zanker et al., 2010).

In conclusion, all discussed related work and all of the mentioned state-of-the-art techniques lack of providing improvements in terms of *runtime performance* and *prediction accuracy* at the same time. Therefore, we introduce our proposed method to improve constraint-based recommendation in terms of both *runtime performance* as well as *prediction accuracy*.

3.4.1. Problem Definition

In order to demonstrate our proposed method, we present an example of an online personalized bike shop. We use a bike recommendation task (according to Definition 2.2.1) which is a small part of a real world knowledge base[1].

The concrete recommendation task of our bike shop example is denoted as RT_{bike}. Thereby, the task is composed of variables and constraints of a personalized bike shop. Furthermore, the bike shop does not offer a fixed product catalog. Instead, it offers any kind of personalized bike which complies with the constraints C_{bike} listed in Table 3.15.

	RT_{bike}
V_{bike}	$frame\_biketype = \{0,..,4\}$, $frame\_internal = \{0,..,1\}$, $extra\_propstand = \{0,..,2\}$, $gear\_internal = \{0,..,1\}$
C_{bike}	$c_1 : (frame\_biketype = (1 \vee 2)) \implies (frame\_internal = 1)$, $c_2 : (frame\_biketype = 4) \implies (gear\_internal = 0)$, $c_3 : (frame\_internal = 0) \implies (gear\_internal = 0)$

Table 3.15.: A constraint-based recommendation task RT_{bike}

Table 3.16 shows an example of different transactions in the context of the given recommendation task RT_{bike}. In this example, there are six users: Alice, Bob, Tom, Ray, Joe and Lisa. The first five users interacted with the online personalized bike shop in the past. The last user Lisa is an active user who has not left the online shop yet and has defined own preferences. In this table, the transaction type is denoted as TRX and explained in the remainder of this section.

[1] https://github.com/CSPHeuristix/CDBC/blob/master/Bike2KB.java

User name:	Alice	Bob	Tom	Ray	Joe	Lisa
TRX ID:	TRX_{Alice}	TRX_{Bob}	TRX_{Tom}	TRX_{Ray}	TRX_{Joe}	TRX_{Lisa}
frame_biketype	3	4	0		2	2
frame_internal	1	0		1		1
extra_propstand	1	1	1			
gear_internal	1	0	1	1	1	
TRX type:	HT1	HT1	HT2	HT2	HT3	AT

Table 3.16.: Transactions based on RT_{bike}

We categorize the transactions into four subtypes as follows:

Historical Transaction 1 (HT1): HT1 represents a complete and historical transaction (i.e., a complete past purchase). A transaction of this type can later serve as valuable input for recommendations in the future. Table 3.16 shows some examples of HT1 transactions. The purchases of Alice and Bob represent complete transactions.

Historical Transaction 2 (HT2): Another possible scenario refers to the situation where a user may leave the online shop without having purchased a recommended product. Such transactions are incomplete and denoted as HT2. These transactions represent stored incomplete transactions which only contain user requirements but not a complete specification of a product. In Table 3.16, Tom and Ray left the system without having finished to configure a bike. In other words, the configuration session was aborted before the transaction was completed.

Historical Transaction 3 (HT3): Another scenario refers to situations where new product features/services are introduced to existing products for which some complete and incomplete transactions of type HT1 and HT2 already exist. With the introduction of new product features/services new columns are added to the *Transaction Matrix*. These new columns cause new blank entries in the matrix for all existing transactions (of type 1 and 2). Consequently, all existing transactions of type 1 and 2 are converted into transactions of type HT3. In Table 3.16, Joe had purchased a personalized bike at a time when both product features *frame_internal* and *extra_propstand* did not exist. After these two features were introduced the former HT1 transaction of Joe was converted to a transaction of type HT3.

Active Transaction (AT): Whenever a new user starts a new configuration session, he or she will receive instant recommendations while he or she is defining requirements in the online bike shop. This scenario is referred to as an active transaction. Likewise HT2 and HT3, an active transaction appears as an incomplete row in the matrix (see example in Table 3.16. In sharp contrast to HT2 and HT3, the transaction is active (i.e., ongoing) and the user can receive instant recommendations based on the requirements provided by him or her.

3.4.2. The Proposed Method

The mentioned two main challenges for constraint-based recommender systems are *producing high quality recommendations* and *delivering instant recommendations for online recommendation tasks*. All discussed related work and state-of-the-art techniques lack of providing performance improvements in terms of *runtime performance* and *prediction accuracy* at the same time. The main objective of our approach is to decrease runtime in accordance with the increase of prediction quality for constraint-based recommenders.

Our proposed method calculates value ordering heuristics with respect to the formal definition of the

recommendation task (see Definition 2.2.1). These heuristics are calculated based on the historical transactions data. We gather the data into a matrix. This transactional data matrix is sparse because of the variants of the transactions. Therefore, we apply matrix factorization to obtain a dense matrix. Using the pre-calculated dense matrix, we calculate value ordering heuristics for a recommendation task which decreases the runtime and increases the prediction quality.

In this section, we show how our approach can be applied in such a way that it can find a consistent recommendation for each recommendation task. Our approach consists of two phases: an *offline phase* and an *online phase*. In the *offline phase*, the prerequisites of the calculations are completed. This phase is applied only once, not before solving each recommendation task. In the *online phase*, based on a given user requirement set REQ, REQ-specific value ordering heuristics are calculated for the recommendation search.

Offline Phase

Prerequisites of the calculations in our approach are gathering transactional data into a sparse bitmap matrix and using it to obtain a dense bitmap matrix. The dense bitmap matrix is calculated using matrix factorization techniques.

Building the bitmap matrix. In the first step of the offline phase, a sparse bitmap matrix, which holds all types of transactions, is generated. In our approach, we even take advantage of HT2/HT3 type (incomplete) transactions by including them besides the list of HT1 type (complete) transactions. We find predicted products for HT2/HT3 transactions in the offline phase and directly recommend those products in the online phase when they come back to the online shop. Furthermore, by including this valuable extra information, the prediction quality of our approach can be further increased.

We use a bitmap matrix (Chan and Ioannidis, 1998; Papagelis and Plexousakis, 2005) to hold the transactions. During the online phase, this bitmap matrix is used in order to obtain the value ordering heuristics. Each row of the matrix represents a *transaction*. Each column of the matrix represents a domain value of a variable. The bitmap matrix contains only 0s and 1s (i.e., bits). In a row (transaction), 1s mean that the variables hold as values the *corresponding* values of these columns. In return, all of the other domain values of these variables are set to 0.

For example, as shown in Table 3.17, the transaction of Alice is mapped into bits where all bits represents a value of the domain of the variable. For example, $frame\_biketype$ has a domain with five values (0,..,4), so its bitmap value has five bits (1s and 0s). $frame\_biketype$ is set to 3 by Alice, so in the bitmap matrix the corresponding bit (3rd bit) is set to 1 and others (1st, 2nd, and 4th bits) to 0.

In the working example, we convert all transactions in Table 3.16 to their corresponding bitmaps as shown in Table 3.17. For example, the transaction of Alice is mapped into bits where all bits represents a value of the domain of the variable. For example, $frame\_biketype$ has a domain with five values (0,..,4), so its bitmap value has five bits (1s and 0s). $frame\_biketype$ is set to 3 by Alice so in the bitmap version the corresponding bit (3rd bit) is set to 1 and others (1st, 2nd, and 4th bits) to 0.

Decomposing the sparse bitmap matrix. In this step, we decompose the sparse bitmap matrix which consists of all types (HT1, HT2, and HT3) of transactions.

In our working example, M_TRX_{bike} is a sparse bitmap matrix composed of transactions of RT_{bike} where each row represents a transaction of a user and each column represents a value of a variable. In terms of *matrix factorization*, the sparse matrix M_TRX_{bike} is decomposed into a $m \times k$ *user-feature matrix* and a $k \times n$ *value-feature matrix* which both contain the relevant information of the sparse matrix. Thereby, k is a variable parameter which needs to be adapted depending on the internal structure of the given data.

	frame_biketype					frame_internal		extra_propstand			gear_internal	
	0	1	2	3	4	0	1	0	1	2	0	1
	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12
Alice	0	0	0	1	0	0	1	0	1	0	0	1
Bob	0	0	0	0	1	1	0	0	1	0	1	0
Tom	1	0	0	0	0			0	1	0		
Ray						0	1				0	1
Joe	0	0	1	0	0						0	1

Table 3.17.: User requirements for bike configuration.

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12
Alice	0	0	0	1	0	0	1	0	1	0	0	1
Bob	0	0	0	0	1	1	0	0	1	0	1	0
Tom	1	0	0	0	0			0	1	0		
Ray						0	1				0	1
Joe	0	0	1	0	0						0	1

(a) Sparse matrix $\boldsymbol{M\_TRX}_{\text{bike}}$

	$uf1$	$uf2$	$uf3$	$uf4$	$uf5$	$uf6$
Alice	0.1	0.3	0.3	-0.4	0.4	0.5
Bob	0.1	0.3	0.3	-0.1	-0.4	0.7
Tom	0.1	0.3	0.4	-0.1	0.2	0.6
Ray	0.2	0.3	0.5	-0.5	0.4	0.4
Joe	0.1	0.3	0.4	-0.7	-0.2	0.2

(b) User factors $\boldsymbol{M\_UF}_{\text{bike}}$

	$vf1$	$vf2$	$vf3$	$vf4$	$vf5$	$vf6$
v1	1	1	0.2	0.3	0.3	0.1
v2	1	1	-1.5	0.2	0	-0.3
v3	1	1	0.1	-0.4	-0.2	-0.3
v4	1	1	-0.5	-0.1	0.5	-0.1
v5	1	1	-0.2	0.3	-0.5	0.2
v6	1	1	0.4	0.2	-0.6	0.3
v7	1	1	1.3	-0.4	0.7	0.2
v8	1	1	-1.3	0.2	-0	-0.3
v9	1	1	3.4	-0.4	0.1	0.8
v10	1	1	-1.3	0.2	-0	-0.3
v11	1	1	-1.5	0.2	-0	-0.3
v12	1	1	3.7	-0.6	-0	0.7

(c) Value factors $\boldsymbol{M\_VF}_{\text{bike}}$

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12
Alice	0.5	-0.3	0.3	0.4	0.1	0.3	1.3	-0.2	2	-0.2	-0.3	2
Bob	0.4	-0.3	0.3	0	0.6	0.9	0.6	-0.2	1.9	-0.2	-0.3	2
Tom	0.5	-0.3	0.2	0.3	0.3	0.5	1.1	-0.3	2.1	-0.3	-0.4	2.1
Ray	0.6	-0.5	0.5	0.5	0.1	0.5	1.7	-0.4	2.7	-0.4	-0.5	2.9
Joe	0.3	-0.4	0.6	0.2	0.2	0.6	1.1	-0.3	2.1	-0.3	-0.4	2.3

(d) Dense matrix $\boldsymbol{M\_TRX'}_{\text{bike}}$

Table 3.18.: Estimating the dense matrix $M\_TRX'_{\text{bike}}$ where latent factor=6.

As shown in Table 3.18, an estimated dense matrix $M\_TRX'_{\text{bike}}$, a user-feature and a value-feature matrices are calculated by applying matrix factorization based on singular-value decomposition (SVD) (Boutsidis and Gallopoulos, 2008) (we set the parameters in SVD as $latentfactors = 3$ and $iterations = 1000$). User factors (uf1..uf6) and value factors (vf1..vf6) are representing user and value features according to the defined latent factor. The matrices M_UF_{bike} and M_VF_{bike} will be used in the online phase to calculate the recommendation task specific value ordering heuristics.

Online Phase

In the online phase, the personalized bike shop recommender system receives the requirements of a user and searches for a consistent recommendation. In order to guide the search, we calculate recommendation task specific value ordering heuristics using the dense bitmap matrix which is calculated in the offline phase. At the end of this phase, a consistent recommendation result is provided to the user.

Calculating the value ordering heuristics for HT2/HT3. Since HT1 is a complete (purchase) transaction, there is no need to predict a recommendation for such transactions. However, for the incomplete transactions HT2 and HT3, we can provide recommendations. When a user of incomplete historical transactions comes back to the online system again, the corresponding transaction in the dense matrix can be used as a value ordering heuristics to guide the search of consistent recommendation.

For example, when *Ray* revisits the online bike shop, the related value ordering heuristics is calculated using the estimated transaction of *Ray* in the dense matrix $M\_TRX'_{\text{Ray}}$. For frame_biketype in $M\_TRX'_{\text{Ray}}$, we have the probabilities as {0.6, -0.5, 0.5, 0.5, 0.1} where the third and fourth probabilities are the highest. Thus, for $M\_TRX'_{\text{Ray}}$, the value ordering heuristic for frame_biketype becomes an ordered set: {2, 3, 0, 4, 1}. The values in this set indicate the assignment order of variable values used by the constraint solver.

Calculating the value ordering heuristics for AT. In order to calculate a value ordering heuristics for AT, we can not use the dense matrix since Lisa's transaction does not yet exist in the transaction matrix. If we include Lisa in the dense matrix $M\_TRX'_{\text{bike}}$ and again decompose it, the runtime would not be feasible for a real-time recommendation task.

Therefore for AT, we select the k nearest neighbors (Cover and Hart, 1967) (k most similar historical transactions) from the dense matrix of transactions. Then the corresponding k user factors from M_UF are aggregated into one user factors which is called predicted user factors for AT ($M\_UF'_{\text{AT}}$). Then we multiply $M\_UF'_{\text{AT}}$ with the value features matrix M_VF to obtain a predicted transaction for AT ($M\_TRX'_{\text{AT}}$).

We use Euclidean Distance (Weinberger et al., 2006) based similarity to find k most similar historical transactions to AT as shown in Formula 3.10. M_TRX' is a historical transaction from the dense matrix and M_TRX_{new} is the bit-mapped requirements of the active user. b_{i} represents the ith bit of the bitmap matrix. The bits of M_TRX_{new} and M_TRX' are compared on the basis of the instantiated bits in the M_TRX_{new}. Therefore n stands for the total number of instantiated bits in M_TRX_{new} and i stands for the index of the instantiated bits in M_TRX_{new}.

$$\Delta(M\_TRX_{\text{new}}, M\_TRX') = \sqrt{\sum_{i=1}^{n} \|M\_TRX_{\text{new}}.b_{\text{i}} - M\_TRX'.b_{\text{i}}\|^2} \tag{3.10}$$

For the working example, to find the most similar transactions, first we need to convert REQ_{Lisa} into a bitmap form (M_TRX_{new}) as shown in Table 3.5.

$$M\_TRX_{\text{Lisa}} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & \_ & \_ & \_ & \_ & \_ \end{bmatrix}$$

Figure 3.5.: The active transaction of Lisa (M_TRX_{Lisa})

We need to select k most similar transaction of M_TRX_{Lisa} from the rows of the dense matrix $M\_TRX'_{\text{bike}}$. In this working example, we set $k = 3$ to make it easy to present. In order to find three most similar transactions to M_TRX_{Lisa}, we find euclidean distance between TRX_{Lisa} and each row of $M\_TRX'_{\text{bike}}$ based on the instantiated bits of M_TRX_{Lisa}.

transactions in $M\_TRX'_{\text{bike}}$	distances to $\text{M\_TRX}_{\text{Lisa}}$
$\text{M\_TRX'}_{\text{Alice}}$	**2.03**
$\text{M\_TRX'}_{\text{Bob}}$	**1.95**
$\text{M\_TRX'}_{\text{Tom}}$	2.07
$\text{M\_TRX'}_{\text{Ray}}$	2.43
$\text{M\_TRX'}_{\text{Joe}}$	**1.89**

Table 3.19.: Euclidean distances to M_TRX_{Lisa}

According to the calculated distances in Table 3.19, the most similar transactions to M_TRX_{Lisa} are selected as M_TRX_{Alice}, M_TRX_{Bob}, and M_TRX_{Joe} since they have the shortest distances to M_TRX_{Lisa}. In order to find the predicted complete transaction $M\_TRX'_{\text{Lisa}}$, the user features of Alice, Bob, and Joe are taken from M_UF_{bike} to be aggregated then the aggregated matrix represents the predicted user features of Lisa $M\_UF'_{\text{Lisa}}$. Then $M\_UF'_{\text{Lisa}}$ is multiplied with M_VF_{Lisa} in order to find the predicted transaction of Lisa $M\_TRX'_{\text{Lisa}}$.

$$\begin{bmatrix} M\_UF_{\text{Alice}} \\ M\_UF_{\text{Bob}} \\ M\_UF_{\text{Joe}} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.3 & 0.3 & -0.4 & 0.4 & 0.5 \\ 0.1 & 0.3 & 0.3 & -0.1 & -0.4 & 0.7 \\ 0.1 & 0.3 & 0.4 & -0.7 & -0.2 & 0.2 \end{bmatrix}$$

$$M\_UF'_{\text{Lisa}} = \begin{bmatrix} 0.1 & 0.3 & 0.3 & -0.4 & -0.1 & 0.5 \end{bmatrix}$$

Figure 3.6.: The predicted user-features of Lisa ($M\_UF'_{\text{Lisa}}$)

The predicted user-feature matrix of Lisa $M\_UF'_{\text{Lisa}}$ is multiplied with the value-feature matrix M_VF_{bike} in order to find the value ordering heuristics for RT_{Lisa} as shown in Figure 3.7.

$$M\_TRX'_{\text{Lisa}} = M\_UF_{\text{Lisa}} x M\_VF_{\text{bike}}$$

$$\begin{bmatrix} 0.4 & -0.2 & 0.7 & 0.3 & 0 & 0.3 & 1.2 & -0.1 & 1.7 & -0.1 & -0.2 & 1.9 \end{bmatrix}$$

Figure 3.7.: The predicted transaction for Lisa ($M\_TRX'_{\text{Lisa}}$)

Using the predicted complete transaction for Lisa $M\_TRX'_{\text{Lisa}}$ as a matrix of probabilities for values (as

used for Ray in Section 3.4.2), we obtain the value orderings to solve RT_{Lisa} as shown in Table 3.20.

value ordering for frame_biketype:	**2**, 0 ,3, 4, 1
value ordering for frame_internal:	**1**, 0
value ordering for extra_propstand:	**1**, 0, 2
value ordering for gear_internal:	**1**, 0

Table 3.20.: Value ordering heuristics to solve RT_{Lisa}

Searching for a consistent recommendation. The predicted transaction (in the form of bitmap matrix) holds the probabilities of the domain values of each variable for RT_{new}. However, the highest probabilities in the predicted transaction matrix (*candidate recommendation*) may be inconsistent. Therefore, the highest values in the predicted transaction can not be recommended directly before checking their consistencies. A consistent recommendation can be found by a CSP solver. In order to guide the search of the CSP solver, we provide our calculated value ordering heuristics to solve the corresponding recommendation task.

Our approach calculates the consistent recommendation result using the probabilities in the predicted transaction matrix as varible and value ordering heuristics. Using the given value ordering heuristics in Table 3.20, a consistent recommendation result is found by the CSP solver.

As seen in Table 3.21, the consistent recommendation holds the first values from the value orderings in Table 3.20 which means the recommendation is found within a minimum search space. Moreover, if this recommendation is purchased by Lisa, the prediction accuracy becomes 1.

TRX_{Lisa}	AT	Consistent Recommendation
$frame\_biketype$	2	**2**
$frame\_internal$	1	**1**
$extra\_propstand$		**1**
$gear\_internal$		**1**

Table 3.21.: Consistent recommendation result for TRX_{Lisa}

3.4.3. Experimental Evaluation

In this section, we conduct a series of experiments to evaluate the effectiveness of the proposed approach for improving runtime performance and prediction accuracy for constraint-based recommendation. We first describe the experimental settings including the knowledge bases, evaluation criteria and parameters of tests. Then, we demonstrate the performance of our approach compared to the available methods.

Knowledge bases

We have used publicly available real-world knowledge based and datasets from *Configuration/Diagnosis Benchmarks in Choco (CDBC)* [2]. The number of historical and active transactions in each knowledge base is presented in detail in Table 3.22.

[2] https://github.com/CSPHeuristix/CDBC

Bike knowledge base. We have used a real-world bike knowledge base from *CDBC* to evaluate the runtime performance of our approach. Using this knowledge base, there can be an online personalized bike shop where users can define their preferences on the 34 variables to generate their own personalized bike. After their specification, a personalized bike is recommended on the online system. Then, they can order this recommended bike or still make new changes on it then order the next recommendation.

PC knowledge base. We have used a real-world personal computer (PC) knowledge base from *CDBC*. Using this knowledge base, there can be an online personalized PC shop where users can define their preferences on the 45 variables to generate their own personalized PC. After their specification, a personalized PC is recommended on the online system. Then, they can order this recommended PC or still make new changes on it then order the next recommendation.

Camera knowledge base. To be able to evaluate the prediction accuracy, we used a real-world digital camera dataset (*CameraKB_ConfigurationDataset*) from *CDBC* which includes historical transactions based on the real product catalog from a digital camera online shop. This dataset contains 264 transactions which contains a list of user requirements and corresponding user's preferred product IDs from the product catalog. Among these 264 transaction, we have used 200 of them as historical transactions and 64 of them as active transactions. Even though the dataset is rather small, it is very important for our prediction accuracy tests since there is no other similar real-world dataset available publicly.

Evaluation Criteria

We compare the performance of our approach with other heuristics (given in Section 3.4.3) based on the following two performance criteria: runtime (τ) and accuracy (π).

Runtime (τ). Runtime, as one of our performance indicators, represents the time spent in between collecting user requirements and providing the recommendation result (n represents the number of recommendation tasks, see Formula 3.11).

$$\tau = \frac{1}{n} \times \sum_{i=1}^{n} runtime(RT_{\mathrm{i}}) \tag{3.11}$$

For the evaluation of our approach, runtime is calculated by dividing the time spent in the online phase (see Section 3.4.2) by the number of recommendation tasks.

Accuracy (π). The recommended product (*REC*) can be purchased by the user or not. If the purchased product (*PP*) has the same values as *REC*, then *REC* is considered as a correct recommendation *CR*, otherwise *REC* is not a correct recommendation *NR*. As shown in Formula 3.12, a correct recommendation has all the values equal to the corresponding values in the purchased product.

$$\pi = \frac{\#(\mathrm{CR})}{\#(\mathrm{NR})+\#(\mathrm{CR})} \tag{3.12}$$

We have used only the camera knowledge base for evaluating the prediction quality. In order to calculate π, we have compared the recommendation result with the selected product of the user.

Parameters

Our experiments are executed on a computer with an Intel Core i5-5200U, 2.20 GHz processor, 8 GB RAM and 64 bit Windows 7 Operating System and Java Run-time Environment 1.8.0. Constraint satisfaction is

Parameters/Features	**Bike KB*** Runtime Tests	**PC KB*** Runtime Tests	**Camera KB*** Runtime Tests	Accuracy Tests
# of HT1	80**	50**	150***	150***
# of HT2/HT3	20**	50**	50***	50***
# of AT	50**	50**	64***	64***
# of variables	31	45	10	10
# of constraints	32	42	20	20
size of the search space	7.92×10^{24}	5.07×10^{34}	1.02×10^{7}	1.02×10^{7}
# of user constraints (r)	3	3	3	{1,2,3,4,5,6}
# of variable ordering heuristics	9	9	9	1
# of value ordering heuristics	7	7	7	1
# of similarity measures	-	-	-	4
# of test cases	63	63	63	35
# of runs on each case	50	50	50	50
# of total runs	3150	3150	3150	1750

*real world knowledge base, **synthetic transactions dataset, ***real world transactions dataset

Table 3.22.: Parameters of the runtime performance and prediction accuracy tests.

applied by a Java based CSP solver *choco-solver* [3]. For decomposing the bitmap matrix, we have used the *SVDRecommender* of Apache Mahout library (Schelter and Owen, 2012) with a latent factor k=100, number of iterations =1000, and number of recommended items is set to number of total variables of the recommendation task. For finding the estimated transaction, three most similar transactions are selected from the historical transactions.

All parameters in the test cases are given in Table 3.22. *Size of search space* of each knowledge base is calculated by multiplying domain sizes of all variables. r is the number of unary constraints in the user requirements. Each unary constraint includes a variable assigned with a random value from the corresponding domain. *# of value ordering heuristics* includes also our approach besides compared six other value ordering heuristics: *IntDomainMin, IntDomainMax, IntDomainMedian, IntDomainMiddle, IntDomainRandom, IntDomainRandomBound*. In accuracy tests, only our approach is used as used value ordering heuristic. Therefore it is set to 7 in runtime tests. *# of similarity measures* are the number of compared methods in accuracy tests which are *Euclidean Distance Similarity, Pearson Correlation, Cosine Similarity, and Tanimoto Coefficient Similarity*. *# of test cases* for runtime tests are calculated by multiplying the number of variable ordering heuristics with the number of value ordering heuristics which gives all number of combinations. *# of test cases* for accuracy tests are calculated by multiplying the number of r with the summation of number of similarity measures and number of value ordering heuristics.

Comparative Approaches

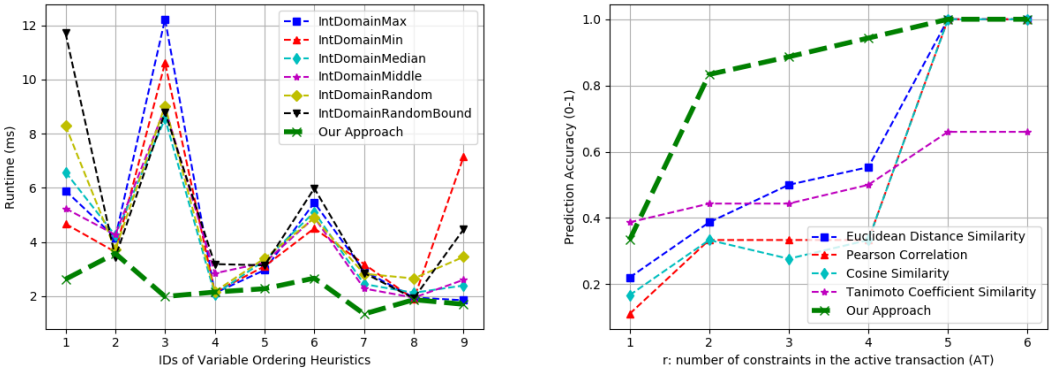
Our approach is learning value ordering heuristics for constraint solvers to solve a constraint-based recommendation task efficiently in terms of *runtime* and *prediction accuracy*. It combines the techniques of constraint satisfaction and collaborative filtering. Therefore, we have compared our approach both with constraint satisfaction heuristics and collaborative filtering based approaches.

Baselines for runtime performance. In order to evaluate the *runtime performance* (τ), we compared our approach with six value ordering heuristics of *choco-solver* (for integers) as following: *IntDomainMin, IntDomainMax, IntDomainMedian, IntDomainMiddle, IntDomainRandom, IntDomainRandomBound*. Choco solver has also 9 variable ordering heuristics: *Smallest, Largest, FirstFail, AntiFirstFail, Occurrence, InputOrder, DomOverWeg, GeneralizedMinDom, Random*. We have tested all combinations of value-variable ordering heuristics of choco-solver.

Baselines for prediction accuracy. To evaluate prediction accuracy (π), we have compared our approach with user neighborhood (*k nearest neighbor*) based recommenders which use *euclidean distance*

[3] http://www.choco-solver.org/

similarity, Pearson correlation, cosine similarity, and Tanimoto coefficient similarity from the collaborative filtering library of Apache Mahout (Schelter and Owen, 2012). Comparative collaborative filtering approaches have calculated the recommendation results using the AT and HT1 transactions of the camera dataset. For each approach, the recommendation result is compared with the purchased product ID of the corresponding historical transaction of the active transaction. If they are same, the accuracy is set to 1, if not to 0.



(a) Runtime Performance (τ)

(b) Prediction Accuracy (π)

Figure 3.8.: Performance results in terms of τ and π.

Results

We have used real-world knowledge bases and transactions datasets in various test cases as shown in Table 3.22. Thereby, we have compared our approach with constraint satisfaction and collaborative filtering based approaches. We show the evaluated performance results (see Figure 3.8) in terms of two performance indicators runtime (τ), and prediction quality (π). In runtime performance comparisons (a), we have used the variable ordering heuristics with the following IDs: #1: Smallest, #2: Largest, #3: FirstFail, #4: AntiFirstFail, #5: Occurance, #6: InputOrder, #7: DomOverWDeg, #8: GeneralizedMinDomain, #9:Random. For the runtime performance evaluation, baseline approaches are the mentioned six value ordering heuristics, whereas for the prediction accuracy evaluation, the baseline approaches are the KNN recommenders based on four different similarity techniques. In both graphs, our approach is shown in a bold line.

Evaluation of runtime performance

Figure 3.8(a) presents a comparison between our approach and six value ordering heuristics. Represented runtime values are the averages of the runtime results of all the tests on each of three datasets.

Each value ordering heuristic is combined with nine different variable-ordering heuristics (as described in Section 3.4.3) to set the search strategy of choco-solver. As observed, in some combinations of heuristics, the runtime results are very similar with our approach, especially when the variable ordering heuristic is set as Largest, Antifirstfail and GeneralizedMinDomain.

However, our approach outperforms all built-in value-ordering heuristics of Choco-Solver on the basis of all variable ordering heuristics combinations. The best performance of our approach is observed as 1.34 ms when it is combined with the variable ordering heuristic *DomOverWDeg* (#7).

Evaluation of accuracy. In Figure 3.8-b, we have compared the prediction quality of the recommendation results of our approach and mentioned collaborative filtering approaches in Section 3.4.3. In accuracy evaluations, we have used the camera knowledge base since the only real transactions dataset is collected based on this knowledge base.

We have tested each approach on the basis of r values where r represents the number of constraints in AT. According to the results, we have observed that when r increases the accuracy also increases. When r is set to 10, the user requirements itself is the same with the recommendation result. That is because there are only 10 variables in the camera knowledge base and r=10 means the user specifies 10 variables.

As observed in our real transactions dataset, user requirements include in average r=3 constraints and the most of the them are with r=(2,3,4). For these most common cases r=(2,3,4), our approach finds recommendations with the best prediction qualities than all of the compared collaborative filtering approaches. For example, when r=2, the accuracy of our approach is 0.83 where the next best approach (Tanimoto coefficient similarity) has the accuracy 0.44.

3.4.4. Conclusions

In particular, constraint-based recommenders are applicable in situations where there are large and potentially complex product assortments and/or cold-starting users. In such situations standard collaborative and content-based filtering techniques are known to encounter serious limitations. For such recommenders, the quick generation of recommendations within a reasonable time (i.e., before users leave the web page) can be very challenging if recommendation results should be in a high prediction quality as well.

We have proposed a novel value-ordering heuristics for constraint-based recommenders which employs matrix factorization techniques and historical transactions. As far as we are aware, no researches exist in constraint-based recommendation domain which exploit matrix factorization techniques. Our approach calculates a historical transactions matrix in the offline phase. This matrix is used to estimate a dense transaction for each new active transactions. Estimated dense transaction is used as a value ordering heuristics for searching a recommendation result in the online phase. According to our experimental results on the basis of real-world constraint-based knowledge bases (PC, bike, and camera), our approach outperforms the compared value ordering heuristics in terms of *runtime efficiency*. Moreover, to be able to evaluate *prediction quality*, we used real historical transactions based on camera knowledge base and we have observed that our approach outperforms nearest neighbor based collaborative filtering methods.

We have employed nearest neighbor based collaborative filtering approaches as prediction accuracy baselines but could not use other constraint-based recommendation approaches since these implementations are not yet available in commonly used recommendation libraries. However, as future work, we would like to compare our approach with other constraint-based recommenders as well.

We have used user constraints which include exact value assignments (e.g. $frame\_biketype$=1). However, in many real world cases, user requirements do not have exact value assignments but a group of preferred values (e.g. $frame\_biketype$= 1 *or* 2). Moreover, there can be also combined constraints in user requirements (e.g. *if* $frame\_biketype$=1 *then* $gear\_internal$=1). As a future work, we plan to focus on covering these kind of constraints as well. By this way, we can provide a full solution for real-world configuration systems which assist its users online with high quality recommendation results within a reasonable response time.

3.5. Matrix Factorization based Heuristics for Direct Diagnosis

When constraints of a *CSP* are inconsistent, no solution can be found. In this context, diagnosis (Bakker et al., 1993) is required to find at least one solution for an inconsistent *CSP*. There are several diagnosis approaches. One of them is *direct diagnosis* which employs queries to check the consistency of the constraint set without the need to identify the corresponding conflict sets. When diagnoses have to be provided in real-time, response times should be less than a few seconds (Card et al., 1991). For example, in communication networks, efficient diagnosis is crucial to retain the quality of service. However, in direct diagnosis approaches, there is a clear trade-off between runtime performance of diagnosis calculation and diagnosis quality (Felfernig et al., 2018c).

To address this challenge, we propose *Learned Constraint Ordering (LCO)* for direct diagnosis. Our approach learns constraint ordering heuristics from inconsistent historical transactions which include inconsistent user requirements. Using historical inconsistent transactions, we build a sparse matrix and then employ matrix factorization techniques to estimate diagnoses. After this offline learning phase, the most similar transaction to the new inconsistent requirement set is found and the corresponding constraint ordering heuristic (which is calculated in the offline phase) is applied to reorder the inconsistent constraints before direct diagnosis. Thanks to the learned ordering of constraints, direct diagnosis algorithms can solve the diagnosis task with a high quality diagnosis result in a shorter runtime compared to direct diagnosis without constraint ordering. We provide a working example to demonstrate the effects of our approach. Finally, based on experimental evaluations, we show that using our constraint ordering approach with direct diagnosis algorithms is superior to the baseline (direct diagnosis algorithms without constraint ordering) on popular benchmark constraint satisfaction problems.

3.5.1. Problem Definition

The following (simplified) assortment of digital cameras (see Table 3.23) and a set of inconsistent user requirements (see Table 3.24) for selecting a digital camera from the camera product table will serve as a working example to demonstrate how our approach works.

Table 3.23.: The Camera Product Table

	Camera_1	Camera_2	Camera_3	Camera_4	Camera_5
effectiveResolution	20.9	6.1	6.1	6.2	6.2
display	3.5	2.5	2.2	1.8	1.8
touch	yes	yes	no	no	no
wifi	yes	yes	no	no	no
nfc	no	no	no	no	no
gps	yes	yes	no	no	yes
videoResolution	UHD	UHD	No	UHD	4K
zoom	3.0	3.0	7.8	5.8	3.0
weight	475	475	700	860	560
price	659	659	189	2329	469

The working example is formed as a configuration task in Table 3.24 on the basis of Definition 2.1.1. As shown in Table 3.24, it has a variable set (V) with 10 variables (which are also listed in the first column of Table 3.23) and only one knowledge base constraint (c_1) which allows to select one camera (from the available cameras in Table 3.23). Other constraints are defined in the set of user requirements (REQ_{Lisa}).

Table 3.24.: An inconsistent configuration task: CSP_{Lisa}

V, D	v_1: **effectiveResolution**: $\{6.1Megapixel, 6.2Megapixel, 20.9Megapixel\}$, v_2: **display**: $\{1.8inches, 2.2inches, 2.5inches, 3.5inches\}$, v_3: **touch**:$\{no, yes\}$, v_4: **wifi**:$\{no, yes\}$, v_5: **nfc**:$\{no, yes\}$, v_6: **gps**:$\{no, yes\}$, v_7: **videoResolution**: $\{No, UHD, 4K\}$, v_8: **zoom**:$\{3.0x, 5.8x, 7.8x\}$, v_9: **weight**: $\{475g, 560g, 700g, 860g, 1405g\}$, v_{10}: **price**: {189 €, 469 €, 659 €, 2329 €, 5219 €}
C_{KB}	c_1: ($\text{Camera}_1 \vee \text{Camera}_2 \vee \text{Camera}_3 \vee \text{Camera}_4 \vee \text{Camera}_5$)
REQ_{Lisa}	c_2: **effectiveResolution**=20.9Megapixel c_3: **display**=2.5inches c_5: **wifi**=yes c_7: **gps**=yes c_9: **zoom**=5.8x

In such inconsistency cases we can help users to resolve the conflicts with a diagnosis Δ which is a set of constraints. Assuming that C_{KB} is consistent, we can say that the knowledge base always will be consistent if we remove REQ. Removing Δ from REQ leads to a consistent knowledge base (see Definition 2.1.5). The *REQ diagnosis task* of *Lisa* (RDT_{Lisa}) can be resolved by two minimal diagnoses. The removal of the set $\Delta_{\text{Lisa\_1}} = \{c_2, c_9\}$ or $\Delta_{\text{Lisa\_2}} = \{c_3, c_9\}$ leads to a consistent configuration knowledge base.

Direct Diagnosis

Algorithmic approaches to provide efficient solutions for diagnosis problems are many-fold. Basically, there are two types of approaches *conflict-directed diagnosis* (Stern et al., 2012) and *direct diagnosis* (Felfernig et al., 2018c).

Conflict-directed diagnosis algorithms first calculate conflicts then find diagnoses. Therefore, their runtime performance are not sufficient for real-time scenarios. *Direct diagnosis* algorithms determine diagnoses by executing a series of queries. These queries check the consistency of the constraint set without the need of pre-calculated conflict sets.

Quality of diagnoses and runtime performance of direct diagnosis algorithms are based on the ordering of the constraints in the set of user requirements: *the lower the importance of a constraint means the lower the index of the constraint*. The lower the ordering conflicting constraint has the higher the probability that this constraint will be part of the diagnosis (Felfernig et al., 2012).

Users typically prefer to keep the important requirements and to change or delete (if needed) the less important ones (Junker, 2004). The major goal of (model-based) diagnosis tasks is to identify the *preferred (leading) diagnoses* (de Kleer, 1990). For the characterization of a preferred diagnosis we will rely on a total ordering of the given set of constraints in REQ. Such a total ordering can be achieved, for example, by directly asking the customer regarding the preferences, by applying multi attribute utility theory where the determined interest dimensions correspond with the attributes of REQ or by applying the orderings determined by conjoint analysis (Schaupp and Bélanger, 2005).

Evaluation Criteria

We can evaluate the performance of a direct diagnosis algorithm based on runtime performance, diagnosis quality (in terms of minimality), and combined performance (runtime and minimality).

Runtime. $runtime(\Delta)$ represents the time spent during the diagnostic search to find Δ. This spent time

can be measured in milliseconds or in the number of consistency checks (**#CC**) applied till a diagnosis is found. For an more accurate runtime measurement (excluding the operating system's effects on runtime, etc.), it can preferred to use the number of consistency checks.

Minimality. Diagnosis quality can be measured in terms of the degree of minimality of the constraints in a diagnosis, the cardinality of Δ compared to the cardinality of $\Delta_{\min}$. $|\Delta_{\min}|$ represents the cardinality of a minimal diagnosis. The highest (best) minimality can be 1 according to Formula 3.13.

$$minimality(\Delta) = \frac{|\Delta_{\min}|}{|\Delta|} \tag{3.13}$$

Combined. Since it is important to satisfy both evaluation criteria runtime performance and minimality at the same time, we also evaluate combined performance based on the Formula 3.14.

$$combined(\Delta) = \frac{minimality(\Delta)}{runtime(\Delta)} \tag{3.14}$$

Related Work

The most widely known algorithm for the identification of minimal diagnoses is *hitting set directed acyclic graph* (HSDAG) (Reiter, 1987). HSDAG is based on conflict-directed hitting set determination and determines diagnoses based on breadth-first search. It computes minimal diagnoses using minimal conflict sets which can be calculated by QUICKXPLAIN (Junker, 2001). The major disadvantage of applying this approach is the need of predetermining minimal conflicts which can deteriorate diagnostic search performance. Many different approaches to provide efficient solutions for diagnosis problems are proposed. One approach (Wotawa, 2001) focuses on improvements of HSDAG.

The direct diagnosis algorithm FLEXDIAG (Felfernig et al., 2018c), utilize an inverse version of the QUICKXPLAIN and an associated inverse version of HSDAG. Therefore, it finds directly a diagnosis from an inconsistent constraint set. FLEXDIAG assures diagnosis determination within certain time limits by systematically reducing the number of solver calls needed using the parameter *m*. Therefore, authors claim that this specific interpretation of anytime diagnosis leads *a trade-off between diagnosis quality (evaluated, e.g., in terms of minimality) and the time* needed for diagnosis determination. Our proposed constraint ordering approach *LCO* improves their direct diagnosis approach in terms of diagnosis quality at the same time with runtime performance.

Another work DIR (Shchekotykhin et al., 2014) determines diagnoses by executing a series of queries. Authors reduce the number of consistency checks by avoiding the computation of minimized conflict sets and by computing some set of minimal diagnoses instead of a set of most probable diagnoses or a set of minimum cardinality diagnoses. Their approach is very similar to (Felfernig et al., 2012), with two modifications: (i) a depth-first search strategy instead of breadth-first and (ii) a new pruning rule to remove a constraint from the set of inconsistent constraints. They compared their approach only with the standard technique (based on QUICKXPLAIN (Junker, 2001) and HSDAG (Reiter, 1987). In their experiments based on a set of knowledge bases created by automatic matching systems, they show that their direct diagnosis approach outperforms the standard diagnosis approach in terms of runtime. In this work, authors gather the constraint ordering directly from users interactively. Our work differs from (Shchekotykhin et al., 2014) in that we do not need active user interactions to determine the constraint ordering. Our approach learns the constraint ordering from historical inconsistent user requirements and their preferred diagnoses.

The importance of constraint ordering are already mentioned in related direct diagnosis work (Felfernig et al., 2018c; Shchekotykhin et al., 2014). In our approach, we predict the most important constraints for the users based on an important collaborative filtering approach *matrix factorization* (Koren, 2010)

Table 3.25.: Historical transactions with inconsistent user requirements.

	Alice	Bob	Tom	Ray	Joe
c_2: resolution	-	6.1	20.9	20.9	6.2
c_3: display	3.5	2.2	-	2.5	2.2
c_4: touch screen	-	-	yes	yes	-
c_5: wifi	-	-	yes	yes	-
c_6: nfc	-	yes	-	-	yes
c_7: gps	-	yes	yes	yes	no
c_8: video resolution	UHD	-	UHD	UHD	UHD
c_9: zoom	3.0	5.8	3.0	5.8	7.8
c_{10}: weight	560	700	475	475	-
c_{11}: price	469	189	469	-	189
Purchase	Camera_1	-	Camera_1	-	Camera_3
Δ_{min}	c_{10}, c_{11}	-	c_3, c_{11}	-	c_2, c_6, cv_7

and employing historical transactions. We learn a constraint ordering where the predicted most important constraints are placed to the highest orderings. This is because, the implemented direct diagnosis algorithm FLEXDIAG first start searching a diagnosis among the lowest raking constraints. Therefore, the highest ordering constraints have low probability to be in the diagnosis set.

3.5.2. The Proposed Method

In this section, our motivation is to solve *the quality-runtime performance trade off problem* of direct diagnosis. For this purpose, our proposed method learns constraint ordering heuristics based on historical transactions in offline phase and then in online phase it employs a direct diagnosis algorithm on the re-ordered constraints of diagnosis tasks (active transactions).

Offline Phase: Learning from Historical Transactions

Our proposed method needs an offline phase in which various constraint ordering heuristics are learned based on historical transactions. For the offline learning, matrix factorization techniques are employed *historical transactions with inconsistent user requirements* are utilized (see Table 3.25).

In Table 3.25, for each user, we have an inconsistent set of user requirements (which leads to "no solution"). After their no-solution situation, some of them (e.g. Alice, Tom, and Joe in Table 3.25) decided to buy a product (*Purchase*) which does not completely satisfy their requirements. Therefore, they had to eliminate a set of initial requirements which is presented as Δ_{min}. These historical transactions which are completed with a purchase are *complete historical transactions*. The rest of historical transactions, in which users did not complete their transactions with a purchase (e.g. Bob and Ray in Table 3.25), is called *incomplete historical transactions*. Therefore, we estimate diagnoses of *incomplete historical transactions* using matrix factorization.

The Sparse Matrix. Matrix factorization based collaborative filtering algorithms (Koren et al., 2009) introduce a rating matrix R (a.k.a., user-item matrix) which describes preferences of users for the individual items the users have rated. Thereby, R represents a $m \times n$ matrix, where m denotes the number of users and n the number of items. The respective element $r_{u,i}$ of the matrix R describes the rating of the item i made by user u. Given the complete set of user ratings, the recommendation task is to predict how the users *would* rate the items which they have not yet been rated by these users.

In our approach, we build a sparse matrix R (user-constraint matrix) using inconsistent historical transactions as shown in Table 3.26-(a) where columns represent constraints. Therefore, each row of the sparse

matrix R represents a set of user requirements (the left half) and if exist their corresponding diagnoses (the right half). User requirements are presented in their normalized values in the range of 0-1, and diagnoses are presented with the presence (1)/non-presence (0) of user requirements.

If there are non-numeric domains in the problem, they are enumerated. For example, the domain v_7: **videoResolution**: $\{No, UHD, 4K\}$ is enumerated as v_7: $\{0,1,2\}$. Besides, domain ranges of all constraints in REQ are mapped to [0..1] since matrix factorization needs to use the same range for all values in the matrix. For this purpose, we have employed *Min-Max Normalization* (Visalakshi and Thangavel, 2009).

	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	pc_2	pc_3	pc_4	pc_5	pc_6	pc_7	pc_8	pc_9	pc_{10}	pc_{11}
Alice		1					0.5	0	0.09	0.05	0	0	0	0	0	0	0	0	1	1
Bob	0	0.23			1	1		0.58	0.24	0										
Tom	1		1	1		1	0.5	0	0	0.05	0	1	0	0	0	0	0	0	0	1
Ray	1	0.41	1	1		1	0.5	0.58	0											
Joe	0.006	0.23			1	0	0.5	1		0	1	0	0	0	1	1	0	0	0	0

(a) The sparse matrix (R)

	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	pc_2	pc_3	pc_4	pc_5	pc_6	pc_7	pc_8	pc_9	pc_{10}	pc_{11}
Alice	1.4	0.2	1.3	1.5	0.3	1.2	0.9	0.6	-0.2	-0.3	0.4	0.2	-0.4	-0.4	0.4	0.4	-0.4	0.2	0.4	1.1
Bob	0.9	0.3	1.2	1.3	1.1	1.3	0.9	1	0	-0.3	0.7	0	-0.4	-0.4	1.1	1.1	-0.4	0.6	0.1	0.5
Tom	1.5	0.3	1.5	1.6	0.3	1.4	0.9	0.6	-0.2	-0.3	0.4	0.3	-0.4	-0.4	0.4	0.4	-0.4	0.3	0.4	1.1
Ray	1.4	0.3	1.5	1.6	0.7	1.5	0.9	0.9	-0.1	-0.3	0.8	0.1	-0.4	-0.4	0.6	0.6	-0.4	0.7	0.2	0.6
Joe	0.7	0.2	0.9	1.1	1.2	0.9	0.8	1.1	0.1	-0.3	0.8	-0.1	-0.3	-0.3	1.1	1.1	-0.4	0.4	0.1	0.4

(b) The estimated dense matrix (PQ^{T})

Table 3.26.: Matrix factorization estimates a dense matrix PQ^{T} (b) from the sparse matrix R (a).

Matrix Factorization. In terms of *matrix factorization*, the sparse matrix R is decomposed into a $m \times k$ *user-feature matrix* P and a $k \times n$ *constraint-feature matrix* Q which both are used to find the estimated dense matrix PQ^{T}. Thereby, k is a variable parameter which needs to be adapted accordingly depending on the internal structure of the given data.

In our example, we apply matrix factorization to the sparse matrix in Table 3.26-(a). Then, the estimated matrix is obtained as shown in Table 3.26-(b) which includes the estimated diagnoses for Bob and Ray.

Online Phase: Diagnosing Active Transactions

After calculating the estimated matrix in the offline phase, in the online phase we diagnose active transactions which includes inconsistencies as in our working example. In active transactions, users still did not leave the configuration system and need real-time help to remove inconsistencies in their configuration to decide on a product to purchase. Therefore, the configuration system should provide a high quality diagnosis in a reasonable time (before users leave the system without a purchase).

The Most Similar Historical Transaction. We find the most similar historical transaction to the new set of inconsistent requirements using Formula 3.15 where HT represents a historical transaction, AT represents the active transaction, $HT.c_{\mathrm{i}}$ represents the value of each constraints in the estimated dense matrix PQ^{T}, and $AT.c_{\mathrm{i}}$ represents the value of each constraints in the active transaction. i represent a constraint index value in the REQ of AT.

$$min(\sqrt{\sum_{i \in AT.REQ} \|HT.c_{\mathrm{i}} - AT.c_{\mathrm{i}}\|^2}) \tag{3.15}$$

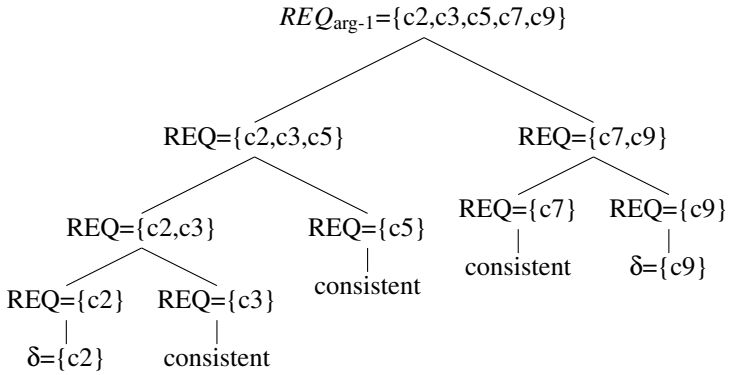
In our camera configuration example, the most similar historical transaction to the active transaction of *Lisa* is the transaction of *Ray*. Therefore, to diagnose RDT_{Lisa}, we use LCO_{Ray}: {c$_2$, c$_9$, c$_6$, c$_7$, c$_{11}$, c$_{10}$, c$_3$, c$_4$, c$_5$, c$_8$}. When we only consider user requirements of *Lisa*, we obtain the constraint ordering for Lisa LCO_{Lisa}: {c$_2$, c$_9$, c$_7$, c$_3$, c$_5$}.

Direct Diagnosis with LCO. After the most similar historical transaction is found and its constraint ordering is applied to the active transaction's user requirements to be reordered, the direct diagnosis algorithm is employed on the active diagnosis task with the reordered user constraints.

As shown in below four search trees, the search trees of FLEXDIAG with *LCO* (Tree-2 and Tree-4) have better combined performance compared to the search trees of FLEXDIAG without *LCO* (Tree-1 and Tree-3). When m=1, *LCO* improved the combined performance with the ratio 32% (0.166 instead of 0.125). When m=2, *LCO* improved the combined performance with the ratio 100% (0.250 instead of 0.125) and the minimality with the ratio 50% whereas the runtime is not improved.

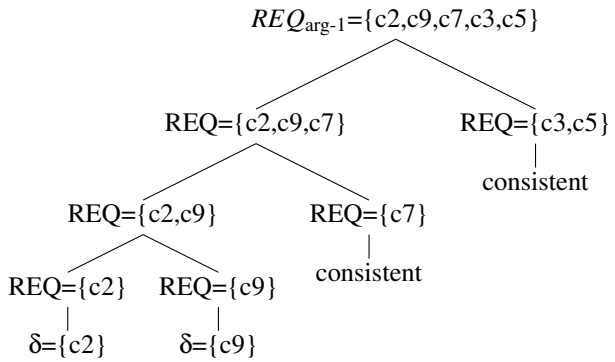
Tree-1: FLEXDIAG (M=1): $\Delta = \{c2, c9\}$

#CC = 8, **minimality = 1**, and **combined = 0.125**



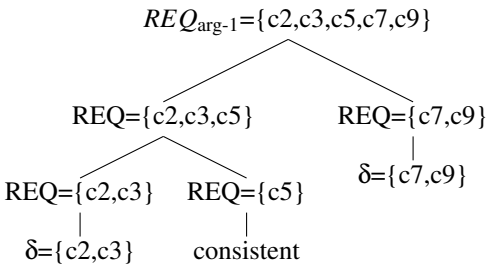
TREE-2: FLEXDIAG (M=1) with LCO: $\Delta = \{c2, c9\}$

#CC = 6, **minimality = 1**, and **combined = 0.166**



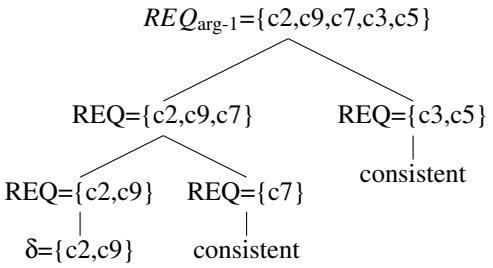
TREE-3: FLEXDIAG (M=2): $\Delta = \{c2, c3, c7, c9\}$

#CC = 4, **minimality = 0.5**, and **combined = 0.125**



TREE-4: FLEXDIAG (M=2) with LCO: $\Delta = \{c2, c9\}$

#CC = 4, **minimality = 1**, and **combined = 0.250**:



Consequently, in this working example, user constraints of the diagnosis task is reordered using LCO_{Lisa} as $\{c_2, c_9, c_7, c_3, c_5\}$ and a minimal diagnosis $\Delta = \{c_2, c_9\}$ is found by FLEXDIAG-(m=1) with performance results (on the basis of evaluation criteria given in Section 3.5.1): #CC = 4, minimality = 1, and combined = 0.250. However, when we employ the diagnosis algorithm FLEXDIAG-(m=1) on the default order of user constraints $\{c_2, c_3, c_5, c_7, c_9\}$, the same diagnosis $\Delta = \{c_2, c_9\}$ is found with performance results: #CC = 8, minimality = 1, and combined = 0.125. Therefore, using LCO with FLEXDIAG-(m=1), we double the combined performance of diagnosis when diagnosing the working example.

3.5.3. Experimental Evaluation

In our experiments[1], we have used Minizinc-2016 benchmark problems (Stuckey et al., 2014) where each problem includes five data files with file extension ".*dzn*".[3] In order to obtain historical and active transactions based on these benchmark problems, we randomly generated 5000 sets of inconsistent user requirements (each with *N* constraints) based on integer variables. We have compared our approach *LCO* with the baseline *no constraint ordering*. In both cases, for diagnostic search, FLEXDIAG is used with three different *m* values 1, 2, and 4. We do not compare *LCO* with more traditional diagnosis approaches. For related evaluations we refer the reader to (Felfernig et al., 2018c) where detailed analyses of our baseline (FLEXDIAG without constraint ordering) can be found.

[1] We have developed our approach in Java and tested on a computer with an Intel Core i5-5200U, 2.20 GHz processor, 8 GB RAM and 64 bit Windows 7 Operating System and Java Run-time Environment 1.8.0. Constraint satisfaction problems have been solved by *Choco3*[2] which is a java library for constraint satisfaction problems with a FlatZinc (the target language of MiniZinc) parser. For matrix factorization needs, we have used the *SVDRecommender* of Apache Mahout (Schelter and Owen, 2012) with a latent factor *k*=100, the number of iterations = 1000.

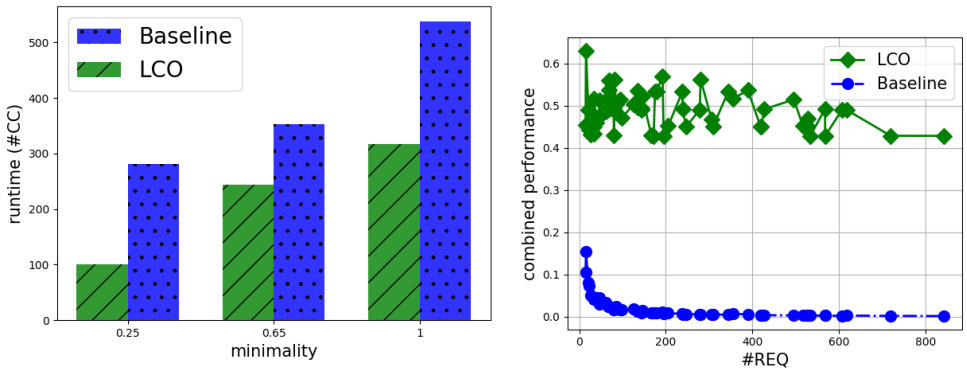
[3] http://www.minizinc.org/challenge2016/results2016.html

EXPERIMENTAL RESULTS																
Minizinc 2016 Benchmark			Inconsistencies		BASELINE						OUR APPROACH					
					runtime			*minimality*			*runtime*			*minimality*		
.mzn	*.dzn*	#vars	#REQs	$\lvert\Delta_{min}\rvert$	m=1	m=2	m=4	m=1	m=2	m=4	m=1	m=2	m=4	m=1	m=2	m=4
1. cc_base	test.02	136	68	5	27	23	15	1	0.333	0.25	19	15	6	1	0.5	0.333
	test.06	478	239	23	120	80	60	1	0.333	0.166	60	53	22	1	1	0.333
	test.11	159	80	5	32	27	18	1	0.5	0.25	20	16	7	1	1	0.333
	test.13	291	146	11	73	42	36	1	0.5	0.166	36	32	13	1	1	0.333
	test.20	688	344	28	172	98	86	1	0.333	0.166	86	76	29	1	1	0.333
2. celar	CEL.6-S0	557	279	21	139	80	70	1	1	0.143	70	62	23	1	1	0.333
	CEL.6-S4	1136	568	45	284	189	126	1	0.333	0.25	162	114	52	1	1	0.25
	CEL.7-S4	1137	569	45	227	190	126	1	0.333	0.143	162	126	52	1	0.5	0.333
	graph05	2680	1340	99	536	447	335	1	0.333	0.143	383	268	122	1	1	0.25
	scen07	6331	3166	211	1583	1055	703	1	1	0.166	791	703	288	1	1	0.25
3. step1	kb128_11	17390	8695	790	4348	2484	1932	1	0.5	0.25	2174	1739	725	1	1	0.25
	kb128_14	17390	8695	828	4348	2898	2174	1	0.333	0.166	2174	1932	725	1	0.5	0.333
	kb128_16	17390	8695	870	4348	2484	2174	1	1	0.25	2174	1739	725	1	1	0.25
	kb128_17	17390	8695	870	3478	2898	2174	1	0.333	0.166	2484	1739	725	1	0.5	0.25
	kb192_10	47278	23639	1970	9456	6754	5253	1	0.5	0.166	6754	5253	2149	1	0.5	0.333
4. depot	att48_6	77	39	3	19	11	9	1	1	0.166	10	9	3	1	1	0.25
	rat99_5	69	35	3	14	12	9	1	0.333	0.143	9	8	3	1	1	0.25
	rat99_6	77	39	4	19	11	9	1	0.333	0.25	10	8	4	1	1	0.25
	st70_5	69	35	2	17	10	9	1	0.5	0.166	10	8	3	1	0.5	0.25
	ulysses	69	35	2	17	10	9	1	0.333	0.166	10	7	3	1	1	0.333
5. dcmst	c_v15_d7	494	247	18	124	71	62	1	0.5	0.166	62	49	22	1	0.5	0.25
	c_v20_d5	856	428	39	214	143	107	1	0.333	0.143	122	95	39	1	0.5	0.25
	s_v20_d4	390	195	14	78	65	49	1	1	0.25	49	39	18	1	1	0.333
	s_v20_d5	385	193	14	77	64	48	1	0.333	0.166	48	43	16	1	0.5	0.333
	s_v40_d5	1214	607	45	304	173	135	1	1	0.2	173	121	51	1	0.5	0.25
6. filter	ar_1_3	121	61	6	30	17	13	1	0.333	0.25	17	12	6	1	1	0.25
	dct_1_3	189	95	9	38	32	21	1	0.5	0.143	24	19	9	1	1	0.333
	ewf_1_2	139	70	6	35	20	15	1	1	0.25	17	14	6	1	0.5	0.333
	fir_1_3	92	46	4	18	15	10	1	1	0.166	13	10	4	1	1	0.333
	fir_1_4	92	46	3	18	15	12	1	0.333	0.2	12	10	4	1	0.5	0.333
7. gbac	UD3	1687	844	77	422	241	211	1	0.5	0.25	241	187	70	1	1	0.25
	UD6	561	281	24	140	94	62	1	0.333	0.166	80	62	23	1	1	0.333
	UD10	619	310	24	155	103	77	1	1	0.2	77	62	28	1	1	0.25
	UD3	1938	969	81	388	277	242	1	0.5	0.143	242	194	88	1	1	0.25
	UD5	1439	720	55	360	206	160	1	0.5	0.2	180	160	65	1	1	0.333
8. gfd-sch.	n25f5.	344	172	11	69	57	43	1	0.333	0.166	49	34	14	1	0.5	0.333
	n35f5.	477	239	20	95	80	53	1	0.333	0.25	60	48	20	1	0.5	0.25
	n55f2.	612	306	21	122	102	77	1	1	0.166	87	61	28	1	0.5	0.333
	n60f7.	1236	618	54	247	177	155	1	0.5	0.143	177	137	56	1	0.5	0.333
	n180f.	4950	2475	177	1238	825	550	1	0.5	0.143	619	495	225	1	0.5	0.25
9. map.	m2x2_1	197	99	9	49	33	22	1	0.333	0.25	25	22	8	1	0.5	0.333
	m2x2	358	179	16	90	60	40	1	1	0.2	45	40	16	1	1	0.333
	m3x3	838	419	38	210	120	105	1	0.333	0.166	105	93	35	1	0.5	0.333
	m4x4	841	421	28	168	120	105	1	1	0.25	120	93	38	1	0.5	0.25
	ring	411	206	15	82	59	46	1	0.5	0.2	51	46	19	1	1	0.25
10. m.dag	25_01	136	68	7	34	19	15	1	0.333	0.166	19	15	6	1	0.5	0.333
	25_03	160	80	5	32	27	18	1	0.333	0.143	20	18	7	1	1	0.25
	25_04	133	67	6	27	19	17	1	0.333	0.166	17	15	6	1	1	0.333
	25_06	164	82	8	33	27	18	1	0.5	0.2	23	16	7	1	1	0.333
	31_02	169	85	6	42	24	19	1	0.5	0.2	21	17	8	1	1	0.25
11. mrcp.	j30_1_10	712	356	28	178	102	89	1	1	0.143	89	79	32	1	0.5	0.333
	j30_15_5	333	167	12	67	56	37	1	0.5	0.2	42	37	15	1	0.5	0.333
	j30_17_10	992	496	34	198	142	124	1	1	0.25	142	99	41	1	0.5	0.25
	j30_37_4	780	390	28	195	111	98	1	0.5	0.2	111	78	33	1	0.5	0.25
	j30_53_3	350	175	12	70	58	44	1	0.333	0.25	44	39	15	1	1	0.25
12. nfc	12_2_5	31	16	1	8	5	3	1	0.333	0.2	4	3	1	1	1	0.25
	12_2_10	29	15	1	6	4	4	1	1	0.166	4	3	1	1	1	0.333
	18_3_5	45	23	2	11	6	6	1	1	0.143	6	5	2	1	0.5	0.333
	18_3_10	42	21	2	11	7	5	1	1	0.2	5	5	2	1	0.5	0.333
	24_4_10	54	27	2	11	8	7	1	0.5	0.25	7	5	2	1	1	0.25
13. oocsp	030_e6_cc	1034	517	43	207	148	129	1	1	0.25	129	103	43	1	0.5	0.25
	030_ea4_cc	1068	534	40	267	178	119	1	0.333	0.25	153	119	45	1	0.5	0.333
	030_f7_cc	1058	529	44	265	151	132	1	1	0.166	151	106	44	1	1	0.25
	030_mii8	1053	527	48	263	176	117	1	1	0.25	132	117	44	1	0.5	0.333
	100_r1	3433	1717	149	858	572	429	1	0.333	0.25	429	381	143	1	1	0.333
14. pc	28-4-7-1	252	126	12	50	42	28	1	0.333	0.143	32	25	11	1	1	0.25
	30-5-6-2	270	135	9	54	39	34	1	1	0.143	39	27	11	1	1	0.333
	30-5-6-8	270	135	11	54	45	30	1	1	0.143	34	27	11	1	0.5	0.333
	32-4-8-2	288	144	10	58	41	36	1	0.333	0.2	36	32	13	1	1	0.333
	32-4-8-5	288	144	10	58	41	36	1	0.5	0.2	36	29	13	1	1	0.333
average		2349	1174	101	530	357	276	1	0.588	0.192	315	249	102	1	0.779	0.296

Table 3.27.: Experimental results based on Minizinc-2016 Benchmark problems.

As shown in Table 3.27, based on the averages (in the last row), *LCO* outperforms the baseline in terms of *runtime* and *minimality* since with each *m* value (1, 2, and 4), *LCO* has lower runtime than the baseline whereas its minimality is higher (or equal) compared to the baseline with each *m* value (1, 2, and 4).

Relations between performance indicators and the number of constraints in the set of user requirements are presented in Figure 3.9. In Figure 3.9-(a), we observe that *minimality* increases when *runtime* (in #CC) increases. As observed, the number of consistency checks (#CC) are at each *m* (*m*=1, 2, and 4) lower when *LCO* is used. Moreover, at each *m* (*m*=1, 2, and 4), *LCO* also provides better of equal minimality results.



(a) Runtime vs Minimality

(b) Combined vs *#REQ*

Figure 3.9.: Comparison Graphs based on the Experimental Results in Table 3.27

As discussed throughout this section, our main objective is to improve the combined performance (runtime performance and diagnosis quality at the same time). We show combined performance results in Figure 3.9-(b). Deviations in the results of *LCO* are more visible than the baseline, because *LCO* has greater performance values. When we zoom into the results of the baseline, we also observe similar deviations due to the variations in problems. As observed, our approach improves combined performance significantly.

3.5.4. Conclusions

In this section, we proposed a novel learning approach for constraint ordering heuristics to solve *the quality-runtime performance trade off problem* of direct diagnosis. We employed matrix factorization for learning based on historical transactions. Taking the advantage of learning from historical transactions, we calculated possible constraint ordering heuristics in offline phase for solving new diagnosis tasks in online phase.

In particular, we applied our constraint ordering heuristics to reorder user constraints of diagnosis tasks and employed a direct diagnosis algorithm FLEXDIAG (Felfernig et al., 2018c) to diagnose the reordered constraints of the diagnosis tasks. The reason to choose this diagnosis algorithm is that *the quality-runtime performance trade off problem* is much more obvious in FLEXDIAG when its *m* parameter is increased. However, our approach can be also applicable to other direct diagnosis approaches (e.g. DIR (Shchekotykhin et al., 2014)). We compared our approach with a baseline: FLEXDIAG without heuristics. According to our experimental results, our approach *LCO* solves the *the quality-runtime performance trade off problem* by improving the diagnosis quality (in terms of minimality) and the runtime performance at the same time.

Even our approach also improves the diagnosis quality in terms of prediction accuracy by learning the preferred diagnoses of users from historical transactions, we could not measure this factor due to the lack of preferred diagnoses information in publicly available real-world configuration benchmarks. As a future work, in order to measure the prediction quality, we can further investigate *LCO*'s prediction accuracy on the basis of a real-world dataset of users preferred diagnoses.

Chapter 4

Improving Recommendation Technologies for IoT Scenarios

Parts of the contents of this chapter have been published in the the journal JIIS'18 *and proceedings of* AIAI'18 *(Felfernig et al., 2018b; Erdeniz et al., 2018b).*

As an emerging topic, the Internet of Things (IoT) (Atzori et al., 2010; Greengard, 2015; Miorandi et al., 2012) represents a networked infrastructure of connected different types of devices. In this context, a huge amount of services and applications is created which makes the identification of the relevant ones a challenging task. One successfully applied domain of IoT applications is health-care.

It is a well-known fact that the average human lifetime is increasing. Living longer implies the risk of age related health problems that reduce significantly the quality of life. Therefore, many people need to improve and maintain their independence, capabilities, health status as well as their physical, cognitive, mental and social wellbeing. Modern mobile and sensor technologies enable the recording of all kinds of data related to a person's daily lifestyle, such as exercises, steps taken, body weight, food consumption, blood pressure, cigarettes smoked, etc. This type of self-data tracking is often referred as the *Quantified-Self (QS)* concept (Swan, 2012). Empowering and motivating people for physical activities is a major challenge. This becomes especially crucial when it comes to the health and the physical condition of an individual (Swan, 2012).

Recent works have shown that tracking measurements such as step counts, spent calories and body weight are very effective in lifestyle changes by motivating a person to engage in physical exercise (McGrath and Scanaill, 2013). Additionally, by tracking measurements over time, he/she gets insights regarding his/her progress and he/she is able to experience the direct relation between his efforts and the actual outcome. For instance, going for jogging twice a week leads to a decrease of body fat percentage (Munson and Consolvo, 2012).

In this chapter, first in Section 4.1 (Felfernig et al., 2018b), we provide an overview of recommendation technologies in IoT . In Section 4.2 (Erdeniz et al., 2018b), we analyze the need of recommendation in QS of AGILE IoT and propose three recommendation approaches to solve problems. Finally, we show the experimental results based on real-world dataset from QS pilot application of AGILE IoT project.

4.1. An Overview of Recommender Systems in IoT

In the IoT domain, recommendation functionalities are required, for example, in IoT workflow development, the recommendation of apps, and domain-specific scenarios such as food recommendation (Valtolina et al., 2014), personalized shopping (Magerkurth et al., 2011), and technology fairs (Munoz-Organero et al., 2010). Example upcoming IoT application domains are *health monitoring*, *animal monitoring*, *enhanced retail services*, *smart homes*, and *sports events* (Felfernig et al., 2016; Greengard, 2015; Leitner et al., 2014; Ray, 2015; Stolpe, 2016). In this section, we show how basic recommenders can be applied in IoT scenarios and propose advanced recommendation approaches for the IoT domain.

Scenarios in AGILE

In health monitoring (so-called *Quantified-Self*) (Erdeniz et al., 2018b; Maglogiannis et al., 2016; Menychtas et al., 2016), users need to know which measuring instruments are needed in their specific context and also how to change personal behaviors (e.g., eating and sports) to improve their situation. The realization of *Quantified-Self* concept requires the integration of several mobile health and IoT elements, where related applications are orchestrated around the IoT Gateway. The gateway connects to the home network and through the gateway's management user interface, the owner has access to all provided features, such as reporting and visualization tools, can manage (store/view/edit) their data and define an access policy to share data with their social network contacts.

In the context of *wildlife animal monitoring*, measuring devices and data collection units (typically drones) have to be selected and parametrized in such a way that the observation area is completely covered, i.e., the needed data can be provided in the required quality. IoT-based *retail services* are developed to support a personalized shopping experience in physical stores. In this context, recommender algorithms help to determine which offers should be recommended to a customer when, where, and in which format. In the context of *smart homes*, recommendation technologies improve the overall applicability of the installed equipment and can also help to optimize the usage of the available resources (e.g., minimizing power consumption). At *large scale sports events* such as marathons or triathlons, recommender systems can help the spectators to determine the current geographical location of certain athletes. This further results in recommended sites at which the athlete can be seen and cheered.

Further Scenarios in IoT

Compared to other recommendation scenarios, IoT-based applications enable a deeper understanding of user preferences and behaviors which can primarily be explained by the availability of heterogeneous information sources (Amato et al., 2013). For instance, *personalized shopping* is a core element of IoT technology based retail environments (Magerkurth et al., 2011). Customers entering a store receive recommendations regarding items and corresponding price offers – these recommendations depend on the condition of the offered items. For example, if the expiration date of some of the offered items is approaching, and this information is detected via their RFID (radio frequency identification) tags (Finkenzeller, 2010), corresponding special offers can be announced to the customer. Important IoT-related aspects are *automated quality control of items*, *context-dependent pricing*, and *targeted product information* (Mashal et al., 2016). The recommendation approach presented in (Magerkurth et al., 2011) follows a knowledge-based (rule-based) paradigm.

Valtolina et al. (Valtolina et al., 2014) introduce a *household scenario* where users ask for recommendations regarding recipes. In the context of an IoT infrastructure, a recommender system does not have to only rely on the preferences of the user but can take into account further information sources. For example, recipe recommendation can take into account the availability of food items in the fridge, personal diet

plans, food consumption information from the last days, planned activities, and also historical data about last day's sports activities. In this scenario, the fridge can read the RFID tags (Finkenzeller, 2010) of items and notify the mobile application over a BLE (bluetooth low energy) (Lee et al., 2007) connection. This availability of orthogonal data sources provided by IoT devices will help to increase the prediction quality of recommendation algorithms. In this scenario, the recommendations are based only on the data of the active user.

In many applications, such as recommending a vacation package, personalized content on a Web site, or a movie, it may not be sufficient to consider only users and items, it is also important to incorporate the *contextual information* into the recommendation process in order to recommend items to users under certain circumstances (Adomavicius and Tuzhilin, 2015). Therefore, some recommender approaches focus on recommending the most relevant items to users by taking into account any additional *contextual information*, such as time, location, or the company of other people (e.g., for watching movies or dining out). *Contextual information* is also important in many IoT use cases. Munoz-Organero et al. (Munoz-Organero et al., 2010) introduce *technology fairs* as a scenario where *context-aware recommender systems* can be applied. In such a scenario, users can receive information about exhibits of relevance and also be informed about lectures to attend depending on their personal preferences. In such scenarios, location tags (e.g. iBeacons (Martin et al., 2014)) are used to provide the location ID to the mobile applications over BLE.

Similar scenarios exist in domains such as museum visits of user groups, where recommendations can take into account aspects, such as time available, accessibility of objects at specific times, and personal interests. Visitors of museums are often overwhelmed by the information available in the space they are exploring. Therefore, finding relevant artworks to see in a limited amount of time is a crucial task. Such context-based recommender systems (Benouaret and Lenne, 2015) for mobile devices adapt to the users profiles and is sensitive to their context (location, time, expertise, etc.). These recommenders improve the visitors' experience and help them build their tours on-site according to their preferences and constraints.

The authors of (Cha et al., 2016; Martino and Rossi, 2016; Yavari et al., 2016) show how basic Artificial Intelligence (AI) approaches such as planning and clustering can be exploited for offering public services in a personalized fashion. They do not include basic recommendation approaches but, for example, scheduling and clustering that empower functionalities such as route planning and recommending points of interest.

Bahirat et al. (Bahirat et al., 2018) present a data-driven approach to the development of a privacy-setting interface for IoT devices. By applying machine learning techniques to an existing dataset of users, a set of "smart" default profiles are generated. Using these smart profiles, privacy settings are recommended to users. The accuracy of their privacy-setting predictions is around 82%.

A Motivating Example: IoT for the Smart Home

The motivation for our study came from the needs of recommender systems for our IoT gateway project *AGILE*. We present a *IoT for the smart home* illustration in Figure 4.1 as the motivating example throughout this section.

In this smart home example, a user (*Alex*) installs an AGILE gateway to make his home smarter. After that, *Alex* buys a gas and a temperature sensor, connects them to his gateway then connects via the local network to the management user interface of his gateway on the web browser of his computer. At this stage, *Alex* needs to receive some *app*, *device* or *communication* protocol (BLE, zigbee, etc.) recommendations according to the overall settings on the gateway. All measured data from IoT devices (motion sensor, light sensor, etc.) are collected on the AGILE gateway over various connection protocols such as 5G, BLE, LORA, and ZigBee. Alex can monitor the collected data or edit gateway settings by connecting to his smart home gateway via WAN/LAN. He can install applications (e.g. fire alarm app) or connect various devices (e.g. gas sensor) to his AGILE gateway to extend his smart home. He can use a cloud service to export his smart home data to utilize powerful and online cloud based applications.

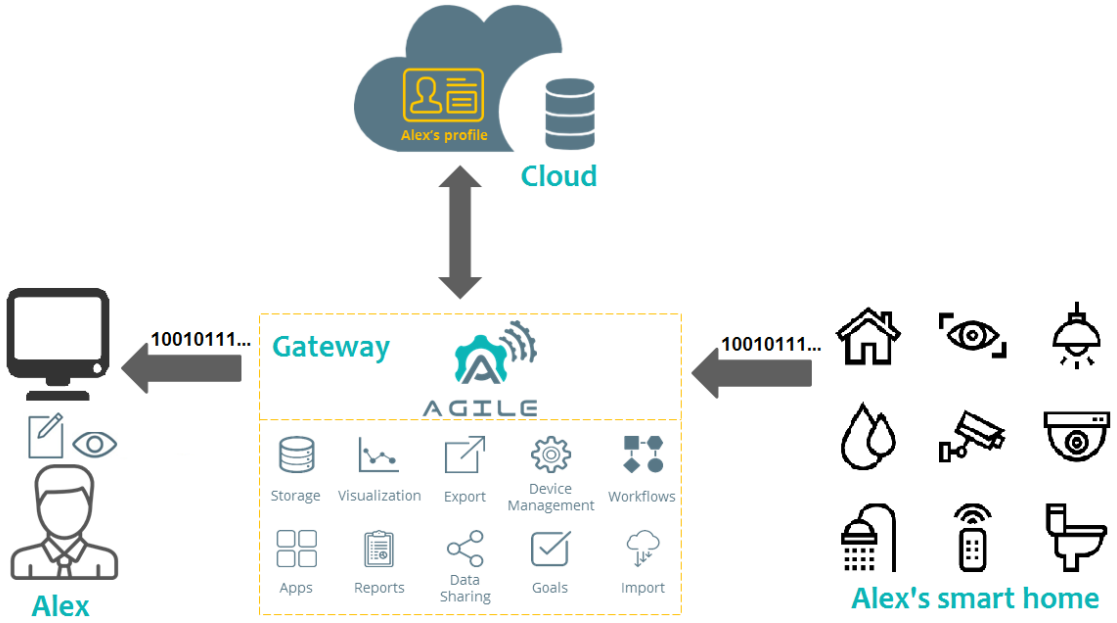


Figure 4.1.: An IoT based smart home on the basis of the AGILE gateway.

Moreover, when Alex wants to extend his smart home with a pollution monitoring system, he needs a special support for choosing the sensors and configuring the system properly. He may also need more help for the workflow development if he has very low know-how about developing workflow with his new pollution monitoring sensors.

We explain how these recommendation needs for *apps*, *devices*, *protocols*, and *workflows/nodes* can be addressed based on Alex's smart home using recommendation technologies in sections 4.1.1 and 4.1.2.

4.1.1. Basic Recommendation Technologies in IoT

In the IoT context, recommender systems can support scenarios such as the recommendation of apps, services, sensor equipment, and IoT workflows (Felfernig et al., 2016). In the following subsections, we show how basic recommendation algorithms can be applied in IoT contexts.

Collaborative Filtering

In order to recommend new apps for *Alex's smart home gateway*, a recommender can provide a list of (additional) apps that can be of relevance on the basis of a given local gateway configuration (e.g., devices and drivers). Collaborative filtering can determine such recommendations on the basis of gateway profile data collected in anonymous form. A simplified example of a related recommendation approach[1] is given in Table 4.1. In this context, information about configurations of other gateways is used to infer relevant apps to be additionally proposed/installed on the *local* gateway. In our example, the devices *temperature sensor* and *gas sensor* are connected to the *Alex's* gateway. The installation bases 1 and 3 (profiles) include all the devices also connected to the local gateway but include additional apps that are currently not installed on the local gateway (these are *fire alarm* and *thief alarm*). Consequently, these apps represent recommendation candidates for *Alex*.

[1] "1.0" denotes the fact that the device is installed on the corresponding gateway (profile).

There are a couple of similarity metrics used in the context of collaborative filtering scenarios for determining nearest neighbors (for details we refer to (Jannach et al., 2010)). We also want to emphasize that the examples provided in this article are using basic recommendation approaches, for example, matrix factorization is a state-of-the-art algorithmic approach to support collaborative filtering. For the purposes of our examples, we introduce a simplified formula that supports the identification of *k-nearest neighbors*[2](see Formula 2.1). If applied to the example of Table 4.1, *profiles* are represented by *items* in Formula 2.1. The apps *fire alarm* and *thief alarm* can be recommended to Alex since they are installed on gateways with the same devices available on the local gateway.

Table 4.1.: Collaborative filtering based app recommendation based on gateway profiles.

Gateway Profiles								
	Devices				Apps			
profile	temp. sensor	motion sensor	gas sensor	camera	temp. alarm	fire alarm	thief alarm	gas alarm
user1	1.0		1.0	1.0			1.0	
user2		1.0					1.0	
user3	1.0		1.0	1.0		1.0	1.0	
Alex	1.0		1.0		1.0			1.0

Alternatively, collaborative filtering can exploit the ratings ([0..5]) of users when interacting with an app marketplace (in this context, *users* have to be associated with the *items* contained in Formula 2.1). The underlying idea is that gateways can be connected to app marketplaces where users can select and download apps that are of interest for their local installation. In this scenario, the evaluation data (ratings) of users serve as a basis for determining recommendations (see Table 4.2).

In Table 4.2, user 1 (the *nearest neighbor*) has provided app evaluations which are similar to those on *Alex's smart home gateway*. The *pollution monitoring* app has been rated by *user-1* but has not been rated by Alex and is therefore recommended. Consequently, a collaborative recommender proposes apps for *Alex* which have been investigated by the nearest neighbor but not by the current user (e.g., the *pollution monitoring* app).

Table 4.2.: Collaborative filtering based app recommendation based on user ratings.

	User Ratings to Apps							
	temp. alarm	fire alarm	thief alarm	gas alarm	heart activity	running app	poll. mon.	dog mon.
user1	1.0		4.5				3.0	
user2		2.0			4.5		2.5	3.5
user3	4.0		3.0			2.5	3.0	
Alex	1.0		4.0	4.0				

[2]For simplicity we assume $k = 1$.

Content-based Filtering

Another alternative for app recommendation is to implement a content-based recommendation approach where apps can be recommended for installation if their required devices (it is assumed that this information is given for each app) are "compatible" with the local gateway configuration (profile).

When applying a content-based filtering based approach, recommended items are determined on the basis of the similarity between the local gateway profile information (e.g., in terms of installed devices) and the profile information of apps available, for example, on a marketplace in the cloud. Similar to collaborative filtering, there are different types of similarity metrics (see, e.g., (Jannach et al., 2010)). For the purposes of our examples, we introduce a simplified formula that supports the identification of, for example, relevant apps for the local gateway.

Table 4.3.: Content-based app recommendation.

	Devices						
Apps	temp. sensor	motion sensor	camera	gas sensor	ZigBee	BLE	WiFi
thief alarm		1.0	1.0			1.0	1.0
fire alarm	1.0	1.0		1.0		1.0	1.0
temp. alarm	1.0	1.0			1.0		
Alex's gateway	1.0			1.0		1.0	

When we apply Formula 2.2, *user* is Alex's gateway, *item* is each app, and features are the devices of each app and the gateway. Therefore, it determines the similarity on the basis of the information about installed devices.[3] However, this approach can be extended to include further information, for example, regarding installed modules and network protocols available on the gateway. In our example of Table 4.3, the *fire alarm* app has the highest similarity with the profile information of *Alex's smart home gateway*, therefore this app is recommended.

Note that content-based recommendation is often applied when the similarity between textual information from different sources has to be determined. Therefore, the approach presented in this article can be extended to the matching of text-based search criteria and the textual description of apps.

Utility-based Recommendation

Modern embedded systems included in IoT scenarios support a rich set of connectivity solutions (e.g., 3G, LTE, TD-LTE, FDD-LTD, WIMAX, and Lora). In this context, recommendation technologies play an important role when it comes to suggesting the best connectivity configurations for the selected communication channel. The recommendation can be based, for example, on location information, available connectivity, performance and reliability requirements, and contractual aspects and costs. Gateway configurations can be manually defined by users but can also be determined on the basis of a configurator that is in charge of keeping the overall system installations consistent. A configurator (e.g., a constraint solver) can determine alternative configurations which have to be ranked. In order to determine a ranking for alternative configurations, a MAUT-based approach can be used. Examples of evaluation *dimensions* (dim) used in MAUT could be *performance*, *reliability*, and *costs* using Formula 2.3. Depending on the current gateway configuration and the usage context, a configurator can determine alternative re-configurations and rank them accordingly.

[3] "1.0" denotes the fact that the device is installed on the corresponding gateway (profile).

An example of the application of a utility-based approach is the following. Table 4.4 includes an example evaluation of connectivity protocols (*BLE* and *ZigBee*) to be used on the gateway. Furthermore, Table 4.5 includes the personal preferences of *Alex*.

Table 4.4.: Utilities of protocols.

protocol	performance	reliability	costs
BLE	9	5	2
ZigBee	5	8	3

Table 4.5.: User preferences w.r.t. *performance*, *reliability*, and *costs* (in between [0..1]).

user	performance	reliability	costs
Alex	1.0	0.3	0.1

In order to recommend a connectivity protocol for *Alex's smart home gateway*, we can apply a utility function (see, e.g., Formula 2.3). When we apply the utility function, *item* stands for a protocol, and dimensions (*dim*) are the protocol utilities. Therefore, *BLE* (utility(*BLE*,Alex)=10.7) is recommended rather than *ZigBee* (utility(*ZigBee*,Alex)= 7.7) to Alex because the utility value of *BLE* for Alex is higher than the utility value of *ZigBee* for Alex.

Group Recommender Systems

In scenarios where a group of users is in charge of making a decision, group recommenders can provide support (Masthoff, 2011). For example, if Alex shares his smart home with two home mates *Bob* and *Tom*, this group of smart home users is in charge of selecting an appropriate smart home solution for the smart home where they live together. User-specific evaluations of different smart home theft protection solutions on Google Playstore[4] are depicted in Table 4.6.

Table 4.6.: Selecting a smart home theft protection solution for Alex and his home mates.

	Group Ratings to Smart Home Apps			
	SalientEye Home Security Alarm	Yale Smart Living Home	At Home Camera Home security video surveillance	Home Security Camera - Alfred
Bob	5.0	4.0	5.0	3.0
Tom	4.0	5.0	3.0	3.0
Alex	4.0	3.0	4.0	3.0

Let's apply the group recommendation algorithm *least misery* (see Formula 2.4) on given group ratings. In Formula 2.4, t is an *item*, I a set of *items*, and *LM* is assumed to return a recommended item for the group. If we apply *least misery* group recommendation, *SalientEye Home Security Alarm* is recommended to the group since the minimum rating for this item is 4.0 (which globally represents the best least misery value for all group members).

[4] https://play.google.com/store/apps

Hybrid Recommendation

Hybrid recommendation (Burke, 2002) is based on the idea of combining basic recommendation approaches in such a way that one helps to compensate the weaknesses of the other.

For example, when combining content-based filtering with collaborative recommendation, content-based recommendation helps to recommend unrated items. If a user has already consumed some items (e.g., purchased some IoT apps), the content description of a new item can be compared with the descriptions of items already purchased by the user. If the new item is similar to some of the already consumed ones (e.g., installed apps), it can be recommended to the user. Combining the recommendations of different algorithms, for example, on the basis of a voting mechanism, can help to significantly increase prediction quality (Jannach et al., 2010).

In the AGILE project, we have developed a hybrid recommender. A workflow recommendation is calculated based on the contents of the active workflow, devices connected to the user's gateway, and other similar gateway profiles (their nodes, workflows, and devices). We have combined the recommendation results of aforementioned basic approaches content-based filtering, and collaborative filtering with the recommendation results of our new approach SEQREQ: Sequences based Recommendation (see Section 4.1.2). As shown in Figure 4.2, the AGILE NodeRed[5] development environment presents recommendation results of workflows and nodes. Hybrid recommendation is applied to recommend workflows and nodes which are the displayed in the *workflow tab*.

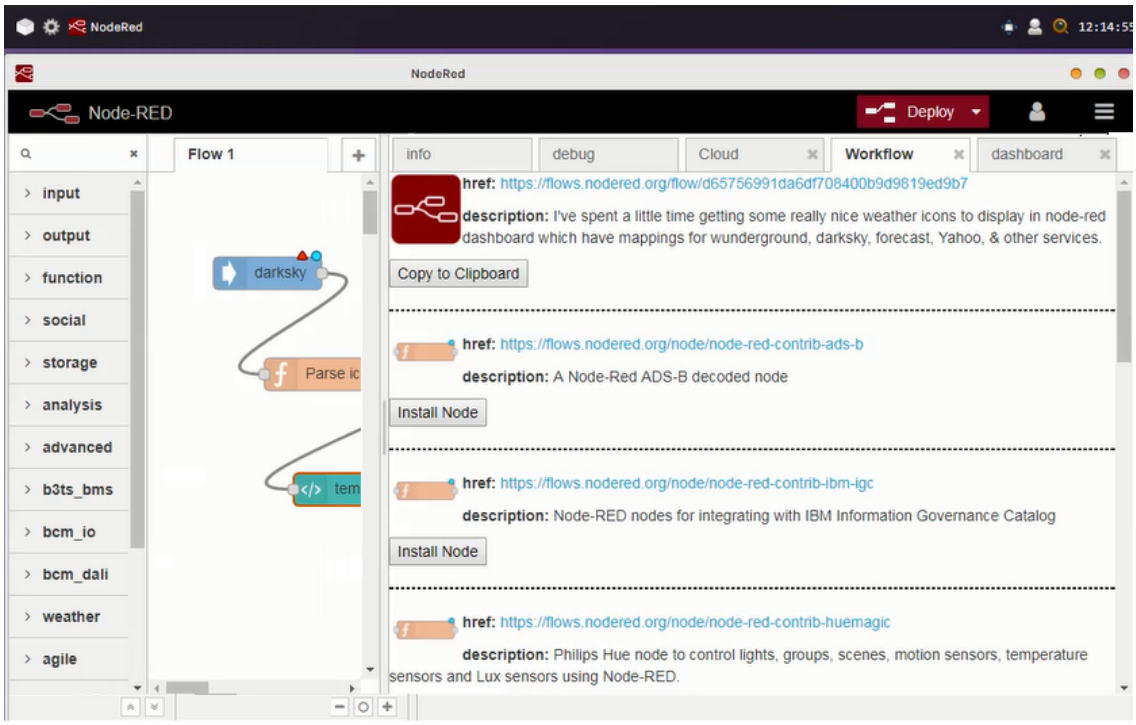


Figure 4.2.: The AGILE Node-Red development environment of *Alex's smart home gateway*.

Node-Red workflows are composed of connected nodes where the order and type of the nodes are important. One node sends its output to the next connected node as an input. For example, in Figure 4.3, the *email* node can not be used before the *openweathermap* node in the workflow because it needs an input (*the*

[5]https://nodered.org/

email text) coming from the connected predecessor *openweathermap* and *function* nodes.[6] The first node *openweathermap* collects the weather forecast online from https://openweathermap.org/ and provides the data to the next the node. The next node *function* (which is named as "*if bad weather*") checks the forecast data whether it is clear or not (rainy, snowy, or stormy). If the forecast says the weather is not clear, this *function* outputs a message to the next connected node. Finally, the next node (*email*) sends the received input message by email to a specified email address.

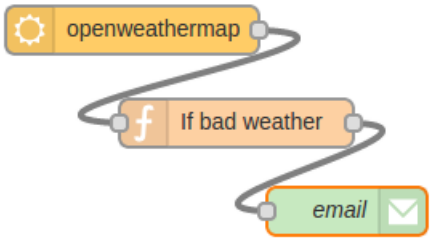


Figure 4.3.: An example sequence of nodes in a workflow.

4.1.2. Advanced Recommendation Approaches in IoT

In this section, we introduce three new recommendation approaches which go beyond the basic ones introduced in Section 3. These approaches have been developed to support specific requirements of the use cases in the AGILE project.

SeqReq: Sequences based Recommendation

In the AGILE project, in situations where sequences of recommendable items play important roles, we required an alternative recommendation approach compared to the basic ones introduced in Section 3. Thus, we have developed SeqReq (Sequences based Recommendation) which recommends items based on sequential pattern mining. In this subsection, we first explain sequential pattern mining, then show on the basis of a node recommendation example from the AGILE Node-Red[7] development environment how SeqReq can be used based (see Figure 4.2).

Sequential pattern mining (Han et al., 2001; Zaki, 2001) is used for finding sequential patterns (sequences). Various algorithms have been proposed to find such patterns. For example, when analyzing user behaviors, a sequential pattern mining algorithm may find that many customers also buy a gas sensor, after having purchased a temperature sensor. Therefore, buying a gas sensor after a temperature sensor is a common sequence as: *[temperature sensor, gas sensor]*.

SeqReq is useful in cases where the sequence of items are important. We explain SeqReq on the basis of a workflow development use case from the AGILE project where sequences (or orders) are very important. In such a setup, we can find many common node sequences in the workflow repository. By searching common sequences of nodes in a workflow repository, we can build a look-up table with sequences an their occurrence frequencies (the number of observations) and distances (the number of links between two nodes). For example, in Figure 4.3, the distance (the number of links) between the nodes *openweathermap* and *email* is 2. If there is another workflow in the repository which includes the nodes *openweathermap* and *email* with a distance 4, then the average distance of the nodes become $(2+4) \div 2 = 3$. Table 4.7 includes an example of the application of pattern mining in the context of workflows and nodes. This

[6] http://developers.sensetecnic.com/article/a-node-red-flow-to-monitor-the-weather/
[7] http://agile-iot.eu/wiki/index.php?title=How_to_develop_an_App

example is using only sequences with two nodes and an active workflow which has any number of (> 1) nodes. *Frequency* is the number of observations of a sequence in the repository and *Distance* is the average distance (number of links) between the nodes in related workflows. AW is the active workflow on *Alex's smart home gateway*.

Table 4.7.: Sequential patterns (sequences) of nodes from a Node-Red repository.

	Nodes	Frequency	Distance	Similarity
sequence-1	[mqtt in, MSSQL]	112	5	22.4
sequence-2	[mqtt in, tingodb]	78	12	6.5
sequence-3	[http out, twitter]	102	8	0
sequence-4	[e-mail, ovh_sms]	27	9	3
sequence-5	[openweathermap, email]	72	3	0
AW	[**mqtt in, http in, email**]	-	-	

The recommender's output is a list of nodes according to Formula 4.1. In this formula, $isFirstNodeInAW_{sequence\text{-}x}$ is 1 if the first node of sequence-x is observed in the active workflow (AW), otherwise it is 0. $Freq_{sequence\text{-}x}$ is the frequency of sequence-x and $Dist_{sequence\text{-}x}$ is the distance between the nodes in sequence-x. When this formula is applied on Table 4.7, the list of the recommended nodes for Alex is {*MSSQL, tingodb, ovh_sms*}. These recommended nodes are from the sequences with $Similarity > 0$.

$$Similarity(\text{sequence-x}, AW) = \frac{Freq_{\text{sequence-x}}}{Dist_{\text{sequence-x}}} \times isFirstNodeInAW_{\text{sequence-x}} \tag{4.1}$$

CONFREQ: Recommendations for Configurators

Existing recommendation technologies focus on simple items, however there is also a need for recommendation technologies for complex items. In the AGILE project, we have developed recommendation technologies to support configuration process for complex products and services. CONFREQ provides recommendations of search heuristics to improve runtime performance of configurators. In this subsection, first we explain the relationship between configuration technologies and constraint satisfaction problems, then show how CONFREQ can be applied on a constraint satisfaction problem from the smart home domain.

The configuration of a new gateway infrastructure requires *configuration technologies* with integrated recommendation functionalities (Falkner et al., 2011). When starting a new configuration, the configuration environment should be able to exploit information about already existing gateway installations in order to reuse some parts of the installation for the new gateway. In this context, recommendation technologies have to be integrated in order to guide search. Similar requirements exist in reconfiguration scenarios where the system has to react on changes in the set of installed applications or connected sensors. In such situations, for example, data transfer protocols have to be adapted in order to optimally take into account changes in the operating environment.

CONFREQ supports CSP solvers by recommending heuristics based on *cluster specific heuristic* (Erdeniz et al., 2017). CONFREQ can support the configurator by recommending *where to start the solution search*. In order to improve the runtime performance of CSP solvers, search is guided by so-called variable and value ordering heuristics. CONFREQ has been introduced to increase the runtime performance of CSP solvers based on learned variable ordering heuristics. CONFREQ clusters past *REQs* using k-means clustering (see Formula 4.2) and calculates variable ordering heuristics for each cluster using a genetic algorithm.

Therefore, the heuristic of a cluster can be used for a new CSP (which is closer to this cluster rather than other clusters). It has been shown that CONFREQ significantly improves the runtime performance of the used CSP solver (Choco Solver (Prud'homme et al., 2016)).

For example, when *Alex* wants to extend his smart home's pool with new IoT based sensors or a IoT based pump, he needs a professional support for installing the suitable devices and configuring them. Therefore, we developed a configuration recommender using CONFREQ to increase the runtime performance of CSP solvers by the help of *the recommendation of heuristics*.

His swimming pool can be configured with one of the available pool pumps as given in Table 4.8. CONFREQ first clusters past user requirements based on K-means clustering.

Table 4.8.: Product table with five types of pool pumps where each has three features.

	pump1	pump2	pump3	pump4	pump5
v1 (power - Watt)	1000	1000	600	600	1200
v2 (price - Euro)	120	140	100	100	160
v3 (size - Centimeter)	12	8.7	14.5	7.2	11

K-means clustering is based on the minimization of distances between the cluster elements and the mean values of clusters as shown in Formula 4.2, in this context k is the number of target clusters, S is a cluster set, μ_{i} is the average value of cluster elements in the S_{i} and x is a cluster element in S_{i}.

$$min \sum_{i=1}^{k} \sum_{x \in S_{\mathrm{i}}} \|Distance(x, \mu_{\mathrm{i}})\|^2 \tag{4.2}$$

Euclidean n-distance is generally applied by K-means clustering as the distance measurement equation between the cluster elements and mean values of clusters as shown in Formula 4.3 where x_{i} is the i^{th} attribute in the cluster element x, y_{i} is the i^{th} attribute in the cluster element y, and n is the number of attributes in one cluster element.

$$Distance(x,y) = \sqrt{\sum_{i=1}^{n} (x_{\mathrm{i}} - y_{\mathrm{i}})^2} \tag{4.3}$$

In our example, we use k-means clustering with *number of clusters: k=2* as shown in Table 4.9 where variables shown with "-", are not assigned by the user. Then, we obtain the clusters as shown in Table 4.10. CONFREQ applies *Min-Max Normalization* (see Formula 4.4) on *REQ*s before clustering.

$$v_{\mathrm{i\_norm}} = \frac{v_{\mathrm{i}} - dom(v_{\mathrm{i}})_{\min}}{dom(v_{\mathrm{i}})_{\max} - dom(v_{\mathrm{i}})_{\min}} \tag{4.4}$$

Table 4.10.: Two clusters generated for six sets of user requirements (see Table 4.9).

cluster1		cluster2	
items	centroid	items	centroid
REQ1,REQ2,REQ3	[1133,100,0]	REQ5,REQ6,REQ4	[0,86.66,2.4]

Table 4.9.: Six sets of user requirements (*REQs*) stored in previous configuration sessions.

REQ1	REQ2	REQ3	REQ4	REQ5	REQ6
v1=1200	v1=1000	v1=1200	v1= -	v1= -	v1=-
v2=160	v2=140	v2= -	v2=100	v2= -	v2= 160
v3= -	v3= -	v3= -	v3= -	v3= 7.2	v3= -

After calculating the clusters, CONFREQ learns cluster specific variable ordering heuristics using supervised learning (Venturini, 1993) based on a genetic algorithm (GA) (Erdeniz et al., 2017) which uses a fitness function (see Formula 4.5). This formula minimizes the total runtime (τ) for finding the first solution for a CSP over all sets of user requirements in a cluster.

$$min(\tau = \sum_{i=1}^{n} runtime(solve(CSP_i))) \tag{4.5}$$

An *individual* is generated by the genetic algorithm in the form of an array which includes all variables of the CSP. According to the defined parameters of the genetic algorithm (the most important ones are: the maximum number of generations, and the mutation rate), it finds an individual which is the best of all individuals in the generated population (see the fitness function Formula 4.5).

The cluster-specific variable ordering heuristics are learned as follows: *h1:[v1,v2,v3]* for cluster1 and *h2:[v1,v3,v2]* for cluster2. *REQ_new={v1=1000}* is a new set of user requirements based on the same pool-configuration problem and it is the closest to cluster1 according to Formula 4.3. Consequently, we use the variable ordering heuristic *h1:[v1,v2,v3]* for solving the corresponding CSP. Using this heuristic, the CSP solver searches for a solution by instantiating the variables in the order of *h1*, which results in the product *pump1:(v1=1000,v2=1200,v3=1200)*. With the help of learned heuristics, a solution can be found in a shorter time. A more detailed discussions of the experimental results of our approach is presented in (Erdeniz et al., 2017).

DIAGREQ: Recommending Diagnoses

In the AGILE project, we also developed recommender algorithms that provide support in *inconsistent situations*. For this purpose, we developed DIAGREQ that provides recommendations for resolving inconsistencies effectively in terms of *runtime* and *accuracy*. In this subsection, first we introduce diagnosis problems, then show how DIAGREQ can be applied on the basis of an example diagnosis problem about pollution-monitoring systems.

In the context of knowledge-based configuration, search interfaces allow the specification of requirements (REQ) and display a solution if the requirements are consistent with the recommendation knowledge base. However, in many cases no solution exists for a given set of requirements and the user needs support in finding a way out of the *"no solution could be found dilemma"* (Jannach et al., 2010).

In this context, diagnoses can be recommended that help to solve inconsistencies. When a constraint set is inconsistent, there is no solution for the corresponding constraint satisfaction problem (CSP) / configuration problem. In this case, a diagnosis algorithm can be applied to find the cause of the inconsistency.

For example, when *Alex* wants to extend his smart home gateway with pollution monitoring sensors, he needs a professional support for installing and configuring these sensors to his home. Therefore, we developed a ramp-up configurator where Alex is responsible for defining the requirements and the ramp-up

configurator finds a solution according to Alex's requirements. The ramp-up configurator uses DIAGREQ to solve inconsistency situations in a reasonable time with a high prediction quality. In this example, the knowledge base is a product catalog (see Table 4.11). Sensors in the product table measure the level of the following pollutants; SO_2: Sulfur Dioxide, NO: Nitric Oxide, PH_3: Phosphine, CO: Carbon Monoxide, H_2S: Hydrogen Sulfide.

Table 4.11.: Five types of air pollutant sensors where each has three features.

	SO_2 sensor	NO sensor	PH_3 sensor	CO sensor	H_2S sensor
v1 (sensitivity - ppm: parts per million)	1000	1000	600	600	1200
v2 (price - euro)	14	12	10	10	16
v3 (size - millimeter)	12	8.7	14.5	7.2	11

Moreover, we know about six previous monitoring station requirement specifications that lead to an inconsistency, i.e., no solution could be found (see Table 4.12). Furthermore, this table includes the information of selected products by other users after changing their initial inconsistent requirements (see Table 4.13).

Table 4.12.: Inconsistent requirements (*REQ*s) collected from previous configuration sessions.

	user1 REQ1	user2 REQ2	user3 REQ3	user4 REQ4	user5 REQ5	user6 REQ6
c1	v1=1200	v1=1000	v1=1200	v1=600	v1=600	v1=1000
c2	v2=10	v2=16	v2=14	v2=12	v2=0	v2=12
c3	v3=12	v3=14.5	v3=11	v3=12	v3=11	v3=7.2

Table 4.13.: Sensors selected by users who install the monitoring station.

user1	user2	user3	user4	user5	user6
SO_2 sensor	SO_2 sensor	NO sensor	H_2S sensor	CO sensor	NO sensor

In order to find a diagnosis recommendation, we developed DIAGREQ (Atas et al., 2017) which can find diagnoses using cluster-specific constraint ordering heuristics that are used by direct diagnosis search. In order to find heuristics, at first, DIAGREQ clusters the inconsistent requirements (see Table 4.12) using k-means clustering. It applies *Min-Max Normalization* (Visalakshi and Thangavel, 2009) on *REQ*s (see Formula 4.4). After clustering the normalized *REQ*s, DIAGREQ uses supervised learning based on a genetic algorithm) to find constraint orderings. An individual generated in the genetic algorithm is represented as array of constraints of the CSP (e.g. *[c2,c3,c1]*). According to the defined parameters of the genetic algorithm (maximum number of generations, mutation rate, etc.), it finds an individual which is the best one among the other individuals in the generated population. This individual has the best fitness value which is calculated using the fitness function (see Formula 4.6 and Formula 4.7).

$$min(\tau = \sum_{i=1}^{n} runtime(\Delta_i)) \tag{4.6}$$

$$max(\pi = \frac{\text{k=\#(correct predictions)}}{n = \text{\#(predictions)}}) \tag{4.7}$$

In supervised learning, runtime and accuracy of a diagnosis are calculated using the requirement specifications in Table 4.12 and selected solutions (see Table 4.13). Based on supervised learning, the correspond-

ing learned constraint orderings are shown in Table 4.14. Runtime (τ) and precision (π) are calculated for each cluster (κ_i).

Table 4.14.: Learned constraint ordering heuristics (H).

κ_1	$H_1\pi$:	{c3, c2, c1}
	$H_1\tau$:	{c2, c3, c1}
κ_2	$H_2\pi$:	{c1, c3, c2}
	$H_2\tau$:	{c1, c2, c3}

Whenever Alex generates a new inconsistent requirements set (see Table 4.15), DIAGREQ can be applied to restore the consistency. First, it finds the closest cluster to this new requirement set which is *cluster1* and applies this cluster's constraint ordering before running the diagnosis algorithm. For example, for a accuracy-efficient solution we apply $H_1\pi$ shown in Table 4.15. Then, DIAGREQ finds the diagnosis Δ. After eliminating Δ from *REQ_new*, *REQ_new_diagnosed* is obtained as shown in Table 4.15. Now a CSP solver can solve the CSP of *REQ_new_diagnosed*. Finally, a CSP solver can find two solutions from the product catalog (see Table 4.11): *PH_3 sensor* and *CO sensor*. Since we used the accuracy-based heuristics to diagnose this problem, the solutions have high probabilities to be accepted by Alex. Related experimental results are presented in our previous work (Atas et al., 2017).

Table 4.15.: Alex generates a new inconsistent set of requirements (*REQ_new*).

	REQ_new	REQ_new_reordered	Δ	REQ_new_diagnosed
c1	v1=1200	v3=10	v3=10	-
c2	v2=10	v2=10	-	v2=10
c3	v3=10	v1=1200	v1=1200	-

Selection of Recommendation Algorithms

The five basic approaches of collaborative filtering (CF), content-based filtering (CBF), knowledge-based recommendation (KBR) (utility-based recommendation is also considered as a subtype of knowledge based recommendation since the utility function is indeed a utility constraint (Felfernig et al., 2010)), and group recommendation (GR) are based on different knowledge sources and also have different strengths and weaknesses – a corresponding overview is shown in Table 4.16.

Table 4.16.: Selection criteria for recommendation algorithms.

	Algorithms			
	Collaborative	Content-based	Knowledge-based	Group
easy setup	yes	yes	no	yes
conversational	no	no	yes	yes
adaptivity	yes	yes	no	no
serendipity effects	yes	no	yes	no
ramp-up problem	yes	yes	no	no
transparency	no	no	yes	no
high involvement items	no	no	yes	yes

Easy Setup. Collaborative filtering and content-based filtering systems are easy to set up since only basic information about item names, descriptions, and graphical representations is needed – the same holds for group recommender systems which rely on pre-defined heuristics to determine recommendations. Knowledge-based recommender systems require a more detailed specification of the recommendation knowledge (represented in terms of attributes, constraints, and/or similarity metrics) and also of the corresponding items (semantic properties have to be specified).

Conversational Approach. Both, group recommender systems and knowledge-based recommender systems are often based on a conversational approach where users have to provide answers to questions (preferences regarding the properties of alternatives) and recommender systems propose solutions (candidate items) which serve as a basis for further user feedback. Critiquing-based recommender systems (Burke et al., 1997) support the specification of critiques which represent user feedback on the properties of an item currently shown to the user. Constraint-based recommender systems allow the specification and re-specification of preferences (similar to the concept of critiques) and then support users in situations where no solution can be identified (Felfernig and Burke, 2008). Collaborative filtering and content-based recommendation approaches are typically not used in the context of conversational scenarios.

Adaptivity. Both, collaborative filtering and content-based recommendation are more adaptive in the sense that new ratings provided by users are automatically taken into account. Knowledge-based recommendation does not support this type of adaptivity since utility schemes (Winterfeldt and Edwards, 1986) are in most of the cases adapted manually, i.e., are not learned. Group recommender systems in their basic form (Felfernig et al., 2018a) do not take new evaluations of items into account.

Serendipity Effects. Serendipity characterizes a situation where a user is confronted with relevant items he/she did not expect. Serendipity effects can be achieved primarily using collaborative filtering and variants thereof (Koren et al., 2009). Since content-based recommendation in its basic from does not take into account the preferences of other users, less serendipity effects can be achieved with this approach. Serendipity effects can be somehow achieved with knowledge-based recommenders, however, in this context serendipity knowledge has to be encoded into the underlying recommendation knowledge base. In critiquing-based systems, this encoding is part of the similarity metrics used to determine new candidate items. Also in basic types of group recommender systems, the serendipity rather depends on the encoding in corresponding group decision heuristics (Masthoff, 2011).

Ramp-up Problems. Ramp-up problems occur if a recommendation algorithm relies on initial information which is sometimes not available. For example, in collaborative filtering, user preferences have to be available in terms of item ratings – if these ratings are not available, no recommendations can be determined. Ramp-up problems primarily exist in the context of collaborative filtering and content-based recommendation. In collaborative filtering, users have to rate items in order to enable the algorithm to determine nearest neighbors. In content-based recommendation, users have to specify which kinds of items are perceived as interesting in order to enable the algorithm identify items with similar characteristics. Knowledge-based recommendation does not have to deal with ramp-up problems since the recommendation knowledge is already pre-specified (in terms of constraints, rules, or similarity metrics). Similarly, group recommenders do not have a ramp-up issue since recommendation calculation is based on predefined decision heuristics.

Transparency. Transparency is a measure that specifies to which extent recommendations can be explained to users. In collaborative filtering, explanations are based on the similarity to nearest neighbors (*this item is recommended since similar users also purchased this one*). Explanations in content-based recommendation scenarios are based on the similarity between the recommended item and those already consumed by the user (*this item is recommended since you purchased similar items in the past*). In both cases, explanations can be regarded as shallow, i.e., do not provide deep insights to the reasons of a specific item recommendation. Group-based recommender systems generate explanations that strongly depend on the used heuristics, for example, *this item is recommended to the group since no misery can be expected by one of the group members* (Felfernig et al., 2017a). The highest degree of transparency can be ex-

pected from knowledge-based recommendation approaches where solutions can be explained on the basis of information gained from the underlying reasoning process. Especially in the context of constraint-based recommendation, it is possible to generate explanations that help to understand as to why no solution could be identified (Felfernig and Burke, 2008).

High Involvement Items. High-involvement items (Felfernig et al., 2017a) are items that are selected and/or purchased in most of the cases after a careful consideration since the impact of suboptimal decisions can be rather high. Examples of related items are IoT based smart homes, IoT based animal monitoring stations, and IoT based pollution monitoring stations. Both, collaborative filtering and content-based filtering are used in most of the cases for recommending low-involvement items such as IoT apps, and IoT sensors. Group recommender systems are exploited for scenarios ranging from decisions such as choosing a smart home alarm system to complex products such as configuring a new pollution monitoring station. Ratings related to high-involvement items are provided less frequently which makes collaborative filtering and content-based recommendation less applicable (for example, preferences regarding an apartment or a car could significantly change over time).

Recommender Libraries. Recommendation algorithms and heuristics are in many cases regarded as a central intellectual property of a company and are therefore often not implemented on the basis of existing recommendation libraries. *Strands*[8] is a commercial recommendation library supporting different types of recommendation algorithms for the retail and finance sector. *MyMediaLite*[9] is a .NET based recommendation library that supports collaborative filtering. *LensKit*[10] is a toolkit from the University of Minnesota that supports different kinds of collaborative filtering algorithms. *Movielens*[11] is a related non-commercial movie recommendation platform – it also provides a couple of publicly available datasets that can be exploited for the evaluation of the predictive quality of recommendation algorithms. *Apache Mahout* (mahout.apache.org) is a machine learning environment that also includes different types of collaborative filtering approaches. *Choco*[12] is an example of an open-source constraint library that can be exploited, for example, for the development of constraint-based recommender applications. Another example of a constraint-based recommendation environment is *WeeVis*[13] that supports the integration of constraint-based recommender applications into Wiki pages. Finally, *Choicla*[14] is a group recommender environment that supports group decision making for non-configurable items.

Further Research Issues

Scalability of Algorithms. In some scenarios, recommendation algorithms can be deployed in the cloud which has no serious limitations regarding computational resources. Typical examples of such a setting are the recommendation of IoT apps (e.g., located on some sort of marketplace) and the recommendation of workflows (e.g., located in a workflow repository). Recommendation functionalities that support the task of resource balancing or functionalities supporting the reconfiguration of a gateway installation should be located directly on the gateway in order to be able to perform reconfigurations even in the case that the gateway is not connected to the Internet. Despite limited computational resources available on gateways, recommendations have to be determined in an efficient fashion (Felfernig et al., 2016).

Datasets for Evaluation Purposes. The development of recommendation technologies for IoT scenarios is a rather young discipline and research in the field would strongly profit from the availability of more IoT datasets that enable corresponding tests of, for example, the prediction quality of recommendation algorithms. In the context of end-user development support in IoT scenarios, datasets are helpful that

[8] strands.com
[9] mymedialite.net
[10] lenskit.org
[11] movielens.org
[12] choco-solver.org
[13] weevis.org
[14] choiclaweb.com

include logs about the development of IoT workflows on remote gateway installations. This information can be exploited to optimize user support, for example, by predicting relevant code-fragments and sensors that should be included.

Distributed Data Analysis. The distributed nature of the Internet of Things and corresponding high amounts of collected data are challenging existing data analysis methods (Stolpe, 2016). While approaches to big data analytics (Chen et al., 2015) often follow the paradigm of parallel and high-performance computing, analysis approaches in IoT scenarios are often limited, for example, in terms of bandwidth and energy supply. This is the major motivation for decentralized analysis algorithms that often have to work (partly) on data-generating IoT devices.

Context-Aware Recommendation Approaches. Compared to traditional context-based recommendation approaches, IoT scenarios increase the number of relevant context dimensions (Felfernig et al., 2016). For example, in group-based scenarios (e.g., a group of tourists interested in a city round trip recommendation) example dimensions are not only related to the items to be recommended (e.g., tourist destinations) but also to additional dimensions such as information about potential traffic jams, weather forecasts, occupancy rates of destinations, and availability of public transport (just to mention a few). All these aspects have to be taken into account when building recommendation solutions which also requires the integration of data sources. Recommendation and configuration technologies supporting the ramp-up of IoT infrastructures have to take into account additional aspects such as topological information relevant for the IoT environment (e.g., in the case of animal monitoring applications) and environmental data (e.g., in the context of air pollution monitoring). Such aspects are not relevant in more traditional recommendation and configuration scenarios (Felfernig et al., 2014b).

4.1.3. Conclusions

In this section, we have provided an overview of existing recommendation approaches besides our proposed recommendation techniques in the Internet of Things (IoT) domain. First, we have given a short overview of existing work related to the application of recommendation technologies in IoT scenarios. Thereafter, we have shown how basic recommendation algorithms can be applied in simple IoT scenarios. Moreover, we have described the challenges that we have faced in the AGILE project. To come over these challenges, we have proposed three recommendation approaches SEQREQ, CONFREQ, and DIAGREQ. SEQREQ provides intelligent workflow/node recommendations whereas CONFREQ and DIAGREQ increases runtime performance and prediction quality of CSP solvers. We have shown how these new approaches can be applied in AGILE project's use cases. After that, we have explained how to select a recommendation approach based on the application domain. Finally, as future work, we discussed further research issues of recommender systems in IoT.

4.2. Recommender Systems for IoT Enabled Quantified Self Applications

Several IoT solutions have been proposed to implement various *Quantified-Self* applications in the areas of health care and assisted living (Wei, 2014; Maglogiannis et al., 2016; Menychtas et al., 2016). The main concept behind AGILE IoT is to enable users to easily build IoT applications and control connected devices through a modular IoT gateway and a set of full stack (OS, runtime and applications) IoT software components. One of five pilot projects of AGILE is *Quantified-Self* which is an IoT enabled m-health (mobile health) system based on the AGILE gateway environment.

In the AGILE project, we have developed new *recommendation approaches* especially useful in IoT scenarios (Valdez et al., 2016; Felfernig et al., 2016, 2017b). *Recommender systems* (Jannach et al., 2010)

suggest items (alternatives, solutions) which are potential interests for a user. Examples of related questions are: *which book should be purchased?*, *which test method should be applied*?, *which method calls are useful in a certain development context?* or *which applications are of potential interest for the current user?* A recommender system can be defined as *any system that guides a user to interesting or useful objects for the user in a large space of options or that produces such objects as output* (Felfernig and Burke, 2008).

In this section, we propose three new recommendation approaches on the basis of *Quantified-Self*:

- *Virtual Coach*,
- *Virtual Nurse*, and
- *Virtual Sleep Regulator*.

Virtual Coach is developed to motivate the users of *Quantified-Self* by recommending new activities on the basis of their demographic information and past activities. On the other hand, *Virtual Nurse* helps chronic patients for reach their targets by recommending an activity plan on the basis of their medical history. We have implemented our proposed *Virtual Coach* approach and evaluated the test results of *Virtual Coach* based on our real-world dataset. *Virtual Sleep Regulator* provides walking and sleeping time recommendations for insomnia patients in order to improve their sleep qualities. Insomnia (Fox, 1999; Zammit et al., 1999) is a sleep disorder that is characterized by difficulty falling and/or staying asleep. These people can not have a good quality sleep easily.

Quantified Self

The Quantified Self (QS) concept refers to the use of technologies for collecting data about peoples' daily activities. Smartphone apps, physical activity trackers, biometric sensors, and IoT devices allow people to monitor important aspects of their daily lives, such as their physical activity, heart rate, and mood, with the aim to learn more about themselves, improve their well-being and adopt a healthier lifestyle. In this context, the user is capable of not only storing their data to a location of their choice, such as their own server or private cloud, but also sharing their data with whomever they choose, such as their social circle or their physician. However, the process of sharing this data remains complex, mainly due to the fact that each vendor uses different communication mechanisms and requires a separate application for the persistent storage and visualization of data.

QS is targeting data acquisition on aspects of a person's daily life in terms of inputs (e.g. food consumed, quality of surrounding air), states (e.g. mood, arousal, blood oxygen levels), and performance (mental and physical activities) through a modern, health centric, social and mobile enabled, communication platform that resides in the gateway (in terms of collecting and visualizing data). The application is developed using the AGILE environment and uses the communication modules of the gateway to collect data from self-tracking devices of users: wristbands or smart watches, weighting scales, oximeters, blood pressure monitors, etc. In addition, the cloud integration modules are exploited for periodically importing activity data and biosignals from other providers and applications through their public APIs (Menychtas et al., 2016; Panagopoulos et al., 2017).

According to Swan et al. (Swan, 2013), QS is starting to be a mainstream phenomenon as 60% of U.S. adults are currently tracking their weight, diet, or exercise routine, and 33% are monitoring other factors such as blood sugar, blood pressure, headaches, or sleep patterns. Further, 27% of U.S. Internet users track health data online, 9% have signed up for text message health alerts, and there are 40,000 smartphone health applications available. Diverse publications have covered the quantified self movement and it was a key theme at CES 2013, a global consumer electronics trade show. Commentators at a typical industry conference in 2012, Health 2.0, noted that more than 500 companies were making or developing self-management tools, up 35% from the beginning of the year, and that venture financing in the commensurate period had

risen 20%. At the center of the quantified self movement is, appropriately, the Quantified Self community, which in October 2012 comprised 70 worldwide meet-up groups with 5,000 participants having attended 120 events since the community formed in 2008[1]. At the "show-and-tell" meetings, self-trackers come together in an environment of trust, sharing, and reciprocity to discuss projects, tools, techniques, and experiences. There is a standard format in which projects are presented in a simplified version of the scientific method, answering three questions: "What did you do?" "How did you do it?" and "What did you learn?" The group's third conference was held at Stanford University in September 2012 with over 400 attendees. Other community groups address related issues, for example Habit Design (www.habitdesign.org), a U.S.-based national cooperation for sharing best practices in developing sustainable daily habits via behavior-change psychology and other mechanisms (Swan, 2013).

A variety of quantified self-tracking projects have been conducted, and a few have been selected and described here to give an overall sense of the diverse activity. One example is design student Lauren Manning's year of food visualization, where every type of food consumed was tracked over a one-year period and visualized in different infographic formats[2]. Another example is Rosane Oliveira's multiyear investigation into diabetes and heart disease risk, using her identical twin sister as a control, and testing *vegan dietary* (not consuming animal products, not only meat but also eggs, dairy products, etc.) and metabolism markers such as insulin and glucose[3].

The range of tools used for QS tracking and experimentation extends from the pen and paper of manual tracking to spreadsheets, mobile applications, and specialized devices. Standard contemporary QS devices include pedometers, sleep trackers and fitness trackers. The Quantified Self web site[4] listed over 500 tools, mostly concerning exercise, weight, health, and goal achievement. Unified tracking for multiple activities is available in mobile applications such as Track and Share[5] and Daily Tracker[6]. Many QS solutions pair the device with a web interface for data aggregation, infographic display, and personal recommendations and action plans. At present, the vast majority of QS tools do not collect data automatically and require manual user data input. A recent emergence in the community is tools created explicitly for the rapid design and conduct of QS experiments, including the Personal Analytics Companion (PACO)[7] and studycure[8].

Consequently, QS projects are becoming an interesting data management and manipulation challenge for big data science in the areas of data collection, integration, and analysis. Therefore, recommender technologies are also becoming very important required in many QS applications.

State-of-the-art

IoT-based applications enable a deeper understanding for recommender systems which can primarily be explained by the availability of heterogeneous information sources (Amato et al., 2013; Frey et al., 2015; Yao et al., 2016). Thanks to this ongoing IoT revolution, huge amounts of data are being collected in clinical databases representing patients' health states. Sensor-based internet-enabled devices equipped with radio frequency identification (RFID) (Want, 2006) tags and other communication enablers (Chen et al., 2012) are opening up exciting new ways of innovative recommendation applications in the health domain. Hence, required digital information is already available for patient-oriented decision making. This means, when this data can be used by recommendation algorithms, important results can be obtained (Chen et al., 2012).

Morales-Torres et al. (Casino et al., 2015) show how recommender systems could be used to provide healthcare services within the context of a smart city in which citizens collaborate with the city to improve

[1] http://quantifiedself.com/
[2] https://flowingdata.com/2011/06/29/a-year-of-food-consumption-visualized/
[3] http://quantifiedself.com/2014/04/rosane-oliveira-quantified-double-self/
[4] http://quantifiedself.com/guide/
[5] www.trackandshareapps.com
[6] www.thedailytracker.com/
[7] https://quantifiedself.appspot.com/
[8] http://studycure.com/

their quality of life. It is observed that many citizens perform physical activities in the city, namely walking, running, cycling, etc. With the aim to promote these healthy habits, it would be desirable to count with a system that could dynamically adapt to the needs of the citizens. The system would consider real-time constraints and information from several sources: (i) citizens' preferences, (ii) citizens' health conditions and, (iii) real-time information provided by the smart city infrastructure. They propose the design of a system that fulfills the following properties:

- Citizens can obtain recommendations of routes that best fit their needs and preferences using regular smartphones. No other special devices are required.
- The system will be dynamic and collaborative, and it will adapt to real-time environmental changes.
- Citizens will be allowed to contribute with their sensing capabilities, knowledge and experience to the system. Also they can inform about dangerous situations.
- Citizens can provide the system with new routes.

In the area of patient health monitoring, Sharma and Kaur (Sharma and Kaur, 2017) discuss how web-based tools can be used for dissemination of health related information and for providing a better quality of care to patients. It concludes that patients are more probable to follow advice from peers and patients with similar diseases.

Anumala and Busetty (Anumala and Busetty, 2015) propose a distributed health platform using IoT devices. User health goals are specified and home smart appliances (e.g., microwave oven, smart TV, etc.) are all involved in monitoring the user health goals. However, their model does not support users in decision making.

Datta et al. (Datta et al., 2015) propose the application of IoT for personalized healthcare in smart homes. An IoT architecture is presented which enables such healthcare services. Continuous monitoring of physical parameters and processing of the medical data form the basis of smarter, connected and personalized healthcare. The core functionalities of the IoT architecture are exposed using Restful web services.

On the other hand, we also observe recommender systems in traditional healthcare systems (without IoT usage). These approaches (Hu et al., 2016; Valdez et al., 2016; Schäfer et al., 2017) generally use the data stored in the centralize personal health records (PHR). PHR management systems may fail to satisfy the individual medical information needs of their users. Personalized recommendations could solve this problem. In (Wiesner and Pfeifer, 2010), a ranking procedure based on a health graph is proposed which enables a match between entries of a PHR management system and health information artifacts. This way, the user of such a system can obtain individualized health information he might be interested in. Nursing care plan recommender systems can provide clinical decision support, nursing education, clinical quality control, and serve as a complement to existing practice guidelines (Duan et al., 2011). Based on rule-based expert system, recommending clinical examinations for patients or physicians is also possible (Pattaraintakorn et al., 2007).

Recent researches also include recommender systems in QS applications. Schafer (Schäfer, 2016) proposed a decision support system that engages users with in- and output functionalities, such as gamified/ automated data insertion and intrigues him with avatar-based self-quantification or explanations. It also motivates users by personalizing on specific user needs and applying social pressure to ensure long term habit development. The system provides both crowd based and expert based recommendations, as well as hybrid recommendations tailored to the users needs and contexts.

Our approaches differ from all cited above in that we aim to achieve an effective decision making system in QS applications by utilizing both physical activity and health monitoring data. Our proposed recommendation approaches can answer a wide range of QS user questions, such as *which new biosignal sensors to buy?, which new apps to install?, when to go for a walk?, how long to go for a walk?, when to go to*

sleep?. This means, our proposed approaches are helping QS application users in multi-dimensions. For example, we propose a recommender called *Virtual Sleep Regulator* which recommends a *daily walking and sleeping plan* for *insomnia patients* (people with sleep disorder) based on their and similar users' QS data. As far as we know, there is not such a QS application based recommendation approach up to now to support these *insomnia patients*. Therefore, our proposed recommender approaches aims to carry the *recommendation systems in QS applications* one step further than the state-of-the-art.

4.2.1. Problem Definition

We describe and test our recommendation approaches on the basis of *AGILE Quantified-Self* (see Fig. 4.4). Within the frame of *AGILE Quantified-Self*, AGILE gateway addresses this issue by creating a single point of communication for these devices, to facilitate their integration into QS concept, and provide advanced functionalities for the utilization and secure sharing of the acquired data, demonstrating the applicability of AGILE in home/personal use (Menychtas et al., 2017).

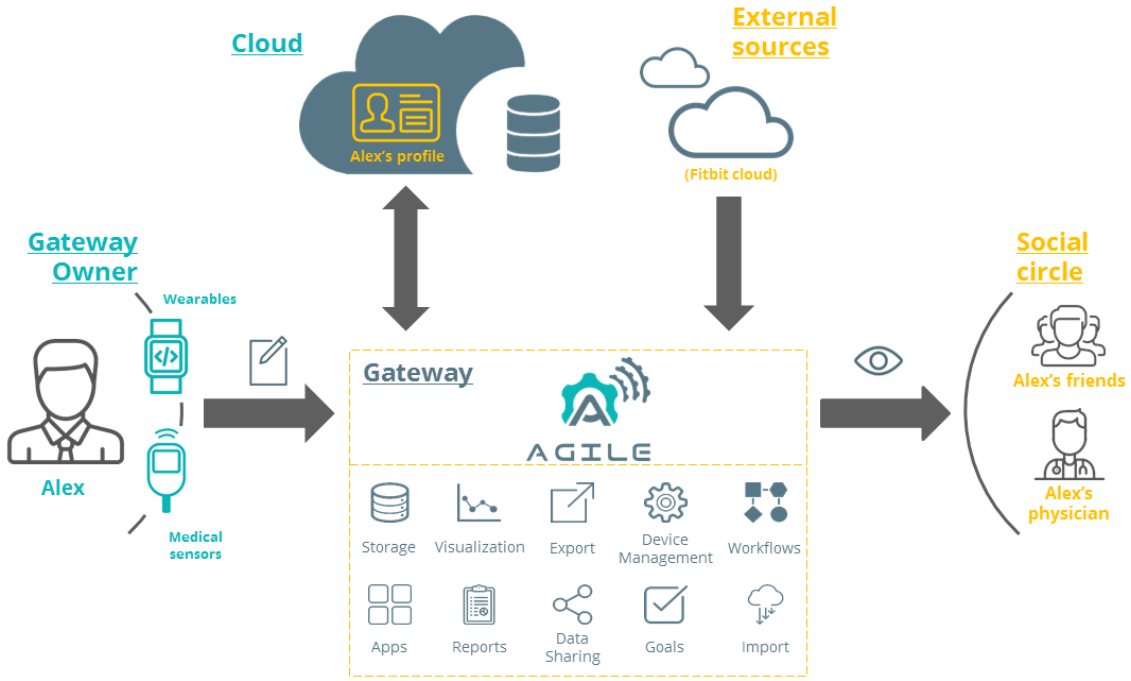


Figure 4.4.: Software architecture of *Quantified-Self* on the basis of the AGILE gateway.

As illustrated in Fig. 4.4, each user of the *Quantified-Self* application is provided with a set of activity tracking devices and biosignals sensors (such as oximeters[9], blood pressure monitors[10] or glucometers[11]), to monitor their daily physical activity and physical condition. All activity data and biosignal measurements are stored locally, on the user's gateway. Users are able to visualize and manage their data, create reports and export the data from the gateway. Furthermore, they can even import past data from other cloud services they might have used before, such as Fitbit[12], GoogleFit[13], etc. In parallel, motion and lifestyle data can be processed and analyzed on the gateway, so as personalized recommendations are sent to the user's smart

[9] https://en.wikipedia.org/wiki/Pulse_oximetry
[10] http://bestreviews.com/best-blood-pressure-monitors
[11] https://en.wikipedia.org/wiki/Glucose_meter
[12] https://www.fitbit.com/at/home
[13] https://www.google.com/fit/

phone in order to encourage them to reach their physical activity goals. Moreover, users are able to share their activity data and achievements with certain people/users from their social circle, including relatives, friends and most importantly their physicians.

The main dataset consists of the "biosignal" measurements of patients and elders, which are acquired directly from the biosignal sensors that users are equipped with. Currently supported biosignal sensors are the following: *activity trackers, pulse oximeters, blood pressure monitors, weighing scales, spirometers, glucometers, and thermometers*. These sensors measure the following biosignal types: *step count, heart rate (bpm), oxygen saturation (%), blood pressure (mmHg), FEV1 (L), peak flow (L/min), blood glucose levels (mg/dL), and body temperature (Celsius)*. It should be noted that the modular design that has been followed, enables the potential integration of additional devices in the same application, which may require different operational workflows and communication patterns.

The biosignals' dataset includes approximately half million records of the aforementioned biosignals from one hundred users in a time frame of three years. Besides the biosignal data, the knowledge base includes information for users' demographics (gender, age, location), as well as their personal health record (lab results, medication and allergies). In order to ensure the smooth communication between the different components of the system (and for the smooth data integration with external systems), well established data models have been used. Therefore, in the proposed approach, all components and workflows which require data exchange and/or storage follow the *Fast Healthcare Interoperability Resources Specification (FHIR)*[14].

4.2.2. The Proposed Method

The utilization of recommendation technologies is essential for improving the health conditions of individuals. Users of *IoT enabled m-health applications* can get recommendations for new activity plans, IoT enabled m-health devices, applications, healthy nutritions. All these recommendations help users to enhance their life style and reach their targeted health conditions easier than before. These recommendations indeed play an important role like a *personal trainer* or *personal coach*.

We explain our proposed three recommendation approaches for *IoT enabled m-health applications* based on our QS application *Agile Quantified-Self* in the following subsections.

Virtual Coach

In order to motivate subscribers/users for sport activities, *Virtual Coach* collects demographic information (age, location, physical condition, medical history, chronic diseases, etc.) of each user. It stores user profiles on its online server and a recommender engine calculates the similarities between users based on their demographic data. Using the similar users' information, a new activity plan (how often, what to measure, which activities) or a new IoT device (a wristband, step counter watch, etc.) can be recommended to users.

In this case, a recommender engine uses *collaborative filtering* as the recommendation technology to find similarities between users. *Virtual Coach* recommends new activities or new devices to users based on these similarities.

[14] https://www.hl7.org/fhir/

Table 4.17.: Profiles of QS Users.

Profiles of Users							
	Demographics				Devices		
	age	gender	location	diseases	oximeter	wristband	BPM
$user_1$	young	male	urban	asthma	✓	✓	-
$user_2$	middle	female	suburban	diabetics	-	-	✓
$user_3$	elder	male	suburban	diabetics	-	✓	-
$user_x$	young	female	urban	asthma	-	✓	-

There are several similarity metrics in the context of collaborative filtering scenarios for determining nearest neighbors (Jannach et al., 2010). For the purposes of our example, we use a simplified formula that supports the identification of *k-nearest neighbors* (For simplicity we assume $k = 1$) (see Formula 4.8).

$$similarity(user_a, user_b) = \frac{1}{1 + \Sigma^{n}_{property=1} |eval(user_a) - eval(user_b)|} \tag{4.8}$$

When Formula 4.8 is applied to the example of Table 4.17, *property* in Formula 4.8 is standing for *demographics* and *devices* of users. Thus, for the seven properties (age, gender, location, chronic diseases, oximeter, wristband, BPM (hearth beats per minute) device) of each user, the calculation result of *eval(user_a) - eval(user_b)* for the i^{th} property is 0 if their values are same, otherwise it is 1. For instance, for the first property *age*, the calculation result of *eval($user_1$) - eval($user_2$)* is 1, because age of $user_1$ is *young* whereas age of $user_2$ is *middle*. Since they do not have the same values, their difference is 1. For another instance, for the sixth property *wristband*, the calculation result of *eval($user_1$) - eval($user_2$)* is 1, because the usage of oximeter of $user_1$ is ✓ whereas for $user_3$ it is ✓. Since they have the same values, when $property = 5$, the result of *eval($user_1$) - eval($user_3$)* is 0.

In order to find a recommendation for the active user $user_x$ in Table 4.17, we first find the *nearest neighbor* based on demographics. $user_1$ is the most similar user to the active user $user_x$. Consequently, a collaborative recommender suggests new devices to the current user which have been used by the nearest neighbor (e.g., an *oximeter* device is recommended to $user_x$).

Virtual Nurse

Virtual Nurse motivates different types of chronic patients (diabetes, asthma, cancer, cardiovascular, etc.) to reach their targets on the basis of a recommended plan. It collects the measured data of patients and checks their health conditional targets. If the measured values are very far from their expected (target) values, then some specific recommendations can be placed for those patients. The recommendations should be personalized and related to the base line of each user's data, which will be permanently updated.

The patient medical data includes the personalized models that shows the behavioral responses of the patients versus the coach interventions to discover best-practices and measure adherence. The recommender could act as a decision support system that gathers information from the patient, finds an activity plan that matches with the available objectives and offers a personalized list. A physician might intervene (semi-supervised recommendation) to select the collected information that are more related with the well-being of the patient. In this case, the recommender engine uses *Content-based Filtering* as the recommendation technology to find a related plan based on the user data. *Virtual Nurse* recommends new activity plans to users based on their actual and expected measurements.

Table 4.18.: Actual and expected (targets) measurements of patient-1.

Medical data of a patient and possible activity plans			
	sys. blood pres. (mm Hg)	heart-rate (bpm)	weight (kg)
targets of plan-1	$\downarrow$	$\downarrow$	$\downarrow$
targets of plan-2	$\downarrow$	$\longleftrightarrow$	$\downarrow$
targets of plan-3	$\longleftrightarrow$	$\downarrow$	$\longleftrightarrow$
targets of plan-4	$\longleftrightarrow$	$\longleftrightarrow$	$\uparrow$
targets of patient-1	120.00 ($\downarrow$)	70.00 ($\downarrow$)	80.00 ($\downarrow$)
actuals of patient-1	142.00	91.00	108.00

A simplified example of a related recommendation approach is given in Table 4.18. The targets of the available activity plans are also represented. Arrows denote increase($\uparrow$)/decrease($\downarrow$)/stay($\longleftrightarrow$) targets of a plan. For instance, plan-2 is targeting to decrease the blood pressure and weight. Therefore, plan-2 includes activities which can decrease these two parameters. When applying a content-based filtering based approach, recommended items (plans) are determined on the basis of the similarity of the patient's targets and available plans. Similar to collaborative filtering, there are different types of similarity metrics (Jannach et al., 2010). For the purposes of our examples, we introduce a simplified formula that supports the identification of, for example, relevant plans for the patient-1 (see Formula 4.9).

$$similarity(patient, plan) = \frac{\#(targets(patient) \cap targets(plan))}{\#(targets(patient) \cup targets(plan))} \tag{4.9}$$

Formula 4.9 determines the similarity on the basis of the targets of plans and *targets of patient-1*. For instance, the similarity between *targets of patient-1* and *targets of plan-3* is calculated as 0.33 where $\#(targets(patient-1) \cap targets(plan-3)) = 1$ since there is only one common target which is decreasing the hearth-rate and $\#(targets(patient-1) \cup targets(plan-3)) = 3$ where all targets include systolic blood pressure, hearth-rate, and weight. In our example of Table 4.18, *targets of plan-1* has the highest similarity with *targets of patient-1*, therefore *targets of plan-1* is recommended to *patient-1*.

Virtual Sleep Regulator

Chronic insomnia, defined as difficulty initiating or maintaining sleep, awakening too early in the morning, or non-restorative sleep, is the most common sleep disorder among adults. Though exercise has long been assumed to improve sleep, surprisingly little research has been conducted on the effect of exercise on chronic insomnia.

Related studies (Guilleminault et al., 1995; Passos et al., 2010; Reid et al., 2010; Passos et al., 2011) show that exercise significantly improves the sleep of people with chronic insomnia. The only study that looked at the effects of a single exercise session found that about of moderate-intensity exercise (e.g., walking) reduced the time it took to fall asleep and increased the length of sleep of people with chronic insomnia compared to a night in which they did not exercise. However, in the same study, vigorous exercise (e.g., running) or lifting weights did not improve sleep. Similar results have been found for studies that examined the effects of long-term exercise on sleep in adults with insomnia. In these studies, after 4 to 24 weeks of exercise, adults with insomnia fell asleep more quickly, slept slightly longer, and had better sleep quality than before they began exercising[15].

[15] https://www.sleepfoundation.org/ask-the-expert/how-does-exercise-help-those-chronic-insomnia

Virtual Sleep Regulator helps insomnia patients to improve their sleep qualities. It uses collaborative filtering techniques to recommend an appropriate walking/sleeping plan for the patients.

4.2.3. Experimental Evaluation

We used collaborative filtering approach to recommend a *steps & sleep* plan for users based on the recommendable activities of other users'. As aforementioned, the first step of a collaborative filtering recommender is to identify the *k-nearest neighbors* (*k* represents the number of users with similar ratings compared to the current user) and to extrapolate from the ratings of these users the preferences of the current user.

We have used the collaborative filtering recommendation library of *Apache-Mahout* (Schelter and Owen, 2012). Apache Mahout is an Apache-licensed, open source library for scalable machine learning. It is well known for algorithm implementations that run in parallel on a cluster of machines. Besides that, Mahout offers one of the most mature and widely used frameworks for non-distributed Collaborative Filtering. We give an overview of this framework's functionality, API and featured algorithms. At the heart of Collaborative filtering applications lie user-item interactions. Mahout models those as a *<user,item,value>* triple. Mahout's recommenders expect interactions between users and items as input. The easiest way to supply such data to Mahout is in the form of a textfile, where every line has the format *<user,item,value>*. Here userID and itemID refer to a particular user and a particular item, and value denotes the strength of the interaction (e.g. the rating given to a movie).

An example of Mahout data file is shown in Table 4.19. Based on the ratings of users for the movies, for the user with userID=2, the movie with the itemID=12 can be recommended (due to the similarities between the ratings of users userID=1 and userID=2).

Table 4.19.: An example *<user,item,value>* Dataset for Mahout: Movie Ratings Dataset

user ID	item ID	rating (value)
1	10	1.0
1	11	2.0
1	12	5.0
1	18	5.0
2	10	1.0
2	15	5.0
2	18	5.0
3	10	5.0
3	14	5.0

Dataset

From 26 users, during six months, we have collected *number of steps* (see Table 4.20) and *quality of sleep* (see Table 4.21) data. In Table 4.20, each row is identified with a unique Activity ID and the duration of each activity is 5 minutes. Number of steps are the taken steps during a 5 minutes period which starts at given Date / Time.

Table 4.20.: Example data from Steps Dataset

Activity ID	User ID	Gender	Age	Date / Time	Number of Steps
as1	u1	female	27	2018-05-21 / 15:05:00	267
as2	u2	male	51	2018-05-21 / 15:05:00	19
as3	u1	female	27	2018-05-21 / 15:10:00	267
as4	u2	male	51	2018-05-21 / 15:10:00	19

In Table 4.21, each row is identified with a unique Activity ID and the duration of each activity is 5 minutes. Quality of Sleep is the measured sleep quality during a 5 minutes period which starts at given Date / Time. Sleep Quality domain is [-10,-20,-30,-40] where -40 is the highest quality (the deepest) sleep whereas -10 is the lowest quality (the lightest) sleep.

Table 4.21.: Example data from Sleep Quality Dataset

Activity ID	User ID	Gender	Age	Date / Time	Quality of Sleep
aq1	u1	female	27	2018-05-21 / 23:50:00	-30
aq2	u2	male	51	2018-05-21 / 23:50:00	-30
aq3	u1	female	27	2018-05-21 / 23:55:00	-40
aq4	u2	male	51	2018-05-21 / 23:55:00	-30
aq3	u1	female	27	2018-05-22 / 00:00:00	-40
aq4	u2	male	51	2018-05-22 / 00:00:00	-30

Based on these two datasets, we generated a <*user,item,value*> triple style dataset (see Table 4.22). In order to do this, we have defined attributes of users as items. All the item values are scaled in [1..5]. For example, gender of user is defined as item ID=1 with values [1.0 (if gender=female), 3.0 (if gender=other), 5.0 (if gender=male)]. Age of user is defined as item ID=2 with values [1.0 (if age<20), 2.0 (if 20<=age>40), 3.0 (if 40<=age>60), 4.0 (if 60<=age>80), 5.0 (if age>=80)]. Other items are the steps and sleep results of users during a day which are encoded according to Table 4.19 with values [1.0 (if sleep quality>-10), 2.0 (if -10>=sleep quality<-20), 3.0 (if -20>=sleep quality<-30), 4.0 (if -30>=sleep quality<-40), 5.0 (if sleep quality=<-40)] where 5.0 means the deepest sleep and 1.0 is the lightest.

Table 4.22.: Steps/Sleep Quality Dataset

user ID	item ID	rating (value)
1	1	1.0
1	2	3.0
1	1001102600	4.0
1	1000010820	4.0
1	1010141710	4.0
1	1000011701	4.0
1	1100100700	5.0
19	1	1.0
19	2	1.0
19	1020000700	5.0
19	1000112510	4.0
19	1030010800	5.0
20	1	1.0
20	2	1.0

As shown in Table 4.23, the steps and sleep plan of a day is encoded as an item ID which holds number of steps, durations of steps (hours) and durations of sleeps (hours). The first character in the item ID is ignored since it is used for padding (to avoid the cases where the item ID starts with 0, because initial 0s are ignored by the recommender). The first row (Enc.) shows the encoding in the item ID and the second row (Act.) holds the actual corresponding values in number of steps or hours. The actual number of steps are calculated by adding 1 to the number of steps before multiplying with 2000. The actual duration of steps is calculated by adding 1 to the duration of steps. The actual duration of sleep is directly taken as the duration of sleep.

Table 4.23.: Mapping in a Steps & Sleep Plan

	number of steps			durations of steps			durations of sleeps		
	00AM-12AM	12AM-18PM	18PM-24PM	00AM-12AM	12AM-18PM	18PM-24PM	00AM-12AM	12AM-18PM	18PM-24PM
Enc.	0	0	1	1	0	2	6	0	0
Act.	2000	2000	4000	2 hours	1 hour	3 hours	6 hours	0	0

Based on the converted steps/sleep quality dataset in Table 4.22, we request two recommendations from the recommender for the user with userID=20 and the recommender finds the nearest neighbor. The most similar user (nearest neighbor) to userID=20 is userID=19, because they both have same gender (gender is represented with itemID=1 and both users have value=1.0 which means both are female) and same age group (age group is represented with itemID=2 and both users have value=1.0 which means both are younger than 20). Therefore the items with highest ratings (values) of userID=19 are recommended to userID=20 which are 1020000700 (value = 5.0) and 1000000800 (value = 5.0).

The web API of *Virtual Sleep Regulator* provides a human readable version of these recommendation results in JSON (JavaScript Object Notation) format as follows:
{ "intro1": "Recent scientific works show that Insomnia (a sleep disorder)
can be solved with a personal physical activity plan.",
"intro2": "Therefore, according to your profile, for a high quality sleep,
we recommend you several walking and sleeping plans as below:",
"activityRecommendation_list": [
{ "steps1": "take 2000 steps in 1 hours before noon",
"steps2": "take 6000 steps in 1 hours in the afternoon",
"steps3": "take 2000 steps in 1 hours in the evening",
"sleep1": "sleep 7 hours between 00:00-12:00",
"sleep2": "sleep 0 hours between 12:00-18:00",
"sleep3": "sleep 0 hours between 18:00-24:00" },
{ "steps1": "take 2000 steps in 1 hours before noon",
"steps2": "take 8000 steps in 2 hours in the afternoon",
"steps3": "take 2000 steps in 1 hours in the evening",
"sleep1": "sleep 8 hours between 00:00-12:00",
"sleep2": "sleep 0 hours between 12:00-18:00",
"sleep3": "sleep 0 hours between 18:00-24:00" }] }

Further Research Issues

The lessons learned during the design and implementation of the proposed recommender approaches can be grouped in several further research issues as stated in the following paragraphs.

Datasets for Evaluation Purposes. The development of recommendation technologies for QS applications is a rather young discipline and research in the field would strongly profit from the availability of more QS datasets that enable corresponding tests of, for example, the prediction quality of recommendation algorithms.

Scalability and Privacy. In many scenarios, recommendation algorithms, especially the ones based on collaborative filtering approaches, can be deployed in the cloud which has no serious limitations regarding computational resources. Typical examples of such a setting are the recommendation of QS apps (e.g., located on some sort of application marketplace) and the recommendation of QS devices (e.g., located on an online store). However, for recommendation functionalities that employs sensitive data (e.g., medical data of users), the recommendation system (e.g., *Virtual Nurse*) and the related content-based dataset (e.g., activity plan options) should be located directly on the local IoT gateway for privacy issues.

Distributed Data Analysis. The distributed nature of the Internet of Things and corresponding high amounts of collected data from QS applications are challenging existing data analysis methods (Stolpe, 2016). While approaches to big data analytics (Chen et al., 2015; Sun et al., 2016) often follow the paradigm of parallel and high-performance computing, analysis approaches in IoT gateways based QS application scenarios are often limited, for example, in terms of bandwidth and energy supply. This is the major motivation for decentralized analysis algorithms that often have to work (partly) on QS devices.

Group Recommender Systems. Compared to traditional group recommender systems, QS application scenarios increase the number of relevant dimensions in the corresponding datasets. For example, in group-based scenarios (e.g., a group of tourists interested in a city round trip recommendation) example dimensions are not only related to the items to be recommended (e.g., tourist destinations) but also to additional dimensions such as information about health status of group members (e.g., according to the heart rate data of group members, some may need a short break rather than further walking, etc.).

4.2.4. Conclusions

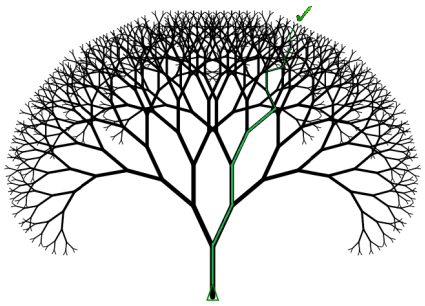
Applications based on *the Quantified-Self (QS)* engage individuals in the self-tracking of any kind of biological, physical, behavioral, or environmental information as individuals or groups. Based on the collected huge amount of self-tracking data, there are new needs for recommender systems to support QS application users. *Recommender systems* can help to more easily identify relevant artifacts for users and thus improve user experiences.

In this section, we have proposed three recommendation approaches *Virtual Coach*, *Virtual Nurse*, and *Virtual Sleep Regulator* which help users/patients to improve their health conditions. This improvement is provided by appropriate device or sportive activity recommendations for users by *Virtual Coach*, activity plan recommendation for patients by *Virtual Nurse*, and walking/sleeping plan recommendation for patients by *Virtual Sleep Regulator*. We have explained these proposed approaches based on real-world datasets collected from a QS application *AGILE Quantified-Self*. Moreover, we have demonstrated how a JSON format recommendation output of *Virtual Sleep Regulator* looks like. Finally, we have discussed further research issues to motivate the researchers working in the domain of recommender systems for QS.

Chapter 5

CSPHeuristix

The open-source library CSPHeuristix is hosted on GitHub: https://github.com/CSPHeuristix.



CSPHeuristix

"CSPHeuristix is an open source library for solving configuration and consistency-based diagnosis problems with the help of its built-in and novel heuristics learning methods."

In order to support further researches and industrial applications of configuration systems, we have implemented the proposed novel methods in this thesis as a free and open-source library dedicated to heuristics learning methods in constraint programming. It is written in Java and aims to solve configuration and consistency-based diagnosis problems in the form of constraint satisfaction problems using constraint programming techniques with the help of its built-in heuristics learning methods.

CSPHeuristix is used for:

- teaching : easy to use for demonstrating performance effects of heuristics in constraint solving,
- research : easy to extend with various constraint solvers and heuristics approaches, and
- real-life applications : easy to integrate into an industrial configuration system.

5.1. About

CSPHeuristix has been implemented based on research outputs of this thesis. All proposed heuristics learning approaches described in Chapter 4 and Chapter 5 are available in CSPHeuristix. This section includes the details of its novel built-in heuristics and user manual. The source code of CSPHeuristix is hosted on GitHub.[1] The informative web page[2] of CSPHeuristix includes a JavaDoc, User Manual, a link to GitHub repository, and a dowloadable package with its .jar file.

Technical overview

CSPHeuristix is a Java 8 library including:

- configurable CSP definition:
  integer variables and unary constraints (or any other CSP content can be defined),
- configurable CSP solving:
  choco-solver (or any other third party solver can be used),
- configurable consistency-based diagnosis:
  FASTDIAG, FLEXDIAG (or any other diagnosis algorithm can be used).

Main contributors

The first version of CSPHeuristix has been released in 2019, January by AIG (Applied Artificial Intelligence Group), Software Technology Institute, Graz University of Technology.

- Core developers: Seda Polat Erdeniz
- Main contributors: Seda Polat Erdeniz, Muesluem Atas, Ralph Samer

How to cite CSPHeuristix?

A reference to CSPHeuristix is made like this:

```
@phdthesis{PolatErdeniz2019,
  title = {Configuration and Recommendation Technologies in Internet of Things},
  school = {Graz University of Technology},
  author = {Polat Erdeniz, Seda},
  year = {2019},
  type = {PhD dissertation},
}
```

[1] https://github.com/CSPHeuristix
[2] http://agile.ist.tugraz.at:8080/CSPHeuristix/

5.2. Using CSPHeuristix

Structure of the main package

You can download CSPHeuristix on the website of CSPHeuristix[1]. You will get a zip file which contains the following files:

- CSPHeuristix-1.0.jar: A ready-to-use jar file including dependencies.
- CSPHeuristix-1.0-apidocs.zip: Javadoc of CSPHeuristix-1.0

Adding CSPHeuristix to your project

After downloading the latest package of CSPHeuristix from the website, simply add CSPHeuristix-1.0.jar to the classpath of your project (in a terminal or in your favorite IDE).

Example Code

A simple example showing how to use CSPHeuristix is given below. In this example, used input files and the corresponding output (the log file) can be found in Appendix (Section A.1 and Section A.2).

As shown in the simple example, at first the library should be initialized with preferences. For example, the first parameter is the chosen heuristic ID, where available heuristics in CSPHeuristix-1.0 are all listed in the array as HeuristicID [] hid = {HeuristicID.CLVVO, HeuristicID.CLCO, HeuristicID.MFVVO, HeuristicID.MFCO}. In the example code, HeuristicID.CLVVO is selected to solve the problems defined in files of the input folder. Input folder is also defined in parameters so that different problem sets can be tested easily by only modifying this parameter. In the input folder, besides problem datasets, required training datasets should be also provided according to the specific format as explained in Section 5.3. Based on the training datasets in the input folder, library can be initialized with learned heuristics for defined performance indicator in parameters.

Optional parameters *did*, *numberOfClusters*, *stoppingCriteria*, and *m* are used only if they are required by the selected heuristics learning method. For example, when HeuristicID.MFVVO or HeuristicID.MFCO is selected, *numberOfClusters* and *stoppingCriteria* are not used. These two parameters are required for the genetic algorithm based learning by HeuristicID.CLVVO and HeuristicID.CLCO. *did* is required by used by HeuristicID.MFCO and HeuristicID.CLCO where diagnosis tasks are solved. *m* is required by the diagnosis algorithm FLEXDIAG, so it is used by HeuristicID.MFCO and HeuristicID.CLCO if the diagnosis algorithm is selected as DiagnoserID.flexdiag.

After initializing the library with required parameters, solving configuration or diagnosis tasks is done by only one line of code. Solutions or diagnoses of given configuration or diagnosis tasks are returned in a CSP array where in each CSP object the problem definition in CSP format, its solution/diagnosis, runtime performance to solve the task, (if expected solutions are available in the problem dataset file) prediction quality of the solution/diagnosis, and used heuristics to solve/diagnose the problem can be found.

[1] http://agile.ist.tugraz.at:8080/CSPHeuristix/

```
package at.tugraz.ist.ase.examples;
import at.tugraz.ist.ase.cspheuristix.Library;
import at.tugraz.ist.ase.solvers.CSP;
import at.tugraz.ist.ase.util.ClusteringAlgorithmID;
import at.tugraz.ist.ase.util.DiagnoserID;
import at.tugraz.ist.ase.util.HeuristicID;
import at.tugraz.ist.ase.util.PerformanceIndicator;
import at.tugraz.ist.ase.util.SolverID;

public class Example {
// supported diagnosers in CSPHeuristix-1.0
DiagnoserID[] did = {DiagnoserID.fastdiag, DiagnoserID.flexdiag};

// supported CSP solvers in CSPHeuristix-1.0
SolverID[] sid = {SolverID.choco};

// supported performance indicators in CSPHeuristix-1.0
PerformanceIndicator[] pid = {PerformanceIndicator.runtime,
    PerformanceIndicator.predictionQuality};

// supported heuristics learning methods in CSPHeuristix-1.0
HeuristicID[] hid = {HeuristicID.CLVVO, HeuristicID.CLCO,
    HeuristicID.MFVVO, HeuristicID.MFCO};

// supported clustering methods in CSPHeuristix-1.0
ClusteringAlgorithmID[] cid = {ClusteringAlgorithmID.kmeans,
    ClusteringAlgorithmID.xmeans, ClusteringAlgorithmID.em,
    ClusteringAlgorithmID.clope,
    ClusteringAlgorithmID.farthestFirst,
    ClusteringAlgorithmID.filteredClusterer};

// number of clusters
int numberOfClusters=4;
// time out for genetic algorithm
String stoppingCriteria = "10";
// m parameter of FlexDiag algorithm
int m=1;

// INPUT FOLDER have to be initialized by users
String inputFolder = "IOFOLDER/INPUT/test/CSPSolving2";
// OUTPUT FOLDER is used to store temporary files
String outputFolder = "IOFOLDER/OUTPUT/test/CSPSolving2";

   public static void main(String[] args) {
        Library lib = new Library(hid[0], sid[0], did[0],
            inputFolder, outputFolder, pid[0], stoppingCriteria,
            cid[0], numberOfClusters, m);
        // results are printed in "CSPHeuristixLogs.log"
        CSP[] results = lib.solveTasks();
   }
}
```

5.3. Modeling

CSPHeuristix are trained with training datasets and can then be used to solve/diagnose new problems datasets. All problems are defined in CSP format in input files.

CSP Object

The object CSP is the key component of CSPHeuristix. All problems are read from input files and converted into CSP objects. After they are solved, library returns these CSP objects with solutions/diagnoses inside:

```
CSP[]  results = lib.solveTasks ();
```

A CSP object are composed of these variables:

```
////////// MANDATORY FIELDS ////////////
String name; // the name of the CSP
Var[] vars;
Const[] AC; // all constraints = BC + REQ
Const[] BC; // basis constraints -> always consistent
////////// OPTIONAL FIELDS ////////////
Const[] REQ; // user requirements -> can be inconsistent
int [] intREQ; // integer values of REQ
// expectedSolution may not null if the CSP is from a training
   dataset
int [] expectedSolution; // to measure the accuracy of the result

////////// FILLED AFTER SOLVING/DIAGNOSING ////////////
int [] solution; // the solution of the CSP if it is consistent
Const[] diagnosis; // the diagnosis of the CSP if it is inconsistent
boolean isSolved; // true if it is solved/diagnosed
float runtime; // time to find solution/diagnosis (nanoseconds)
// predictionQuality is calculated if expectedSolution is not null
float predictionQuality; // accuracy of the solution/diagnosis
```

Var Object

Var object represents an integer variable with a bounded domain:

```
String name; // the name of the Var
int minDomain; // lower bound of the domain
int maxDomain; // upper bound of the domain
```

The domain of an integer variable is bounded and represented through an interval of the form [a, b] where a and b are integers such that a <= b. This representation is pretty light in memory (it requires only two integers) but it cannot represent holes in the domain. For instance, if we have a variable whose domain is [0, 10] and a constraint enables to detect that values 2, 3, 7 and 8 are infeasible, then this learning will be lost as it cannot be encoded in the domain (which remains the same).

Const Object

Const object represents an arithmetic unary constraint:

```
int varID; // variable ID
String operator; // "=", "!=","<", ">", "<=, or ">="
int value; // an integer value
```

Input Files

CSPHeuristix need seven input files as follows:

1. **File name: "basisCSP"**
   It holds the definition of the root CSP with name (<name>), variables (<min>,<max>) and constraints (<id>,<op>,<val>). Variables are defined with their domain boundaries (lower bound (<min>) and upper bound (<max>)) and automatically their ID numbers are given by the library starting from 0 (ID of first variable is 0, second variable is 1, ...). Constraints are defined as unary constraints with the ID number of a variable (<id>), a operator (<op>), and a value (<val>). An operator can be =, !=, <, >, <=, or >=.

```
% NAME;<name>
% VARS;<min>,<max>;...;<min>,<max>
% CONSTS;<id>,<op>,<val>;...;<id>,<op>,<val>
```

   For example, here is a simple "basisCSP" content with four variables (variable IDs are automatically assigned respectively as: 0,1,2,3) and one constraint:

```
% NAME;Test
% VARS;0,10;0,10;0,8;0,9
% CONSTS;1,<,6
```

2. **File name: "newConsistentReqs"**
   It holds new sets of user requirements to be solved. Each row represents a user's transaction. Each transaction holds user requirements for all variables of the basis CSP. Therefore, if there are n variables in the basis CSP, it holds n values are user requirements (the first value is for the first variable: <reqk.0>, the second value is for the second variable: <reqk.1>,...). When a variable is not initialized in a user requirement set, it is stated with "-1". The last value on each row gives the user ID (uid1,...,uidk).

```
<req1.0>, <req1.1>,...,<req1.n>, uid1
.....
<reqk.0>, <reqk.1>,...,<reqk.n>, uidk
```

   For example, here is a simple "newConsistentReqs" content with two user transactions (with only user requirements) for configuration of a product with four variables:

```
3, 3, -1, -1, 0
3, 3, -1, -1, 1
```

3. **File name: "newInconsistentReqs"**
   It holds new sets of inconsistent user requirements to be diagnosed. Its format is same as "newConsistentReqs".

4. **File name: "pastConsistentReqs" (Incomplete Historical Transactions)**
   It is used as a training dataset and holds past user requirements without a purchased product information. Its format is same as "newConsistentReqs".

5. **File name: "pastInconsistentReqs" (Incomplete Historical Transactions)**
   It is used as a training dataset holds past inconsistent user requirements without a purchased product information. Its format is same as "newInconsistentReqs".

6. **File name: "pastSolutions" (Complete Historical Transactions)**
   It is used as a training dataset and holds past purchased products information. Each row represents a user transaction with a user ID (uidk). Each transaction holds past user requirements (<req1.0>,...,<reqk.n>) and past purchased product information (<solnk.0>,...,<solnk.n>). Therefore, if there are n+1 variables in the basis CSP, it holds all n+1 values which are purchased product information.

```
<req1.0>,...,<req1.n>, <soln1.0>,...,<soln1.n>,uid1
.....
<req1.0>,...,<reqk.n>, <solnk.0>,...,<solnk.n>,uidk
```

For example, here is a simple "pastSolutions" content with two sets of user transactions (user requirements and purchased product information) where the product has four variables:

```
3, 3, -1, -1, 3, 3, 3, 3, 0
3, 3, -1, -1, 3, 3, 3, 3, 1
```

7. **File name: "pastDiagnoses" (Complete Historical Transactions)**
   It is used as a training dataset and holds past inconsistent user requirements and their purchased product information. Each row represents a user transaction. Each transaction holds past user requirements for all variables of the basis CSP and the purchased product information. Therefore, if there are n variables in the basis CSP, first n values are user requirements and the following n values are the purchased product information. The rest format is same as "pastSolutions".

5.4. Solving

After the library is initialized, new configuration/diagnosis problems in defined input file can be solved by the library in one line:

```
Library lib = new Library(hid[0], sid[0], did[0], inputFolder,
    outputFolder, pid[0], stoppingCriteria, cid[0],
    numberOfClusters, m);
CSP[]  results = lib.solveTasks ();
```

In order to initialize the library, 10 parameters should be defined:

1. hid: Heuristic ID
2. sid: CSP Solver ID
3. did: Diagnosis Algorithm ID
4. inputFolder: folder of input files
5. outputFolder: folder for temporary output files (clusters, etc.)
6. pid: Performance Indicator ID

7. stoppingCriteria: stopping criteria of the genetic algorithm
8. numberOfClusters: number of clusters
9. m: the anytime diagnosis parameter of FLEXDIAG

Heuristics

CSPHeuristix-v1.0 provides four novel built-in heuristics learning methods:

1. CLVVO: Cluster-and-Learn based Variable and Value Ordering Heuristics (see Section 3.2)
2. CLCO: Cluster-and-Learn based Constraint Ordering Heuristics (see Section 3.3)
3. MFVVO: Matrix Factorization based Variable and Value Ordering Heuristics (see Section 3.4)
4. MFCO: Matrix Factorization based Constraint Ordering Heuristics (see Section 3.5)

CSP Solvers

CSPHeuristix-v1.0 provides one CSP solver: choco solver. It can be extended with various CSP solvers or even ASP or SAT solvers.

Consistency-based Diagnosis Algorithms

CSPHeuristix-v1.0 provides two consistency-based direct diagnosis algorithms: FASTDIAG and FLEXDIAG. It can be extended with various diagnosis algorithms.

Performance Indicators

CSPHeuristix-v1.0 provides two performance indicators: runtime and predictionQuality. It can be extended with various performance indicators.

Clustering Algorithms

CSPHeuristix-v1.0 provides two clustering algorithms: kmeans, xmeans, em, clope, farthestFirst, filteredClusterer. It can be extended with various clustering algorithms.

Chapter 6

Conclusions and Future Work

Artificial intelligence (AI) methods are very important to support emerging IoT applications. With a growing need to process big data, users need help adapting these new technologies. This thesis introduces and evaluates novel artificial intelligence approaches (specifically configuration and recommendation technologies) to assist IoT environment stakeholders. In the following section, we reflect upon our research questions and contributions. This chapter concludes with an outlook on future research issues.

6.1. Conclusion

Below, we provide an overview of our research questions (for a detailed discussion see Section 1.2) and our contributions to answer these questions.

Research Question Q1:

How to adapt state-of-the-art configuration and recommendation systems to IoT applications in order to improve experiences of IoT environment stakeholders?

In Section 3.1 (Felfernig et al., 2016), Section 4.1 (Felfernig et al., 2018b) and Section 4.2, we have analyzed configuration and recommendation scenarios based on AGILE IoT. For these scenarios, we proposed adapting state-of-the-art (basic) configuration and recommendation technologies and also proposed new approaches.

Basic recommendation technologies have been developed and tested in AGILE IoT (Felfernig et al., 2018b) (see Section 4.1) for application recommendation, workflow/node (code) recommendation, device (sensors, IoT connectivity devices, etc.) recommendation, and activity recommendation in quantified self applications (Erdeniz et al., 2018b) (see Section 4.2).

Additionally, basic configuration technologies for optimizing gateway configurations and assisting IoT environment installing process (ramp-up configurator) are introduced and developed for AGILE IoT (Felfernig et al., 2016) (see Section 3.1).

Research Question Q2.1:

How to learn heuristics using genetic algorithms in order to improve runtime performance of configuration systems?

In Section 3.2 (Erdeniz et al., 2017; Erdeniz and Felfernig, 2018b,a) and Section 3.3 (Erdeniz et al., 2018a), we have proposed novel heuristics learning methods based on a similar idea in which historical transactions are clustered and a genetic algorithm is employed on each cluster to learn cluster-specific heuristics. When a new configuration/diagnosis task need to be solved, the closest cluster is identified and its pre-learned heuristic is used for solving the new problem. These heuristics (CLVVO and CLCO) are provided in the CSPHeuristix Library as well (see Chapter 5). Using this approach,

- variable and value ordering heuristics for constraint solving (Erdeniz et al., 2017; Erdeniz and Felfernig, 2018a,b) (see Section 3.2), and
- constraint ordering heuristics for consistency based diagnosis (Erdeniz et al., 2018a) (see Section 3.3)

are learned. As presented in experimental evaluations of Section 3.2 and Section 3.3, both proposed VVO (variable and value ordering) and CO (constraint ordering) heuristics improve runtime performance of configuration systems.

Research Question Q2.2:

How to learn heuristics using matrix factorization techniques in order to improve runtime performance of configuration systems?

In Section 3.4 (Erdeniz et al., 2019b) and Section 3.5 (Erdeniz et al., 2019a), we have proposed novel heuristics learning methods based on a similar process by which historical transactions are gathered into a matrix (a sparse matrix) and matrix factorization is employed on the sparse matrix to estimate a full matrix in which we obtain learned heuristics. When a new configuration/diagnosis task need to be solved, the most similar historical transaction is identified and its pre-learned heuristic is used for solving the new problem. These heuristics (MFVVO and MFCO) are provided in the CSPHeuristix Library as well (see Chapter 5). Using this approach,

- variable and value ordering heuristics for constraint solving (Erdeniz et al., 2019b) (see Section 3.4), and
- constraint ordering heuristics for consistency based direct diagnosis (Erdeniz et al., 2019a) (see Section 3.5)

are learned. As presented in experimental evaluations of Section 3.4 and Section 3.5, both proposed VVO (variable and value ordering) and CO (constraint ordering) heuristics improve runtime performance of configuration systems.

Research Question Q3.1:

How to learn heuristics using genetic algorithms in order to improve prediction quality of configuration systems?

In Section 3.3 (Erdeniz et al., 2018a), we have proposed a novel heuristics learning method with which historical transactions are clustered and a genetic algorithm is employed on each cluster to learn cluster-specific heuristics. When a new configuration/diagnosis task need to be solved, the closest cluster is identified and its pre-learned heuristic is used for solving the new problem. This heuristic (CLCO) is provided in the CSPHeuristix Library as well (see Chapter 5). Using this approach,

- constraint ordering heuristics for consistency based diagnosis (Erdeniz et al., 2018a) (see Section 3.3)

are learned. As presented in experimental evaluations of Section 3.3 (Erdeniz et al., 2018a), CO (constraint ordering) heuristics improve quality of configuration in terms of prediction quality and minimality.

Research Question Q3.2:

How to learn heuristics using matrix factorization techniques in order to improve prediction quality of configuration systems?

In Section 3.5 (Erdeniz et al., 2019a), we have proposed novel heuristics learning method with which historical transactions are gathered into a matrix (a sparse matrix) and matrix factorization is employed to estimate a full matrix from which we then obtain learned heuristics. When a new configuration/diagnosis task need to be solved, the most similar historical transaction is identified and its pre-learned heuristic is used for solving the new problem. This heuristic (MFCO) is provided in the CSPHeuristix Library as well (see Chapter 5). Using this approach,

- constraint ordering heuristics for consistency based diagnosis (Erdeniz et al., 2019a) (see Section 3.5)

are learned. As presented in experimental evaluations of Section 3.5 (Erdeniz et al., 2019a), proposed CO (constraint ordering) heuristics improve quality of configuration in terms of diagnosis minimality.

6.2. Future Work

In configuration and recommendation technologies, little research exists to simultaneously improve both the runtime and prediction quality. In this context, we have provided novel solutions based on learning heuristics using learning algorithms and historical transactions. Our future work will include further heuristics learning approaches. In this chapter, relevant topics for future research are discussed.

Further Learning Approaches

Machine learning algorithms are made up of three main components: representation, evaluation, and optimization. A classifier must be represented in a formal language that the computer can understand and interpret. Creating a set of classifiers that the learner can learn is crucial. Evaluation function is needed

to distinguish good classifiers from bad ones. Finally we need a method to search in a specialized language for the highest scoring classifier. The choice of optimization technique is key to the efficiency of the algorithm (Domingos, 2012). The goal, of course, is to generate a rule that makes the most accurate predictions possible on new test examples. On one hand, building a highly accurate prediction rule is certainly a difficult task. On the other hand, it is not hard at all to come up with very rough rules of thumb that are only moderately accurate. An example of such a rule is something like the following: "If the phrase 'buy now' occurs in the email, then classify it as spam.". Such a rule will not even come close to covering all spam messages; for instance, it does not say anything about what to predict if 'buy now' does not occur in the message. However, this rule will make predictions that are significantly better than random guessing (Schapire, 2003).

Based on the training dataset, four groups of learning methods can be used. They are as follows: supervised learning (Linear Regression, Support Vector Machines, Neural Networks, Decision Trees, Naive Bayes, Nearest Neighbor, etc.), unsupervised learning (k-means clustering, Association rule, etc.), semi-supervised learning, and reinforced learning (Q-Learning, Deep Adversarial Networks, etc.).

In this thesis, we have utilized historical transaction as the training dataset and unsupervised learning methods based on this dataset. We have employed k-means clustering algorithm, genetic algorithms, and matrix factorization techniques. However, even based on the same historical transactions, various combinations of various machine learning techniques can be also investigated.

Datasets for Evaluation Purposes

In the world of recommender systems, it is a common practice to use public available datasets from different application environments (e.g. MovieLens, Book-Crossing, or Each-Movie) in order to evaluate recommendation algorithms. These datasets are used as benchmarks to develop new recommendation algorithms and to compare them to other algorithms in given settings (Verbert et al., 2011). The relevance of these datasets however has faded over the years as they become outdated (e.g., most recent movie in MovieLens 100K dataset is from 1998). While still useful for offline evaluation, online experiments with actual users may fail because of the lack of recent and relevant movies in the dataset. Moreover, datasets themselves are also often filtered so to only contain users with e.g. a minimum number of ratings (e.g., 20 ratings for MovieLens). Because of this filtering, a systematic bias is introduced which may prevent experimental results to be generalizable to real-life scenarios (Harper and Konstan, 2016).

Configuration systems need a larger and richer open-source evaluation datasets as is the case with recommendation systems (e.g. Movielens Dataset). In these datasets, configuration and diagnosis problems (with many variables and constraints) should be defined with their corresponding real-world user preferences (e.g., user requirements and their purchase transactions on a online shopping with personalization features). On the other hand, the development of recommendation technologies for IoT scenarios is a rather young discipline and research in the field would strongly profit from the availability of more IoT datasets that enable corresponding tests of, for example, the prediction quality of recommendation algorithms. In the context of end-user development support in IoT scenarios, datasets that include logs are helpful for development of IoT workflows on remote gateway installations. This information can be exploited to optimize user support, for example, by predicting relevant code-fragments and sensors that should be included. We are interested in gathering such big datasets for our quality based evaluations of configuration and recommendation systems in IoT.

CSPHeuristix Library

New heuristics learning approaches can be easily tested and compared with our approaches by using our open-source library, CSPHeuristix. CSPHeuristix is a modular and extendable library, so these new ap-

proaches can be implemented inside the library using its already-implemented constraint solving objects. CSPHeuristix can also be extended with new constraint-solvers, diagnosis algorithms, complex variable/-constraint structures, and even various input file formats.

CSPHeuristix-v1.0 supports:

- **Heuristics:** four novel heuristics learning methods which are described in this thesis in Section 3.2, Section 3.3, Section 3.4, and Section 3.5,
- **Variables:** integer,
- **Constraints:** unary,
- **Constraint solvers:** choco-solver (as a constraint solver)
- **Consistency-based diagnosis algorithms Solvers:** FASTDIAG and FLEXDIAG

Appendix A

Appendix

A.1. Example Input for CSPHeuristix

Used input files in the example code are as follows:

1. **File Name: "basisCSP"**

```
% NAME;Test
% VARS;0,10;0,10;0,8;0,9
% CONSTS;1,<,6
```

2. **File Name: "newConsistentReqs"**

```
3, 3, -1, -1, 0
3, 3, -1, -1, 1
3, 3, -1, -1, 2
3, 3, -1, -1, 3
3, 3, -1, -1, 4
1, 1, -1, -1, 5
1, 1, -1, -1, 6
1, 1, -1, -1, 7
1, 1, -1, -1, 8
1, 1, -1, -1, 9
1, 1, -1, -1, 10
1, 1, -1, -1, 11
1, 1, -1, -1, 12
1, 1, -1, -1, 13
1, 1, -1, -1, 14
1, 1, -1, -1, 15
1, 1, -1, -1, 16
5, 5, -1, -1, 17
5, 5, -1, -1, 18
```

3. **File Name: "newInconsistentReqs"**

```
3, 8, -1, -1, 0
```

```
3, 8, -1, -1, 1
3, 6, -1, -1, 2
3, 7, -1, -1, 3
3, 7, -1, -1, 4
1, 6, -1, -1, 5
1, 7, -1, -1, 6
1, 7, -1, -1, 7
1, 8, -1, -1, 8
1, 8, -1, -1, 9
1, 8, -1, -1, 10
1, 6, -1, -1, 11
1, 6, -1, -1, 12
1, 6, -1, -1, 13
1, 7, -1, -1, 14
1, 9, -1, -1, 15
1, 8, -1, -1, 16
5, 6, -1, -1, 17
5, 9, -1, -1, 18
```

4. **File Name: "pastConsistentReqs"**

```
3, 3, -1, -1, 0
3, 3, -1, -1, 1
3, 3, -1, -1, 2
3, 3, -1, -1, 3
3, 3, -1, -1, 4
1, 1, -1, -1, 5
1, 1, -1, -1, 6
1, 1, -1, -1, 7
1, 1, -1, -1, 8
1, 1, -1, -1, 9
1, 1, -1, -1, 10
1, 1, -1, -1, 11
1, 1, -1, -1, 12
1, 1, -1, -1, 13
1, 1, -1, -1, 14
1, 1, -1, -1, 15
1, 1, -1, -1, 16
5, 5, -1, -1, 17
5, 5, -1, -1, 18
```

5. **File Name: "pastDiagnoses"**

```
3, 7, 3, 3, 3, 3, 3, 3, 0
3, 7, 3, 3, 3, 3, 3, 3, 1
3, 9, 3, 3, 3, 3, 3, 3, 2
3, 6, 3, 3, 3, 3, 3, 3, 3
3, 8, 3, 3, 3, 3, 3, 3, 4
1, 4, 1, 1, 1, 1, 1, 1, 5
1, 2, 1, 1, 1, 1, 1, 1, 6
1, 7, 1, 1, 1, 1, 1, 1, 7
1, 6, 1, 1, 1, 1, 1, 1, 8
1, 7, 1, 1, 1, 1, 1, 1, 9
1, 6, 1, 1, 1, 1, 1, 1, 10
```

```
1, 9, 1, 1, 1, 1, 1, 1, 11
1, 8, 1, 1, 1, 1, 1, 1, 12
1, 6, 1, 1, 1, 1, 1, 1, 13
1, 7, 1, 1, 1, 1, 1, 1, 14
1, 9, 1, 1, 1, 1, 1, 1, 15
1, 8, 1, 1, 1, 1, 1, 1, 16
5, 7, 5, 5, 5, 5, 5, 5, 17
5, 8, 5, 5, 5, 5, 5, 5, 18
```

6. **File Name: "pastInconsistentReqs"**

```
3, 8, -1, -1, 0
3, 8, -1, -1, 1
3, 6, -1, -1, 2
3, 7, -1, -1, 3
3, 7, -1, -1, 4
1, 6, -1, -1, 5
1, 7, -1, -1, 6
1, 7, -1, -1, 7
1, 8, -1, -1, 8
1, 8, -1, -1, 9
1, 8, -1, -1, 10
1, 6, -1, -1, 11
1, 6, -1, -1, 12
1, 6, -1, -1, 13
1, 7, -1, -1, 14
1, 9, -1, -1, 15
1, 8, -1, -1, 16
5, 6, -1, -1, 17
5, 9, -1, -1, 18
```

7. **File Name: "pastSolutions"**

```
3, 3, 3, 3, 0
3, 3, 3, 3, 1
3, 3, 3, 3, 2
3, 3, 3, 3, 3
3, 3, 3, 3, 4
1, 1, 1, 1, 5
1, 1, 1, 1, 6
1, 1, 1, 1, 7
1, 1, 1, 1, 8
1, 1, 1, 1, 9
1, 1, 1, 1, 10
1, 1, 1, 1, 11
1, 1, 1, 1, 12
1, 1, 1, 1, 13
1, 1, 1, 1, 14
1, 1, 1, 1, 15
1, 1, 1, 1, 16
5, 5, 5, 5, 17
5, 5, 5, 5, 18
```

A.2. Example Output of CSPHeuristix

The output of the example code is printed in the log file as follow:

File Name: "CSPHeuristixLogs.log"

```
2019-01-12 10:44:38 INFO  Library:72 -
##################################################
2019-01-12 10:44:38 INFO  Library:73 - START (2019/01/12 10:44:38)
2019-01-12 10:44:38 INFO  Library:74 - heuristicsID: CLVVO
2019-01-12 10:44:38 INFO  Library:75 - solverID: choco
2019-01-12 10:44:38 INFO  Library:76 - ClusteringAlgorithmID: kmeans
2019-01-12 10:44:38 INFO  Library:77 - PerformanceIndicator: predictionQuality
2019-01-12 10:44:38 INFO  Library:78 - stoppingCriteria: 10
2019-01-12 10:44:38 INFO  Library:79 - diagnosisAlgorithmID: fastdiag
2019-01-12 10:44:38 INFO  Library:80 - inputFile: IOFOLDER/INPUT/test/LibraryInputFolder1
2019-01-12 10:44:38 INFO  Library:81 - outputFolder: IOFOLDER/OUTPUT/test/LibraryOutputFolder1
2019-01-12 10:44:40 INFO  Library:112 - LEARNED HEURISTICS:
Heuristics-0: Variable and Value Ordering: Var-3:(0,3,1,7,2,8,4,6,5,); Var-0:(0,5,8,7,9,4,6,3,2,1,);
     Var-2:(4,5,6,2,1,3,0,7,); Var-1:(5,2,3,8,0,1,7,9,6,4,);
Heuristics-1: Variable and Value Ordering: Var-2:(2,4,7,3,6,5,0,1,); Var-0:(4,5,9,7,6,0,3,8,2,1,);
     Var-1:(0,3,2,6,4,8,1,7,9,5,); Var-3:(0,1,6,7,3,2,4,8,5,);
Heuristics-2: Variable and Value Ordering: Var-1:(8,0,7,1,4,9,5,3,6,2,); Var-2:(0,4,6,3,1,5,2,7,);
     Var-3:(7,0,2,4,5,1,8,3,6,); Var-0:(4,3,1,8,5,2,7,0,9,6,);

2019-01-12 10:44:40 INFO  Library:151 - RESULTS:
CSP-Test0: , var-0= 3, var-1= 3, var-2= 0, var-3= 0, runtime: 64381.0 ns, prediction quality: 0.5
CSP-Test1: , var-0= 3, var-1= 3, var-2= 0, var-3= 0, runtime: 27992.0 ns, prediction quality: 0.5
CSP-Test2: , var-0= 3, var-1= 3, var-2= 0, var-3= 0, runtime: 22394.0 ns, prediction quality: 0.5
CSP-Test3: , var-0= 3, var-1= 3, var-2= 0, var-3= 0, runtime: 22860.0 ns, prediction quality: 0.5
CSP-Test4: , var-0= 3, var-1= 3, var-2= 0, var-3= 0, runtime: 22860.0 ns, prediction quality: 0.5
CSP-Test5: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 25192.0 ns, prediction quality: 0.5
CSP-Test6: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 23793.0 ns, prediction quality: 0.5
CSP-Test7: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 23326.0 ns, prediction quality: 0.5
CSP-Test8: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 22860.0 ns, prediction quality: 0.5
CSP-Test9: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 23326.0 ns, prediction quality: 0.5
CSP-Test10: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 23327.0 ns, prediction quality: 0.5
CSP-Test11: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 22860.0 ns, prediction quality: 0.5
CSP-Test12: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 22860.0 ns, prediction quality: 0.5
CSP-Test13: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 25193.0 ns, prediction quality: 0.5
CSP-Test14: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 24259.0 ns, prediction quality: 0.5
CSP-Test15: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 24726.0 ns, prediction quality: 0.5
CSP-Test16: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 22859.0 ns, prediction quality: 0.5
CSP-Test17: , var-0= 5, var-1= 5, var-2= 0, var-3= 0, runtime: 913927.0 ns, prediction quality: 0.5
CSP-Test18: , var-0= 5, var-1= 5, var-2= 0, var-3= 0, runtime: 48052.0 ns, prediction quality: 0.5
2019-01-12 10:44:40 INFO  Library:152 - END (2019/01/12 10:44:38)
2019-01-12 10:44:40 INFO  Library:153 -
##################################################

2019-01-12 10:44:40 INFO  Library:72 -
##################################################
2019-01-12 10:44:40 INFO  Library:73 - START (2019/01/12 10:44:40)
2019-01-12 10:44:40 INFO  Library:74 - heuristicsID: CLCO
2019-01-12 10:44:40 INFO  Library:75 - solverID: choco
2019-01-12 10:44:40 INFO  Library:76 - ClusteringAlgorithmID: kmeans
2019-01-12 10:44:40 INFO  Library:77 - PerformanceIndicator: predictionQuality
2019-01-12 10:44:40 INFO  Library:78 - stoppingCriteria: 10
2019-01-12 10:44:40 INFO  Library:79 - diagnosisAlgorithmID: fastdiag
2019-01-12 10:44:40 INFO  Library:80 - inputFile: IOFOLDER/INPUT/test/LibraryInputFolder1
2019-01-12 10:44:40 INFO  Library:81 - outputFolder: IOFOLDER/OUTPUT/test/LibraryOutputFolder1
2019-01-12 10:44:41 INFO  Library:112 - LEARNED HEURISTICS:
Heuristics-0: Constraint Ordering: VAR-0; VAR-3; VAR-2; VAR-1;
Heuristics-1: Constraint Ordering: VAR-2; VAR-1; VAR-3; VAR-0;
Heuristics-2: Constraint Ordering: VAR-2; VAR-3; VAR-1; VAR-0;

2019-01-12 10:44:41 INFO  Library:151 - RESULTS:
CSP-Test0: CONSTRAINT: var-1 = 8;, runtime: 360159.0 ns, prediction quality: -1.0
CSP-Test1: CONSTRAINT: var-1 = 8;, runtime: 245860.0 ns, prediction quality: -1.0
CSP-Test2: CONSTRAINT: var-1 = 6;, runtime: 241661.0 ns, prediction quality: -1.0
CSP-Test3: CONSTRAINT: var-1 = 7;, runtime: 237463.0 ns, prediction quality: -1.0
CSP-Test4: CONSTRAINT: var-1 = 7;, runtime: 243061.0 ns, prediction quality: -1.0
CSP-Test5: CONSTRAINT: var-1 = 6;, runtime: 250058.0 ns, prediction quality: -1.0
CSP-Test6: CONSTRAINT: var-1 = 7;, runtime: 238862.0 ns, prediction quality: -1.0
CSP-Test7: CONSTRAINT: var-1 = 7;, runtime: 240728.0 ns, prediction quality: -1.0
CSP-Test8: CONSTRAINT: var-1 = 8;, runtime: 248193.0 ns, prediction quality: -1.0
CSP-Test9: CONSTRAINT: var-1 = 8;, runtime: 247260.0 ns, prediction quality: -1.0
CSP-Test10: CONSTRAINT: var-1 = 8;, runtime: 245393.0 ns, prediction quality: -1.0
CSP-Test11: CONSTRAINT: var-1 = 6;, runtime: 233730.0 ns, prediction quality: -1.0
```

```
CSP-Test12: CONSTRAINT: var-1 = 6;, runtime: 238395.0 ns, prediction quality: -1.0
CSP-Test13: CONSTRAINT: var-1 = 6;, runtime: 374622.0 ns, prediction quality: -1.0
CSP-Test14: CONSTRAINT: var-1 = 7;, runtime: 239329.0 ns, prediction quality: -1.0
CSP-Test15: CONSTRAINT: var-1 = 9;, runtime: 235596.0 ns, prediction quality: -1.0
CSP-Test16: CONSTRAINT: var-1 = 8;, runtime: 275718.0 ns, prediction quality: -1.0
CSP-Test17: CONSTRAINT: var-1 = 6;, runtime: 232331.0 ns, prediction quality: -1.0
CSP-Test18: CONSTRAINT: var-1 = 9;, runtime: 229532.0 ns, prediction quality: -1.0
2019-01-12 10:44:41 INFO  Library:152 - END (2019/01/12 10:44:40)
2019-01-12 10:44:41 INFO  Library:153 -
##################################################

2019-01-12 10:44:41 INFO  Library:72 -
##################################################
2019-01-12 10:44:41 INFO  Library:73 - START (2019/01/12 10:44:41)
2019-01-12 10:44:41 INFO  Library:74 - heuristicsID: MFVVO
2019-01-12 10:44:41 INFO  Library:75 - solverID: choco
2019-01-12 10:44:41 INFO  Library:76 - ClusteringAlgorithmID: kmeans
2019-01-12 10:44:41 INFO  Library:77 - PerformanceIndicator: predictionQuality
2019-01-12 10:44:41 INFO  Library:78 - stoppingCriteria: 10
2019-01-12 10:44:41 INFO  Library:79 - diagnosisAlgorithmID: fastdiag
2019-01-12 10:44:41 INFO  Library:80 - inputFile: IOFOLDER/INPUT/test/LibraryInputFolder1
2019-01-12 10:44:41 INFO  Library:81 - outputFolder: IOFOLDER/OUTPUT/test/LibraryOutputFolder1
2019-01-12 10:44:41 INFO  Library:112 - LEARNED HEURISTICS:
Heuristics-0: Variable and Value Ordering: Var-1:(1,3,5,7,9,2,0,6,8,4,);
     Var-0:(1,3,5,4,6,9,8,0,7,2,); Var-2:(1,3,5,0,4,6,2,7,); Var-3:(1,3,5,8,2,6,0,7,4,);
Heuristics-1: Variable and Value Ordering: Var-1:(1,3,5,9,7,2,8,0,4,6,);
     Var-0:(1,3,5,6,4,7,9,0,2,8,); Var-2:(1,3,5,4,0,6,2,7,); Var-3:(1,3,5,8,2,0,7,6,4,);
Heuristics-2: Variable and Value Ordering: Var-1:(1,3,5,7,0,9,2,8,4,6,);
     Var-0:(1,3,5,0,4,9,2,6,8,7,); Var-2:(1,3,5,4,6,0,7,2,); Var-3:(1,3,5,7,8,6,4,2,0,);
Heuristics-3: Variable and Value Ordering: Var-1:(1,3,5,9,7,6,0,2,8,4,);
     Var-0:(1,3,5,8,4,6,7,2,0,9,); Var-3:(1,3,5,2,8,0,6,4,7,); Var-2:(1,3,5,2,0,6,4,7,);
Heuristics-4: Variable and Value Ordering: Var-1:(1,3,5,8,2,9,7,0,4,6,);
     Var-0:(1,3,5,9,6,7,4,0,2,8,); Var-2:(1,3,5,4,7,6,0,2,); Var-3:(1,3,5,8,4,7,2,0,6,);
Heuristics-5: Variable and Value Ordering: Var-1:(1,3,5,2,9,7,8,6,4,0,);
     Var-0:(1,3,5,4,6,9,7,0,2,8,); Var-3:(1,3,5,8,2,0,4,7,6,); Var-2:(1,3,5,4,6,7,0,2,);
Heuristics-6: Variable and Value Ordering: Var-1:(1,3,5,0,7,2,4,9,8,6,);
     Var-0:(1,3,5,0,6,9,4,7,2,8,); Var-2:(1,3,5,4,6,0,7,2,); Var-3:(1,3,5,7,8,6,4,0,2,);
Heuristics-7: Variable and Value Ordering: Var-1:(1,3,5,9,8,2,7,6,0,4,);
     Var-0:(1,3,5,9,6,4,7,0,2,8,); Var-3:(1,3,5,8,2,6,0,4,7,); Var-2:(1,3,5,4,7,6,2,0,);
Heuristics-8: Variable and Value Ordering: Var-1:(1,3,5,0,9,7,2,8,4,6,);
     Var-0:(1,3,5,4,7,6,0,8,2,9,); Var-3:(1,3,5,4,7,0,6,8,2,); Var-2:(1,3,5,4,6,7,2,0,);
Heuristics-9: Variable and Value Ordering: Var-1:(1,3,5,7,0,9,2,4,8,6,);
     Var-0:(1,3,5,4,0,7,2,6,8,9,); Var-3:(1,3,5,7,4,0,8,2,6,); Var-2:(1,3,5,4,6,7,0,2,);
Heuristics-10: Variable and Value Ordering: Var-1:(1,3,5,7,0,2,4,9,6,8,);
     Var-0:(1,3,5,4,0,9,6,7,8,2,); Var-2:(1,3,5,4,6,0,7,2,); Var-3:(1,3,5,7,4,6,8,0,2,);
Heuristics-11: Variable and Value Ordering: Var-1:(1,3,5,7,0,2,4,9,6,8,);
     Var-0:(1,3,5,6,9,4,0,7,8,2,); Var-2:(1,3,5,4,0,6,2,7,); Var-3:(1,3,5,8,6,7,4,0,2,);
Heuristics-12: Variable and Value Ordering: Var-1:(1,3,5,7,2,0,9,8,4,6,);
     Var-0:(1,3,5,0,9,6,4,2,7,8,); Var-2:(1,3,5,4,6,7,0,2,); Var-3:(1,3,5,7,8,4,6,0,2,);
Heuristics-13: Variable and Value Ordering: Var-1:(1,3,5,0,9,7,4,6,8,2,);
     Var-0:(1,3,5,8,7,6,2,4,0,9,); Var-3:(1,3,5,7,0,8,2,6,4,); Var-2:(1,3,5,2,0,4,6,7,);
Heuristics-14: Variable and Value Ordering: Var-1:(1,3,5,4,7,0,6,9,2,8,);
     Var-0:(1,3,5,6,4,7,8,0,2,9,); Var-2:(1,3,5,0,4,2,6,7,); Var-3:(1,3,5,7,0,4,2,8,6,);
Heuristics-15: Variable and Value Ordering: Var-1:(1,3,5,0,7,4,6,9,2,8,);
     Var-0:(1,3,5,4,6,8,0,7,9,2,); Var-2:(1,3,5,4,0,6,2,7,); Var-3:(1,3,5,7,4,0,6,8,2,);
Heuristics-16: Variable and Value Ordering: Var-1:(1,3,5,7,6,0,2,4,9,8,);
     Var-0:(1,3,5,4,8,6,0,9,7,2,); Var-2:(1,3,5,0,4,6,2,7,); Var-3:(1,3,5,0,6,8,2,7,4,);
Heuristics-17: Variable and Value Ordering: Var-1:(1,3,5,9,8,2,0,7,6,4,);
     Var-0:(1,3,5,6,7,9,4,0,2,8,); Var-3:(1,3,5,8,4,6,7,0,2,); Var-2:(1,3,5,4,7,6,2,0,);
Heuristics-18: Variable and Value Ordering: Var-1:(1,3,5,0,6,9,4,7,8,2,);
     Var-0:(1,3,5,8,6,4,7,0,9,2,); Var-3:(1,3,5,6,4,8,0,7,2,); Var-2:(1,3,5,4,2,6,0,7,);
Heuristics-19: Variable and Value Ordering: Var-1:(1,3,5,7,9,0,2,6,8,4,);
     Var-0:(1,3,5,4,6,0,8,7,9,2,); Var-2:(1,3,5,4,6,0,2,7,); Var-3:(1,3,5,8,6,0,7,4,2,);
Heuristics-20: Variable and Value Ordering: Var-1:(1,3,5,9,7,0,2,8,4,6,);
     Var-0:(1,3,5,4,6,0,7,9,2,8,); Var-2:(1,3,5,4,6,0,7,2,); Var-3:(1,3,5,8,7,2,6,4,0,);
Heuristics-21: Variable and Value Ordering: Var-1:(1,3,5,7,2,6,0,9,4,8,);
     Var-0:(1,3,5,4,9,6,0,8,7,2,); Var-2:(1,3,5,4,6,0,7,2,); Var-3:(1,3,5,8,6,0,4,2,7,);
Heuristics-22: Variable and Value Ordering: Var-1:(1,3,5,7,2,9,6,4,8,0,);
     Var-0:(1,3,5,4,6,9,0,7,8,2,); Var-2:(1,3,5,4,6,0,7,2,); Var-3:(1,3,5,8,2,0,7,4,6,);
Heuristics-23: Variable and Value Ordering: Var-1:(1,3,5,7,2,0,9,6,4,8,);
     Var-0:(1,3,5,4,6,9,0,8,7,2,); Var-2:(1,3,5,4,6,0,7,2,); Var-3:(1,3,5,8,6,0,4,2,7,);
Heuristics-24: Variable and Value Ordering: Var-1:(1,3,5,7,0,6,2,9,4,8,);
     Var-0:(1,3,5,4,8,9,6,0,7,2,); Var-2:(1,3,5,4,6,0,2,7,); Var-3:(1,3,5,6,8,0,4,2,7,);
Heuristics-25: Variable and Value Ordering: Var-1:(1,3,5,7,0,2,4,6,9,8,);
     Var-0:(1,3,5,4,6,0,9,8,7,2,); Var-2:(1,3,5,4,6,0,2,7,); Var-3:(1,3,5,8,0,7,4,6,2,);
Heuristics-26: Variable and Value Ordering: Var-1:(1,3,5,7,2,0,9,6,8,4,);
     Var-0:(1,3,5,4,9,6,0,8,7,2,); Var-2:(1,3,5,4,6,0,7,2,); Var-3:(1,3,5,8,6,4,0,7,2,);
Heuristics-27: Variable and Value Ordering: Var-1:(1,3,5,7,6,9,0,2,4,8,);
     Var-0:(1,3,5,4,8,6,7,0,9,2,); Var-3:(1,3,5,8,0,2,6,4,7,); Var-2:(1,3,5,2,4,0,6,7,);
```

```
Heuristics-28: Variable and Value Ordering: Var-1:(1,3,5,7,2,9,0,4,8,6,);
     Var-0:(1,3,5,4,0,6,9,8,2,7,); Var-2:(1,3,5,4,0,6,7,2,); Var-3:(1,3,5,8,7,2,0,6,4,);
Heuristics-29: Variable and Value Ordering: Var-1:(1,3,5,9,8,0,7,2,6,4,);
     Var-0:(1,3,5,2,7,8,0,6,4,9,); Var-3:(1,3,5,8,2,6,0,4,7,); Var-2:(1,3,5,2,7,4,6,0,);
Heuristics-30: Variable and Value Ordering: Var-1:(1,3,5,7,9,2,0,8,6,4,);
     Var-0:(1,3,5,4,0,8,9,6,2,7,); Var-2:(1,3,5,4,6,0,2,7,); Var-3:(1,3,5,8,6,2,0,7,4,);
Heuristics-31: Variable and Value Ordering: Var-1:(1,3,5,0,7,2,6,4,9,8,);
     Var-0:(1,3,5,4,9,0,8,6,7,2,); Var-2:(1,3,5,4,6,0,2,7,); Var-3:(1,3,5,6,8,4,7,0,2,);
Heuristics-32: Variable and Value Ordering: Var-1:(1,3,5,7,9,2,0,4,8,6,);
     Var-0:(1,3,5,4,6,7,0,2,8,9,); Var-3:(1,3,5,7,2,0,8,4,6,); Var-2:(1,3,5,4,0,6,7,2,);
Heuristics-33: Variable and Value Ordering: Var-1:(1,3,5,0,7,6,2,4,9,8,);
     Var-0:(1,3,5,4,9,6,0,8,7,2,); Var-2:(1,3,5,4,6,2,0,7,); Var-3:(1,3,5,6,8,4,7,0,2,);
Heuristics-34: Variable and Value Ordering: Var-1:(1,3,5,7,2,0,4,9,6,8,);
     Var-0:(1,3,5,4,6,0,9,8,7,2,); Var-2:(1,3,5,4,0,6,2,7,); Var-3:(1,3,5,7,8,0,2,4,6,);
Heuristics-35: Variable and Value Ordering: Var-1:(1,3,5,7,2,9,0,6,4,8,);
     Var-0:(1,3,5,4,8,0,6,9,7,2,); Var-2:(1,3,5,4,0,6,2,7,); Var-3:(1,3,5,8,2,0,6,7,4,);
Heuristics-36: Variable and Value Ordering: Var-1:(1,3,5,0,7,9,4,6,2,8,);
     Var-0:(1,3,5,8,4,0,6,7,2,9,); Var-3:(1,3,5,7,6,8,0,4,2,); Var-2:(1,3,5,4,2,0,6,7,);
Heuristics-37: Variable and Value Ordering: Var-1:(1,3,5,7,2,9,0,6,4,8,);
     Var-0:(1,3,5,4,9,6,0,8,7,2,); Var-2:(1,3,5,4,6,0,7,2,); Var-3:(1,3,5,8,2,6,0,7,4,);

2019-01-12 10:44:41 INFO  Library:151 - RESULTS:
CSP-Test0: , var-0= 3, var-1= 3, var-2= 0, var-3= 0, runtime: 91906.0 ns, prediction quality: 0.5
CSP-Test1: , var-0= 3, var-1= 3, var-2= 0, var-3= 0, runtime: 61115.0 ns, prediction quality: 0.5
CSP-Test2: , var-0= 3, var-1= 3, var-2= 0, var-3= 0, runtime: 9331.0 ns, prediction quality: 0.5
CSP-Test3: , var-0= 3, var-1= 3, var-2= 0, var-3= 0, runtime: 7465.0 ns, prediction quality: 0.5
CSP-Test4: , var-0= 3, var-1= 3, var-2= 0, var-3= 0, runtime: 7464.0 ns, prediction quality: 0.5
CSP-Test5: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 6998.0 ns, prediction quality: 0.5
CSP-Test6: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 6998.0 ns, prediction quality: 0.5
CSP-Test7: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 6531.0 ns, prediction quality: 0.5
CSP-Test8: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 6998.0 ns, prediction quality: 0.5
CSP-Test9: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 6531.0 ns, prediction quality: 0.5
CSP-Test10: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 6998.0 ns, prediction quality: 0.5
CSP-Test11: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 7464.0 ns, prediction quality: 0.5
CSP-Test12: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 8864.0 ns, prediction quality: 0.5
CSP-Test13: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 9797.0 ns, prediction quality: 0.5
CSP-Test14: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 8398.0 ns, prediction quality: 0.5
CSP-Test15: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 7464.0 ns, prediction quality: 0.5
CSP-Test16: , var-0= 1, var-1= 1, var-2= 0, var-3= 0, runtime: 6998.0 ns, prediction quality: 0.5
CSP-Test17: , var-0= 5, var-1= 5, var-2= 0, var-3= 0, runtime: 19594.0 ns, prediction quality: 0.5
CSP-Test18: , var-0= 5, var-1= 5, var-2= 0, var-3= 0, runtime: 14928.0 ns, prediction quality: 0.5
2019-01-12 10:44:41 INFO  Library:152 - END (2019/01/12 10:44:41)
2019-01-12 10:44:41 INFO  Library:153 -
##################################################

2019-01-12 10:44:41 INFO  Library:72 -
##################################################
2019-01-12 10:44:41 INFO  Library:73 - START (2019/01/12 10:44:41)
2019-01-12 10:44:41 INFO  Library:74 - heuristicsID: MFCO
2019-01-12 10:44:41 INFO  Library:75 - solverID: choco
2019-01-12 10:44:41 INFO  Library:76 - ClusteringAlgorithmID: kmeans
2019-01-12 10:44:41 INFO  Library:77 - PerformanceIndicator: predictionQuality
2019-01-12 10:44:41 INFO  Library:78 - stoppingCriteria: 10
2019-01-12 10:44:41 INFO  Library:79 - diagnosisAlgorithmID: fastdiag
2019-01-12 10:44:41 INFO  Library:80 - inputFile: IOFOLDER/INPUT/test/LibraryInputFolder1
2019-01-12 10:44:41 INFO  Library:81 - outputFolder: IOFOLDER/OUTPUT/test/LibraryOutputFolder1
2019-01-12 10:44:42 INFO  Library:112 - LEARNED HEURISTICS:
Heuristics-0: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-1: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-2: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-3: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-4: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-5: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-6: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-7: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-8: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-9: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-10: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-11: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-12: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-13: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-14: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-15: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-16: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-17: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-18: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-19: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-20: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-21: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-22: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
```

```
Heuristics-23: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-24: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-25: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-26: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-27: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-28: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-29: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-30: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-31: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-32: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-33: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-34: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-35: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-36: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;
Heuristics-37: Constraint Ordering: VAR-1; VAR-3; VAR-0; VAR-2;

2019-01-12 10:44:42 INFO  Library:151 - RESULTS:
CSP-Test0: CONSTRAINT: var-1 = 8;, runtime: 735248.0 ns, prediction quality: -1.0
CSP-Test1: CONSTRAINT: var-1 = 8;, runtime: 334500.0 ns, prediction quality: -1.0
CSP-Test2: CONSTRAINT: var-1 = 6;, runtime: 218801.0 ns, prediction quality: -1.0
CSP-Test3: CONSTRAINT: var-1 = 7;, runtime: 215069.0 ns, prediction quality: -1.0
CSP-Test4: CONSTRAINT: var-1 = 7;, runtime: 216936.0 ns, prediction quality: -1.0
CSP-Test5: CONSTRAINT: var-1 = 6;, runtime: 208538.0 ns, prediction quality: -1.0
CSP-Test6: CONSTRAINT: var-1 = 7;, runtime: 210871.0 ns, prediction quality: -1.0
CSP-Test7: CONSTRAINT: var-1 = 7;, runtime: 220201.0 ns, prediction quality: -1.0
CSP-Test8: CONSTRAINT: var-1 = 8;, runtime: 223466.0 ns, prediction quality: -1.0
CSP-Test9: CONSTRAINT: var-1 = 8;, runtime: 223933.0 ns, prediction quality: -1.0
CSP-Test10: CONSTRAINT: var-1 = 8;, runtime: 211337.0 ns, prediction quality: -1.0
CSP-Test11: CONSTRAINT: var-1 = 6;, runtime: 217402.0 ns, prediction quality: -1.0
CSP-Test12: CONSTRAINT: var-1 = 6;, runtime: 218801.0 ns, prediction quality: -1.0
CSP-Test13: CONSTRAINT: var-1 = 6;, runtime: 217402.0 ns, prediction quality: -1.0
CSP-Test14: CONSTRAINT: var-1 = 7;, runtime: 205272.0 ns, prediction quality: -1.0
CSP-Test15: CONSTRAINT: var-1 = 9;, runtime: 200607.0 ns, prediction quality: -1.0
CSP-Test16: CONSTRAINT: var-1 = 8;, runtime: 215536.0 ns, prediction quality: -1.0
CSP-Test17: CONSTRAINT: var-1 = 6;, runtime: 209004.0 ns, prediction quality: -1.0
CSP-Test18: CONSTRAINT: var-1 = 9;, runtime: 248193.0 ns, prediction quality: -1.0
2019-01-12 10:44:42 INFO  Library:152 - END (2019/01/12 10:44:41)
2019-01-12 10:44:42 INFO  Library:153 -
##################################################
```

Bibliography

Gediminas Adomavicius and Alexander Tuzhilin. 2015. *Context-Aware Recommender Systems*. Springer US, Boston, MA, 191–226. https://doi.org/10.1007/978-1-4899-7637-6_6 (Cited on page 77.)

Ethem Alpaydin. 2016. *Machine learning: the new AI*. MIT press. (Cited on page 6.)

F. Amato, A. Mazzeo, V. Moscato, and A. Picariello. 2013. A Recommendation System for Browsing of Multimedia Collections in the Internet of Things. In *Internet of Things and Inter-Cooperative Computational Technologies for Collective Intelligence*, N. Bessis, F. Xhafa, D. Varvarigou, R. Hill, and M. Li (Eds.). Studies in Computational Intelligence, Vol. 460. Springer, 391–411. (Cited on pages 5, 76, and 93.)

H. Andersen. 1999. An Introduction to Binary Decision Diagrams. In *Lecture Notes for Efficient Algorithms and Programs*. 1–35. (Cited on pages 31 and 32.)

Hariprasad Anumala and Shiva Murthy Busetty. 2015. Distributed device health platform using Internet of Things devices. In *Data Science and Data Intensive Systems (DSDIS), 2015 IEEE International Conference on*. IEEE, 525–531. (Cited on page 94.)

Liliana Ardissono, Alexander Felfernig, Gerhard Friedrich, Anna Goy, Dietmar Jannach, Markus Meyer, Giovanna Petrone, Ralph Schäfer, Wilken Schuetz, Markus Zanker, et al. 2002. Personalising on-line configuration of products and services. In *ECAI*, Vol. 2. 225–229. (Cited on pages 4 and 53.)

Muesluem Atas, Alexander Felfernig, Seda Polat Erdeniz, Stefan Reiterer, Amal Shehadeh, and Thi Ngoc Trang Tran. 2017. Cluster-Based Constraint Ordering for Direct Diagnosis. In *19 th International Configuration Workshop*. 68. (Cited on pages 87 and 88.)

L. Atzori, A. Iera, and G. Morabito. 2010. The Internet of Things: A Survey. *Computer Networks* 54 (2010), 2787–2805. Issue 15. (Cited on pages 2, 22, 28, 32, and 75.)

Paritosh Bahirat, Yangyang He, Abhilash Menon, and Bart Knijnenburg. 2018. A Data-Driven Approach to Developing IoT Privacy-Setting Interfaces. In *23rd International Conference on Intelligent User Interfaces*. ACM, 165–176. (Cited on page 77.)

René R Bakker, F Dikker, Frank Tempelman, and Petronella Maria Wognum. 1993. Diagnosing and solving over-determined constraint satisfaction problems. In *IJCAI*, Vol. 93. 276–281. (Cited on pages 27, 43, and 65.)

Marcello Balduccini. 2011. Learning and Using Domain-specific Heuristics in ASP Solvers. *AI Commun.* 24, 2 (2011), 147–164. (Cited on page 4.)

Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot. 2017. Dynamic graph coloring. In *Workshop on Algorithms and Data Structures*. Springer, 97–108. (Cited on page 34.)

Nicolas Barnier and Pascal Brisset. 2004. Graph coloring for air traffic flow management. *Annals of operations research* 130, 1-4 (2004), 163–178. (Cited on page 35.)

J Christopher Beck, Patrick Prosser, and Richard J Wallace. 2004. Variable ordering heuristics show promise. *CP* 4 (2004), 711–715. (Cited on page 33.)

D. Benavides, A. Felfernig, J. Galindo, and F. Reinfrank. 2013. Automated Analysis in Feature Modelling and Product Configuration. In *13th International Conference on Software Reuse (ICSR 2013) (LNCS)*. Pisa, Italy, 160–175. (Cited on page 33.)

Idir Benouaret and Dominique Lenne. 2015. Personalizing the museum experience through context-aware recommendations. In *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*. IEEE, 743–748. (Cited on page 77.)

Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. 2004. Boosting systematic search by weighting constraints. In *Proceedings of the 16th European Conference on Artificial Intelligence*. IOS Press, 146–150. (Cited on page 42.)

Christos Boutsidis and Efstratios Gallopoulos. 2008. SVD based initialization: A head start for nonnegative matrix factorization. *Pattern Recognition* 41, 4 (2008), 1350–1362. (Cited on page 58.)

Daniel Brélaz. 1979. New methods to color the vertices of a graph. *Commun. ACM* 22, 4 (1979), 251–256. (Cited on page 35.)

R. Burke. 2000. Knowledge-based recommender systems. *Encyclopedia of Library and Information Systems* 69, 32 (2000), 180–200. (Cited on page 19.)

R. Burke. 2002. Hybrid Recommender Systems: Survey and Experiments. *UMUAI Journal* 12, 4 (2002), 331–370. (Cited on pages 3, 17, 21, 53, and 82.)

R. Burke, A. Felfernig, and M. Goeker. 2011. Recommender Systems: An Overview. *AI Magazine* 32, 3 (2011), 13–18. (Cited on page 3.)

Robin D Burke, Kristian J Hammond, and BC Yound. 1997. The FindMe approach to assisted browsing. *IEEE Expert* 12, 4 (1997), 32–40. (Cited on page 89.)

Massimiliano Caramia and Paolo Dell'Olmo. 2002. Constraint propagation in graph coloring. *Journal of Heuristics* 8, 1 (2002), 83–107. (Cited on page 35.)

Stuart K Card, George G Robertson, and Jock D Mackinlay. 1991. The information visualizer, an information workspace. In *Proceedings of the SIGCHI Conference on Human factors in computing systems*. ACM, 181–186. (Cited on pages 43 and 65.)

Fran Casino, Edgar Batista, Constantinos Patsakis, and Agusti Solanas. 2015. Context-aware recommender for smart health. In *Smart Cities Conference (ISC2), 2015 IEEE First International*. IEEE, 1–2. (Cited on page 93.)

Carlos Castro. 1996. Solving binary CSP using computational systems. *Electronic Notes in Theoretical Computer Science* 4 (1996), 246–265. (Cited on page 5.)

S. Cha, M. Ruiz, M. Wachowicz, L. Tran, H. Cao, and I. Maduako. 2016. The role of an IoT Platform in the Design of Real-time Recommender Systems. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. Reston, VA, USA, 448–453. (Cited on page 77.)

Chee-Yong Chan and Yannis E Ioannidis. 1998. Bitmap index design and evaluation. In *ACM SIGMOD Record*, Vol. 27. ACM, 355–366. (Cited on page 56.)

Feng Chen, Pan Deng, Jiafu Wan, Daqiang Zhang, Athanasios V Vasilakos, and Xiaohui Rong. 2015. Data mining for the internet of things: literature review and challenges. *International Journal of Distributed Sensor Networks* 11, 8 (2015), 431047. (Cited on pages 91 and 102.)

Hsinchun Chen, Roger HL Chiang, and Veda C Storey. 2012. Business intelligence and analytics: From big data to big impact. *MIS quarterly* 36, 4 (2012). (Cited on page 93.)

Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27. (Cited on page 58.)

Giacomo Da Col and Erich C Teppan. 2017. Learning Constraint Satisfaction Heuristics for Configuration Problems. In *19 th International Configuration Workshop*. 8. (Cited on page 33.)

Soumya Kanti Datta, Christian Bonnet, Amelie Gyrard, Rui Pedro Ferreira Da Costa, and Karima Boudaoud. 2015. Applying Internet of Things for personalized healthcare in smart homes. In *Wireless and Optical Communication Conference (WOCC), 2015 24th*. IEEE, 164–169. (Cited on page 94.)

Lawrence Davis. 1991. Handbook of genetic algorithms. (1991). (Cited on page 6.)

Johan de Kleer. 1990. Using crude probability estimates to guide diagnosis. *Artificial Intelligence* 45, 3 (1990), 381–391. (Cited on page 66.)

Rina Dechter and Judea Pearl. 1988. Network-based heuristics for constraint-satisfaction problems. In *Search in artificial intelligence*. Springer, 370–425. (Cited on page 33.)

Claudio Di Ciccio, Fabrizio Maria Maggi, Marco Montali, and Jan Mendling. 2017. Resolving Inconsistencies and Redundancies in Declarative Process Models. *Inf. Sys.* 64 (2017), 425–446. (Cited on page 4.)

Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. 1990. Solving large combinatorial problems in logic programming. *The Journal of Logic Programming* 8, 1-2 (1990), 75–93. (Cited on page 35.)

Pedro M Domingos. 2012. A few useful things to know about machine learning. *Commun. acm* 55, 10 (2012), 78–87. (Cited on page 116.)

Lian Duan, W Nick Street, and E Xu. 2011. Healthcare information systems: data mining methods in the creation of a clinical recommender system. *Enterprise Information Systems* 5, 2 (2011), 169–181. (Cited on page 94.)

S. L. Epstein and R. J. Wallace. 2006. Finding Crucial Subproblems to Focus Global Search. In *18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*. 151–162. (Cited on page 4.)

Seda Polat Erdeniz and Alexander Felfernig. 2018a. CLUSTER AND LEARN: Cluster-Specific Heuristics for Graph Coloring. In *International Conference on the Practice and Theory of Automated Timetabling*. Elsevier, 401–404. (Cited on pages 7, 27, and 114.)

Seda Polat Erdeniz and Alexander Felfernig. 2018b. OCSH: optimized cluster specific heuristics for the university course timetabling problem. In *Proceedings of the 8th International Conference on Information Systems and Technologies*. ACM, 13–18. (Cited on pages 7, 27, and 114.)

Seda Polat Erdeniz, Alexander Felfernig, and Muesluem Atas. 2018a. Learn Diag: A Direct Diagnosis Algorithm Based On Learned Heuristics. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*. Springer, 190–197. (Cited on pages 27, 114, and 115.)

Seda Polat Erdeniz, Alexander Felfernig, and Muesluem Atas. 2019a. Matrix Factorization based Heuristics for Direct Diagnosis. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE'19)*. Springer. (Cited on pages 7, 27, 114, and 115.)

Seda Polat Erdeniz, Alexander Felfernig, Muesluem Atas, Thi Ngoc Trang Tran, Michael Jeran, and Martin Stettinger. 2017. Cluster-Specific Heuristics for Constraint Solving. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 21–30. (Cited on pages 7, 27, 33, 43, 84, 86, and 114.)

Seda Polat Erdeniz, Alexander Felfernig, Ralph Samer, and Muesluem Atas. 2019b. Matrix factorization based heuristics for constraint-based recommenders. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC'19)*. ACM, 1655–1662. (Cited on pages 7, 27, and 114.)

Seda Polat Erdeniz, Ilias Maglogiannis, Andreas Menychtas, Alexander Felfernig, and Thi Ngoc Trang Tran. 2018b. Recommender Systems for IoT Enabled m-Health Applications. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 227–237. (Cited on pages 7, 75, 76, and 113.)

A. Falkner, A. Felfernig, and A. Haag. 2011. Recommendation Technologies for Configurable Products. *AI Magazine* 32, 3 (2011), 99–108. (Cited on page 84.)

A. Falkner and H. Schreiner. 2014. SIEMENS: Configuration and Reconfiguration in Industry. In *Knowledge-based Configuration – From Research to Business Cases*, A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen (Eds.). Morgan Kaufmann, 199–210. (Cited on pages 2 and 32.)

Boi Faltings and Rainer Weigel. 1994. Constraint-based knowledge representation for configuration systems. *Technical Report No. TR-94/59, Department D'Informatique* (1994). (Cited on pages 10 and 11.)

Alexander Felfernig, Muesluem Atas, Thi Ngoc Trang Tran, Martin Stettinger, Seda Polat Erdeniz, and Gerhard Leitner. 2017a. *An Analysis of Group Recommendation Heuristics for High- and Low-Involvement Items*. Springer International Publishing, Cham, 335–344. https://doi.org/10.1007/978-3-319-60042-0_39 (Cited on pages 89 and 90.)

Alexander Felfernig, Ludovico Boratto, Martin Stettinger, and Marko Tkalčič. 2018a. *Group Recommender Systems: An Introduction*. Springer. (Cited on pages 17, 21, and 89.)

A. Felfernig and R. Burke. 2008. Constraint-based Recommender Systems: Technologies and Research Issues. In *ACM International Conference on Electronic Commerce (ICEC08)*. Innsbruck, Austria, 17–26. (Cited on pages 3, 4, 17, 19, 20, 53, 89, 90, and 92.)

A. Felfernig, S. Polat Erdeniz, P. Azzoni, M. Jeran, A. Akcay, and C. Doukas. 2016. Towards Configuration Technologies for IoT Gateways. In *International Workshop on Configuration 2016 (ConfWS'16)*. Toulouse, France, 73–76. (Cited on pages 7, 27, 76, 78, 90, 91, and 113.)

Alexander Felfernig, Seda Polat Erdeniz, Michael Jeran, Arda Akcay, Paolo Azzoni, Matteo Maiero, and Charalampos Doukas. 2017b. Recommendation Technologies for IoT Edge Devices. *Procedia Computer Science* 110 (2017), 504–509. (Cited on page 91.)

Alexander Felfernig, Gerhard Friedrich, and Dietmar Jannach. 2001. Conceptual modeling for configuration of mass-customizable products. *Artificial Intelligence in Engineering* 15, 2 (2001), 165–176. (Cited on page 33.)

Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Stumptner. 2004. Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence* 152, 2 (2004), 213–234. (Cited on pages 1, 29, 30, and 33.)

Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Markus Stumptner, et al. 2000. Consistency-based diagnosis of configuration knowledge bases. In *ECAI*. 146–150. (Cited on page 33.)

A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. 2006. An Environment for the Development of Knowledge-based Recommender Applications. *International Journal of Electronic Commerce (IJEC)* 11, 2 (2006), 11–34. (Cited on page 32.)

Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. 2015a. Constraint-based recommender systems. In *Recommender Systems Handbook*. Springer, 161–190. (Cited on page 17.)

A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen. 2014a. *Knowledge-based Configuration: From Research to Business Cases* (1st ed.). Elsevier/Morgan Kaufmann Publishers. (Cited on page 3.)

Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen. 2014b. *Knowledge-based configuration: From research to business cases*. Newnes. (Cited on pages 9, 28, 29, and 91.)

Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen. 2014c. *Knowledge-based Configuration: From Research to Business Cases* (1 ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. (Cited on pages 12 and 16.)

Alexander Felfernig, Monika Mandl, Stefan Schippel, Monika Schubert, and Erich Teppan. 2010. Adaptive utility-based recommendation. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 641–650. (Cited on page 88.)

Alexander Felfernig, Seda Polat-Erdeniz, Christoph Uran, Stefan Reiterer, Muesluem Atas, Thi Ngoc Trang Tran, Paolo Azzoni, Csaba Kiraly, and Koustabh Dolui. 2018b. An overview of recommender systems in the internet of things. *Journal of Intelligent Information Systems* (2018), 1–25. (Cited on pages 7, 75, and 113.)

Alexander Felfernig, Monika Schubert, and Christoph Zehentner. 2012. An Efficient Diagnosis Algorithm for Inconsistent Constraint Sets. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AIEDAM)* 26, 1 (2012), 53–62. (Cited on pages 66 and 67.)

Alexander Felfernig, Rouven Walter, José A Galindo, David Benavides, Seda Polat Erdeniz, Müslüm Atas, and Stefan Reiterer. 2018c. Anytime diagnosis for reconfiguration. *Journal of Intelligent Information Systems* (2018), 1–22. (Cited on pages 33, 42, 43, 47, 65, 66, 67, 71, and 73.)

A. Felfernig, R. Walter, and S. Reiterer. 2015b. FlexDiag: AnyTime Diagnosis for Reconfiguration. In *16th International Workshop on Configuration*. Vienna, Austria, 105–110. (Cited on page 30.)

Klaus Finkenzeller. 2010. *RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication*. John Wiley & Sons. (Cited on pages 5, 76, and 77.)

G. Fleischanderl, G. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner. 1998. Configuring Large Systems Using Generative Constraint Satisfaction. *IEEE Intelligent Systems* 13, 4 (1998), 59–68. (Cited on page 29.)

Charles Fleurent and Jacques A Ferland. 1996. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research* 63, 3 (1996), 437–461. (Cited on page 35.)

Kenneth R Fox. 1999. The influence of physical activity on mental well-being. *Public health nutrition* 2, 3a (1999), 411–418. (Cited on page 92.)

R. Frey, R. Xu, and A. Ilic. 2015. A Novel Recommender System in IoT. In *5th International Conference on the Internet of Things (IoT 2015)*. Seoul, South Korea, 1–2. (Cited on pages 3, 11, and 93.)

S. Greengard. 2015. *The Internet of Things*. MIT Press. (Cited on pages 22, 75, and 76.)

Chris Groër, Bruce Golden, and Edward Wasil. 2010. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation* 2, 2 (2010), 79–101. (Cited on pages 4 and 53.)

Christian Guilleminault, Alex Clerk, Jed Black, Michael Labanowski, Rafael Pelayo, and David Claman. 1995. Nondrug treatment trials in psychophysiologic insomnia. *Archives of Internal Medicine* 155, 8 (1995), 838–844. (Cited on page 98.)

Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and MC Hsu. 2001. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *proceedings of the 17th international conference on data engineering*. 215–224. (Cited on page 83.)

F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2016), 19. (Cited on page 116.)

Alain Hertz and Dominique de Werra. 1987. Using tabu search techniques for graph coloring. *Computing* 39, 4 (1987), 345–351. (Cited on page 35.)

Hengyi Hu, Adam Elkus, and Larry Kerschberg. 2016. A Personal Health Recommender System incorporating personal health records, modular ontologies, and crowd-sourced data. In *Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on*. IEEE, 1027–1033. (Cited on page 94.)

Anil K Jain. 2010. Data clustering: 50 years beyond K-means. *Pattern recognition letters* 31, 8 (2010), 651–666. (Cited on page 46.)

Dietmar Jannach. 2013. Toward Automatically Learned Search Heuristics for CSP-encoded Configuration Problems - Results from an Initial Experimental Analysis. In *Proceedings of the 15th International Configuration Workshop, Vienna, Austria, August 29-30, 2013*. 9–13. (Cited on pages 4, 5, 15, and 33.)

D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. 2010. *Recommender Systems – An Introduction*. Cambridge University Press. (Cited on pages 3, 17, 20, 21, 79, 80, 82, 86, 91, 97, and 98.)

Y. Jia and M. Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering* 37, 5 (2011), 649–678. (Cited on page 30.)

M Johnston and Steven Minton. 1994. Analyzing a heuristic strategy for constraint satisfaction and scheduling. *Intelligent scheduling* (1994), 257–289. (Cited on page 33.)

Ulrich Junker. 2001. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints*. (Cited on page 67.)

Ulrich Junker. 2004. Preferred explanations and relaxations for over-constrained problems. In *AAAI-2004*. (Cited on page 66.)

Narendra Jussien, Guillaume Rochart, and Xavier Lorca. 2008. Choco: an open source java constraint programming library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*. 1–10. (Cited on page 42.)

Elias B Khalil, Bistra Dilkina, George L Nemhauser, Shabbir Ahmed, and Yufen Shao. 2017. Learning to Run Heuristics in Tree Search.. In *Proceedings of the international joint conference on artificial intelligence. AAAI Press, Melbourne, Australia*. 659–666. (Cited on page 43.)

J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. 1997. GroupLens: applying collaborative filtering to Usenet news Full text. *Comm. of the ACM* 40, 3 (1997), 77–87. (Cited on pages 17 and 18.)

Yehuda Koren. 2010. Collaborative filtering with temporal dynamics. *Commun. ACM* 53, 4 (2010), 89–97. (Cited on pages 3 and 67.)

Y. Koren, R. Bell, and C. Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer* 42, 8 (2009), 30–37. (Cited on pages 6, 68, and 89.)

T. Krebs, L. Hotz, and A. Günter. 2002. Knowledge-based Configuration for Configuring Combined Hardware/Software Systems. In *Proceedings of PuK'2002*. Freiburg, Germany, 1–6. (Cited on pages 2 and 32.)

Vipin Kumar. 1992. Algorithms for constraint-satisfaction problems: A survey. *AI magazine* 13, 1 (1992), 32. (Cited on pages 12, 13, 14, 33, and 34.)

Jin-Shyan Lee, Yu-Wei Su, and Chung-Chou Shen. 2007. A comparative study of wireless protocols: Bluetooth, UWB, ZigBee, and Wi-Fi. In *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*. Ieee, 46–51. (Cited on page 77.)

Frank Thomson Leighton. 1979. A graph coloring algorithm for large scheduling problems. *Journal of research of the national bureau of standards* 84, 6 (1979), 489–506. (Cited on pages 34 and 35.)

G. Leitner, A. Felfernig, A. Fercher, and M. Hitz. 2014. Disseminating Ambient Assisted Living in the Rural Area. *Sensors* 14, 8 (2014), 13496–13531. (Cited on page 76.)

Xingjian Li and Susan L Epstein. 2010. Learning Cluster-based Structure to Solve Constraint Satisfaction Problems. *Annals of Mathematics and AI* 60, 1–2 (2010), 91–117. (Cited on pages 4, 5, and 33.)

Y. Liu, Y. Jiang, and H. Qian. 2008. Topology-based Variable Ordering Strategy for Solving Disjunctive Temporal Problems. In *15th International Symposium on Temporal Representation and Reasoning*. IEEE, 129–136. (Cited on pages 4, 5, and 33.)

Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137. (Cited on page 38.)

James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA., 281–297. (Cited on page 38.)

C. Magerkurth, K. Sperner, S. Meyer, and M. Strohbach. 2011. Towards Context-Aware Retail Environments: An Infrastructure Perspective. In *Mobile-HCI 2011*. Stockholm, Sweden, 1–4. (Cited on pages 3, 5, and 76.)

Ilias Maglogiannis, Charalampos Ioannou, and Panayiotis Tsanakas. 2016. Fall detection and activity identification using wearable and hand-held devices. *Integrated Computer-Aided Engineering* 23, 2 (2016), 161–172. (Cited on pages 76 and 91.)

J. Marques-Silva, F. Heras, M. Janota, A. Previti, and A. Belov. 2013. On computing minimal correction subsets. In *IJCAI*. 615–622. (Cited on page 30.)

Paul Martin, Bo-Jhang Ho, Nicholas Grupen, Samuel Munoz, and Mani Srivastava. 2014. An ibeacon primer for indoor localization: demo abstract. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. ACM, 190–191. (Cited on page 77.)

S. Di Martino and S. Rossi. 2016. An Architecture for a Mobility Recommender System in Smart Cities. *Procedia Computer Science* 98 (2016), 425–430. (Cited on page 77.)

Dániel Marx. 2004. Graph coloring with local and global constraints. (2004). (Cited on page 34.)

I. Mashal, O. Alsaryrah, and Tein-Yaw Chung. 2016. Performance evaluation of recommendation algorithms on Internet of Things Services. *Physica A* 451 (2016), 646–656. (Cited on page 76.)

J. Masthoff. 2011. Group Recommender Systems. *Recommender Systems Handbook* (2011), 677–702. (Cited on pages 17, 21, 81, and 89.)

Michael J McGrath and Cliodhna Ní Scanaill. 2013. Wellness, fitness, and lifestyle sensing applications. In *Sensor Technologies*. Springer, 217–248. (Cited on page 75.)

Andreas Menychtas, Charalampos Doukas, Panayiotis Tsanakas, and Ilias Maglogiannis. 2017. A Versatile Architecture for Building IoT Quantified-Self Applications. In *2017 IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS)*. IEEE, 500–505. (Cited on page 95.)

Andreas Menychtas, Panayiotis Tsanakas, and Ilias Maglogiannis. 2016. Automated integration of wireless biosignal collection devices for patient-centred decision-making in point-of-care systems. *Healthcare Technology Letters* 3, 1 (2016), 34–40. (Cited on pages 76, 91, and 92.)

Elie Merhej, Steven Schockaert, and Martine De Cock. 2017. Repairing inconsistent answer set programs using rules of thumb: A gene regulatory networks case study. *International Journal of Approximate Reasoning* 83 (2017), 243–264. (Cited on page 4.)

Laurent Michel and Pascal Van Hentenryck. 2012. Activity-based search for black-box constraint programming solvers. *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems* (2012), 228–243. (Cited on page 42.)

D. Miorandi, S. Sicari, F. DePellegrini, and I. Chlamtac. 2012. Internet of Things: Vision, Applications and Research Challenges. *Ad Hoc Networks* 10 (2012), 1497–1516. Issue 7. (Cited on page 75.)

CA Morgenstern and HD Shapiro. 1986. *Chromatic number approximation using simulated annealing, Department of Computer Science, The University of New Mexico*. Technical Report. Albuquerque, Technical Report, CS86-1. (Cited on page 35.)

Malek Mouhoub and Bahareh Jafari. 2011. Heuristic techniques for variable and value ordering in CSPs. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, 457–464. (Cited on page 33.)

M. Munoz-Organero, G. Ramirez-Gonzalez, P. Munoz-Merino, and C. Loos. 2010. A Collaborative Recommender System Based on Space-Time Similarities. *IEEE Pervasice Computing* 9, 3 (2010), 81–87. (Cited on pages 3, 76, and 77.)

Sean A Munson and Sunny Consolvo. 2012. Exploring goal-setting, rewards, self-monitoring, and sharing to motivate physical activity. In *Pervasive computing technologies for healthcare (PervasiveHealth), 2012 6th international conference on*. IEEE, 25–32. (Cited on page 75.)

V. Myllärniemi, J. Tiihonen, M. Raatikainen, and A. Felfernig. 2014. Using Answer Set Programming for Feature Model Representation and Configuration. In *Workshop on Configuration*. Novi Sad, 1–8. (Cited on page 29.)

Nina Narodytska and Toby Walsh. 2007. Constraint and Variable Ordering Heuristics for Compiling Configuration Problems.. In *IJCAI*. 149–154. (Cited on page 14.)

Iulia Nica, Ingo Pill, Thomas Quaritsch, and Franz Wotawa. 2013. The Route to Success-A Performance Comparison of Diagnosis Algorithms.. In *IJCAI*, Vol. 13. 1039–1045. (Cited on page 43.)

I Nica, F Wotawa, R Ochenbauer, C Schober, H Hofbauer, and S Boltek. 2014. Kapsch: reconfiguration of mobile phone networks. *Knowledge-based Configuration–From Research to Business Cases, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen* (2014), 287–300. (Cited on page 43.)

B. O'Sullivan, A. Ferguson, and E. C. Freuder. 2004. Boosting constraint satisfaction using decision trees. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*. 646–651. (Cited on pages 4, 5, and 33.)

Yunhe Pan. 2016. Heading toward artificial intelligence 2.0. *Engineering* 2, 4 (2016), 409–413. (Cited on page 1.)

Christos Panagopoulos, Foteini Malli, Andreas Menychtas, Efstathia-Petrina Smyrli, Aikaterini Georgountzou, Zoe Daniil, Konstantinos I Gourgoulianis, Panayiotis Tsanakas, and Ilias Maglogiannis. 2017. Utilizing a Homecare Platform for Remote Monitoring of Patients with Idiopathic Pulmonary Fibrosis. In *GeNeDis 2016*. Springer, 177–187. (Cited on page 92.)

Manos Papagelis and Dimitris Plexousakis. 2005. Qualitative analysis of user-based and item-based prediction algorithms for recommendation agents. *Engineering Applications of Artificial Intelligence* 18, 7 (2005), 781–789. (Cited on page 56.)

Giselle Soares Passos, Dalva Poyares, Marcos Gonçalves Santana, Carolina Vicaria Rodrigues D'Aurea, Shawn D Youngstedt, Sergio Tufik, and Marco Túlio de Mello. 2011. Effects of moderate aerobic exercise training on chronic primary insomnia. *Sleep medicine* 12, 10 (2011), 1018–1027. (Cited on page 98.)

Giselle S Passos, Dalva Poyares, Marcos G Santana, Sergio Tufik, Marco Tú, et al. 2010. Effect of acute physical exercise on patients with chronic primary insomnia. *Journal of clinical sleep medicine* 6, 03 (2010), 270–275. (Cited on page 98.)

Puntip Pattaraintakorn, Gregory M Zaverucha, and Nick Cercone. 2007. Web based health recommender system using rough sets, survival analysis and rule-based expert systems. In *International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*. Springer, 491–499. (Cited on page 94.)

M. Pazzani and D. Billsus. 1997. Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning* 27 (1997), 313–331. (Cited on pages 17 and 18.)

Judea Pearl. 1984. Heuristics: intelligent search strategies for computer problem solving. (1984). (Cited on pages 5 and 33.)

C. Perera, C. Liu, and S. Jayawardena. 2015. The Emerging Internet of Things Marketplace From an Industrial Perspective: A Survey. *IEEE Transactions on Emerging Topics in Computing* 3, 4 (2015), 585–598. (Cited on page 32.)

C. Perera, A. Zaslavsky, M. Compton, P. Christen, and D. Georgakopoulos. 2013. Semantic-driven Configuration of Internet of Things Middleware. In *9th International Conference on Semantics, Knowledge & Grids (SKG)*. Beijing, China, 66–73. (Cited on pages 2 and 32.)

Steven Prestwich. 2001. Local search and backtracking vs non-systematic backtracking. In *AAAI 2001 Fall symposium on using uncertainty within computation*. 109–115. (Cited on page 35.)

Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. 2016. Choco solver documentation. *TASC, INRIA Rennes, LINA CNRS UMR* 6241 (2016). (Cited on pages 13, 42, and 85.)

P. Ray. 2015. Generic Internet of Things Architecture for Smart Sports. In *International Conference on Control, Instrumentation, and Communication Technologies (ICCICCT)*. 405–410. (Cited on page 76.)

Philippe Refalo. 2004. Impact-based search strategies for constraint programming. *CP* 3258 (2004), 557–571. (Cited on page 42.)

Kathryn J Reid, Kelly Glazer Baron, Brandon Lu, Erik Naylor, Lisa Wolfe, and Phyllis C Zee. 2010. Aerobic exercise improves self-reported sleep and quality of life in older adults with insomnia. *Sleep medicine* 11, 9 (2010), 934–940. (Cited on page 98.)

R. Reiter. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32, 1 (1987), 57–95. (Cited on pages 29 and 67.)

Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 1–35. (Cited on pages 4 and 53.)

Francesco Ricci, Lior Rokach, and Bracha Shapira. 2015. Recommender systems: introduction and challenges. In *Recommender systems handbook*. Springer, 1–34. (Cited on pages 1, 18, and 19.)

Daniel Sabin and Rainer Weigel. 1998. Product Configuration Frameworks - A Survey. *IEEE Intelligent Systems* 13, 4 (1998), 42–49. (Cited on pages 9, 27, and 28.)

Norman Sadeh and Mark S. Fox. 1996. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *AI Journal* 86, 1 (1996), 1–41. https://doi.org/10.1016/0004-3702(95)00098-4 (Cited on pages 5, 15, and 33.)

Jeff J Sandvig, Bamshad Mobasher, and Robin Burke. 2007. Robustness of collaborative recommendation based on association rule mining. In *Proceedings of the 2007 ACM conference on Recommender systems*. ACM, 105–112. (Cited on pages 4 and 53.)

Hanna Schäfer. 2016. Personalized Support for Healthy Nutrition Decisions. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 455–458. (Cited on page 94.)

Hanna Schäfer, Santiago Hors-Fraile, Raghav Pavan Karumur, André Calero Valdez, Alan Said, Helma Torkamaan, Tom Ulmer, and Christoph Trattner. 2017. Towards health (aware) recommender systems. In *Proceedings of the 2017 international conference on digital health*. ACM, 157–161. (Cited on page 94.)

Robert E Schapire. 2003. The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*. Springer, 149–171. (Cited on page 116.)

L Christian Schaupp and France Bélanger. 2005. A Conjoint Analysis of Online Consumer Satisfaction1. *Journal of electronic commerce research* 6, 2 (2005), 95. (Cited on page 66.)

S Schelter and S Owen. 2012. Collaborative Filtering with Apache Mahout Categories and Subject Descriptors. *Recommender Systems Challenge at ACM RecSys, i* (2012). (Cited on pages 62, 63, 71, and 99.)

Pallavi Sharma and Pankaj Deep Kaur. 2017. Effectiveness of web-based social sensing in health information dissemination—A review. *Telematics and Informatics* 34, 1 (2017), 194–219. (Cited on page 94.)

Kostyantyn M Shchekotykhin, Gerhard Friedrich, Patrick Rodler, and Philipp Fleiss. 2014. Sequential diagnosis of high cardinality faults in knowledge-bases by direct diagnosis generation.. In *ECAI*, Vol. 14. 813–818. (Cited on pages 43, 67, and 73.)

Barbara M Smith. 1996. Succeed-first or fail-first: A case study in variable and value ordering. (1996). (Cited on page 6.)

Stephen F Smith. 1983. Flexible Learning of Problem Solving Heuristics Through Adaptive Search.. In *IJCAI*, Vol. 83. 422–425. (Cited on page 6.)

ILOG Solver. 2003. 6.0 User Manual. *ILOG SA* (2003). (Cited on page 13.)

Roni Tzvi Stern, Meir Kalech, Alexander Feldman, and Gregory M Provan. 2012. Exploring the Duality in Conflict-Directed Model-Based Diagnosis.. In *AAAI*, Vol. 12. 828–834. (Cited on page 66.)

M. Stolpe. 2016. The Internet of Things: Opportunities and Challenges for Distributed Data Analysis. *ACM SIGKDD Exlorations Newsletter* 18 (2016), 15–34. Issue 1. (Cited on pages 76, 91, and 102.)

Peter J Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. 2014. The minizinc challenge 2008–2013. *AI Magazine* 35, 2 (2014), 55–60. (Cited on page 71.)

M. Stumptner. 1997. An Overview of Knowledge-Based Configuration. *AICOM* 10, 2 (1997), 111–125. (Cited on page 28.)

Y. Sun, H. Song, A. Jara, and R. Bie. 2016. Internet of Things and Big Data Analytics for Smart and Connected Communities. *IEEE Access* 4 (2016), 766–773. (Cited on page 102.)

Melanie Swan. 2012. Sensor mania! the internet of things, wearable computing, objective metrics, and the quantified self 2.0. *Journal of Sensor and Actuator Networks* 1, 3 (2012), 217–253. (Cited on page 75.)

Melanie Swan. 2013. The quantified self: Fundamental disruption in big data science and biological discovery. *Big Data* 1, 2 (2013), 85–99. (Cited on pages 92 and 93.)

J. Tiihonen and A. Felfernig. 2010. Towards Recommending Configurable Offerings. *International Journal of Mass Customization* 3, 4 (2010), 389–406. (Cited on pages 2 and 32.)

Edward Tsang. 1993. *Foundations of Constraint Satisfaction*. Academic Press. (Cited on pages 12, 27, 33, and 44.)

André Calero Valdez, Martina Ziefle, Katrien Verbert, Alexander Felfernig, and Andreas Holzinger. 2016. Recommender systems for health informatics: state-of-the-art and future perspectives. In *Machine Learning for Health Informatics*. Springer, 391–414. (Cited on pages 91 and 94.)

S. Valtolina, M. Mesiti, and B. Barricelli. 2014. User-Centered Recommendation Services in Internet of Things Era. In *CoPDA2014 Workshop*. Como, Italy, 1–4. (Cited on pages 3 and 76.)

Gilles Venturini. 1993. SIA: a supervised inductive algorithm with genetic search for learning attributes based concepts. In *European conference on machine learning*. Springer, 280–296. (Cited on pages 46 and 86.)

Katrien Verbert, Hendrik Drachsler, Nikos Manouselis, Martin Wolpers, Riina Vuorikari, and Erik Duval. 2011. Dataset-driven research for improving recommender systems for learning. In *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*. ACM, 44–53. (Cited on page 116.)

N Karthikeyani Visalakshi and K Thangavel. 2009. Impact of normalization in distributed k-means clustering. *international Journal of Soft computing* 4, 4 (2009), 168–172. (Cited on pages 46, 69, and 87.)

R. Walter, A. Felfernig, and W. Küchlin. 2016. Constraint-Based and SAT-Based Diagnosis of Automotive Configuration Problems. *Journal of Intelligent Information Systems (JIIS)* (2016), 1–32. (Cited on page 30.)

Kun Wang, Zhan-shan Li, Yang Ai, and Yong-gang Zhang. 2009. Computing minimal diagnosis with binary decision diagrams algorithm. In *Fuzzy Systems and Knowledge Discovery, 2009. FSKD'09. Sixth International Conference on*, Vol. 1. IEEE, 145–149. (Cited on page 43.)

Roy Want. 2006. An introduction to RFID technology. *IEEE pervasive computing* 5, 1 (2006), 25–33. (Cited on page 93.)

Joseph Wei. 2014. How Wearables Intersect with the Cloud and the Internet of Things: Considerations for the developers of wearables. *IEEE Consumer Electronics Magazine* 3, 3 (2014), 53–56. (Cited on page 91.)

Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. 2006. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*. 1473–1480. (Cited on page 58.)

Darrell Whitley. 1994. A genetic algorithm tutorial. *Statistics and computing* 4, 2 (1994), 65–85. (Cited on page 37.)

Martin Wiesner and Daniel Pfeifer. 2010. Adapting recommender systems to the requirements of personal health record systems. In *Proceedings of the 1st ACM International Health Informatics Symposium*. ACM, 410–414. (Cited on page 94.)

D. Winterfeldt and W. Edwards. 1986. *Decision Analysis and Behavioral Research*. Cambridge University Press. (Cited on pages 20, 31, and 89.)

Franz Wotawa. 2001. A variant of Reiter's hitting-set algorithm. *Inform. Process. Lett.* 79, 1 (2001), 45–51. (Cited on pages 43 and 67.)

Dong Yang, Ming Dong, and Xiao-Kun Chang. 2012. A dynamic constraint satisfaction approach for configuring structural products under mass customization. *Engineering Applications of Artificial Intelligence* 25, 8 (2012), 1723–1737. (Cited on page 33.)

L. Yao, Q. Sheng, A. Ngu, and X. Li. 2016. Things of Interest Recommendation by Leveraging Heterogeneous Relations in the Internet of Things. *ACM Transactions on Internet Technology* 16, 9 (2016), 1–25. (Cited on pages 3 and 93.)

A. Yavari, P. Prakash Jayaraman, and D. Georgakopoulo. 2016. Contextualised Service Delivery in the Internet of Things. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. Reston, VA, USA, 454–459. (Cited on page 77.)

Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. 2017. Effective and efficient dynamic graph coloring. *Proceedings of the VLDB Endowment* 11, 3 (2017), 338–351. (Cited on page 34.)

Mohammed J Zaki. 2001. SPADE: An efficient algorithm for mining frequent sequences. *Machine learning* 42, 1 (2001), 31–60. (Cited on page 83.)

Gary K Zammit, Julie Weiner, Nicola Damato, George P Sillup, and Charlotte A McMillan. 1999. Quality of life in people with insomnia. *Sleep: Journal of Sleep Research & Sleep Medicine* (1999). (Cited on page 92.)

Markus Zanker. 2008. A collaborative constraint-based meta-level recommender. In *Proceedings of the 2008 ACM conference on Recommender systems*. ACM, 139–146. (Cited on pages 4 and 53.)

Markus Zanker, Markus Aschinger, and Markus Jessenitschnig. 2007. Development of a collaborative and constraint-based web configuration system for personalized bundling of products and services. In *International Conference on Web Information Systems Engineering*. Springer, 273–284. (Cited on page 53.)

Markus Zanker, Markus Jessenitschnig, and Wolfgang Schmid. 2010. Preference reasoning with soft constraints in constraint-based recommender systems. *Constraints* 15, 4 (2010), 574–595. (Cited on pages 5 and 54.)