CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Finding optimal alternatives based on efficient comparative preference inference
Authors	Trabelsi, Walid
Publication date	2013
Original Citation	Trabelsi, W. 2013. Finding optimal alternatives based on efficient comparative preference inference. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2013. Walid Trabelsi - http://creativecommons.org/licenses/by-nc-nd/3.0/
Download date	2026-11-07 02:47:22
Item downloaded from	https://hdl.handle.net/10468/1113



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Finding Optimal Alternatives based on Efficient Comparative Preference Inference

Walid Trabelsi



A thesis submitted to the National University of Ireland, Cork
in fulfilment of the requirements for the degree of
Doctor of Philosophy in the Faculty of Science

May 2013

Dr Nic Wilson Supervisor
Dr Derek Bridge Co-Supervisor
Prof Barry O'Sullivan Head of Department

Department of Computer Science,
National University of Ireland, Cork.

Contents

Abstract viii

Acknowledgements x

Dedication xi

Declaration xii

1 Introduction 1

1.1 Background . . . 1

1.1.1 Preference representation . . . 1

1.1.2 Recommender systems . . . 2

1.1.3 Conversational recommender systems . . . 3

1.1.4 Constrained optimisation . . . 4

1.2 Problem Statement . . . 5

1.3 Research Goals and Objectives . . . 6

1.4 Contributions . . . 7

1.5 Publications . . . 9

1.6 Structure of the Thesis . . . 9

2 Literature Review and Related Work 11

2.1 Introduction . . . 11

2.2 Preferences: Description . . . 12

2.2.1 The concept . . . 12

2.2.2 Elicitation . . . 13

2.3 Decision Theory . . . 14

2.3.1 Rationale . . . 14

2.3.2 Need for preferences in decision making . . . 14

2.3.3 Different decision-making approaches . . . 15

2.3.3.1 Normative decision-making . . . 16

2.3.3.2 Descriptive decision-making . . . 16

2.4 Representing Preferences . . . 16
2.4.1 Multi-attribute preferences . . . 17
2.4.2 Graphical models of preferences . . . 17
2.5 Preference Representation in Decision-Making . . . 18
2.5.1 Utility-based assessment . . . 19
2.5.2 Pairwise comparison-based assessment . . . 20
2.6 Comparative Preference Languages . . . 20
2.6.1 Lexicographic preference models . . . 21
2.6.2 Preferences with other features held constant . . . 23
2.6.3 CP-nets: overview . . . 23
2.6.4 Some extensions of CP-nets . . . 26
2.7 Even More Expressive Languages . . . 26
2.8 Preferences-Based Systems . . . 28
2.8.1 Individual decision aiding . . . 28
2.8.2 Collective decision aiding . . . 29
2.8.3 Preferences-based database requests . . . 30
2.9 Recommender Systems . . . 30
2.9.1 The task . . . 31
2.9.2 Human computer interaction . . . 32
2.9.3 Different techniques of recommendation . . . 33
2.9.4 Case-based recommender systems . . . 35
2.9.4.1 Single-shot recommender systems . . . 36
2.9.4.2 Conversational recommender systems . . . 36
2.10 Preference Handling Methods in Conversational Recommender Systems . . . 38
2.10.1 Critiquing . . . 38
2.10.2 Some approaches for conversational recommender systems . . . 39
2.10.2.1 FindMe . . . 39
2.10.2.2 More Like This & Partial More Like This . . . 40
2.10.2.3 Information Recommendation . . . 40
2.10.3 Some applications of conversational recommender systems . . . 41
2.10.3.1 Travel planning advisors . . . 41
2.10.3.2 Restaurant advisors . . . 44
2.10.3.3 Music advisor . . . 45
2.10.3.4 Other applications . . . 45
2.11 Constraint Satisfaction Problem . . . 46
2.11.1 Definitions . . . 46
2.11.2 CSP solution methods . . . 48
2.12 Constrained Optimisation . . . 50

2.12.1 Constrained optimisation problems: different solution methods 51
2.12.1.1 Complete search . . . 51
2.12.1.1.1 Exact methods . . . 51
2.12.1.1.2 Search heuristics . . . 52
2.12.1.2 Incomplete search . . . 52
2.12.2 Constrained optimisation: coupled and decoupled approaches 53
2.12.3 Branch and bound . . . 53
2.12.4 Preference-based complete search . . . 54
2.12.5 Conditional preferences-based constrained optimisation . . . 55
2.12.5.1 CP-net-based constrained optimisation . . . 55
2.12.5.2 Hard and optimality constraints-based constrained optimisation . . . 57
2.12.5.3 Constrained CP-net-based constrained optimisation 57
2.12.5.4 Constrained FCP-net-based constrained optimisation 58
2.12.5.5 CP-net-based formulation for constrained optimisation . . . 59
2.12.5.6 Polynomial constrained optimisation for partial acyclic CP-net . . . 59
2.12.5.7 Comparative preference theories-based constrained optimisation . . . 60
2.13 Conclusion . . . 60

3 Dominance for Comparative Preferences 62
3.1 Introduction . . . 62
3.2 Preference Relations . . . 62
3.2.1 Preference relations properties . . . 63
3.2.2 Preference relations application . . . 64
3.3 A Comparative Preference Language . . . 65
3.3.1 Conditional preference theories-like statements . . . 65
3.3.2 CP-nets-like statements . . . 66
3.3.3 TCP-nets-like statements . . . 66
3.4 Total Pre-orders-Based Dominance: Formal Semantics . . . 66
3.4.1 Total pre-orders-based semantics . . . 67
3.4.2 CP-tree-based semantics . . . 68
3.4.2.1 Description of a cp-tree . . . 68
3.4.2.2 CP-tree: variable and value orderings . . . 69
3.4.2.3 Comparing two outcomes . . . 70
3.4.2.4 Generation of a compatible ordering of outcomes . 71
3.5 CP-Tree-Based Preference Computation . . . 72

3.6 Other Preferential Semantics . . . 76
3.7 Summary . . . 79

4 Preferences Deduction for Conversational Recommender Systems 80
4.1 Introduction . . . 80
4.2 The Case Study: Information Recommendation . . . 82
4.2.1 The advisor . . . 83
4.2.2 The queries . . . 86
4.2.3 The dialogue . . . 86
4.2.4 The user . . . 89
4.3 A Framework for Preference Dominance . . . 90
4.3.1 Ultimate goal: inference . . . 90
4.3.2 Logical settings . . . 91
4.3.3 Application . . . 93
4.4 Sum of weights-Model Approach . . . 94
4.4.1 Models . . . 94
4.4.2 Constraint language . . . 94
4.4.3 Dominance relation . . . 95
4.4.4 Dominance computation . . . 95
4.5 CP-tree Model Approach . . . 97
4.5.1 Models . . . 97
4.5.2 Constraint language . . . 99
4.5.3 Dominance relation . . . 100
4.5.4 Dominance computation . . . 100
4.6 Induction of Constraints on Preferences Within the Framework . . . 101
4.6.1 Inducing constraints in the sum of weights model . . . 101
4.6.2 Inducing constraints in the cp-tree model . . . 103
4.7 Experimentation and Comparative Study . . . 105
4.7.1 Settings . . . 106
4.7.1.1 Offline experiments . . . 106
4.7.1.2 Products . . . 106
4.7.1.3 The initial query . . . 107
4.7.1.4 User Modeling . . . 107
4.7.1.5 System runs . . . 109
4.7.1.6 The pruning . . . 109
4.7.2 Comparative study . . . 110
4.7.2.1 Representing true preferences in the sum of weights model . . . 111
4.7.2.2 Representing true preferences in the cp-tree model 114

4.8 Generalization for Non-Boolean Features . . . 115
4.9 Other Kinds of Models of Preferences . . . 119
4.9.1 Larger sets of models . . . 119
4.9.2 Towards a stronger pruning . . . 120
4.9.3 Lexicographic inference . . . 121
4.9.4 Application to group recommender systems . . . 122
4.9.5 Application to other forms of conversations in recommender systems . . . 124
4.10 Conclusions . . . 125

5 Constrained Optimisation for Comparative Preferences 126
5.1 Introduction . . . 126
5.2 Personalized Branch And Bound . . . 128
5.2.1 Model of the search tree . . . 128
5.2.2 Preference relation for optimisation . . . 129
5.2.3 Preference-based branch and bound . . . 129
5.3 Dominance Pruning Rules . . . 130
5.3.1 The root-dominates rule . . . 131
5.3.2 The deciding-node dominance rule . . . 133
5.3.3 Projection-dominance condition . . . 135
5.4 p(Γ): A sufficient Condition for Dominance based on Unsound Dominance Relation . . . 136
5.5 Non-Dominance Pruning Rules . . . 137
5.6 Example: Computer Configuration Problem . . . 138
5.6.1 Computer configuration problem . . . 139
5.6.2 Example . . . 139
5.7 Implementation Issues . . . 142
5.7.1 The search tree . . . 144
5.7.2 Relevant undominated solutions . . . 146
5.7.3 The *Reduce* procedure . . . 147
5.7.4 The dominanceTest procedure . . . 148
5.7.5 The CPOptimizer procedure . . . 148
5.8 Experimental Testing . . . 149
5.8.1 Experimental setup . . . 149
5.8.2 Discussion of results . . . 154
5.8.3 Synthesis of discussion . . . 158
5.9 Conclusion . . . 159

6 Conclusion **160**

6.1 Summary 160

6.2 Future directions 161

Glossary **165**

Abstract

Choosing the right or the best option is often a demanding and challenging task for the user (e.g., a customer in an online retailer) when there are many available alternatives. In fact, the user rarely knows which offering will provide the highest value.

To reduce the complexity of the choice process, automated recommender systems generate personalized recommendations. These recommendations take into account the preferences collected from the user in an explicit (e.g., letting users express their opinion about items) or implicit (e.g., studying some behavioral features) way. Such systems are widespread; research indicates that they increase the customers' satisfaction and lead to higher sales. Preference handling is one of the core issues in the design of every recommender system. This kind of system often aims at guiding users in a personalized way to interesting or useful options in a large space of possible options. Therefore, it is important for them to catch and model the user's preferences as accurately as possible.

In this thesis, we develop a comparative preference-based user model to represent the user's preferences in conversational recommender systems. This type of user model allows the recommender system to capture several preference nuances from the user's feedback. We show that, when applied to conversational recommender systems, the comparative preference-based model is able to guide the user towards the best option while the system is interacting with her. We empirically test and validate the suitability and the practical computational aspects of the comparative preference-based user model and the related preference relations by comparing them to a sum of weights-based user model and the related preference relations.

Product configuration, scheduling a meeting and the construction of autonomous agents are among several artificial intelligence tasks that involve a process of constrained optimisation, that is, optimisation of behavior or options subject to given constraints with regards to a set of preferences. When solving a constrained optimisation problem, pruning techniques, such as the branch and bound technique, point at directing the search towards the best assignments, thus allowing the bounding functions to prune more branches in the search tree. Several constrained optimisation problems may exhibit dominance relations. These dominance relations can be particularly useful in constrained optimisation problems as they can instigate new ways (rules) of pruning non optimal solutions. Such pruning methods can achieve dramatic reductions in the search space while looking for optimal solutions.

A number of constrained optimisation problems can model the user's preferences using the comparative preferences. In this thesis, we develop a set of pruning rules used in the branch and bound technique to efficiently solve this kind of optimisation problem. More specifically, we show how to generate newly defined pruning rules from a dominance algorithm that refers to a set of comparative preferences. These rules include pruning approaches (and combinations of them) which can drastically prune the search space. They mainly reduce the number of (expensive) pairwise comparisons performed during the search while guiding constrained optimisation algorithms to find optimal solutions. Our experimental results show that the pruning rules that we have developed and their different combinations have varying impact on the performance of the branch and bound technique.

Acknowledgements

The time for carrying out this research has been hugely enjoyable, valuable and inspiring since the outset. This research would have not been possible without the tremendous amount of unconditional support that was received from various sources. I would like to seize this opportunity to express my sincere gratitude and would like to thank some of them in particular.

I would like to express my deep appreciation and gratitude to my supervisor Nic Wilson for his intellectual guidance, constructive critiques and continuous encouragement which ensure the successful completion of this thesis. I am also extremely grateful to my second supervisor Derek Bridge who contributed to my work with his continuous collaboration and valuable suggestions. Their patience and kindness, as well as their academic experience, have been invaluable to me.

My sincere thanks must also go to the exam committee: Professor Barry O'Sullivan, as my internal examiner, and Professor Francesca Rossi, as my external examiner. Their constructive criticism helped me develop a broader perspective to my thesis.

I would like to show my gratitude to the Science Foundation Ireland which made this work possible by funding it under Grant No. 08/PI/I1912.

I greatly appreciate the company of all current and previous members of the Cork Constraint Computation Centre (4C). Also, the laboratory administrative staff (Eleanor O'Riordan, Catriona Walsh, Linda O'Sullivan and Olivia Frawley) and technical staff (Peter MacHale and Joe Scanlon) for their professional guidance throughout the complex administrative procedures. They made the lab a friendly environment for working. Special regards go to Barry O'Sullivan whose attitude to people and research has been inspiring for me. His advices and insight were invaluable to me.

I owe my parents and friends a debt of gratitude for their permanent moral support and prayers. Undoubtedly this work would have not been completed without their unconditional help.

I would like to express my appreciation to my brother Brahim Douar and my nephew Ali Douar for their constant support in so many ways. Without question, their moral support and availability were crucial at any stage of this research.

Dedication

To my parents, Abdelkader Trabelsi and Hadhria Ziadi, my mum Chadlia Miladi, and my sisters, Yosra and Emna.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Signed:
Walid Trabelsi

List of Tables

2.1 Laptop components. 25
2.2 CPT for TCP-net *N*. 27
2.3 Movie's features. 28

3.1 Laptop components. 72

4.1 Two databases of hotels 107
4.2 The pruning rates (true preferences represented in sum of weights model) 111
4.3 The average number of steps per dialogue (true preferences represented in weights vector model) 112
4.4 The same final query rate (true preferences represented in weights vector model) 112
4.5 The average shortfalls (true preferences represented in weights vector model) 113
4.6 The computation time (true preferences represented in weights vector model) in millisecond (ms) 113
4.7 The pruning rates (true preferences represented in cp-tree model) . . 114
4.8 The computation time (true preferences represented in cp-tree model) in millisecond (ms) 115
4.9 Recapitulative table of results (users as weights vectors) 118
4.10 Recapitulative table of results with lexicographic models (users as weights vectors) 119
4.11 The pruning rates (true preferences represented in cp-tree model) . . 122
4.12 The average normalized shortfalls (users as cp-trees) 123
4.13 The execution time (users as cp-trees) 123

5.1 Computer components. 140

5.2 Mean number of optimal solutions for each preference family, and CPU time (ms), number of visited nodes and number of dominance checks at *end nodes* for each preference family and each method. The CSPs averaged around 500 solutions. 155
5.3 Mean number of optimal solutions for each preference family, and running times (ms) for each family and each method. 155
5.4 CPU time (Time), mean number of optimal solutions (#sol), number of visited nodes (#nd), ratio, standard deviation from the average time (SDTime) and frac-sol for **CP-nets family** and each method. The CSPs averaged around 500 solutions. 155
5.5 Mean number of optimal solutions for each preference family, and CPU time (ms), the ratio (Time/#sol), the frac-sol (#sol/#sol for Basic) and SDTime (standard deviation from the average time) for **Rand-W family** and each method. The CSPs averaged around 500 solutions. . 156
5.6 Mean number of optimal solutions for each preference family, and CPU time (ms), the ratio (Time/#sol), the frac-sol (#sol/#sol for Basic) and SDTime (standard deviation from the average time) for **Lex family** and each method. The CSPs averaged around 500 solutions. . 156
5.7 Mean number of optimal solutions for each preference family, and CPU time (ms), the ratio (Time/#sol), the frac-sol (#sol/#sol for Basic) and SDTime (standard deviation from the average time) for **CPn-to family** and each method. The CSPs averaged around 500 solutions. . 156

List of Figures

2.1 A CP-net for laptop configuration. 25
2.2 The induced preference graph for laptop configuration. 25
2.3 Classification of recommender systems. 33
2.4 Search tree example 50

3.1 A cp-tree σ, along with its associated ordering $\succcurlyeq_\sigma$ on outcomes, with $\gamma = 1$ (i.e., with at most one variable associated with a node) 69
3.2 A cp-tree σ, along with its associated ordering $\succcurlyeq_\sigma$ on laptop outcomes, with $\gamma = 1$ (i.e., with a single variable associated with a node) 72
3.3 A cp-tree σ, along with its associated ordering $\succcurlyeq_\sigma$ on laptop outcomes, with $\gamma = 2$ (i.e., with at most two variables associated with a node) 73

4.1 A single-shot recommendation scenario 82
4.2 A conversational recommendation scenario 83
4.3 Interaction model of the user with the recommender. 90
4.4 A cp-tree σ with $\gamma = 1$, along with its associated ordering $\succcurlyeq_\sigma$ on configurations 98
4.5 A cp-tree σ with $\gamma = 2$, along with its associated ordering $\succcurlyeq_\sigma$ on configurations 99
4.6 A preference dominance framework 105

5.1 Search trees using basic approach 143
5.2 Search trees using deciding-node-dominance rule 143
5.3 Example of a search tree 146
5.4 Model of transferring the set of undominated solutions between nodes in the search tree 148
5.5 Running time(ms) for each family of preferences for $n = 10, 15, \ldots, 40$ variables (having 3 values each). Each CSP has approximately 1000 solutions. 157

1 Introduction

The goal of this dissertation is to develop a comparative preferences approach for eliciting and reasoning with preferences in Recommender Systems (RSs) and constraint optimisation-based systems.

1.1 Background

1.1.1 Preference representation

Choosing the colour of a car or choosing among different mortgage loans are examples in which preferences can direct humans from simple to very important decisions. Preferences are a multi-disciplinary topic that has been extensively studied in economics, psychology, philosophy, logics and other human-centered disciplines. Nevertheless, it is a relatively new topic in Artificial Intelligence (AI) (Kaci 2011, Domshlak et al. 2011) and has become of great interest for the development of reasoning mechanisms in intelligent systems (Brafman & Domshlak 2009, Chen & Pu 2004).

The preference relation, which is used to specify the preferred alternatives, is often defined over a very large set, so we cannot afford to explicitly represent it, but needs a compact implicit representation. Such approaches have been developed in the AI literature. The above concern has fostered the study of compact preference

representation languages (Benferhat et al. 2001, Brewka et al. 2003, Brewka, Benferhat & Berre 2004, Boutilier et al. 2004*a*, Brewka, Niemelä & Syrjänen 2004, Kärger et al. 2008). Generally speaking, these specifications define a preference model in a compact way and a preference relation to rank the outcomes of the model.

The way in which preferences are encoded and the ranking is obtained mainly depends on the specifications used. Preference representation languages can be categorized into:

- Approaches which extend classical logic such as Qualitative Choice Logic (QCL) (Brewka, Benferhat & Berre 2004) and Possibilistic Logic (Benferhat et al. 2001);
- Approaches based on Conditional Preference Networks (CP-nets) (Boutilier et al. 2004*a*) and CP-nets extensions such as TCP-nets (Brafman, Domshlak & Shimony 2006), CP-theories (Wilson 2004*b*), Conditional Importance Networks (CI-nets) (Bouveret et al. 2009*a*), and comparative preference theories (Wilson 2009*b*);
- Approaches based on logic programming such as logic programs with ordered disjunction (*LPOD*) (Brewka 2002, Brewka, Niemelä & Syrjänen 2002), Answer Set Optimisation (*ASO*) programs (Brewka et al. 2003), Logic Programs with Ordered and Unordered Disjunction (DLPODs) (Kärger et al. 2008), CRProlog with Ordered Disjunction (Balduccini & Mellarkod 2003), and Resourced Answer Set Programming (*ASP*) (Costantini & Formisano 2009).

1.1.2 Recommender systems

People increasingly face the difficult task of having to select the best option from a large set of multi-attribute alternatives. An online electronic catalog system might provide access to large numbers of items, such as apartments to rent, notebook computers to buy, or financial products in which to invest. The user has to navigate through the catalog to find the most suitable one. RSs are tools that help people find their most desired items based on a model of their preferences.

Research and development of RSs has been an exciting and vibrant field for over a decade, having produced proven methods for preference-aware computing. The purpose of a RS is to help users identify interesting, personalized items or content from a large search space. For example, recommenders have successfully helped users find books and media of interest from a massive inventory base, e.g., (*Amazon* 2012), and personalized movie suggestions, e.g., (*netflix* 2012, *Movielens* 2012). A RS can be considered as an information agent that provides suggestions for items which are likely to be of interest to the user (McGinty & Reilly 2011).

These systems infer user preferences from data gathered either explicitly, e.g., in the form of product ratings, or implicitly by observing user behaviour. These preferences help them make predictions for products not yet assessed (e.g., rated) by the user. The prediction algorithms used within RSs include collaborative filtering, content-based filtering and case-based reasoning, and various hybrid approaches that combine these (Adomavicius & Tuzhilin 2005, Anand & Mobasher 2005, Bridge et al. 2006). However, these classical approaches have typically supported only a simple "single-shot" form of human-computer interaction, where a user who has previously supplied a set of ratings, identifies herself to the system and is then given a set of product recommendations.

1.1.3 Conversational recommender systems

Regardless the quality of RSs, they are unlikely to be sufficiently prescient that their first set of recommendations always satisfies the user. Indeed, users are *rarely* satisfied with the first set of recommendations (Bridge et al. 2006); they usually want to see more options and they exploit the initial recommendations to refine their preferences and articulate new requests. In fact, they are usually not very familiar with the available products and their characteristics. Thus, their preferences are not well established, but constructed while learning about the available products (Payne et al. 1993).

Conversational RSs allow for this, and recognise that their users may be willing and able to reveal more of their constraints and preferences, over a short dialogue, thereby moving away from "single-shot" interaction. This is also an opportunity for the RS to guide the user by asking questions, giving advice, displaying candidate products, and giving explanations (Reilly et al. 2004, McSherry 2005, Bridge et al. 2006, Ricci et al. 2006, Schmitt 2002*a*, Thompson et al. 2004*b*, Pu et al. 2006).

Conversational RSs typically involve iteratively showing the user a small set of options (e.g., products) for them to choose between. To select an appropriate set to display at each stage, from a much larger collection of options, the recommender needs information regarding which options are likely to be preferred to others by the user, based on previous responses the user has given in the dialogue; if one assumes that the user has some kind of preference relation over products, this amounts to determining if certain products are dominated (less preferred than other products) according to this preference relation.

Pruning less optimal items, with regards to the user preferences, is a key task for RSs. This was confirmed and validated by experiences from fielded applications with conversational RS described by (Felfernig et al. 2006, Felfernig & Gula 2006) based on user questionnaires after using systems. The results showed that interactive

recommenders help users to better steer themselves when being confronted with large sets of choices.

Furthermore, solutions are characterized by a set of attributes in classical decision theory (Keeney & Raiffa 1976). As utility theory provides a solid mathematical foundation for recommendations (Keeney & Raiffa 1976), users might express their preferences by the relative weights they give to attributes or combinations of attributes. Such approaches are described for example in (Keeney 1992). An example of its application in a decision aid system is the Personal Computer (PC) selection tool of IBM described in (Schiex 1992), where users can adjust the weights of different criteria and interactively see how different PC models rank according to these weights.

However, there are many practical situations where the space of possible criteria is so large that stating a numerical preference model for all of them would be cognitively impossible (Faltings, Torrens & Pu 2004). In fact, such situations assume the supply of a large amount of information from the user and it implies complex preference models that cannot be obtained in e-commerce scenarios because people are not willing to go through lengthy preference elicitation processes (Brafman & Domshlak 2008). The user may wish to state simple comparisons. She may want to make no explicit quantification of preference or utility, leaving the preference purely qualitative. This could be the case for example in a travel problem, where there is a large number of possible attributes involving times, means of transportation, locations, that vary from one user to another. In such a case, the user may want to say that she likes to travel to a country during the summer in that country whatever the country is, all other attributes being equal. The user will then avoid having to communicate an accurate numerical model. It has also been claimed that the qualitative specification of preferences (e.g., pairwise comparisons between tuples) is more general than the quantitative one, as not all preference relations can be expressed by scoring functions (Stefanidis et al. 2011).

1.1.4 Constrained optimisation

Several tasks, like configuring products for an online shopper or scheduling meetings for a busy executive officer, require balancing the user's desires with hard constraints. Intelligent automated agents that perform such tasks are driven by several established optimisation algorithms whose aim is to significantly improve their ability to traverse through the constrained space. Constrained optimisation is searching for optimal solutions, which are solutions that are feasible regarding hard constraints and that best meet the user's preferences. Computational approaches for solving constrained optimisation problems consist of mainly two fundamental principles:

exploring a large solution space toward a desired solution while trying to eliminate sub-parts of the solution space which are guaranteed not to have a better solution (Kadioglu 2012). This guarantees that constrained optimisation algorithms perform some pruning that will aid the search for optimal solutions, without missing any potentially optimal solution.

Branch and Bound (B & B) is a common technique for solving constrained optimization problems (COPs) using backtracking search. The B & B technique consists of splitting the original problem recursively into subproblems which become sooner or later easy to solve. It allows one to reduce the feasible region to several parts by exploiting properties of the problem (e.g., user's preferences). One of the main advantages of pruning away subspaces from the search space is the reduction of the number of pairwise comparisons performed during the search. Optimality in several constrained optimisation approaches can refer to a preference relation. Thus, progress made on preference approaches may contribute in the development and expansion of the constrained optimisation approaches.

1.2 Problem Statement

Preferences are a crucial notion guiding our choices and actions. Intelligent systems require a concise and processable representation of preference information in order to enable them to act autonomously and to intelligently support users. In several decision making problems (e.g., in planning or in scheduling), there is a concern about which alternative(s) to choose. Alternatives involve multiple attributes and so defined in terms of a number of variables. These alternatives can be represented by a set of combinations that are depicted explicitly (e.g., as database of outcomes), or implicitly (e.g., a mathematical programming model which describes a set of outcomes). Choosing the best alternative(s) in such settings implies the need for a preference relation that is used to compare two solutions based on preferences expressed over the attributes of these solutions.

With regards to RSs, providing personalized recommendations to users requires the modeling of their preferences and needs. Eliciting and representing users' preferences is a crucial task for such automated systems to successfully accomplish their task. It is usually too complex to get a complete and accurate model of the users' preferences, especially regarding the tradeoffs between different attributes (Faltings, Torrens & Pu 2004).

Conversational RSs belong to a category of RSs that provides challenges for more elaborate formalisms that represent the user's preferences while conversing with her. Conversational RSs would be more reliable and efficient if provided with adapted and suitable preference formalisms in order to model and reason with the user's

preferences. Therefore, it would be interesting to look for supporting this process by integrating recently developed preference formalisms, e.g., (Wilson 2004*b*, Wilson 2009*b*). These formalisms would enable conversational RSs to manage a variety of types of user feedback without losing an efficient pruning capability.

Besides, researchers are investigating clever and new mechanisms to be integrated with constrained optimisation algorithms to efficiently produce optimal solutions. A number of COPs use comparative preferences to express the user preferences. One way to find solutions for these COPs is to extend the B & B technique with the related preference relation. Then, we will be able to solve COPs in which preferences are expressed as comparative preferences. The integration of new methods which are based on general preference formalisms and enriched with a computationally efficient dominance operation can offer flexibility to the B & B technique and expand the set of COPs that can be tackled by B & B.

1.3 Research Goals and Objectives

One goal of this thesis is to integrate pairwise comparisons-based preference models that represent the users' preferences into a conversational RS. The preference model would express the user's preferences as accurately as possible and needs to offer key comparison operations (e.g., dominance) that give more efficiency and flexibility to the pruning engine of the RS. We need to effectively evaluate the comparative preferences-based preference models and the dominance for comparative preferences and its practical computational aspects. One way to do this is to experimentally compare them to another preference model supported by a dominance operation which is based, for instance, on utility-based preferences (e.g., importance weights).

A second goal of this thesis is to enrich the B & B technique by the integration of new comparative preference-based pruning rules that can be used to ease the search for optimal solutions for COPs where preferences are expressed as comparative preferences. Thus, this will widen the range of problems the B & B technique can deal with and solve.

Our objectives in this thesis can be summarised as follows:

- To integrate such a preference model in an existing conversational Recommender System (RS).
- To accomplish relevant computations with the preference model in a practical way.

- To perform, within offline experiments, an effective evaluation of two preference dominance approaches: weights vector-based approach and comparative preferences-based approach where the (simulated) user is modeled as a weights vector (Balabanovic 1998, Shen 2007, Kim et al. 2011, Shen et al. 2005) or as a cp-tree (Wilson 2009*b*). Experiments with different user models allow for examining whether the matching or the mismatching of the way the simulated users' preferences are represented and the model of the users' preferences induced from the system can have an influence on the performance of the pruning method.
- To define new pruning rules associated with the B & B technique for constraint optimisation for comparative preferences.
- To perform an evaluation of the pruning methods with a number of binary constraint satisfaction problem (CSP) instances by comparing their pruning power and efficiency regarding other measures (e.g., time).

1.4 Contributions

The contributions being claimed by this dissertation are twofold:

- A novel use of a formalism for preference elicitation in conversational RSs. This formalism is based on comparative preferences.

  In fact, we consider a framework of preference dominance and its rationale. This framework is illustrated through two instances. One fundamental step in the process of recommending, which is preferences induction, is then presented in the two instances. We have developed the second instance that we have implemented, experimentally tested and compared it with the first instance. We report the experimentations and a comparative study of the two instances.

  The first instance is based on a simple quantitative preference formalism, involving a weights vector-based user model, with an associated language of linear inequalities. This is a very commonly used model for preference representation, specifically, in multi-attribute utility theory (MAUT) (Figueira et al. 2005). The second instance of the framework is a qualitative preference formalism, where models adopt a kind of generalised lexicographic order (i.e., cp-trees), and constraints are expressed as comparative preference statements in a language generalising CP-nets (Boutilier et al. 2004*a*).

  To the best of our knowledge, this is the first time that the type of comparative preferences that we use in the second instance has been deployed with RSs.

We show that comparative preference theories are able to give freedom and flexibility to the system to handle the user's preferences by allowing the system to capture preference nuances and various forms of preferences without giving up the attractive computational properties of the preference dominance relation.

We also prove that the comparative preferences-based dominance algorithm that we select and use to compare outcomes works efficiently for a range of comparative preference statements when checking dominance between two outcomes α and β. We verify and validate the suitability and attractiveness of the comparative preference theories-based approach for RSs through several experiments that include repeated scenarios with simulated users represented by models that are based on different semantics (e.g., weights vector vs cp-tree).

- New comparative preferences-based pruning rules that extend the B & B technique for comparative preferences.

  We have developed a first group of newly defined sufficient conditions that help the constrained optimisation algorithm locate the extra spurious data and remove it from the solution space. During search, we use the previously found solutions to bound the search space up to the point that either there is an optimal solution that dominates the subspace or there is no such solution. One of the main advantages of pruning away subspaces from the search space is the reduction of the number of pairwise comparisons performed during the search.

  Another way of eliminating useless comparisons is to avoid involving any optimal solution that is unable to better any extension (complete assignment) in a sub-space. In other words, an optimal solution α, which was already found, might be unable to dominate any outcome that extends the partial assignment obtained so far in the search tree. If we prove, through a pruning method, that this is the case then we do not need to involve α in any dominance check below the current node. Thus, we have also developed a second category of pruning rules which temporarily disengage any subset of optimal solutions below some node of the search tree from playing a role in the comparisons performed in the sub-space newly created. This pruning rule checks whether any optimal solution already found is unable to dominate any other possible assignment in the considered sub-space below the current node. If the condition is checked to be true for one solution α, then α is no longer involved in eventual comparisons below the current node in the search tree.

In order to assess the efficiency of the developed pruning rules in practice, we performed experiments and discuss the results that we found.

1.5 Publications

The work presented in this thesis has been developed via a collaboration with Nic Wilson, Derek Bridge and Francesco Ricci. Part of it has been published in the proceedings of international conferences, and a peer reviewed journal as we describe below.

- Pruning Rules for Constrained Optimisation for Conditional Preferences, Nic Wilson and Walid Trabelsi, in Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP-2011).
- Preference Dominance Reasoning for Conversational Recommender Systems: a Comparison between a Comparative Preferences and a Sum of Weights Approach, Walid Trabelsi, Nic Wilson, Derek Bridge and Francesco Ricci, in the International Journal of Artificial Intelligence Tools (IJAIT) volume 20 number 4.
- Comparing Approaches to Preference Dominance for Conversational Recommenders, Walid Trabelsi, Nic Wilson, Derek Bridge and Francesco Ricci, in Proceedings of the 22nd International Conference on Tools with Artificial Intelligence (ICTAI-2010).
- Preference Dominance Approaches for Conversational Recommender Systems, Walid Trabelsi, Nic Wilson, Derek Bridge and Francesco Ricci, in Proceedings of the 5th Multidisciplinary Workshop on Advances in Preference Handling (MPREF-2010).

1.6 Structure of the Thesis

The thesis is organized as follows:

- Chapter 2:

  Conversational RSs are found at the intersection of RSs research, dialogue system and preference handling. In this chapter, we present the background material and literature review related to conversational RSs and preference handling, which are the two primary research domains of our thesis work.

- Chapter 3:

  Dominance testing represents a key operation when looking for the most preferred set of alternatives among a set of possible alternatives with regards to the user's preferences. In this chapter, we present the preference language, and the dominance semantics and computation we are using in this thesis. The chapter also presents other dominance approaches that adopt different semantics of the dominance relation. In this chapter, we also detail the algorithms and data structures that we use in the key operations in the dominance testing.

- Chapter 4:

  In this chapter, we mainly define a novel use of a formalism for preference elicitation based on comparative preferences and integrate it into a conversational RS. We carefully assess two approaches based on a sum of weights-based dominance relation and comparative preference theories-based dominance relation through several experiments. We conduct these experiments with simulated users and performed a comparative study between these two approaches.

- Chapter 5:

  In this chapter, we propose new methods for finding non-dominated solutions of a CSP with respect to a set of comparative preferences. We derive these pruning rules from a dominance algorithm that refers to a set of comparative preferences theories, that represent the user's desires, in an attempt to guide constrained optimisation algorithm to find optimal solutions. To make the presentation concrete, we apply the new methods to random CSPs of different sizes.

2

Literature Review and Related Work

2.1 Introduction

Preferences handling in conversational RSs and constrained optimisation are two among the main research streams that are studied in this dissertation. In this chapter, we present the background knowledge and the literature review related to the domains we are concerned with in this thesis.

The topic of preferences has recently attracted considerable attention in AI research and plays an increasingly important role in several AI-related fields. Several AI applications, including expert systems, autonomous agents, decision support systems (DSSs), RSs, configuration software, and constrained optimisation applications, rely on the ability to make decisions regarding the user's preferences (D'Ambrosio & Birmingham 1995, Junker 2001, Chajewska et al. 2000, Mura & Shoham 1999, Nguyen & Haddawy 1999). In Section 2.2 we present the concept of preferences. We will talk about decision making in Section 2.3. Section 2.4 presents examples of formalisms devoted to represent preference relations (Domshlak et al. 2011).

In Section 2.5, we present two categories of preference representation in decision-making. A group of languages, which is used to express and reason with preferences in these applications, is presented in Section 2.6. Section 2.7 recalls some expressive preference languages. A large spectrum of AI applications need preferences to be expressed and reasoned with to make decisions. These applications are presented in Section 2.8. Section 2.9 is devoted to giving a description of a group of RSs. The

concept of CSP and the corresponding solution methods are presented in Section 2.11. Section 2.12 presents COPs and different solution methods for these problems.

2.2 Preferences: Description

A crucial concept in modeling user preferences is that of preference relation. This approach is related to the concept of similarity/dissimilarity between values/items (Stahl 2002) and is extensively used in the RSs field (McSherry 2002, Stahl 2006). In this section, we will talk about the concept of preference relations. We will also present a key challenge in preference handling: preference elicitation.

2.2.1 The concept

Preferences are modeled to guide the choices made by a decision-making entity. In AI, an artificial agent can be a decision-making entity. The agent generally acts on behalf of another physical or moral entity, a user for instance. The degrees of desirability, inherent to preferences, are represented, most commonly, quantitatively by means of utility functions for instance, or qualitatively by means of pairwise comparisons. Given a set of options, a rational agent chooses the one that maximizes its expected usefulness, that is the one that most probably leads to the outcome it prefers the most, according to the preferences of the user or organization it acts for.

Preferences over some set of possible options order them so that a more attractive option precedes a less desirable one. Examples of sets of possible options could be possible flights, vacation packages or cameras.

Thus, the notion of preference aims at bringing a natural and effective way of coming to the most appropriate solutions among a large number of choices. In e-commerce, the user preferences are used to evaluate the alternatives so that the most preferred products are most likely to be selected by a customer (Stolze 2000). McSherry and Aha suggested a model for a RSs where four types of preferences are identified (McSherry & Aha 2007).

In their words, *Assumed* preferences illustrate the fact that consumer preferences for particular attributes can be assumed to be consistent among all decision makers based on the characteristics of an attribute, also referred to as *Less-is-Better* (McSherry 2003) or *cost type* attribute (Xu 2007). For example, preference for Price attribute is typically to minimize the value of the price.

Explicit preferences are directly stated by a decision maker. Typically explicitly stated preferences need to be taken into account since the consumer is not willing to accept any compromise on the value. Then, any generated outcome has to comply with all the preferences.

Implicit preferences are indirectly discovered from the available information about the consumer. There are RSs which are based on example critiquing which is one interaction model that allows users to build their preferences by examining or reviewing (we also call it critiquing) examples shown to her by the system (Burke et al. 1996, Burke et al. 1997*a*). Each time the user reviews a product, the system induces some preferences. This technique is explained in Section 2.10.1 of Chapter 2. For example, during a dialog in a critiquing-based system, when a user indicates her critique (i.e., review) on the presented values, the attributes' values she is not critiquing can be assumed to be suitable. Thus, these values will be regarded as the user's implicit preferences.

Predicted preferences appear when a history of previous decisions, or preference discovery dialogs are available, allowing some preferences may be predicted with reasonable accuracy. For example, in progressive critiquing RSs (Bridge et al. 2005), the preferred value of an attribute can be predicted as the nearest available value that satisfies the recent critique on the value of an attribute in the presented item (Ricci, Mirzadeh & Venturini 2002).

2.2.2 Elicitation

One key challenge in preference handling is usually the elicitation of the preference information from users. Decision-makers have a limited capacity for information processing (Bettman et al. 1998). Decision-makers tend to build their preferences when they are prompted to express evaluative judgment or to make a decision (Payne et al. 1992). When alternatives are described with a number of decision attributes, decision makers typically do not have specific predefined strategies of the selection of attribute importance and tradeoffs they are ready to make (Häubl & Murray 2001).

Preference elicitation methods aim at easing the cognitive burden of ordering a set of outcomes, or finding an optimal item. Preference elicitation considers questions such as how best to query the user about her preference model (e.g., utility values) and how much information is needed. These methods range from approaches based on the utility function (Savage 1954, Fishburn 1967, C. 1968, Fishburn 1970, Fishburn 1974, Fishburn 1999), which conveys the user's preferences on a particular option, to computer-aided elicitation approaches (e.g., critiquing, collaborative filtering, etc). Work on the latter approaches indicates that the process of preference elicitation needs to be sophisticated as it involves, in addition to conventional assumptions about the user's preferences, a number of relevant elements related to human-computer interaction (Faltings, Pu, Torrens & Viappiani 2004, Pu & Faltings 2004).

2.3 Decision Theory

2.3.1 Rationale

Decision making is a daily activity. We all make decisions constantly, from the simplest "which shoes should I wear today?" or "should I take my umbrella?" (Poole 1992) to the more complex "how should we deal with the deforestation?" (P. Journee & Vanderpooten 1998). These decisions are made at all levels such as organisational ("how do we schedule the crew shifts?" (A. Caprara & Fischetti 1998), inter-organisational ("which trace (or route) for the highway?") (Ostanello & Tsoukiàs 1993). Quite often, we cannot make decisions by ourselves. Indeed, during decision processes, we normally ask for help, advice or support from friends, experts or consulting companies for instance.

Decision problems are often complex (Alexander 1977). They can involve large volumes of information that have to be managed, by means of filtering or mining for instance, to come out with a critical decision that might have serious consequences. Inappropriate decisions can have heavy consequences (e.g., air traffic and aerospace accidents). The need for decision support is felt when decisions start to be more difficult or critical. This was one of the main objectives the decision support techniques were developed for.

In order to help the decision maker, DSSs need to embrace a reasoning model that is similar to the way people would normally behave. This makes the proposed actions natural as much as possible. In recent decades, there has been a growing interest in "behavioural decision theory" (von Winterfeldt & Edwards 1996) which has contributed to what rational decision-making requires (Tsoukiàs 2008).

The current scientific study of decision making originated from the form of operational research during the World War II era as a response to the need for optimal solutions to complex military planning and decision making problems (Tsoukiàs 2008). The practical application of decision theory aims at finding tools, methodologies and software to help people make better decisions. Systematic and comprehensive software tools developed in this way are called decision support systems (DSSs). The representation of the decision maker's preferences allows DSSs to operate effectively on her behalf. This preference representation is then a key element in the success of such applications.

2.3.2 Need for preferences in decision making

Stefanidis et al. stated that preferences guide human decision making from early childhood (e.g., "which ice cream flavor do you prefer?") up to complex professional and organizational decisions (e.g., "which investment funds to choose?") (Stefanidis

et al. 2011). Furthermore, Visser et al. claimed that there are no intelligent DSSs without knowing the preferences of the user (Visser et al. 2009). We can then infer that the representation of preferences is a central topic in decision making.

Therefore, extensive research has been conducted to study the fundamental structures on which decision aiding models rely: the structure of preference relations (Dushnik & Miller 1941, Luce 1956, Scott & Suppes. 1958*a*), which are described in Section 3.2 in Chapter 3, and the functions that represent them (Fishburn 1970, Koopman 1956). A survey of preference models was carried out in (Öztürk et al. 2004). This survey presented the representations that were conceived to express our preferences for the purpose of automated decision making. The survey considered a broad spectrum of representations ranging from qualitative to quantitative; and simple to complex. Representations employed in existing decision making tools are considered in addition to those describing preferences in a form suitable for manipulation by an automated process.

Several preference representations, including *ceteris paribus*-based preference languages such as Conditional Preference Networks (CP-nets) and their extensions, were described in (Boutilier et al. 2004*a*, Brafman & Domshlak 2002, Bouveret et al. 2009*b*). Other examples of preference representations are the "prototypical" preference logic presented in (Bienvenu et al. 2010), the temporal preference representation described in (Bienvenu et al. 2011), and prioritized goals, which have been used by (Sardiña & Shapiro 2003) and (Benferhat et al. 1993) in decision making, and preference formulae described by (Delgrande et al. 2007) as a logical combination of queries.

2.3.3 Different decision-making approaches

The way people can and do make decisions varies considerably. For decades, research focused on the way the user makes decisions in practice and the way she should theoretically behave when a decision situation is presented to her. Then, an array of decision making models, depending on their methodological foundations, have emerged.

Tsoukiàs presented and discussed different decision aiding approaches that were introduced during the last six decades of existence of the decision theory including the normative and descriptive approaches (Tsoukiàs 2008). The author explained the differences among these approaches by examining the origin of their specific "models of rationality" (Savage 1954) which is a key concept in decision aiding.

The idea of rational choice is a central idea: people tend to compare the costs and benefits of certain actions, and choices will be made by decision makers as they try to maximize their benefits and minimize their costs. Tsoukiàs (Tsoukiàs 2008)

considered a rational model as a tool that enables the interpretation of informal information and its conversion to a formal representation which will be used as an input in the decision making process. Thus, choices and preference information are expressed in the rational model, and the system then comes up with a set of optimal choices.

2.3.3.1 Normative decision-making

Normative methods are based on the belief that there is an ideal standard of rationality that should guide our choices. They are called normative because they use norms. A prominent approach within this class of decision making is the expected utility theory (Neumann & Morgenstern 1944). According to this theory, the behaviour of a rational decision maker adheres to a set of axioms that govern the nature of its preferences (Savage 1954). A rational decision maker in this setting is one that makes decisions which maximise its expected utility, that is, the sum of the utilities of possible decision outcomes multiplied by the probability of their occurrence.

2.3.3.2 Descriptive decision-making

Enforcing a model of ideal rationality, which is a theory of how we should make decisions, is often at odds with real human decision-making. A contrasting branch of research is the study of descriptive decision-making (Bell et al. 1988). Descriptive approaches analyze human decision-making behaviour with the goal of understanding or replicating it. Kahneman and Tversky described the Prospect theory as a model of decision-making amongst risky alternatives, which are choices, the outcome of which occurs with a given probability (Kahneman & Tversky 1979). Through experiment, the two authors found that a real decision maker often violates the principles of expected utility theory.

2.4 Representing Preferences

Effective representations for preferences play a key role in the success of many AI applications (Boutilier et al. 2001). A targeted preference representation is one which captures intuitive statements, that can be for instance easily evaluated by users, and enables a compact representation of the description of the problem, without having to enumerate a prohibitive number of alternatives. There are several formalisms devoted to represent preference relations (Domshlak et al. 2011). In the remainder of the section, we present examples such as utility-based formalisms (a function giving scores to the alternatives) and ordinal preference-based formalisms (a binary relation over the pairs of alternatives).

2.4.1 Multi-attribute preferences

Clearly, it is difficult for humans to compare outcomes over multiple dimensions. It is also complicated for machines to store and analyze preferences over a number of outcomes that is exponential in the number of attributes. It is, therefore, important for multi-attribute decision theory to study, through mathematical methods for instance, the decomposition of utility (preference) functions into combinations of subfunctions that correspond to different attributes or sets of attributes.

Multi-attribute decision theory adopted some assumptions about the user's preferences structures (e.g., additive independence (Keeney & Raiffa 1976)). This theory presented theorems that allow inferences about several forms of independence of attributes from the subfunctions that make up an overall preference theorem (see, for example, (Hansson 1989, von Stengel 1988, Wellman & Doyle 1992)). Structural properties of preferences help automated systems reduce the burden of preference elicitation (Engel 2008).

2.4.2 Graphical models of preferences

One key technique in achieving a considerable simplification of the task of elicitation is the use of graphical models that usually take advantage of the independence for preference (utility) functions in a multi-attribute space (Bacchus & Grove 1995, Gonzales et al. 2008). They also speed up computing and reasoning (Lang 2010). Besides and more generally, graphical languages are a powerful way of expressing preferences in combinatorial domains (Lang 2010).

For combinatorial domains, we need languages that allow a compact preference representation. A significant number of these languages have been studied in the AI research community. They consist of a graphical component describing preferential dependencies between variables, together with a collection of local preferences on single variables or small subsets of variables, compatible with the dependence structure.

Part of the motivation (and also inspiration) for graphical modeling of multi-attribute preferences is driven by the success of Bayesian Networks (BNs) (Pearl 1988) in achieving compact representation and graphical reasoning algorithms for probabilities, based on conditional independence concepts (Pearl & Paz 1986, Pearl 1988). Therefore, researchers explored the conceptual similarities between probability and preferences, and examined the opportunity to transfer ideas from one field to another.

The extension of influence diagrams by (Howard & Matheson 2005) in order to decompose value functions into sums and products of multiple value nodes was probably one of the first tendencies to exploit separable preferences in a graphical model.

Bacchus and Grove (Bacchus & Grove 1995) had developed a graphical model based on the conditional independence structure (Engel & Wellman 2008). They proposed an undirected graph that captures conditional additive utility independencies. The presented models are called generalized additive independence (GAI) models, and they enable efficient dominance testing, i.e., for determining whether a possible outcome has higher utility than another (Engel & Wellman 2008). They introduced GAI-networks, which are similar to the junction graphs in BNs. Dominance relations and semantics for comparative preferences are described in Chapter 3.

(Boutilier et al. 2004*a*) initiated another stream of work by introducing CP-nets that use a particularly simple graphical model which captures users' qualitative conditional preferences over objects. CP-nets are similar to BNs from a syntactical point of view. But, they differ with respect to their semantics. CP-nets received a considerable amount of study in subsequent literature (Kristensen et al. 2005, Purrington & Durfee 2007, Domshlak et al. 2006, Brafman & Dimopoulos 2004*a*), and various graphical models of preferences were developed. Examples are TCP-nets (Brafman & Domshlak 2002), FCP-nets (Gavanelli & Pini 2008*a*), CI-nets (Bouveret et al. 2009*b*) and UCP-nets (Boutilier et al. 2001). Each graphical model of these tried to extend the applicability of graphical models. For instance, UCP-nets (Boutilier et al. 2001) found a way to take the advantages of both GAI-models and CP-nets by combining qualitative and quantitative preferences.

2.5 Preference Representation in Decision-Making

Preferences aim at offering the user the ability to express her relative or absolute satisfaction when faced with a choice between different options (Chevaleyre et al. 2006). Preferences are modeled in such a way that it is possible to derive a final recommendation for the decision maker. In the vast landscape of literature on preference handling, two distinguishing labels are used to classify representations; Delgrande, Schaub and Tompits distinguished two forms of preference representation: absolute (describing the desirability of a particular state-of-affairs) and comparative (describing the desirability of one state-of-affairs relative to others) (Delgrande et al. 2007). Similar distinctions were made throughout the literature on preference - often referring to absolute representations as monadic and comparatives as dyadic (Hansson 2001).

Similarly, according to (Tsoukiàs 2008), multiple criteria decision aiding methods could be grouped in two categories, based on how the set of the potential alternatives is explored:

- the establishment of a utility function synthesizing the different criteria;

- the use of pairwise comparison procedures and majority principles for establishing a final recommendation.

2.5.1 Utility-based assessment

The model of preference representation within economics and early decision theory has been the (absolute) expected utility function (Neumann & Morgenstern 1944). When applied to a choice, this function returns a numeric evaluation of its desirability or interest. The choice is then given a score. A choice *A* is preferred over another choice *B* if and only if its score is higher than the score of *B*. The maximisation of this function determines which is the best choice according to the standard prescription of rationality as recommended by (Savage 1954).

As traditional decision-theoretic approaches were growing, a variety of techniques for the elicitation of utility functions from clients or users emerged (Keeney & Raiffa 1976), (Saaty 1980), (Keeney 1982), (Farquhar 1984) and (Wakker & Deneffe 1996). Several authors (Farquhar 1984, Wakker & Deneffe 1996, Öztürk et al. 2004) argued that a utility function can be used to obtain an intelligible preference relation that can be axiomatically justified. Tsoukias claimed that the construction of a utility function is restrictive, since it needs a number of conditions to be fulfilled (Tsoukiàs 2008). Domshlak et al. argued that a utility function requires a considerable cognitive effort on the part of the decision makers as they would need to determine a value function for a large number of alternatives described by multiple attributes (Domshlak et al. 2011). Thus, they are not often willing to express their preferences directly in terms of a value function (Domshlak et al. 2011). To express liking one option exactly twice as much as an alternative option for instance is not that intuitive and natural (Visser et al. 2009).

(Brafman & Domshlak 2008) described the difficulties inherent in the elicitation of a utility function and explained the burden of this process, as it requires the supply of a large amount of information from the user. These issues were discussed in Chapter 6 of (Bouyssou et al. 2000) and Chapters 4, 5 and 6 of (Bouyssou et al. 2006). These discussions concluded that the associated cognitive burden of ordering a set of outcomes or finding the best outcome makes such approaches not entirely convenient for supporting complex scenarios where the set of possible decisions tends to be either too large to be described explicitly or the information is incomplete (Öztürk et al. 2004).

2.5.2 Pairwise comparison-based assessment

A decision maker may wish to state simple comparisons. She may want to make no explicit quantification of preference or utility, leaving the preference purely qualitative. She may want to say, that other things being equal, she likes to stay in a hotel with a swimming pool rather than in a hotel without a swimming pool. Wellman and Doyle assert that humans place value on compact representations of information "that directly entail the most important conclusions" (Wellman & Doyle 1994). Superlatives (such as "fastest", "cheapest", and "largest") and comparatives (such as "faster", "tastier", and "happier") are recognised as being common ways in which humans describe their preferences (Domshlak 2008). They represent examples of how human try to express their knowledge compactly. Comparisons of objects along some dimension suppose a linear scale which is used to measure to which extent humans exhibit their preferences. For instance, a superlative statement expressed for one object allows it to hold maximal degree in the assumed scale (Wellman & Doyle 1994).

To have the ability of expressing preferences and in response to the limitations of utility functions and variants claimed in the previous section, a wide range of qualitative forms of preference expression have been developed. Methods for qualitative decision-making are discussed in depth in (Roy 1991), (Vincke 1992) and (Roy 1996). A category of these qualitative methods are based on the Ceteris Paribus principle (Wellman & Doyle 1991, Wellman & Doyle 1994, Boutilier et al. 2004*a*) which is described in Section 2.6.2. Among these methods, we cite CP-nets (Boutilier et al. 2004*a*) described in Section 2.6.3 and their various extensions such as Tradeoff-Conditional Preference Networks (TCP-nets) (Brafman & Domshlak 2002) which are described in Section 2.6.4, FCP-nets (Gavanelli & Pini 2008*a*) and CI-nets (Bouveret et al. 2009*b*).

2.6 Comparative Preference Languages

Many decision problems are defined over large multi-attribute domains. This raises a key challenge in preference research which is to develop preference representation languages that support user-friendly elicitation techniques and have reasoning capabilities. These capabilities are needed to cope with the exponential size of the outcome space which often has a combinatorial structure. In fact, an appropriate preference formalism would include a preference representation that captures statements that are natural for users to assess, and a reasoning engine that supports effective inference. The selection of a language for representing preferences requires one to look at a number of characteristics, among others: expressive power which

should be high, relative compactness, complexity which should be low, elicitation-friendliness, and cognitive appropriateness (Chevaleyre et al. 2008).

In order to make decisions, the decision maker may need to determine the preference relation: a ranking of all outcomes determined by her own preferences. A simple and unsophisticated idea would consist in revealing explicitly the user's preferences: simply by stating the possible options tagged by their respective utilities (in the case of cardinal preferences) or listing all different pairwise preferences among the possible options (in the case of ordinal preferences). This approach is too simplistic and not realistic when the set of alternatives has a combinatorial structure which would make the process potentially infeasible (Rossi et al. 2011). It can be problematic for machines to store and analyze preferences over a number of outcomes that is exponential in the number of attributes (Engel & Wellman 2010).

Because of this, the automated systems adopted some assumptions about the user's preference structures to optimize the process by having less complicated and more manageable preferences elicitation methods. Such assumptions include conditional preferential independence (Boutilier et al. 2004*a*). Several compact preference languages emerged and were studied in the AI research community. We should note that comprehending user preferences and finding the way to a suitable representation of them in order to perform meaningful inferences are real challenges (Stefanidis et al. 2011). One way to express preferences is to consider preferences over some subset of attributes, considering the rest of the attributes held fixed. Such a statement is also often referred to as a ceteris paribus preference statement (described in Section 2.6.2). A second approach is to give an importance ranking to the attributes and consider first (or give priority to) the most important attributes when comparing outcomes. This ranking adopts a so called Lexicographic model (described in Section 2.6.1).

Work on several real-world decision problems has taken advantage from the progress made in preference representation (Vig et al. 2011, Brush et al. 2010, Konstan & Riedl 2012, Chen & Pu 2010, Mahmood & Ricci 2009, Pu et al. 2008, Jannach et al. 2011). One example is the recommenders who have successfully helped users find their targets (e.g., books (*Amazon* 2012), news (Hurley & Tewksbury 2012), movies (*netflix* 2012), Movielens (*Movielens* 2012)). We will present RSs in more details in Section 2.9.

2.6.1 Lexicographic preference models

A well-known kind of preference model is the lexicographic preference ordering (see e.g., (Fishburn 1974, Schiex et al. 1995, Brewka 2004*b*, Freuder et al. 2010)). The

idea of a lexicographic ordering is often used in qualitative approaches for multi-criteria decision making. Here, preferences over outcomes are based on a set of relevant variables, which are ranked according to their importance. The importance ranking of variables is defined by a total order which disposes the set of variables in importance levels. A local value ordering associated with variables in each node depends on the values taken by the parents of that node.

The lexicographic preference ordering first considers the highest importance level. If some outcome has better value for the variable on that level than another, then the first is preferred over the second. If two outcomes have the same value for the variable on this level, the next importance level is considered, and so on. Two outcomes are equally preferred if and only if they have the same value for the variable on every level.

The definition implies that if one item is preferred over another on the most important variable on which they differ, it is considered better overall, regardless of its values for the remaining less important variables. For instance, if we have four variables (or attributes) F_1, F_2, F_3 and F_4 ordered as F_i is more important than F_{i+1} ($i = \{1, 2, 3\}$), and α has better value than β for F_1, whereas β is has better value than α for the remaining variables (i.e., F_2, F_3 and F_4) then α is still considered better than β. Two outcomes α and β are compared in linear time when preferences have the form of lexicographic models.

For example, a student who wants to go by plane to Paris prefers a cheap flight to an expensive one; for equally priced flights, the student will check whether the flight is direct or not as she prefers the direct one. The student's preferences could then be represented by a lexicographic preference model as the price will be the most important criterion followed by the itinerary.

Lexicographic preference models are regarded as simple and reasonably intuitive preference representations, and so lexicographic ordering can be well-understood by humans that use it to make preference decisions (Yaman et al. 2011). The psychology literature shows evidence that lexicographic preferences are often an accurate model for human decisions (Gigerenzer & Goldstein 1996).

These models implement a stronger form of preference statements than ceteris paribus statements. They demand stronger judgments. They represent a situation where the value of variable *X* is much more important than the values of any less important variable; we prefer any outcome that does better on variable *X*. This may limit the usefulness of lexicographic models (Wallace & Wilson 2009). One disadvantage is that lexicographic orderings models are unable to handle tradeoffs as the choice of values of a variable dominate the assignments to a set of other less important variables. However, they have attractive computational properties: two alternatives can be efficiently compared which is not the case for some other preference

models (e.g., CP-nets), and a single best solution can always be specified (Wallace & Wilson 2009).

2.6.2 Preferences with other features held constant

One class of preference statements, called ceteris paribus statements, allows preferences that apply preferences over features' values with keeping other features equal. Such a preference might be "other things being equal, I prefer chocolate to strawberry ice cream". These preferences capture the intuitive idea that other undisclosed qualities might affect the decision making process. If, for example, chocolate ice cream turns out to be much more expensive than strawberry ice cream, and hence everything else is not equal, then this preference no longer applies. In (Doyle et al. 1991), the authors propose reasoning systems based on the ceteris paribus preference semantics. Their system allows preferences such as: pet(cat) > pet(dog), meaning other things being equal, I prefer to have a cat rather than a dog.

2.6.3 CP-nets: overview

The CP-nets model captures complex user preferences in a graphical representation. The latter exploits conditional preferential independence in order to structure the decision maker's preferences, over a set of features V, under a ceteris paribus assumption. They were introduced in (Boutilier et al. 1999) and extensively studied in many subsequent papers, in particular (Boutilier et al. 2004*a*, Boutilier et al. 2004*b*). CP-nets provide a compact and natural representation of ordinal preferences in multi-attribute domains (Boutilier et al. 1999, Domshlak et al. 2001, Brafman & Domshlak 2002, Domshlak & Brafman 2002, Boutilier et al. 2004*a*, Goldsmith et al. 2008, Brafman, Domshlak & Shimony 2006, Brafman & Domshlak 2008).

A CP-net *N* over V is an annotated directed graph G over $\{X_1, ..., X_n\}$, in which nodes are the problem variables. Each node X_k is annotated with a conditional preference table denoted by *CPT*(X_k), which associates a total order over the values domain of X denoted by $\underline{X_k}$ with each instantiation $u \in \underline{U}$ ($\underline{U}$ is the set of possible assignments to the set of variable U) of X_k's parents (Boutilier et al. 2004*a*). Indeed, given such a structural information, the agent explicitly specifies her preferences over the values of X_k for each assignment $u \in \underline{U}$. This preference is assumed to take the form of total order over $\underline{X_k}$. An outcome o is a map that assigns a value to each variable $X_k \in V$. For instance, let $V = \{X_1, X_2, X_3\}$, all three variables being binary, and assume that the preference of a given agent over all possible outcomes can be defined by a CP-net whose structural part is the directed acyclic graph $G = \{(X_1, X_2), (X_1, X_3), (X_2, X_3)\}$; this means that the agent's preferences over

the values of X_1 is unconditional and the preference over the values of X_2 (respectively X_3) is fully determined given the values of X_1 (respectively the values of X_1 and X_2). The preference statements contained in the conditional preference tables are written with notation, such as, $x_1 x_2' : x_3' > x_3$ which means that when $X_1 = x_1$ and $X_2 = x_2'$ then $X_3 = x_3'$ is preferred to $X_3 = x_3$. An improving flip is the change of the value of a single variable X_k within an outcome to directly compute a preferred (better) outcome based on CPT (X_k). In other words, an improving flip is a change in the value of a variable towards a more preferred value according to the conditional preference table for that variable.

A CP-net *N* induces a preference ranking over the outcome space (Boutilier et al. 2004*a*): for any pair of outcomes o_1 and o_2, *N* entails $o_1 \succ o_2$ (denoted by $N \models o_1 \succ o_2$) if and only if there exists a sequence of improving flips from o_2 to o_1 respecting the conditional preference tables of *N*; otherwise, $N \not\models o_1 \succ o_2$. This definition induces a pre-order over the outcomes, which is a partial order if the CP-nets is acyclic (see below). Note that the preference relation induced from a CP-net is generally not complete (Boutilier et al. 2004*a*): two outcomes o_1 and o_2 may also be incomparable according to *N* (written as $N \models o_1 \bowtie o_2$). Therefore, a partial order among the outcomes of a CP-net is obtained by means of the CPT associated to each node in the graph. $N \models o_1 \bowtie o_2$ if and only if $N \not\models o_1 \succ o_2$ and $N \not\models o_2 \succ o_1$.

A CP-net is said to be acyclic if its directed graph is acyclic, and tree-structured if its directed graph is a forest, that is, a disjoint union of trees (Domshlak & Brafman 2002). It should be noted that a tree-structured CP-net is an acyclic preference network where arcs are directed from the root down to the leaves in such a way that each node has at most one parent.

When the CP-net is traversed in a topological order of variables, we obtain an induced preference graph of outcomes whose nodes determine the various possible combinations and every edge in the graph represents an improving flip. Although the induced graph specifies the worst and best preferences, respectively, it does not necessarily show any ranking between several intermediate solutions (Boutilier et al. 2004*a*).

Comparing two outcomes is NP-hard for both cyclic and acyclic CP-net. Given an acyclic CP-net, finding an optimal solution can be done in linear time. However, for cyclic CP-nets, it becomes NP-hard (Goldsmith et al. 2008).

Example 1. *Let us assume a student is looking for a suitable laptop configuration. The student will focus only on two components: The Operating System(OS) and The Memory(M) (unit=megabyte). The possible values of variables are defined in Table 2.1. If the memory is equal to* 1024*MB, UBUNTU is preferred over XP because UBUNTU can be personalized and optimized in order to operate more conveniently even with a small*

amount of memory which is not the case for XP, which it needs at least $2048MB$ *of memory. When the laptop has* $2048MB$ *of memory, the student prefers XP over UBUNTU. The CP-net which illustrates the preferences of the student is depicted in Figure 2.1. The induced preference graph is depicted in Figure 2.2. An arc in this graph directed from outcome* β *to* α *indicates that a preference for* α *over* β *can be determined directly from one of the CPTs in the CP-net. For example, the fact that the outcome* $\{1024, UB\}$ *is preferred to* $\{1024, XP\}$ *(as indicated by the direct arc between them) is a direct consequence of the semantics of the CPT drawn next to the OS node in Figure 2.1 which says if the memory is equal to* $1024MB$*, UBUNTU is preferred over XP. The top element* $\{1024, XP\}$ *in Figure 2.1 is the worst outcome while the bottom element* $\{2048, XP\}$ *in the figure is the best.*

Table 2.1: Laptop components.

Variables	Values
Operating System (OS)	{*XP, UBUNTU11.4 (UB)*}
Memory (M)	$\{1024, 2048\}$

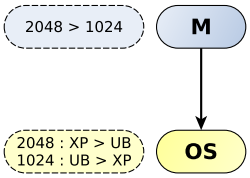


Figure 2.1: A CP-net for laptop configuration.

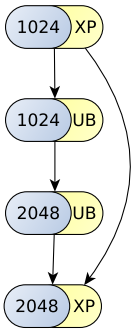


Figure 2.2: The induced preference graph for laptop configuration.

2.6.4 Some extensions of CP-nets

Although CP-nets are a representation language which is well-suited to expressing preferential (in)dependencies, they do not allow, for instance, to express relative important statements between variables.

Therefore, various extensions of CP-nets were proposed for the purpose of being more expressive without losing the advantages of CP-nets. For instance, TCP-nets (Brafman & Domshlak 2002, Brafman, Domshlak & Shimony 2006) are CP-nets with additional importance statements between variables. TCP-nets implement, additionally to CP-nets, variable importance statements, as lexicographic orders do. They generalize CP-nets by introducing the ability of expressing a relative importance and conditional relative importance of object attributes. Thus, TCP-nets are a more refined tool for comparing objects than CP-nets.

Languages for cardinal preference representation in the style of CP-nets have been defined as well, for instance UCP-nets (Boutilier et al. 2001), which are based on generalised additive independence (GAI). UCP-nets capture quantitative preferences and relative importance information using utility functions. They combine the theory of CP-nets and GAI-nets (generalized additive decomposable utility functions). Other extensions of CP-nets were developed as well such as FCP-nets (Gavanelli & Pini 2008*a*) which are described in Section 2.12.5.4.

2.7 Even More Expressive Languages

The user's preferences often have different forms of statements (e.g., comparisons between arbitrary partial or complete assignments). The user's preference statements might be acyclic and the induced preference relation might be inconsistent. These facts motivate the development of new preference languages that can handle more general forms of preference inputs which, for instance, include cyclic preference statements. The approach presented in (Wilson 2004*b*) introduces a new form of preference statements named *conditional preferences theories* and denoted *cp-theories*. It uses a logical framework for expressing conditional preference statements that consists of a formalism along the same lines of CP-nets but with a richer language allowing to express not only the usual CP-nets ceteris paribus statements but also TCP-nets statements. Indeed, this approach allows for tradeoffs between somewhat more expressive preferences.

The conditional preferences theories involve statements of the form $u : x > x'[W]$ which means that given value u for variable U, we prefer value x to x' for variable X, as long as variables outside of W are held equal. The values of W are allowed to be arbitrarily different. Sections 3.1 and 3.2 in (Wilson 2011) states that

CP-nets and TCP-nets can be represented by the means of statements by having $W = \emptyset$, and $0 \leq |W| \leq 1$ respectively.

Example 2. *Let N be a TCP-net on a set of variables $V = \{V_1, V_2, V_3, V_4\}$. A conditional preference table (CPT) is defined in Table 2.2.*

Table 2.2: CPT for TCP-net *N*.

$x_1 > x_1'$		
x_1	$x_2 > x_2'$	$x_3' > x_3$
x_1'	$x_2' > x_2$	$x_3 > x_3'$
x_3	$x_4 > x_4'$	
x_3'	$x_4' > x_4$	

Variable importance states X_2 is more important than X_4 (i.e., $X_2 \longrightarrow X_4$). One conditional importance statement is $X_2 \longrightarrow_{x_1} X_3$ which says that given a tuple x_1, X_2 is more important than X_3.

Let Γ_N be a cp-theory that can represent preference information brought by N. the CPT of N, described in Table 2.2, can be represented as follows: $x_1 > x_1'[\emptyset]$, $x_1 : x_1 > x_1'[\emptyset]$, $x_3' > x_3[\emptyset]$ and $x_3 : x_4 > x_4'[\emptyset]$. The variable importance $X_2 \longrightarrow X_4$ can be represented by the statements $x_1 : x_2 > x_2'[X_4]$ and $x_1' : x_2' > x_2[X_4]$. The conditional importance statement $X_2 \longrightarrow_{x_1} X_3$ can be represented by the statement $x_1 : x_2 > x_2'[X_3]$.

Example 3. *Let us consider a movie database scenario. In this example, the features of a movie are represented by set of variables V. V ={Director (D), Genre (G), Scenario (S), Principal Actor(A)}. Values of the features are presented in Table 2.3.*

A cp-theory allows one to inform the preferences on the values of an attribute depending on the values of some other attributes. Thus, in a movie database scenario, a user can specify her preferences as follows:

$d_1 : g_1 > g_2[A]$ and $d_2 : g_3 > g_4[A]$;

The first statement reveals the user prefers comedy to suspense for movies whose director is Woody Allen regardless of the name of the principal actor. The second statement says she prefers action films to dramas for movies whose director is Steven Spielberg regardless of the name of the principal actor.

Furthermore, Wilson has defined a novel preference language where preference statements compare not only single values of variables but also tuples of values of a set of variables (Wilson 2009*b*). A set of such preference statements is called a *comparative preferences theory* and is described in Section 3.3 in Chapter 3.

Table 2.3: Movie's features.

Variables	Values
Director (D)	{*Allen* (d_1), *Spielberg* (d_2)}
Genre (G)	{Comedy (g_1), Suspense (g_2), Action (g_3), Drama (g_4)}
Scenario (S)	{Fiction (s_1), Real Story (s_2))}
Principal Actor(A)	{Robin Williams (a_1), Will Smith (a_2), Morgan Freeman) (a_3)}

2.8 Preferences-Based Systems

Many real-world problems require people to select the most preferred item, which is called in some contexts the target solution, from a set of options, which might have a combinatorial structure. For example, an electronic catalog system might provide access to millions of products, and the user has to navigate through the catalog to find the most preferred one (e.g., (*netflix* 2012)).

The problem of building personalized preference-based information systems has been investigated in various settings such as querying databases (Stefanidis et al. 2011), getting automated product recommendations, autonomous agents (Rossi et al. 2004), planning (Domshlak et al. 2001, Bienvenu et al. 2006) or receiving support in group decision making (Rossi et al. 2004, Lang & Xia 2009, Xia et al. 2008, Xia et al. 2007).

2.8.1 Individual decision aiding

As stated in several works (Doyle & Thomason 1999, Boutilier et al. 1999, Boutilier et al. 2004*a*), preference applications were first targeting the decision maker in order to help her make the right choice from a relatively large number of alternatives. In (Domshlak et al. 2001), the authors presented a framework for preference-based configuration of web page content. An optimal configuration of web page content is determined according to the preferences of the web author as well as to the current interests of the viewer.

GP-CSP (Do & Kambhampati 2001) is a system that does planning by automatically converting a Graphplan planning graph (Blum & Furst 1995) into a CSP encoding which is solved by standard CSP solvers. In (Brafman & Chernyavsky 2005), the authors showed how to extend GP-CSP in order to find optimal plans for planning problems with preferences expressed as TCP-nets (Brafman & Domshlak 2002, Brafman, Domshlak & Shimony 2006). In that paper, preferences among goal states are specified using a TCP-net, and they look for a plan resulting in an optimal outcome, that is, a state s_1 such that no other reachable state betters s_1. Another example is

presented in (Giunchiglia & Maratea 2011); the authors investigated the relative performance of preference-based planners, including qualitative preference-based planners, in terms of solving problems.

Another example using CP-nets for individual decision making was presented in (Boubekeur et al. 2006), in which an information retrieval approach was detailed in which CP-nets are used for expressing preferences over documents.

Several tasks in decision support require the selection of an optimal subset of items. The work in (Binshtok et al. 2007) considered the problem of computing optimal subsets given a preference specification based on the methodology presented in (Brafman, Domshlak, Shimony & Silver 2006) which proposed a generic way to specify preferences over sets of objects. Indeed, in (Brafman, Domshlak, Shimony & Silver 2006) the set preferences were specified through set property values which are based on TCP-nets. The preferences for each set are based on the attribute values of individual elements in the set.

2.8.2 Collective decision aiding

Most RSs target single individuals. In recent years, however, systems which provide recommendations for groups have emerged, based on the idea that some types of recommended items are at least as likely to be used by groups as by individuals, for example vacations, movies, restaurants or cultural events like concerts or exhibitions.

On the other hand, there are situations in which we cannot make decisions by ourselves and we need to set trade-offs and discuss our preferences with those of others in order to collaborate properly. An example of this phenomenon is encountered when voting in multiple referenda, where we have to decide on a set of propositions to accept, or electing a committee, where we have to decide how to fill each seat.

The work in (Rossi et al. 2004) built on top of CP-nets, which represented the preferences of each agent, a formalism to model and handle the qualitative and conditional preferences of multiple agents. Li et al. generated the set of all Pareto-optimal outcomes from a collection of agents whose preferences were represented by CP-nets (Li et al. 2010). CP-nets were used in voting theory to represent the voter's preferences in (Lang & Xia 2009, Xia et al. 2008, Xia et al. 2007). Xia et al. proposed a general methodology for voting, as a group decision, that allowed the aggregation of preferences when voters express CP-nets that can be cyclic (Xia et al. 2008). Lang et al. proposed a new approach which elicits the voter's preferences for each variable at a time by aggregating the voter's preferences locally (Lang & Xia 2009). Endriss claimed that the most widely used compact preference representation language in

computational social choice are CP-nets (Endriss 2011). CP-nets were used to represent and analyze games (Bonzon et al. 2009). The work in (Apt et al. 2005) related the two formalisms of strategic games and CP-nets and showed reasoning techniques with CP-nets can be mutually used to reason with strategic games.

The authors in (Mukhtar et al. 2011) proposed an approach for dynamic user task composition and assignment in which preferences are specified qualitatively for various components' capabilities. Those preferences are modeled using CP-nets. Such applications can be deployed in a pervasive work environment so that the user will be able to use efficiently the hardware and software resources in the environment to dynamically achieve her task for better ergonomics, entertainment, or special needs.

2.8.3 Preferences-based database requests

Comparative preferences have also been the subject of several research fields including databases in which they capture soft criteria for queries (Stefanidis et al. 2011). Databases brought a novel perspective to the preference-based applications because of the huge amount of information faced by users on a daily basis. Customizing database queries via user preferences is a research topic that raised a lot of interest in the database community in recent years. Such preferences were used for sorting and selecting the best tuples, that is, records in the database, those which most fulfill the user's wishes (Chomicki 2002). (Endres & Kiessling 2006) examined the translation of TCP-nets into preference statements written in the preference framework introduced in (Kießling 2002), an approach based on an algebraic formalism through which a preference is obtained by composing simpler preferences, and whose preference models are based on strict partial orders. This work introduced a new preference constructor to all the resulting algebra expressions to capture the ceteris paribus semantics after the transformation of TCP-nets.

Another example of adaptation of CP-nets to the context of databases is presented in (Mindolin & Chomicki 2007) which extends CP-nets to a structure called hierarchical CP-nets in which each edge of the network expresses both the conditional dependence and the relative importance between different attributes among which parents are more important than all their descendants in the preference graph. Another work (Ciaccia 2007) dealt with a similar case where the user's preferences are not completely specified.

2.9 Recommender Systems

The amount of data accessible on the web expands rapidly and goes beyond limited human cognitive capabilities. Moreover, in several purchase-intention scenarios the

customer is faced with a set of alternative items or services, but she is not ready to make the right choice straightaway due, for instance, to a lack of appropriate knowledge to take the decision.

The set of available alternatives is stored in large electronic product catalogs containing organized information about products and their features. Most e-commerce web sites, such as Amazon (*Amazon* 2012), Expedia (*expedia* 2012), or eBay (*ebay* 2012), use this kind of catalogue. Such large catalogs used by online retailers point the increasing need for a decision agent that takes the customer's needs and preferences as inputs and returns a set of recommended items.

With the growth of the Internet and the business to consumer e-Commerce, the need for intelligent systems increases in electronic commerce field. These automated systems, called *recommender systems* (RSs), have the mission of delivering personalized information services that match the user needs. RSs have become an important stream of research in electronic commerce (Salton & McGill 1983, Jannach et al. 2011).

Since the middle 1990s, several research streams emphasized RSs (Hammond et al. 1994, Burke et al. 1996, Resnick & Varian 1997, Burke et al. 1997*a*, Sarwar et al. 2001, Schafer et al. 2001, Lorenzi & Ricci 2003, Burke 2004, Bridge et al. 2005, Ricci & Nguyen 2005). They are now gaining widespread acceptance in the e-commerce field.

2.9.1 The task

RSs combine ideas from a number of research domains including information retrieval (IR), user modeling, machine learning, and human computer interaction (HCI) (Adomavicius & Tuzhilin 2005). They share the common goal of helping the user carry out tasks, from discovering a product to constructing a plan by eliciting the user preferences, inferring a preference model, and using the model to make a decision about when and how to act. These tasks are interrelated and they strive for a main goal: make the user content and most importantly allow her to get (e.g., buy) what she wants.

Thus, RSs will typically supply the user with a list of recommended items she might like, or predict to which degree the user might prefer each item (e.g., predict ratings of items by a given user). These systems help users decide on appropriate items, and lessen the task of finding preferred items in the large collection of available products, services, and other opportunities.

They assist users by identifying interesting products and services that better fit their preferences in situations where the number and complexity of offers sharply exceeds the user's capability to survey them and reach a decision.

A popular example is Netflix (*netflix* 2012), the DVD rental provider, which gives personalized movie recommendations to each user, using past movie ratings of the user and her circle of friends (e.g., it displays predicted ratings for every movie that is shown to the user). The most successful online book retailer Amazon (*Amazon* 2012) provides average user ratings for displayed books, and a list of other books that are bought by users who buy a particular related book.

E-commerce web sites use RSs, wishing to increase their online retailers' revenues, and looking to the best and most efficient way of guiding the user through their collections and catalogues and encouraging her to buy their services or products.

While preferences and contextual conditions are specified, it is possible that the user might be able to query the system for recommendations and might then get constructive suggestions that she can act upon (Jannach et al. 2011). For instance, to plan a travel RS the user would describe her preferences over features like day of the trip, airline, itinerary. Then, the user might request suitable suggestions.

2.9.2 Human computer interaction

Typical RSs used to support only a simple type of human-computer interaction with their users (Mahmood et al. 2009). Assuming the system assimilates the user model which lodges all the user's preferences, these RSs advise the user through a set of recommendations selected with regards to the user model.

However, the user model will not often be shaped totally at the beginning of the interaction (i.e., not all the user's preferences can be elicited). Moreover, users are rarely satisfied with the initial recommendations (Bridge et al. 2006). Therefore, it is more convenient to employ a more natural approach, similarly to a human-human interaction, where users can specify their preferences over an extended dialogue. Conversational RSs (McGinty & Smyth 2006) support a more interactive and diverse human-computer interaction in which the system supports many types of actions, and interacts with the user in order to determine her preferred products in an iterative, more engaging, and an effective manner.

An example of interaction model starts with the user who needs information (Moreno et al. 2011). Then, she formulates and sends her enquiry to a RS that she uses to help her search in a collection of items. Afterwards, the user evaluates the results sent from the system. The interaction ends, for instance, if the user is satisfied or she does not want to resend another query to the system, or there are no more queries to be sent.

2.9.3 Different techniques of recommendation

Many information filtering and recommendation methods have been developed in the literature. The proliferation of these approaches has meant that there is no accepted clustering of approaches, nor accepted names for all the approaches. Researchers usually use descriptive terms that characterize the approaches. In this work, we adopt the vision of Burke in (Burke 2002) who claimed RS techniques can be classified into four main categories based on the technique used to match users profiles with recommended items characteristics: Collaborative Filtering, Content Based Filtering, Knowledge Based systems and Hybrid systems. In Figure 2.3, we tried to draw a clustering of RSs that we consider quite reasonable. We included two criteria to create RSs clusters which are the interaction mode ("single-shot" or conversational) and the conversation strategy which specifies how the system approaches the user (navigation-by-proposing, navigation-by-asking, or navigation-by-information recommendation (described in Section 2.10.2.3)).

Interaction	Matching / Navigation strategy	Collaborative filtering RS	Content-based RS	Knowledge-based RS
Single				
Conversational	Navigation-by-proposing			
	Navigation-by-asking			
	Navigation-by-information recommendation			

Figure 2.3: Classification of recommender systems.

Two of the best known strategies of RSs that were presented so far in the community of RSs are content-based filtering and collaborative filtering (Smyth 2007).

- **Collaborative filtering** uses preferences of similar users in the same reference group as a basis for recommendation (Resnick et al. 1994, Shardanand & Maes 1995, Hill et al. 1995). Collaborative filtering-based RSs are based on past ratings of users with similar preferences under the assumption that human preferences are correlated (Resnick et al. 1994). Based on these ratings, the recommender tries to find either users that are most similar to the active user or items that are similar to the item for which the user's rating is being predicted. The former approach is called user-based collaborative filtering (Resnick et al. 1994), whereas the latter is known as item-based collaborative filtering (Sarwar et al. 2001). For example, if two customers bought similar Compact Disks (CD) and rated them similarly, the system would recommend to

one customer CDs that the other customer bought and rated positively. Therefore this approach makes recommendation based on collective preferences of the crowd.

In the context of preferences, this type of RS requires recommendation seekers to express preferences by giving a degree of desirability (e.g., rating) to a set of items. These systems focus on algorithms for matching people based on their preferences and weighting the interests of people with similar preferences (e.g., tastes) to produce a recommendation for the service/product seeker.

Collaborative filtering can find similarities among different users but can find it difficult to handle new items that do not have existing usage information.

- **Content-based filtering** uses keywords or other product-related attributes to make recommendations (Maes 1994, Lieberman 1997, Pazzani & Billsus 2007). In fact, this kind of approach for RSs matches product attributes to user-profiles under the assumption that each user operates independently. Its main purpose is to reduce irrelevant content and provide users with more pertinent products or services. This approach uses features of items the user liked in the past to infer new recommendations (e.g., they attempt to recommend items that are similar to items the user liked in the past). For instance, it selects and recommends products that are selected on the basis of purchasing information available from the current user.

In the context of preferences, this type of RS uses only the preferences of the recommendation seeker. Their focus is on algorithms for learning user preferences and filtering a stream of new items for those that most closely match user preferences.

The content-based approach can classify products and services based on their nature, but often have difficulties in identifying related interests of the same user since it recommends items that are highly ranked regarding the user profile. But it is unlikely to recommend to the user items liked by similar users (e.g., people who have the same purchase patterns) (Ge et al. 2010, Lops et al. 2011).

- **Knowledge-based recommendation** exploits deep domain knowledge in the form of mappings between the user preferences and the required product characteristics. Knowledge-based RSs have knowledge about how particular items meet particular user needs, and employ this knowledge to recommend items based on inferences about each user's needs and preferences (Burke 2002, Burke 2004, Burke 2007). In this RS, a suitable recommendation relies on product descriptions and learns an individual preference model from characteristics

of the current preferred product (Felfernig & Gula 2006, Mandl, Felfernig, Teppan & Schubert 2011) . However, the drawback of all knowledge-based systems is the need for knowledge acquisition.

An example of Knowledge-based RS would be a digital camera retailer as it needs the user to be aware of the camera features such as manufacturer, camera type, sensor resolution, optical zoom level, output format and storage type and size.

- **Hybrid Systems** combine several RSs techniques to overcome the limitations of pure techniques (Burke 2002). Hybrid systems often provide better accuracy than pure techniques (Schclar et al. 2009). Different efforts for combining collaborative and content-based methods into a hybrid RS can be classified as (Adomavicius & Tuzhilin 2005):
  - Combining predictions of separate implementations of collaborative and content-based methods.
  - Incorporating some content-based characteristics into collaborative approach.
  - Incorporating come collaborative characteristics into content-based approach.
  - Constructing a general unified model that incorporates both approaches.

2.9.4 Case-based recommender systems

The customers who purchase complex products such as financial services, laptops, or digital cameras need both information and intelligent interaction mechanisms that support the selection of appropriate solutions. One particular class of knowledge-based RSs is case-based recommenders (Smyth 2007, Bridge et al. 2005). This style of recommenders is suitable to help customers identify complex products. Indeed, for products like laptops, digital cameras, cars or houses, or even financial services, customers are not very keen on depending totally on other customers' reviews to be comfortable with a recommendation and make the decision to purchase products. For products that are relatively expensive or professionally critical, people are willing to put enough effort for the purpose of reaching a choice which satisfies the user's needs as best as possible with a minimum of risk (e.g., without substantial financial loss). In fact, people can spend time in interacting with the system in case they want it to recommend a suitable product which has a great value for them (e.g., a house).

Accordingly, such products are usually constrained by a set of features (for example, the digital camera has features like price, optimal zoom, resolution, etc.) for

which customers are able to specify particular preferences (e.g., to specify values, etc.) to filter out the available large data set.

Such organized information about products allow case-based recommenders to adopt and provide required formalisms for intelligent interaction mechanisms that support the selection of the most appropriate solutions within a sizeable set of possible items.

Instead of relying on the collective preferences that collaborative filtering uses, case-based recommendation suggests explicit sales dialogues that emulate the conversation between a customer and a sales assistant. These dialogues help to proactively suggest products to a given customer by predicting her preferences on products.

2.9.4.1 Single-shot recommender systems

Case-based recommendation systems used to operate in a "single-shot" fashion. When a "single-shot" RS receives a query from the user, it recommends a set of products to the user only once. If the user is not satisfied, there is only one way to proceed in order to readjust the "shot": to amend the query and start again. In "single-shot" RSs, the same results are often returned to the user despite being of no interest to the user (Smyth 2007).

This recommendation model supposes the user knows what she is looking for to a point that the system does not consider to have more than a single "exchange" with the user. Figure 4.1 in Chapter 4 illustrates the single-shot recommendation scenario. This would not be supported by the actual extreme diverse products that are available in the web. In fact, the ability to refine the search in large catalogs through multiple interactions could be helpful for the users. Using multiple interactions, a user can navigate a product space much more effectively while taking the time to express her preferences progressively.

2.9.4.2 Conversational recommender systems

Generally speaking people do not state their preferences up-front because initially they only have a vague idea of the product they would like to have (Bridge et al. 2005). Usually, criteria about the product the customer would like to purchase are specified during the dialog with the seller. This is still the case even for knowledgable customers in the domains where expert users need to be assisted because available products dynamically change. A distinctive example is the list of special offers (e.g., flight tickets) which change frequently.

In order to hold natural-like and interactive dialogue, conversational RSs (Bridge et al. 2005) were proposed. These systems support a dialogue where, at each stage,

the system can select one from a set of available system actions, e.g., recommend some products or ask the user for more information. The particular action selected by the system is determined by its *recommendation strategy*.

The recommendation strategy usually characterizes conversational RSs since it determines the action plan adopted by the recommender while interacting with the user, in order to help her reach her ultimate goal: obtain the most suitable possible offer from the system (e.g., product, service or combination of both). The recommendation strategy ultimately depends on the complexity of the user's goal, as it should accompany and guide the user while she is searching for her target. For instance, in the prototype of the Austrian Tourism portal (Mahmood & Ricci 2007, Mahmood et al. 2008), there are three possible targets for the user: 1) "Build a Travel Plan ", 2) "Window Shop ", i.e., just browse through the products, and 3) "Book a Product ", i.e., search for a single product and add it to the travel plan.

In a conversational scenario, which is depicted in Figure 4.1 in Chapter 4, the strategy is implemented by specifying the particular actions that the system executes at each stage of a given interaction session. Indeed, at each interaction cycle, the system can either request from the user a preference or propose a product to the user. The user can either answer the question posed or criticize the system proposal. As the recommendation strategy offers various ways of interacting with the user, conversational RSs are able to recommend a variety of products for diverse categories of users.

Requirements acquisition is regarded as a key element in e-commerce. The product search, which probably needs a filter-based retrieval, can take place in tandem with preference elicitation. Bridge et al. (Bridge et al. 2005) described two ways of requirements elicitation by which conversational recommenders can be identified: the first approach is called *navigation-by-asking* according to which the RS sets a dialogue with the user during which the RS selects and asks questions to the user, whether up-front (e.g., form-filling) or incrementally. The user's preferences are elicited when the user answers the questions (Goker & Thompson 2000, Doyle & Cunningham 2000, Shimazu 2001, Shimazu 2002, Schmitt 2002*b*, Thompson et al. 2004*a*). The second approach is called *navigation-by-proposing* according to which the RS may show the user intermediary products so that it gets a form of feedback from the user's *critiques* to the proposed products when invited to give feedback about what was presented to her (Burke et al. 1996, Burke et al. 1997*a*, Shimazu 2002, Faltings, Pu, Torrens & Viappiani 2004, McCarthy et al. 2004, Pu & Faltings 2004, McCarthy, McGinty, Smyth & Reilly 2005, McCarthy, Reilly, McGinty & Smyth 2005, Chen & Pu 2009).

Conversational RSs may implement only the asking/answering conversation mode (Linden et al. 1997), only the proposing/ criticizing conversation mode (Burke 2002),

(Pu & Chen 2005), or both (Shimazu 2001). Switching between these two kinds of navigation has also been addressed in (McGinty & Smyth 2003).

2.10 Preference Handling Methods in Conversational Recommender Systems

The acquisition of preferences is a central challenge in interactive systems like RSs (Chen & Pu 2004). There are two major approaches in today's RSs: utility functions (Fishburn 1967, C. 1968, Fishburn 1970, Fishburn 1974, Fishburn 1999) and relational preference structures (Kießling 2002, Öztürk et al. 2004). A utility function assigns a numerical score to each data item. The score of an item depends only on the item's properties and represents its overall desirability. Relational preference structures link pairs of items through the notions of "is preferred to" and "is equally preferable as" thus leading to qualitative preference orderings.

Typically, the task of the recent online conversational recommenders is to elicit the customer requirements, while interacting with her, in a personalized way. These recommenders exchange information with the user and suggest products to get to know her needs. Then, they provide additional help and guidance depending on the user's background, find the most suitable alternatives, eventually provide explanations for the recommendation and explain the differences and similarities between products (Jannach et al. 2007). We intend to study preference handling with the interaction mechanism that we consider comparable to the way Information Recommendation (IR) interacts with the user (e.g., navigation-by-proposing or navigation-by-asking).

2.10.1 Critiquing

Conversational RSs are used to help users navigate through product databases by alternatively making suggestions and soliciting user feedback in order to guide subsequent suggestions. There has been an interest in developing effective interfaces that support user interaction in domains of limited user expertise. Critiquing has been proven to be a popular and successful user feedback mechanism in this regard (McCarthy, Reilly, McGinty & Smyth 2005, McGinty & Reilly 2011). This approach is one interaction model that allows users to build their preferences by examining or reviewing examples shown to her by the system. McCarthy et al. (McCarthy, Reilly, McGinty & Smyth 2005) state that critiquing was introduced as a form of feedback for recommender interfaces as part of the *FindMe* recommender systems (Burke et al. 1996, Burke et al. 1997*b*). Example critiquing was observed in (Tou et al. 1982) as a new interface paradigm for database access, especially for non-expert users to

specify queries. This approach alternates phases of intermediate product recommendation and requirements/preferences refinement by critiquing the presented alternatives whose retrieval is based on similarity between items. It thus facilitates the incremental construction of the user's preferences. Moreover, it enables the user to specify her requirements at an informal level, either directly via a graphical representation of the requirements or indirectly by critiquing available products. Thus, the system updates the user's preferences during the critiquing process and reacts promptly by providing her with immediate feedback about the consequences of her critiquing wishes: the subsequent shown products are selected regarding the updated user's model of preferences.

Moreover, in (Reilly et al. 2005), the authors use the *incremental critiquing* as they exploit both the critiquing history of users as well as their current critique in a PC recommendation scenario. Furthermore, *dynamic critiquing* is utilised by the authors in (Reilly et al. 2007). In fact, they apply two approaches to dynamically generate critiques in a conversational RS for recommending laptop computers.

The user's model can be implicit or explicitly shown to the user, via an interface (Torrens et al. 2002), while the user is critiquing shown alternatives.

2.10.2 Some approaches for conversational recommender systems

Different approaches and applications of RSs that are based on example critiquing have emerged. We will present a number of these approaches and systems.

2.10.2.1 FindMe

One approach to using the implicit user's model is called *FindMe* (Burke et al. 1996, Burke et al. 1997*a*) where knowledge (or collected preferences) about the current product selected by the user is used to rank and recommend a list of catalogue products from the case base. A FindMe system first retrieves and displays the best matching product from the database based on the user's initial query. It then retrieves other products based on the user's critiques of the current best item. The interface implementing the critiquing model is called tweaking, a technique that allows users to express preferences with respect to a current example, such as "look for an apartment similar to this, but with a better ambience.". According to this concept, a user navigates in the space of available products by tweaking the current best option to find her target choice. A number of systems adopt the FindMe approach (e,g., Entree (Burke 2000), RentMe (Burke et al. 1997*b*)); they support a similar interaction flow with the users, that is the user initially provides some information about her desired

product, either by constraining and executing a product query, or by providing some specification about the desired product.

The FindMe approach has been successfully applied in the *Entree* system for recommending restaurants (Burke 2000), and the *Wasabi* tool for assisting the user in browsing an online database (Burke 1999). The *PTV* system is another application of FindMe approach which recommends TV programs (Smyth & Cotter 2000, Smyth & Cotter 1999).

2.10.2.2 More Like This & Partial More Like This

The user feedback employed in conversational RSs was also studied in (McGinty & Smyth 2002*a*, McGinty & Smyth 2002*b*) through a proposed comparison-based recommendation algorithm that works as follows: It recommends a set of k items that are likely to satisfy the user's needs. Then, the user is enabled to select the best match, considered by the system as "positive feedback". Next, the system will analyze the user feedback by evaluating the information about the difference between the selected and the $(k-1)$ remaining items. This analysis determines the set of recommended items for the next iteration. The recommendation session terminates either when the user is presented with a suitable item or when she gives up. *More Like This* (MLT) and *Partial More Like This* (PMLT) are two approaches in this work whose role is to induce preferences when the user reacts to the recommended items. They both generate preference statements stating the preference of features that mark the selected item over those that characterize the rejected items during an interaction stage. While this work was applied to a conversational RS for PC, a similar approach has been employed in (Smyth & McGinty 2003) for whisky recommendation.

2.10.2.3 Information Recommendation

Information Recommendation (IR) is a conversational approach on which we will focus in this thesis and it will be presented in more detail in Section 4.2 in Chapter 4. Briefly, this approach aims at suggesting to the user how to reformulate her queries to a product catalogue in order to find the products that maximize her utility. In (Bridge & Ricci 2007), the authors showed that, by observing the queries selected by the user among those suggested, the system can make inferences on the true user utility function and eliminate from the set of suggested queries those products with an inferior utility (these queries are called dominated queries). The computation of the dominated queries was based on solving several linear programming problems.

Blanco et al. proposed a new technique for the computation of the dominated queries (Blanco et al. 2012*a*). In fact, while the work in (Bridge & Ricci 2007) assumes that the user utility function is an arbitrary one (i.e., coming from an infinite

set), Blanco et al. assume the user utility function is drawn from a finite set of user profiles that are known by the system. This set represents the possible different users that the system considers that it may interact with. They show that under this assumption the computation of the query suggestions is simplified and the number of query suggestions is strongly reduced regarding those shown by (Bridge & Ricci 2007). Besides, the authors artificially assumed that the true user utility function is included among the finite set of user profiles contemplated by the system. In (Blanco et al. 2012*b*), they treat this as a crude simplification since a totally unknown user approaching the system may have an arbitrary profile and the system has no knowledge about that. Then, they have removed that assumption and they have also extended the type of query editing operations previously described in (Bridge & Ricci 2007).

2.10.3 Some applications of conversational recommender systems

Recently, different applications based on conversational recommenders have emerged. Several real-life applications (e.g., travel planning, restaurants) took advantage of the critiquing approach while combining several ideas to help the user navigate through the product databases and obtain what she wants.

2.10.3.1 Travel planning advisors

Tourism is a primary application area for mobile applications and a large number of services are now offered to support the traveller before, during and after travel (Ricci & Nguyen 2005). Several RSs have been applied to travel and tourism applications. We present some of these applications.

Trip@dvice

Tripdvice is one instance of the proposed approach (Cavada et al. 2003, Venturini & Ricci 2006) that supports the selection of travel-related products (e.g., a hotel or a visit to a museum or a climbing school) and the building of a travel plan. The Tripdvice approach has been successfully validated in an online evaluation with real users (Zins, Bauernfeind, Missier, Mitsche, Ricci, Rumetshofer & Schaumlechner 2004, Zins, Bauernfeind, Missier & Rumetshofer 2004). This approach was employed within several travel-based conversational systems: Dietorecs (Mirzadeh & Ricci 2007), NutKing (Mirzadeh & Ricci 2007, Mahmood & Ricci 2007), ITR (Ricci, Arslan, Mirzadeh & Venturini 2002, Mirzadeh & Ricci 2007).

Tripdvice offers two different product search functions for the users, i.e., Query Search and Seeking for Inspirations.

The "Query Search" function integrates Case-Based Reasoning (CBR) techniques (Smyth 2007, Bridge et al. 2005) along with user query management functions, i.e., query tightening and relaxation suggestions, so that the advisor is able to retrieve and recommend a manageable subset of interesting tourist products to the users. During the interaction, the system performs system actions of a different kind so as to give the users the chance to update their queries, e.g., suggest some attribute for tightening or relaxing, or offer the user the opportunity to provide some travel characteristics. The user can answer back in many ways, e.g., by accepting the suggestions, reformulating her query autonomously, specifying the travel characteristics. For example, the user asks the system for recommendations about a product type (e.g., an activity). Then, the system reacts to this query either by recommending some products, or, in case the query fails (the product does not exist in the database), by suggesting some query refinements. The user and the system keep on challenging each other within the conversation until the user's travel plan is complete.

Trip@dvice identifies and recommends ranked tourist items. The basic idea is to use a hybrid "cascade" recommendation methodology (Burke 2002) where first a conversational approach is used to select a set of options and then a *collaborative via content* approach approach is used for ranking (Pazzani 1999). High scores are assigned to products that are similar to a product chosen by a user with similar needs and preferences (Mahmood 2009).

Being guided to the desired product through queries might not suit all users. Such a user is offered an alternative product-based search: "Seeking for Inspirations", is the second search function implemented by Trip@dvice. It allows users to seek for products that stimulate their liking, i.e., products with which they would be happy. The system shows suitable proposals to the user, regarding the user's preferences and using CBR techniques, and gives her the opportunity to select or request more products. This continues until the user is happy with her findings, i.e., a travel plan.

INTRIGUE

Another example of a travel-based conversational recommender is *INTRIGUE*, which is a prototype system that is capable of recommending interesting destinations and itineraries in a restricted geographical area (Ardissono et al. 2003). The navigation-by-asking strategy is applied in INTRIGUE.

SmartClient

The *SmartClient* system is one example using an explicit user's model which is used for travel planning (Pu & Faltings 2000, Torrens et al. 2002). The approach to electronic catalogs adopts interaction sequences that are supported through the use of a CSP as a basic selection mechanism. In fact, user preferences are explicitly modeled as a CSP. The user can discover her preferences through checking the available choices. The user model is updated when the user gives critiques about the shown choices. The update consists of posting or retracting constraints which results in a conversation that leads the user to refine her needs.

Other applications that work in a similar fashion are ATA (Linden et al. 1997) and the incremental dynamic-critiquing systems (McCarthy, Reilly, McGinty & Smyth 2005). Letting the user have a look at and update the user model can help users understand the system's moves when they change from one interaction to the next and thus will have confidence in the system (McGinty & Reilly 2011, Tintarev & Masthoff 2011, Pu et al. 2011). With an explicit user model, the system is unlikely to recommend products that have already been disliked by the users.

Two action plans, among several others, that can be adopted by conversational RSs are the following:

- to be querying the user about her preferences, and acquiring enough information from her, in order to select a candidate set of products;
- to be proposing some candidates and acquiring user preferences as critiques, so as to to personalize the future recommendations.

Several approaches combine the two strategies mentioned above in an attempt to bring "the best of both worlds".

VIBE

VIBE (http://www.warmbad.at) is a virtual advisor at an Austrian spa resort. It actively promotes the variety of different tourism products that ranges from hotel accommodation in four and five-star categories, a variety of recreational and sporting facilities to health and beauty therapies. This advisor guides the user and engages her in a preference elicitation dialogue through a personalized series of questions in which the customer's preferences are incrementally elicited. It is a successful application within *ADVISOR SUITE* (Jannach et al. 2007): an approach which provides an off-the-shelf software framework for the rapid and cost-efficient development of an online pre-trip travel advisory service.

2.10.3.2 Restaurant advisors

Restaurants business have also adopted conversational approaches. We present two examples of restaurant recommenders.

Entree

The *Entree* restaurant recommender (Burke 2000) makes its recommendations by finding restaurants similar to restaurants the user knows and likes. The system allows users to navigate by stating their preferences with respect to a given restaurant, thereby refining their search criteria in a number of iterations. During each cycle, Entree presents users with a fixed set of critiques to accompany a suggested restaurant case, allowing users to tweak or critique this case in a variety of directions; for example, the user may request another restaurant that is cheaper or more formal, for instance, by critiquing its price and style features.

The Adaptive Place Advisor

Adaptive Place Advisor (APA) is a human dialogue-based conversational RS, used for restaurant recommendation (Thompson et al. 2004*a*). The approach to acquiring user models is to infer user preferences unobtrusively, by examining normal online behavior (Rafter et al. 2000). It supports an interactive conversation by querying the user or making recommendations, in response to a diverse set of user requests. For instance, the system might recommend products or allow the user to constrain some product attribute in her query, for example, to tighten her query, or offer her to remove one of the attributes in case no products are retrieved (i.e., relax her query). The user is then free to either accept or reject these offers. The APA exploits the user's feedback to narrow down the list of subsequent alternatives and hence helps the user select her desired restaurant. It personalizes and guides the interaction by updating and exploiting a model of the users' preferences throughout the conversation. This activity is the source of the APA's adaptive behavior. The authors have successfully validated APA in an experimental evaluation with real users, where the goal is to recommend preferred restaurants in the San Francisco Bay Area. The most important distinction of APA is that the interaction takes the form of a sequence of questions (which mimic natural language) that are designed to eliminate some items from consideration; the goal is to remove alternatives rather than to give a ranking of the items.

The system keeps interacting with the user without proposing final products until at most a few choices remain. This aims at avoiding showing an unmanageable number of products to the user. The system alternates between asking the user to

revise the query and showing items. There are three situations where the user model is updated:

First, the authors presume that when a user accepts an item, she is indicating: (1) a preference for the item itself, (2) preferences for the attributes she constrained to find this item, and (3) preferences for the values she provided for those attributes. Thus, when a user accepts an item presented by the system, the probabilities and weights relative to the appropriate item, attributes, and values are updated.

Second, when a user rejects an item presented by the system, it was only assumed that she has a dislike for the particular item. Therefore, for rejected items the system simply adds one to the presentation count, that counts how many times the item was presented to the user.

The third situation in which the system updates the user model is when, after the query has become over-constrained, it presents an attribute for relaxation and the user accepts that relaxation. In this situation, it was assumed that, if there was a matching item, the user would have been satisfied with it, since the characteristics specified in the conversation so far were satisfactory. Therefore, before the relaxation occurs, the system increases the attribute preferences for the constrained attributes and increases the value preferences for user-specified values.

Thus, both acquisition and utilization occur not only when items are presented to and chosen by the user, but also during the search for those items. Finally, the system's representation of models goes beyond item preferences to include preferences about both item characteristics and particular values of those characteristics.

2.10.3.3 Music advisor

In (Warnestal 2007), the author presented a behavior-based dialogue model called *BCORN*. BCORN utilizes a user preference modeling framework that supports and utilizes natural language dialogue, and allows for descriptive, comparative, and superlative preference statements, in various situations. BCORN was successfully applied to then evaluated with a music conversational recommender called *CoreSong*. It was shown that BCORN is able to generate coherent, flexible, and effective dialogue during its dialogues with the users.

2.10.3.4 Other applications

In the following, we cite some others applications of conversational RSs. In (Burke et al. 1997*b*), the authors present different systems that are based on the FindMe approach: The *Car Navigator* system (Burke et al. 1996) recommends cars models. Videos are recommended in the *Video Navigator* and *PickAFlick* systems. The *Kenwood* system recommends audio-related products (e.g., home theater systems), and

the *RentMe* (Burke et al. 1996) system recommends rental accommodation.

Another example of conversational RSs is described by Bridge through a prototype for conversational RSs that deals with accommodations for rent (Bridge 2002). It presented a framework within which users' preferences were elicited from the user who is allowed to constrain her query in order to manage the number of suitable products and recommend a reasonable number of items in each iteration of the dialogue.

The conversational approach has also been applied within *ExpertClerk* (Shimazu 2001, Shimazu 2002), which was designed and developed to imitate the conversation techniques of human sales clerks. In order to elicit the shoppers' preferences during the dialogue, the assistant typically alternates between asking questions and proposing suggestions to the customer until the customer is satisfied or all possibilities have been presented. ExpertClerk inherits the navigation-by-asking idea from ExpertGuide and solves the scalability problem of *ExpertGuide* (Shimazu et al. 2001), which is using only techniques of navigation-by-asking. In navigation-by-asking, the advisor's goal is to choose a sequence of questions that most effectively homes in on desirable products. Questions are often selected based on the user's queries and the product distribution. Information gain is usually used to select questions that partition the product space effectively.

2.11 Constraint Satisfaction Problem

Many problems ranging from resource allocation to fault diagnosis and design, involve constraint satisfaction as a key and natural component.

2.11.1 Definitions

A constraint satisfaction problem (CSP) involves the assignment of values to variables that have a set of constraints which disallow individual or combinations of values. A large variety of problems in AI can be modeled as CSPs. Examples include problems in machine vision (Montanari 1974, Mackworth 1977), belief maintenance (Dechter & Dechter 1988), scheduling (Sycara et al. 1991), and natural language processing (Menzel 1998).

The formal study of CSPs was initiated by (Montanari 1974) who used constraint networks to describe combinatorial problems arising in image processing. A great deal of research in constraint satisfaction has focused on algorithms which, given a constraint network as input, automatically find a solution. This is useful in applications where, once the problem has been formulated as a constraint network, no user interaction is required. CSPs can be hard to solve, and all known solution methods

have worst-case exponential time performance.

Constraints are requirements that impose limitations on the possible scenarios, regarding the resolution of a problem, and they have to be met (Dechter 2003, Rossi et al. 2006, Rossi 2005). Constraints can arise for a variety of reasons. For example, when choosing a laptop, we may be interested only in those that have an Intel processor and Nvidia graphics card. If an e-commerce web site does not have laptops in his catalogue with wanted features we would leave it for another web site.

Each variable involved in a CSP has a domain which represents all possible and allowed values to be associated to the variable. The constraints restrict the assignment of the values to a variable, or a combination of variables. Altogether, variables, domains, and constraints form a constraint network which is also called an instance of CSP.

Definition 1. *A constraint network is a triple $R = (X, D, C)$ where*

- *$X = \{x_1, ..., x_n\}$ is a finite set of n variables.*
- *D is a function that maps each variable x_i in X ($i = \{1, ..., |X|\}$) to a finite set of values, written $D(x_i)$, which it is allowed to take. The set $D(x_i)$, called the domain of x_i, is also denoted Dx_i.*
- *A constraint is a requirement that determines a set of possible combinations of values of the variables the constraint deals with. The arity denotes the number of variables the constraint is dealing with. Accordingly, constraints can be classified in three types. The first type is the unary constraint, which restricts the values of a single variable. Every unary constraint can be eliminated by simply preprocessing the domain of the corresponding variable to remove any value that violates the constraint. A binary constraint just relates two variables. When all constraints in a network are binary, the network is called a binary constraint network. A non-binary constraint includes three or more variables. A constraint can be defined explicitly as a set of permitted combinations of values. Conversely, a constraint can be defined implicity. Examples of implicit definitions can be a mathematical expression (e.g., $x_1 < x_2$), or a specific semantic (e.g., the all-different constraint which is equivalent to saying that variables the constraint involves must be taking different values), etc. Set of variables S is the scope of the constraint which means that the constraint involves every variable in S, and $|S|$ denotes the arity of the constraint. A tuple on scope S is an assignment of a value to each variable in S. A constraint c with scope S represents a relation on variables in S, and thus c corresponds with a set of assignments to S.*
- *C is a finite set of constraints. Let $S \subseteq X$.*

Definition 2. *Let $R = (X, D, C)$ be a constraint network. An instantiation of a set of variables $S \subseteq X$ is a simultaneous assignment of values to the variables S. We denote the instantiation briefly as b_S. Let C_S be a constraint whose scope is S. b_S satisfies a constraint C_S if $b_S \in C_S$ which means that the combination of values b_S satisfies (is allowed by) C_S. An instantiation b_T, where $T \subseteq X$, is consistent with regards to R if and only if b_T satisfies all constraints $C_S \in C$ such that $S \subseteq T$.*

Definition 3. *A solution α of the constraint network $R = (X, D, C)$ is an instantiation of all variables in X which is consistent relatively to R. The set of all solutions to a constraint network R is denoted $Sol(R)$.*

A network R is called satisfiable if $Sol(R) \neq \emptyset$ and unsatisfiable if $Sol(R) = \emptyset$. Two networks defined on the same set of variables are considered equivalent if they have the same set of solutions. For a constraint network $R = (X, D, C)$, any subset of variables $I \subseteq X$ induces a subnetwork of R which has variables I, domains D and the set of constraints is a subset of C: C_S such that $C_S \in C, S \cap I \neq \emptyset$.

Given a constraint network R, we can consider the following tasks:

- Determine whether the network R is satisfiable.
- Find a solution to the network R, with no preference as to which one.
- Find the set of all solutions $Sol(R)$.
- Find an optimal solution to the network R, where optimality is defined by a function on the variables in R.

2.11.2 CSP solution methods

Solving a CSP means assigning a value to each variable such that all constraints are satisfied. That is, a CSP solution is an assignment of values to variables which satisfies every constraint. A CSP can be solved using the generate-and-test method (also know as "the British Museum Algorithm" according to (Hoare 1989)), where each time a possible instantiation of all the variables is generated, it is tested against all the constraints to gauge the feasibility of the assignment newly created.

A more ingenious way of solving a CSP is backtracking search (Bitner & Reingold 1975), which is the principal complete mechanism for solving a CSP. The term backtracking search is used for a depth first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign. The term backtracking search implements a search tree which is composed of nodes, edges (arcs) and an ordering $\succ$ on the outgoing arcs of each node (Bessière et al. 2004).

When searching for (feasible) solutions, variables are instantiated in a chosen order. By default, the procedure *SelectUnassignedVariable*, mentioned in Algorithm 1,

simply selects the next unassigned variable in the order given by the list of variables X. Similarly, the procedure *SelectUnassignedValue*, mentioned in Algorithm 1, selects the next unassigned value. The functions *SelectUnassignedVariable* and *SelectUnassignedValue* can involve variable and value selection heuristics (Brélaz 1979, Haralick & Elliott 1980, Pirlot 1996, Boschetti et al. 2009).

The constraint is checked every time one of the variables, involved in this constraint, is instantiated with an untried value. A violation of a constraint provokes the search to step back to the most recently instantiated variable that will be assigned a value among the values in the domain which are not yet instantiated. This backtracking takes place by eliminating a subspace from the Cartesian product of the variable domains.

A simple backtracking algorithm (Russell et al. 1996), which is also called chronological backtracking, is presented in the following algorithm (see Algorithm 1).

Algorithm 1 BACKTRACK(X, D, C, b)

Input : Set of variables X
Set of constraints C
Set of variables values D
Partial assignment b
Output: Set of assignments

```
1  if X = ∅ then
2      return b;
3  else
4      x ← SelectUnassignedVariable(X);
       foreach value v ∈ SelectUnassignedValue(D_x) do
5          if v is consistent with assignment b then
6              b ← b ∪ (x,v);
               X ← X \ x
               r ← BACKTRACK(X, D, C, b);
7              if r ≠ NULL then
8                  return r;
9              else
10                 b ← b \ (x,v);
11     return NULL;
```

Given a network $R = (X, D, C)$ the call $BACKTRACK(X, D, C, b)$ returns a solution if the network is satisfiable, otherwise $NULL$ is returned. b represents a partial assignment. The $BACKTRACK$ procedure is initially called with b that is equal to the null assignment to the empty set of variables. Figure 2.4 shows the

search tree when $BACKTRACK$ is applied to an unsatisfiable problem. The left subtree (a) illustrates the search space where all possible nodes were visited as no constraints were applied unless the algorithm reaches a leaf node. The right subtree (b) is an example of a search tree where there is constraint propagation. It shows the search space where one subtree is eliminated because the first variable was assigned a value that cannot be involved in a consistent assignment.

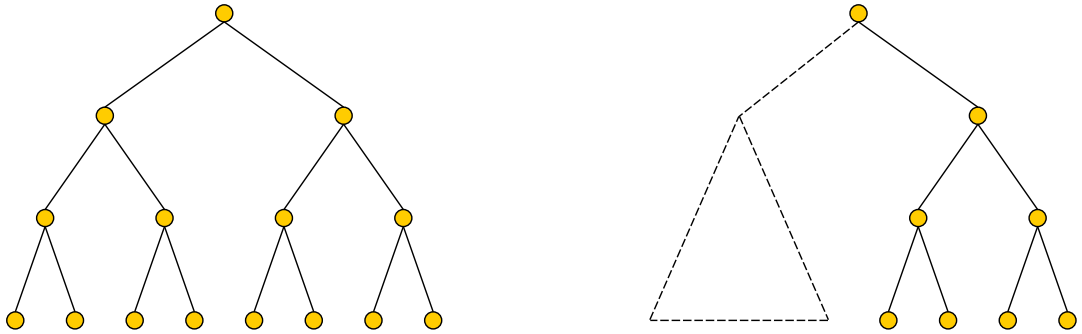


Figure 2.4: Search tree example

Modern CSP solvers tend to integrate forward-checking strategies that enable them to avoid portions of the search space when constraints take effect (Vion 2006). There is a massive literature on search techniques. For example, Miguel and Shen (Miguel & Shen 2001*b*, Miguel & Shen 2001*a*) gave a survey on many of the known search strategies. Kondrak and van Beek (Kondrak & van Beek 1997) also presented a theoretical evaluation of several backtracking algorithms.

2.12 Constrained Optimisation

While decision problems require a boolean answer, optimisation problems require the best solution(s), or reasonably good solutions. Computational approaches for solving COPs consist of mainly two fundamental principles: exploring a vast solution space toward a desired solution while trying to eliminate sub-parts of the solution space which are guaranteed not to have a better solution (Kadioglu 2012).

Thus, there is a need for mechanisms that determine how to compare one solution to another to differentiate between possible solutions and so find the optimal ones. Preference handling has active research lines in optimisation (Rossi et al. 2008) since the latter may rely on preferences that express cost or objective fulfillment and favour the search for optimal solutions.

2.12.1 Constrained optimisation problems: different solution methods

A large number of real-world problems involve combinatorial optimisation problems. Combinatorial optimisation problems are usually easy to state and involve finding the best solution(s) among a very large number of feasible solutions might be difficult (Neumaier 2004). Most of these problems have no known polynomial methods to solve them. Examples are production planning, crew scheduling, and vehicle routing; these problems are NP-hard (Perron & Trick 2008).

Many other real-life applications benefit from constrained optimisation. In computer animation, for example, many activities and tasks, such as seeking boundaries and interpolations, utilise optimisation under constraints. Traveling support is a massively used web-based service (e.g., booking a flight or a hotel). Known e-commerce web sites devoted to tourism, such as TISCover (www.tiscover.com), Priceline (www.priceline.com) and Expedia (www.expedia.com), support the tourist with constrained optimisation-based personalized products.

Therefore, constraint optimisation can benefit to several fields which arouse the curiosity of researchers during the past half century (Agarwal & Sharir 1998). Then, research has focused on ways of solving COPs.

Having a set of possible combinations represented either explicitly (e.g., as a set of outcomes) or implicitly (e.g., as a CSP which describes a set of outcomes), the goal of solving an optimisation problem is to find some or all assignments of values to the variables that are not dominated (i.e., preferred) by other outcomes.

The existing search methods can be classified into two main classes: complete and incomplete search methods. Complete search algorithms are guaranteed to find the optimal solution and to prove its optimality. If optimal solutions cannot be computed efficiently in practice, one alternative is the incomplete methods which trade optimality for efficiency. That is, the guarantee of finding optimal solutions can be sacrificed for the sake of getting very good solutions quickly for instance.

2.12.1.1 Complete search

These methods go through all possible solutions while storing the current best ones that they find.

2.12.1.1.1 Exact methods Branch and Bound (B & B), Branch and Cut, Brute-force search are considered among the most well-known exact methods (Kadioglu 2012). The first two methods are well described by (Chinneck 2007). The first method

is discussed in Section 2.12.3. The third method is a very simple but also a very inefficient method where all possible solutions are enumerated and checked for satisfaction of the problem objectives. Using exact methods has a confirmed advantage: any found optimal solution is global. The disadvantage of such methods is their time inefficiency.

2.12.1.1.2 Search heuristics The enumeration method suffers from the plague of dimensionality (Kadioglu 2012). One argument is that complete search is essential with examples but is often intractable in practice. Thus, ways around such a problem are to look for complete search techniques that can shrink a number of possible but non-dominated combinations at the same time. Search heuristics (Edelkamp & Schrödl 2012, Boschetti et al. 2009) are one example of such techniques that might predict the non-optimality of some paths in the search tree without necessarily visiting all nodes of the search tree. They guide the search towards areas of the search space that are likely to contain optimal solutions. To determine optimality, it has to be proven that there is no better solution and every possibility allowed by the constraints has to be explored. In this case, (variable and value) ordering heuristics are recommended (Meisels et al. 1997, Haralick & Elliott 1980, Dechter & Pearl 1987, Topaloglu & Ozkarahan 2004). Indeed, when solving an optimisation problem, these ordering heuristics aim at guiding the search towards the best assignments, thus allowing the bounding functions to prune more branches. More information about these heuristics can be found in (Rossi et al. 2006, Barták 2012, Kadioglu 2012).

2.12.1.2 Incomplete search

Unlike exact methods, a number of heuristics and metaheuristics (Voss et al. 1999, Polya 1971, Glover 1986, Glover & Laguna 1997) do not provide feasible solutions which are proved to be optimal in a global sense. They, instead, can provide pretty good solutions in shorter time. Examples of the most common methods that are used to solve combinatorial optimisation problems are: Ant colony optimisation (Dorigo & Caro 1999, Dorigo et al. 1999, Dorigo et al. 1996), Bees algorithm (Pham et al. 2006), Local search and Generalized local search (Hoos & Stützle 2004, Pirlot 1996), Nearest neighbor heuristic (Carey & Schneider 1995), Clarke and Wright savings heuristic (Clarke & Wright 1964) and Tabu search (Glover & Laguna 1997).

2.12.2 Constrained optimisation: coupled and decoupled approaches

Preferences and constraints can be represented either by unique structures or separated in different structures. Thus, the COPs can be broadly divided into two categories (Boerkoel et al. 2010): 1) The first is denoted by *coupled* approaches and includes problems where a preference and a constraint are represented by means of the same kind of information (e.g, soft constraint). 2)The second is denoted by *decoupled* approaches and includes problems where preferences and constraints are represented by two separate kinds of information.

Sometimes specific requirements (constraints) may be even contradictory which complicates selection of any solution. After relaxing some constraints in the over-constrained problem, the latter can be transformed into an optimisation problem where the goal or the target may express understanding of the "best" possible solution. Soft constraints can represent quantitative preferences in coupled approaches which have included approaches such as Max-CSP (Serna et al. 2005), weighted CSP (Bistarelli et al. 1999), fuzzy CSP (Schiex 1992) and temporal CSP (Peintner et al. 2005).

There are examples which involve a large number of soft constraints to express preferences over a relatively small number of variables (Boerkoel et al. 2010). Besides, typically the preference degree of an assignment to some subset of variables is not conditioned on assignments to other subsets of variables. Thus, on the one hand authors in (Boerkoel et al. 2010) argued coupled approaches to constrained optimisation are not very suitable for problems with strongly conditional preferences (e.g., CP-nets). On the other hand, the same authors claimed that decoupling constraints and preferences have the advantage of allowing the user to focus only on expressing her preferences. Thus, it saves the user the burden of expressing what is possible (i.e., hard or feasibility constraints). Product configuration might be an example of this and where the knowledge that is required for solving a problem might come from different places or people: a vendor knows what products can be built (e.g., which product components can be put together), while the vendee knows what she wants to buy.

In this dissertation, we deal with decoupled approaches that combine a CSP and compact comparative preferences to obtain a COP.

2.12.3 Branch and bound

Branch and bound (B & B) is a common technique for solving COPs using backtracking search. The basic concept for almost all complete global optimisation algorithms is the branching principle. This technique consists of splitting (branching) the

original problem recursively into subproblems which become sooner or later easy to solve. In pure branching methods, the more likely branches are split more frequently, while in B & B methods one computes for each subproblem bounds with respect to the preference relation in the hope of being able to eliminate many subproblems at an early stage. Indeed, a global upper bound is maintained (in case we are minimizing the cost) which represents the cost of the best known solution to the problem. At each node of the search tree, B & B selects an unassigned variable X, whose current domain is $\mathcal{D}(X)$, and branches on each unpruned value in $\mathcal{D}(X)$ (i.e., each of these assignments generates a new branch in the search tree). Branching on the individual values of a variable is called value branching. After assigning a value to X, a lower bound on the cost of any complete assignment extending the current assignment, is calculated. If the bound is greater than the global upper bound (in case we are minimizing the cost), B & B backtracks, since no optimal solutions can be found in the subtree below the current assignment (the bound closes the branch that cannot improve the current best solution). Thus, B & B continues its search until either a bound is violated or the domain of every variable has been reduced to a single value. In the second case where every variable domain is reduced to a single value, the combination of these values constructs a basis for a new best solution.

This is a very useful technique since it allows one to reduce the feasible region in many cases by exploiting properties of the problem (e.g., user's preferences). B & B algorithms provide suitable methods for solving many global optimisation problems (Parker & Rardin 1988).

When hard constraints are included, the search space can be traversed by any of the search methods reported in the literature (Brélaz 1979, Lecoutre 2009). The CSP paradigm yields several techniques, such as constraint network decomposition (Sprecher 2002) and constraint propagation (Dongen & Lecoutre 2010), which can be harnessed to facilitate a B & B search for optimal solutions.

A thorough discussion of B & B in discrete optimisation, with many algorithmic choices that are of potential interest in general global optimisation, is given in (Parker & Rardin 1988). Search trees are generated by a backtracking algorithm for generating solutions of a CSP.

2.12.4 Preference-based complete search

The standard backtracking algorithm used to solve a CSP, described in Section 2.11.2, can be adjusted to solve a COP. The above standard backtracking search algorithm stops as soon as it finds a feasible solution. Instead of stopping, the algorithm could continue to search for more solutions, always comparing any newly found solution with other solutions that were already generated. Thus, non-dominated solution(s)

cannot be identified until all feasible assignments (with regards to the CSP) are enumerated then compared to each other. When the algorithm has exhausted the search space, it guarantees all optimal solutions are generated.

The ultimate goal of a COP-solving backtracking algorithm is to enumerate non-dominated (i.e., optimal) solutions. A B & B approach checks whether a solution that is already found is preferred over (i.e., dominates) any outcome that extends the partial assignment associated to some node N of the search tree. If it is the case then the algorithm backtracks since the subtree rooted at node N cannot contain a non-dominated solution. Thus, B & B improves the search efficiency by avoiding searching in parts of the search space where all possible assignments are dominated. Comparison between assignments is based on a preference relation and performed using the dominance testing operation (Boutilier et al. 2004*b*). The semantics behind this kind of dominance relation is described in Section 3.4 in Chapter 3.

Definition 4. *Let Ω be a set of outcomes and $\succeq$ be a pre-order preference relation. We say that outcome α is optimal in Ω if there exists no outcome β such that $\beta \succ \alpha$. If Ω is represented as the set of solutions of a CSP then we refer to such tasks as constrained optimisation.*

Therefore, given a dominance relation $\trianglerighteq$ based on user preferences, an appropriate search tree can be used to enumerate the set Ω of $\trianglerighteq$-undominated solutions: when a new solution α is met, it is checked whether it is dominated by any of the $\trianglerighteq$-undominated solutions already found. If it is not the case then α is added to Ω; α cannot be $\trianglerighteq$-dominated by any solution found later as the variable and value instantiation ordering is compatible with preferences. Two approaches using different dominance relations are presented in (Boutilier et al. 2004*b*) and (Wilson 2006, Wilson 2011). These dominance relations are described in more details in Chapter 3.

2.12.5 Conditional preferences-based constrained optimisation

Several works have explored the use of conditional preferences (e.g., CP-nets, cp-theories) in order to select optimal solutions for a problem whose requirements are expressed as hard constraints and optimality is described though conditional preferences, e.g., (Prestwich et al. 2005, Gavanelli & Pini 2008*b*, Purrington & Durfee 2008) for CP-nets, and (Castell et al. 1996, Rosa et al. 2010, Jin & Somenzi 2005, Jin et al. 2005, McMillan 1992) for SAT.

2.12.5.1 CP-net-based constrained optimisation

COPs combine the ordering induced by the preferences with the constraints representation (Boutilier et al. 2004*b*). Thus, the first optimal (regarding a CP-net *N*)

assignment in a CSP search tree can be found by the CSP search algorithm that instantiate variables in an ordering compatible with the variable and value ordering induced by N. To find more non-dominated solutions, we basically need to perform comparisons between assignments, see e.g., (Boutilier et al. 2004*b*, Wilson 2004*a*, Wilson 2006, Wilson 2011).

One ground breaking work in CP-net-based COPs was achieved by Boutilier et al. (Boutilier et al. 2004*b*) by specifying a particular algorithm for solving the COPs and proving its correctness. The authors also demonstrated that, in some sense, finding one optimal solution is no harder than solving the corresponding CSP. The work showed that non-dominated solutions can be enumerated exhaustively using a CSP search that adopts a variable instantiation order that is compatible (topologically) with the CP-net structure, instead of using standard heuristics (Meisels et al. 1997, Haralick & Elliott 1980, Dechter & Pearl 1987, Sadeh & Fox 2003, Topaloglu & Ozkarahan 2004).

In (Boutilier et al. 2004*b*), COPs are solved by generating assignments in non-increasing order of preference, regarding variable and value instantiation, that is, an order consistent with the partial order over induced outcomes, until the first feasible outcome is found. Hence, the latter is a non-dominated solution. More precisely, during the search process, once constraint propagation took place at some node of the search tree and an assignment had to be made, the algorithm selects one variable and assigns to it the most preferred value among those that remain in the variable's domain. The variable and value selection is performed with respect to the CP-net's structure as variables are instantiated and values are assigned to the variables in a top-down manner according to a topological ordering of the CP-net. The values for a variable X to be instantiated are considered according to the preferential ordering induced by the assignment to the parents of X (with regards to CP-net N). The algorithm, called *Search-CP* starts with an empty set of solutions. At each stage, every assignment newly created is considered as a new candidate and it is tested against all the solutions generated up to that point.

Every time the optimality of an assignment needs to be checked, *Search-CP* perform a number of dominance tests that is equal to the number of optimal outcomes already computed at most. The new assignment is added to the set of optimal solutions if there is no existing solution that dominates it. The *Search-CP* algorithm outputs a set of non-dominated (feasible) solutions.

One particular property of the approach of (Boutilier et al. 2004*b*) is that the set of solutions never shrinks as outcomes are considered in a preferentially non-increasing order.

2.12.5.2 Hard and optimality constraints-based constrained optimisation

The algorithms that are reasoning with CP-nets benefit from the efficient test of optimality of CP-nets and try to avoid the complexity of the dominance test which is in P-SPACE (Goldsmith et al. 2008). The work in (Prestwich et al. 2005) is one example where one of the contributions is a technique that avoids dominance testing in some cases, with regards to the approach introduced in (Boutilier et al. 2004*b*).

The idea presented in (Prestwich et al. 2005) is to encode the CP-net statements of a given CP-net N into a set of hard constraints which allows for finding the optimal outcomes through solving a CSP including these hard constraints (called *optimality constraints*). The solutions of the obtained CSP are also the optimal solutions of the CP-net N. The authors identified two sets of constraints: hard constraints that are related to the CSP, and the optimality constraints that are related to the CP-net. A solution that satisfies the first set of constraints is called *feasible* and the solution which satisfies the second set of constraints is called *Pareto optimal*. The proposed algorithm is called *Hard-Pareto*. It aims at finding the feasible Pareto optimal outcomes. *Hard-Pareto* finds all feasible Pareto optimals by finding the outcomes which are both feasible and optimal in the CP-net. If there are optimals for the CP-net (regarding the optimality constraints) and they are all feasible (regarding the hard constraints), then there are no other feasible Pareto optimals and thus the algorithm may stop. Otherwise, *Hard-Pareto* must perform dominance testing over the feasible outcomes.

Hard-Pareto offers computational advantages over *Search-CP*. Unlike the latter, *Hard-Pareto* avoids dominance tests when there are no feasible solutions and when the set of optimal solutions coincides with the solutions of a CSP built by adding optimality constraints to the set of hard constraints. Unlike the approach presented in (Boutilier et al. 2004*b*) (i.e., *Search-CP*), the proposed algorithm also works even with cyclic CP-nets. In addition, the algorithm is not tied to CP-nets, but can work with any preference formalism which produces a pre-order over the outcomes (Prestwich et al. 2005).

2.12.5.3 Constrained CP-net-based constrained optimisation

There are situations where both *Search-CP* and *Hard-Pareto* can be computationally too expensive. For instance, there are RSs which require prompt feedback and so fast algorithms (Jannach et al. 2011). In (Prestwich et al. 2005), authors propose a method to find approximately optimal outcomes. They dealt with CP-nets-based constrained optimisation by giving a different semantics for a constrained CP-net (Prestwich et al. 2004).

Definition 5. *A constrained CP-net is a CP-net plus some constraints on subsets of its variables (additional constraints on assignments). A constrained CP-net is then denoted by a pair (N, C) where N is a set of conditional preference statements defining a CP-net and C is a set of constraints.*

Definition 6. *Given a constrained CP-net (N, C), outcome α is better than outcome β if and only if there is a chain of flips from α to β, where each flip is worsening for N and each outcome in the chain satisfies C.*

A worsening flip is defined very similarly to how it is defined in a regular (unconstrained) CP-net in Section 2.6.3 of this chapter. Given a constrained CP-net (*N*, *C*), where *N* is a CP-net and *C* is a set of hard constraints, an outcome is said to be approximately optimal if and only if no other outcome is approximately better. The method outputs a set of feasible outcomes that are undominated by one-flip-away outcomes.

The construction of the optimality constraints for constrained CP-nets can be adapted to work with soft constraints. Soft constraints (Rossi et al. 2006) extend the classical constraint formalism to model preferences in a quantitative way, by expressing several degrees of satisfaction (that can be either totally or partially ordered). Let C be a set of constraints and C_{opt} a set of optimality constraints. To find an optimal outcome, we need to find the optimal solutions for C that are also feasible for C_{opt}.

They also propose a technique which returns the best outcomes w.r.t. both constraints and preferences even if there are no feasible Pareto optimals. The technique is based on a reformulation of the problem of finding optimal outcomes of a (cyclic) constrained CP-net as a Weighted Constraint Satisfaction Problem (WCSP). A WCSP is a set of variables plus a set of constraints where each constraint has a weight representing the cost of violating it. An optimal solution of a WCSP is an assignment of values to all the variables which minimizes the sum of the costs of the violated constraints. The WCSP corresponding to a constrained CP-net is constructed in such a way that more desirable outcomes have lower weight, and a WCSP solver can be used to find the most desirable outcomes.

2.12.5.4 Constrained FCP-net-based constrained optimisation

(Gavanelli & Pini 2008*b*) introduces a new formalism, called constrained *FCP-net*. Such a formalism extends the classical constrained CP-net formalism, by considering, besides hard constraints and the qualitative aspect of the CP-net, a quantitative aspect as well, given by an objective function, that may relate some of the variables of the CP-net. The quantitative aspect is used to break ties in case the CP-net, which can be cyclic or acyclic, is unable, alone, to select one (or more) preferred outcomes.

2.12.5.5 CP-net-based formulation for constrained optimisation

Authors in (Boerkoel et al. 2010) describe and assess empirically a parameterized formulation for decoupled constrained optimisation problems that subsumes the state of art algorithm of Boutilier et al. in (Boutilier et al. 2004*b*), representing a wider family of alternative algorithms. They define three approaches including two distinct parametrisations that result in two algorithmic variations and a hybrid approach interleaving the two first approaches.

- The first is to solve the underlying CSP and enumerate all feasible assignments. They apply ordering queries to compare those assignments in order to find a non-dominated assignment. In fact, they use the CP-net ordering query: if α is orderable over β, then β cannot be strictly preferred to α; thus, they conclude α is not dominated by β.

- The second is to generate assignments in descending order of preference until the first feasible assignment is found. The assignments are generated in an order that is consistent with the partial order induced from the CP-net. Thus, the first feasible assignment should be non-dominated.

- They also hybridize the two approaches by interleaving constraint propagation and preferential reasoning. Thus, once constraint propagation takes place and an assignment needs to be made, the interleaved approach selects the most preferred assignment with regards to the CP-net (in a way similar to variable and value selection in (Boutilier et al. 2004*b*)). This approach is similar to the constrained optimisation approach in (Boutilier et al. 2004*b*) except the fact that the interleaved approach terminates after finding the first non-dominated solution.

They empirically demonstrated that the first and second approaches within their parametrisation are dominated by the interleaved approach similar to the approach (Boutilier et al. 2004*b*).

2.12.5.6 Polynomial constrained optimisation for partial acyclic CP-net

In (Purrington & Durfee 2008), the authors present a polynomial-time backward sweep algorithm that finds optimal outcomes for a specialized class of partial acyclic CP-net structures. This work bears some resemblance to the problem of CP-nets-based constrained optimisation addressed in (Boutilier et al. 2004*b*) in the sense that external evidence, that they use in their work, can be viewed as a hard constraint placed on the preferential optimisation problem. However, the constraints considered in that work are richer than simply fixing the values of some variables, and the

resulting optimisation problem becomes quite different. With the addition of richer constraints, the problem naturally takes on the character (and complexity) of traditional constraint processing, and solution techniques involve some variety of pruned backtracking search.

2.12.5.7 Comparative preference theories-based constrained optimisation

Wilson studied the task of finding optimal outcomes for COPs when preferences are expressed using conditional preference theories (Wilson 2004*a*, Wilson 2011), denoted by *cp-theories*. He suggested three ways of gathering optimal solutions by merging the CSP with cp-theories.

One way is to consider the preference statements as constraints and then join them to the set of constraints to form a CSP that can be solved by a CSP solver. A number of optimal solutions can be obtained. We should notice that for this method, a feasible solution is also an optimal one since preferences were regarded as constraints that were satisfied. The resulting CSP might be infeasible.

A second alternative is to enumerate all optimal solutions by using, similarly to (Boutilier et al. 2004*b*), a search tree which is chosen to be compatible with the set of preferences. Methods for finding such search trees have been developed in Section 4 of (Wilson 2011). As we mentioned above in this section, comparing solutions is a very hard problem (Goldsmith et al. 2005) which makes this alternative not always feasible particularly for large problems.

A third option offered by Wilson (Wilson 2009*a*) is to keep the same optimisation scheme of the second alternative but adopt a polynomial dominance testing (Wilson 2009*a*) instead of an exact method with expensive dominance testing (Goldsmith et al. 2005). This alternative yields some, but not usually all, optimal solutions.

2.13 Conclusion

Conversational RSs take place in the intersection of RSs research, dialogue system and preference handling. In this chapter, we have presented the background material and literature review related to conversational RSs and preference handling, which are the two primary research domains of our thesis work. Thus, we presented the relevance of preferences in decision theory and their handling in DSSs, then we surveyed most of the preferences-based systems. These personalized preference-based systems provide challenges for more elaborate comparative preference languages that we overview by presenting them and describing their mechanisms of representation and reasoning.

Conversational RSs can be considered as RSs that employ specific dialogue strategies and preference elicitation methods adapted to a human-like conversations. We reviewed most of the existing conversational approaches and discussed their dialogue strategies. Preference approaches also contributed to the development and expansion of constrained optimisation approaches. We gave a review of the main constrained optimisation strategies that are based on qualitative preference approaches.

3

Dominance for Comparative Preferences

3.1 Introduction

Many AI applications such as planning and scheduling call for techniques for representing and reasoning about qualitative preferences over a set of alternatives (Oster et al. 2011, Santhanam et al. 2011). In such settings, there is often a need to choose alternatives that are most preferred among the set of alternatives with respect to a set of user preferences. In order to identify this set, there is a need for a procedure which determines that one alternative A_1 is preferred over another alternative A_2 or vice versa (or neither) with respect to the user's preferences. In Section 3.2 we present the concept of the preference relations and their properties. In Section 3.3, we present the comparative preference theories introduced in (Wilson 2009*b*). The semantics behind the dominance relation are presented in Section 3.4. The main procedures behind the dominance procedure are described in Section 3.5.

3.2 Preference Relations

Preference relations (Dushnik & Miller 1941, Luce 1956, Scott & Suppes 1958*b*) are among the fundamental structures on which decision aiding models rely. They are used to filter out the dominated outcomes (a dominated outcome is any option for

which there is at least one other outcome in the results that is preferable with regards to the relation).

3.2.1 Preference relations properties

A preference relation is a binary relation between possible outcomes.

Definition 7. *Given an outcome space Ω and two different outcomes o and o', we say that $o \succcurlyeq o'$ if and only if o is preferred (or indifferent) over o'. We say $o \sim o'$ if and only if the decision maker is indifferent between the two outcomes. We call the first relation a preference relation and we call the second relation an indifference relation.*

When ordering, we consider a set of objects (in our case, outcomes) ordered through different ordering types. Below, we review a few concepts to provide the appropriate background in understanding the various preference orderings that are representable. Next, we list some standard properties of binary relations that are useful in classifying preference relations (Woronowicz & Zalewska 2004, Roubens 1989, Hansson & Grüne-Yanoff 2011, Joseph et al. 2007). A binary relation R over a set Ω is called:

- reflexive, if, $\forall a \in \Omega$, (aRa),
- irreflexive, if, $\forall a \in \Omega$, $\neg(aRa)$,
- symmetric, if, $\forall a, b \in \Omega$, $(aRb) \implies (bRa)$,
- asymmetric, if, $\forall a, b \in \Omega$, $(aRb) \implies \neg(bRa)$,
- antisymmetric, if, $\forall a, b \in \Omega$, $(aRb) \wedge (bRa) \implies (a = b)$,
- transitive, if, $\forall a, b, c \in \Omega$, $(aRb) \wedge (bRc) \implies (aRc)$,
- complete, if, $\forall a, b \in \Omega$, $(aRb) \vee (bRa)$,

The above properties are not independent. For instance, asymmetry implies irreflexivity, while irreflexivity and transitivity imply asymmetry.

Via the composition of such properties, the relation R induces different ordering classes. We give the definition for some useful kinds of ordering here.

Based on its properties, a preference relation R is characterized as follows:

- A binary relation is a *pre-order* or *quasi order*, if it is reflexive and transitive. If in addition, it is antisymmetric then it is a *partial order*.
- A binary relation is a *strict partial order* (or irreflexive partial order), if it is irreflexive, asymmetric and transitive.

- A binary relation is a *total order*, if it is a partial order and it is also complete. If a preference relation R is a total order, any two outcomes are mutually comparable under R.

- A binary relation is a *weak order*, if it is a complete pre-order.

Given a pre-order $\succeq$, we can define the induced strict order $\succ$ as follows: $o \succ o'$ if and only if $o \succeq o'$, but $o' \not\succeq o$. We define relation $\sim$ by $o \sim o'$ if and only if $o \succeq o'$, and $o' \succeq o$, that is, o and o' are equally preferred. Then, $\succ$ is a strict partial order and $\sim$ is an equivalence relation, i.e., a reflexive, symmetric and transitive relation.

3.2.2 Preference relations application

Many real-life problems call for identifying the best possible set of solutions. In order to solve such problems, techniques for both quantitative and qualitative preference representation and reasoning over a set of attributes have been extensively studied in the literature, e.g., (Benferhat et al. 2001, Brewka 2002, Brewka, Niemelä & Syrjänen 2002, Brewka et al. 2003, Balduccini & Mellarkod 2003, Boutilier et al. 2004*a*, Boutilier et al. 2004*b*, Brewka, Benferhat & Berre 2004, Wilson 2004*b*, Brafman, Domshlak & Shimony 2006, Kärger et al. 2008, Costantini & Formisano 2009, Bouveret et al. 2009*a*, Wilson 2011).

The preference relation is used to compare two solutions in terms of preferences over attributes of those solutions. The preference relation is also used by algorithms that identify the set of most preferred solutions. A solution is said to be optimal if and only if there is no other solution in the set of feasible solutions that strictly dominates it, i.e., a solution α strictly dominates β if and only if α dominates β and β does not dominate α. The set of solutions which are non-dominant to each other is called the non-dominated set.

Research has been performed on multi-attribute decision theory; a considerable part of this research has focused on reasoning with quantitative preferences (Fishburn 1967, Fishburn 1970, Keeney & Raiffa 1993, Fishburn 1999). However, in many applications it is more natural for users to express preferences in qualitative terms (Doyle & Thomason 1999, Doyle & McGeachie 2003, Dubois et al. 2002, Santhanam et al. 2011). We will be talking about dominance relations for qualitative preferences. Preference reasoning engines define their dominance testing strategies based on particular semantics. For instance, these semantics may give an indication about how relatively weak or strong the dominance might be as we discuss in Section 4.9 in Chapter 4. These strategies are well studied in the AI literature (Boutilier et al. 2004*a*, Wilson 2004*b*, Wilson 2006, Brafman, Domshlak & Shimony 2006, Santhanam et al. 2008, Wilson 2009*b*, Santhanam et al. 2010*b*, Santhanam et al. 2010*a*). Dominance properties (e.g., weak and strong dominance)

along with a number of optimality notions in the context of CP-nets were studied in (Brafman & Dimopoulos 2004*b*).

The preference relation is relevant in many applications where we need to know whether a solution is (among) the best, in particular when there is a large number of outcomes that might be feasible within a problem (Boutilier et al. 1997, Boutilier et al. 2004*a*, Brafman, Domshlak & Shimony 2006). Modifying the dominance relation helps the number of optimal solutions be controlled in the sense that the size of the optimal set can shrink or expand. In fact, to stimulate the dominance relation to be more or less strict makes it hard or easy to find non-dominated solutions. Thus, dominance relation tuning allows to have better control of the set of solutions.

3.3 A Comparative Preference Language

Wilson presented comparative preference theories which involve preference statements of the form $p > q||T$ where p is an assignment to a set of variables P, q is an assignment to set of variables Q, and T is a set of variables (Wilson 2009*b*). Such a statement expresses a preference for an assignment p over another assignment q with variables T held constant. In other words this statement expresses the preference of any complete assignment α which extends p over another complete assignment β which extends q, when α agrees with β on variables T.

Formally, the semantics of this statement is also given by the relation Γ^* which is defined to be the set of pairs (α, β) of outcomes such that α extends p, and β extends q, and α and β agree on T: α(T) = β(T). Given that p and q do agree on common variables in T, Wilson (Wilson 2009*b*) assumed that $P \cap T = \emptyset$ and $Q \cap T = \emptyset$.

We are especially interested in the statements where $P = Q$. The statement can then be written as $us > us'||T$ where U, S and T are disjoint sets of variables, and $u \in \underline{U}$, and s and s' are assignments to S which differ on each variable: $s(Z) \neq s'(Z)$ for all $Z \in S$.

3.3.1 Conditional preference theories-like statements

A logical and qualitative framework for preference elicitation was introduced in (Wilson 2004*b*). It is also presented in Section 2.7 in Chapter 2. It consists of conditional preference rules that are able to represent the user's preferences.

A set of these rules, which is denoted by the term *cp-theory*, have the form $u : x > x'[W]$, for each rule, $(W \subseteq (V \setminus (U \cup X)))$ which means that given an assignment u for a set of variables U, we prefer value x to value x' for variable X, as long as variables outside of W are held equal.

In comparative preference theories, a cp-theory statement $u : x > x'[W]$ is

equivalent to statement $us > us'||T$ when we set $S = \{X\}$, $x = s$, $x' = s'$ and $T = V \setminus (U \cup X \cup W)$.

3.3.2 CP-nets-like statements

A CP-net, which is described in Section 2.6.3 in Chapter 2, consists of a directed graph in which nodes represent variables (over a given domain) and edges express preference links between them. Let τ be a CP-net over variables V. Each variable $X \in V$ is represented by a node in CP-net τ, the node is associated with a conditional preference table that maps all possible assignments of X to the parents of X (denoted by $Pa(X)$) to a total order over $\underline{X}$. $Pa(X)$ are instantiated before X in CP-net and have a direct edge to the node that contains X. Let μ be the conditional preference table corresponding to τ. CP-nets can be expressed in terms of comparative preference theories. In particular, $us > us'||T$ is regarded as a CP-net statement when we set $S = \{X\}$, $x = s$, $x' = s'$ and $T = V \setminus (U \cup X)$. Notice that CP-nets can also be expressed as conditional preference theories, using statements $u : x > x'[W]$ with $W = \emptyset$. This was explained and shown with more details in Section 3.1 in (Wilson 2011).

3.3.3 TCP-nets-like statements

TCP-nets (Brafman & Domshlak 2002, Brafman et al. 2003, Brafman, Domshlak & Shimony 2006) are CP-nets with additional importance statements between variables. TCP-nets preference statements can be expressed in terms of comparative preference theories. For TCP-nets, the preference statement $us > us'||T$ is regarded as a TCP-net statement when we set $S = \{X_1, X_2\}$, and $T = V \setminus (U \cup \{X_1, X_2\})$. In general, ceteris paribus preferences are represented by comparative preference statements $us > us'||T$ with $T = V \setminus (U \cup S)$. Notice that TCP-nets can also be expressed as conditional preference theories. This was explained and shown in detail in Section 3.2 in (Wilson 2011).

3.4 Total Pre-orders-Based Dominance: Formal Semantics

Let $\mathcal{M}$ be a set of models and $\mathcal{L}$ be a preference language. Let $\models$ be a relation between $\mathcal{M}$ and $\mathcal{L}$. A model $M \in \mathcal{M}$ can represent an ordering associated with a user. For a model $M \in \mathcal{M}$ and $\Gamma \in \mathcal{L}$, we interpret $M \models \Gamma$ (or M satisfies Γ) to mean that Γ holds for the preferences of M. A model M satisfies Γ if it satisfies every element of Γ.

Let us consider the orderings on outcomes which hold for every model M satisfying statements in Γ. Formally, we can define the relation $\succcurlyeq_\Gamma$ on outcomes as follows: $\alpha \succcurlyeq_\Gamma \beta$ if and only if $\alpha \succcurlyeq_M \beta$ for all M satisfying (every member of) Γ.

One meaning of $\alpha \succcurlyeq_\Gamma \beta$ is that every user (represented by a model $M \in \mathcal{M}$) who agrees with Γ considers that outcome α is at least as desirable as outcome β (assuming this particular model of users (i.e., $\mathcal{M}$)). It is possible that we also have $\beta \succcurlyeq_\Gamma \alpha$, in which case every user considers that α and β are equally desirable. We define the relation $\succ_\Gamma$ to be the strict part of $\succcurlyeq$, so that $\alpha \succ_\Gamma \beta$ if and only if $\alpha \succcurlyeq_\Gamma \beta$ and $\beta \not\succcurlyeq_\Gamma \alpha$. Relation $\succ_\Gamma$ is irreflexive and transitive. We say that *given* Γ, α *strictly dominates* β, if $\alpha \succ_\Gamma \beta$, that is, if all users (represented by models M in $\mathcal{M}$) agreeing with Γ regard α as at least as preferable as β (i.e., $\alpha \succcurlyeq_M \beta$), and at least one such user regards α as strictly preferable to β (i.e., $\beta \not\succcurlyeq_M \alpha$).

Dominance computation is one of the most fundamental queries in preference representation formalisms. Besides, dominance testing is computationally intensive in general. Thus, different preference formalisms brought several algorithms which define specific dominance relations and adapted ways to compute these relations. In this section, we talk about the semantics used by formalisms like CP-nets in order to compute the dominance relation.

Given a language $\mathcal{L}$ whose statements express the user's preferences and a set of models $\mathcal{M}$, it is possible to define different semantics that deal with the concept of dominance.

3.4.1 Total pre-orders-based semantics

Dominance semantics between two different outcomes α and β can be based on total pre-orders, e.g., (Boutilier et al. 2004*a*, Santhanam et al. 2010*a*).

Definition 8. *Let $\succcurlyeq$ be a total pre-order and φ be a comparative preference statement written as $p > q||T$ where p is an assignment to a set of variables P, q is an assignment to set of variables Q, and T is a set of variables. $\succcurlyeq$ satisfies φ if and only if $\alpha \succcurlyeq \beta$ for all alternatives α and $\beta \in \underline{V}$ such that: (i) α extends p; (ii) β extends q; (iii) α and β agree on variables T (i.e., $\alpha(T) = \beta(T)$.) We then say that φ directly implies $\alpha \succcurlyeq \beta$. For a set of comparative preferences statements $\Gamma \in \mathcal{L}$, we say $\succcurlyeq$ satisfies Γ if and only if $\succcurlyeq$ satisfies every element of Γ. $\succcurlyeq$ is also said to be a model of Γ.*

Definition 9. *For $\Gamma \in \mathcal{L}$, $\varphi \in \mathcal{L}$ and $\succcurlyeq$ is a total pre-order, we define the semantic entailment relation by $\Gamma \models \varphi$ if and only if $\succcurlyeq \models \varphi$ for all $\succcurlyeq$ such that $\succcurlyeq \models \Gamma$. Let α and β be two different outcomes $\in \underline{V}$. For α and β, we also say that $\Gamma \models (\alpha, \beta)$ if $\alpha \succcurlyeq \beta$ holds for all models $\succcurlyeq$ of Γ.*

An example of a total pre-order-based dominance is brought by authors in (Brafman & Dimopoulos 2004*a*) where preferences are expressed as a CP-net. The authors state

that confirming that an outcome α dominates another outcome requires to prove that all total pre-orders that satisfy the set of preference statements (i.e., a CP-net) order α before β. Boutilier et al. also prove that a sequence of improving flips from β to α, which are presented in Section 2.6.3 in Chapter 2, yielded a proof that α is preferred over β in all total pre-orders satisfying the set of preferences (Boutilier et al. 2004*a*).

3.4.2 CP-tree-based semantics

In this section, we will present the concept of a cp-tree, which was introduced in (Wilson 2009*b*), its structure and its characteristics. Then, we will present dominance semantics and computation based on cp-trees. The definitions stated in this section are reproduced from (Wilson 2009*b*).

3.4.2.1 Description of a cp-tree

A cp-tree is a directed rooted tree, where edges are directed away from a root node so that all nodes apart from the root node have a unique parent node. Associated with each node N in the tree is a set of variables Y_N. The maximum number of variables in Y_N is equal to γ.

Y_N is instantiated with a different assignment in each of the node's children. The node has also an associated weak order $\succeq_N$ of the values of Y_N. This weak order depends on the values taken by the parents of that node, that is $Y_{N'}$ where N' is the parent of N.

Thus, cp-trees represent a tree that links nodes, represented each by a structure called "cp-node". This tree represents a form of lexicographic order where the importance ordering on cp-nodes, or variables in cp-nodes, can depend on more important cp-nodes and their assigned values, that is values in $\underline{Y_{N'}}$, as can the value orderings in $\underline{Y_N}$.

Definition 10. *We define a cp-node N (usually abbreviated to just "node") to be a tuple $\langle A_N, a_N, Y_N, \succeq_N \rangle$, where $A_N \subseteq V$ is a set of variables, $a_N \in \underline{A_N}$ is an assignment to those variables, $Y_N \subseteq \{V - A_N\}$ is a non-empty set of variables; $\succeq_N$ is a weak order on the set $\underline{Y_N}$ of values of Y_N which is not equal to the trivial full relation on $\underline{Y}$; so there exists some $y, y' \in \underline{Y}$ with $y \not\succeq_N y'$.*

Definition 11. *We define a cp-tree σ to be a directed tree that links cp-nodes to each other through edges that are directed away from a root node. Every cp-node has a unique parent, except the root, which is the first instantiated node and the ancestor of all cp-nodes.*

It is assumed that the weak orders $\succeq_N$ associated with each node of the cp-tree σ do not allow $\succeq_N$-equivalence between values in $\underline{Y_N}$ in order to ensure that the associated

ordering on outcomes is transitive. To do so, the weak order $\succeq_N$ in every node of σ needs to obey the following condition: if there exists a child of node N associated with instantiation $Y_N = y$, then y is not $\succeq_N$-equivalent to any other value of Y so that y $\succeq_N$ y' $\succeq_N$ y only if $y' = y$.

Definition 12. *We define a pre-ordered search tree δ to be a cp-tree with $\gamma = 1$. δ is abbreviated to a pos-tree.*

Example 4. *Let $V = \{X, Y, Z\}$ be a set of variables whose values domains are $\underline{X} = \{x1, x2\}$, $\underline{Y} = \{y1, y2\}$ and $\underline{Z} = \{z1, z2\}$ respectively. Figure 3.1 represents an example of a cp-tree with $\gamma = 1$. Each node in the cp-tree depicted in Figure 3.1 is labeled with a number of variables that varies from 1 to γ. The root is labeled by the most important variable, X in this example. Each node is also associated with a preference ordering of the values of the variable which is discussed in Section 3.4.2.2. We can see the total pre-order of the outcomes below the cp-tree in 3.1. This ordering is generated as stated in Section 3.4.2.4.*

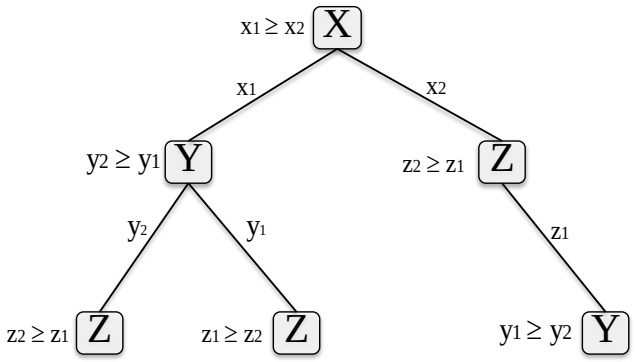


Figure 3.1: A cp-tree σ, along with its associated ordering $\succcurlyeq_\sigma$ on outcomes, with $\gamma = 1$ (i.e., with at most one variable associated with a node)

3.4.2.2 CP-tree: variable and value orderings

Let $N1$ be the root node of a cp-tree σ, and Y_{N1} be a set of variables associated to $N1$. Node $N1$ has $\langle A_{N1}, a_{N1}, Y_{N1}, \succeq_{N1}\rangle$ where $A_{N1} = \emptyset$, $a_{N1} = \top$ and $\succeq_{N1}$ weakly orders the values in $\underline{Y_{N1}}$. Y_{N1} is instantiated with a value $y_{11} \in \underline{Y_{N1}}$ which is associated to the edge that links the current node to the second node $N2$. The set of variables Y_{N2}, associated with $N2$, is determined with regards to y_{11}. Another edge, linking $N1$ to another child node, is eventually created being associated with another value $y_{12} \in \underline{Y_{N1}}$. Node $N2$ has $\langle A_{N2}, a_{N2}, Y_{N2}, \succeq_{N2}\rangle$ where $A_{N2} = A_{N1} \cup Y_{N1}$, a_{N2}

is a_{N1} extended with assignment $Y_{N1} = y_{11}$. If Nk is a leaf node in the cp-tree then A_{Nk} is the union of sets of variables associated with all ancestors in the path from the root node to Nk. a_{Nk} consists of the combination of all assignments made on the path from the root to Nk.

For instance, in Figure 3.1 of Example 4, we can see that the root node (i.e., $\succeq_{N1}$) orders the values in $\underline{X} = \{x1, x2\}$ as $x1$ is preferred over $x2$. Then, after having been instantiated with $x1$, X was instantiated with $x2$ which was associated to the edge that links the most important variable (i.e., X) to the second most important variable given $X = x2$, which is Z. Given $Z = z1$, the third most important variable's values ordering states $y1$ is preferred over $y2$.

3.4.2.3 Comparing two outcomes

Given a cp-tree σ, the comparison of any two outcomes α and β is based on a particular type of node in σ: the *decisive* node (Wilson 2006, Wilson 2011).

Definition 13. *Let σ be a cp-tree. A node N is defined to be decisive for outcomes β and α if it is the deepest node (i.e., furthest from the root) which is both on the path to α and on the path to β, from the root node. For outcome α, define the path to α to be the path from the root which includes all nodes N such that α extends a_N.*

For all variables Y already instantiated in σ before reaching decisive node N, we have $\alpha(Y) = \beta(Y)$. The variable Y_N associated with node N is not assigned the same value in outcomes α and β (that is, $\alpha(Y_N) \neq \beta(Y_N)$). We also say that node N decides α and β. Node N decides which of the outcomes is preferred over the other regarding σ by comparing the two outcomes' values for the root node's associated variable Y_N (i.e., $\alpha(Y_N)$ and $\beta(Y_N)$) with respect to the ordering associated with node N. For instance, if $\alpha(Y_N) \succcurlyeq_r \beta(Y_N)$ then α is preferred over β.

As every cp-tree σ implements a kind of generalized lexicographic order, any two outcomes α and β are compared as follows: α and β are first compared on the value they have for the set of variables associated with the first most important node $N1$, using the relation $\succeq_{N1}$, if one of them has better value then we conclude that outcome is preferred over the other one regarding σ; node N is said to be *decisive*. If not then the two outcomes are compared based on the next most important node whose importance is determined regarding the value both outcomes have for the set of variables associated with the previous node. This comparison continues through the following nodes in a lexicographic fashion until a decisive node is found; otherwise α and β are said to be equivalent.

Definition 14. *Let σ be a cp-tree. The associated relation $\succeq_\sigma$ on outcomes is defined as follows: For outcomes α and $\beta \in \underline{V}$ outcomes, we define $\alpha \succeq_\sigma \beta$ to hold if and only if $\alpha(Y_N) \succeq_N \beta(Y_N)$, where N is the node which decides α and β.*

In Example 4, outcome $\{x1y2z1\}$ is preferred over outcome $\{x1y1z1\}$. The second node on the left to which variable Y is associated with value $y2$ is preferred over value $y1$.

3.4.2.4 Generation of a compatible ordering of outcomes

The cp-tree ordering is generated by instantiating sets of variables Y_N associated to nodes in an order that depends on the variables and values of the ancestor nodes.

To generate this, for each node N, starting from the root, we choose the child associated with the most preferred and uninstantiated value of the root Y_N. Given that value, we instantiate the next node with the most preferred and uninstantiated value. We keep instantiating sets of variables regarding the variable and value importance ordering till there is no more child node to instantiate in the path from the root to a leaf node; we then obtain an outcome α. α is at least as good as the next outcome to be instantiated by backtracking to the current node and instantiating the associated set of variables to the next most preferred uninstantiated value. We continue instantiating the outcomes in an ordering that is compatible with the variable and value ordering implemented by the cp-tree.

According to Definition 14 in Section 3.4.2.3, $\succeq_\sigma$ is similar to a lexicographic ordering in that two outcomes are compared on the first set of variables (associated with a node) on which they differ. The definition implies immediately that $\succeq_\sigma$ is complete; it is easily shown to be transitive, and is hence a weak order. Examples of ordering are shown in Figures 3.1, 3.2 and 3.3.

Given a comparative preference statement φ written as $p > q||T$, φ^* is a set which represents all pairs of outcomes (α, β) such that: (i) α extends p; (ii) β extends q; (iii) α and β agree on variable T (i.e., $\alpha(T) = \beta(T)$.)

Given a set of comparative preference statements denoted by Γ, $\Gamma^* = \bigcup_{\varphi\in\Gamma} \varphi^*$. We say that cp-tree σ satisfies φ (respectively, Γ) if and only if $\succeq_\sigma$ satisfies φ (respectively, Γ) i.e., $\succeq_\sigma$ extends φ^* (respectively, Γ^*), that is, $(\alpha, \beta) \in \varphi^* \Rightarrow \alpha \succeq_\sigma \beta$ (respectively, $(\alpha, \beta) \in \Gamma^* \Rightarrow \alpha \succeq_\sigma \beta$).

Example 5. *Let us assume a student is looking for a suitable laptop. The student will express her preferences over four laptop components which are represented by the following variables: Operating System(OS), Memory(M) (unit=megabyte), Central Processing Unit(CPU), Monitor(Mon) (unit=inch) and Graphics Card(GC). The values of these variables are shown in Table 3.1.*

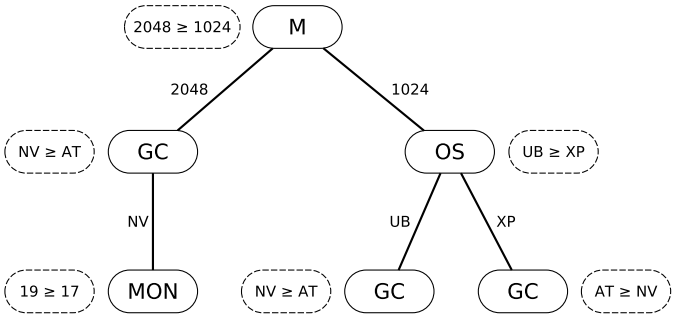
Regarding memory (M) which is the most important variable for the student, $2048MB$ *is preferred over* $1024MB$*. If the memory is equal to* $2048MB$*, the student prefers GC = NV over GC = AT. In the case when GC = NV, the student prefers Mon = 19 over Mon = 17. If the memory is equal to* $1024MB$*, she prefers OS = UB over OS = XP. When the*

Table 3.1: Laptop components.

Variables	Values
Operating System (OS)	{*XP, UBUNTU11.4 (UB)*}
Memory (M)	$\{1024, 2048\}$
Monitor (Mon)	$\{17, 19\}$
Graphics Card (GC)	{*ATI_Radeon_HD5000(AT), Nvidia_GeForce_MX440(NV)*}

operating system is fixed to be UBUNTU11.4 (i.e., OS = UB), she prefers GC = NV over GC = AT. In case the operating system is XP (i.e., OS = XP), she prefers GC = AT over GC = NV. Two possible ways of representing the student's preferences are represented by two cp-trees depicted in Figure 3.2 which represents a cp-tree (with $\gamma = 1$) and Figure 3.3 which represents another cp-tree (with $\gamma = 2$).

The difference between the second one (depicted in Figure 3.3) and the first one (depicted in Figure 3.2) is that when the memory is equal to $2048MB$, the student shall express her preferences over the combinations of values of two variables (i.e., GC and Mon) (in Figure 3.3) instead of expressing preferences over one variable (i.e., GC) and then expressing over the second variable (i.e., Mon) with regards to the value of the first variable (e.g., GC = NV). The total pre-orders generated by the cp-trees in Figure 3.2 and Figure 3.3 are the same.

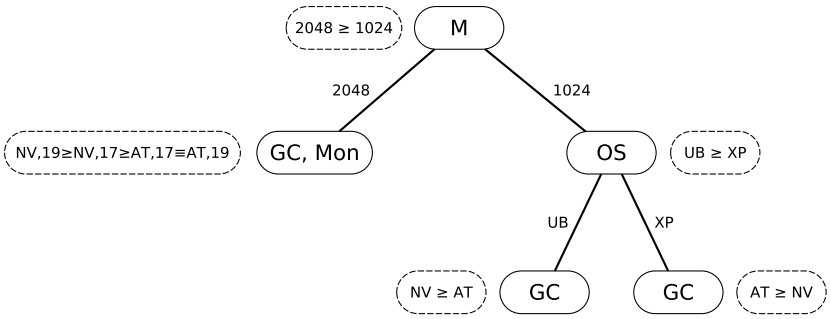


{2048,NV,19,XP} ≡ {2048,NV,19,UB} ≥ {2048,NV,17,XP} ≡ {2048,NV,17,UB}
≥ {2048,AT,17,XP} ≡ {2048,AT,17,UB} ≡ {2048,AT,19,XP} ≡ {2048,AT,19,UB}
≥ {1024,NV,19,UB} ≡ {1024,NV,17,UB} ≥ {1024,AT,19,UB} ≡ {1024,AT,17,UB}
≥ {1024,AT,19,XP} ≡ {1024,AT,17,XP} ≥ {1024,NV,19,XP} ≡ {1024,NV,17,XP}

Figure 3.2: A cp-tree σ, along with its associated ordering $\succcurlyeq_\sigma$ on laptop outcomes, with $\gamma = 1$ (i.e., with a single variable associated with a node)

3.5 CP-Tree-Based Preference Computation

Let Γ be a set of comparative preference statements. Let $\succeq_\Gamma$ be the associated preference relation of Γ on outcomes, $\mathcal{Y}$ be a set of subsets of variables Y ($Y \in \mathcal{Y}$). Let ψ,



{2048,NV,19,XP} ≡ {2048,NV,19,UB} ≥ {2048,NV,17,XP} ≡ {2048,NV,17,UB}
≥ {2048,AT,17,XP} ≡ {2048,AT,17,UB} ≡ {2048,AT,19,XP} ≡ {2048,AT,19,UB}
≥ {1024,NV,19,UB} ≡ {1024,NV,17,UB} ≥ {1024,AT,19,UB} ≡ {1024,AT,17,UB}
≥ {1024,AT,19,XP} ≡ {1024,AT,17,XP} ≥ {1024,NV,19,XP} ≡ {1024,NV,17,XP}

Figure 3.3: A cp-tree σ, along with its associated ordering $\succcurlyeq_\sigma$ on laptop outcomes, with $\gamma = 2$ (i.e., with at most two variables associated with a node)

of the form $\alpha \succeq \beta||\emptyset$, be a comparative preference statement saying α is preferred over β. thus, ψ* is represented by the pair of outcomes (α, β). The definitions are reproduced from (Wilson 2009*b*).

Definition 15. *Let α and β be two outcomes. α dominates β if and only if all possible cp-trees (every cp-tree represents a total pre-order) that satisfy Γ, prefer α over β. In other words, $\alpha \succeq_\Gamma \beta$ holds if every cp-tree σ that extends all preferences in Γ has α come before β.*

Definition 16. *Let $\mathcal{Y}$ be a set of non-empty subsets of V such that if $Y \in \mathcal{Y}$ and non-empty Y' is a subset of Y then $Y' \in \mathcal{Y}$. A $\mathcal{Y}$-cp-tree is defined to be a cp-tree σ such that for any node N of σ, we have $Y_N \in \mathcal{Y}$.*

Definition 17. *Γ entails ψ if and only if $\Gamma^* \models \psi^*$.*

Definition 18. *$\alpha \succeq_\Gamma \beta$ if and only Γ entails ψ.*

In (Wilson 2009*b*), the author shows that the $\mathcal{Y}$-cp-trees that satisfy Γ and also satisfy ψ* (i.e., (α, β)) map to particular sequences of sets in $\mathcal{Y}$, which are called *decisive sequences*. In fact, showing that Γ does not entail $\alpha \succeq \beta$ requires to show there is a $\mathcal{Y}$-cp-tree satisfying Γ but not ψ*. A $\mathcal{Y}$-cp-tree that satisfies Γ but not ψ* corresponds to a chain of subsets Y. More precisely, it maps to a decisive sequence of pickable subsets $Y \subset \mathcal{Y}$ ending with a decisive subset Y. A pickable subset Y corresponds to a node in the $\mathcal{Y}$-cp-tree that satisfies Γ as well as ψ. A decisive subset Y corresponds to a node in the $\mathcal{Y}$-cp-tree that satisfies Γ but not ψ.

Thus, the $\mathcal{Y}$-entailment of ψ by Γ can be determined by checking for the existence of a decisive sequence. A dominance testing algorithm was presented in (Wilson 2009*b*) and reported in Section 3.5 (see Algorithm 2).

The entailment algorithm

Let V be a set of n variables. Let U, S and T be disjoint subsets of variables included in V. Let $u \in \underline{U}$, $s \in \underline{S}$ and $s' \in \underline{S}$. Let α and β be two outcomes which agree with u on U (i.e., $\alpha(X) = \beta(X) = u(X)\ \forall\ X \in U$) and differ on S ($\alpha(X) = s(X) \neq \beta(X) = s'(X)\ \forall X \in S$). ψ, written as $\alpha \succeq \beta||\emptyset$ in the previous paragraph, can be written as $us \succeq us'||\emptyset$.

Let Y be a subset of variables ($Y \subset V$). Let $\underline{Y}$ be the set of possible values of the subset of variables Y. Let y, y_i and y_j be values of Y in $\underline{Y}$. Let a be an assignment to set of variables $A \subset (V \setminus Y)$. Relation $\sqsupset_a^Y$ on $\underline{Y}$ is defined to be the transitive closure of the set of pairs (y_i, y_j) of values of Y over all preference statements $p \geq q\ ||T$ in Γ such that a is compatible with p and q, and $y_i(Y \cap T) = y_j(Y \cap T)$ with y_i being compatible with p and y_j being compatible with q. (y_i, y_j) means y_i is preferred over y_j. A value y is $\sqsupset_a^Y$-equivalent to another value y' if $y \sqsupset_a^Y y'$ and $y' \sqsupset_a^Y y$.

Definition 19. *Given two outcomes us and us' and $Y \subseteq U$, If there is no value $y \in \underline{Y}$ which is $\sqsupset_a^Y$-equivalent to $us(Y)$ then Y is said to be pickable; otherwise Y is considered as not pickable.*

Definition 20. *Given two outcomes us and us' and $Y \cap U = \emptyset$, If there is at least one pair (y, y′) such that $y \not\sqsupset_a^Y y'$ with $y \models us$ and $y' \models us'$ then Y is said to be pickable and decisive (Y is called decisive for us and us'); otherwise Y is considered as not pickable.*

As shown in the previously, $\alpha \succcurlyeq_\Gamma \beta$ involves reasoning with cp-trees and the total pre-orders they generate. Algorithm 2 was presented in Section 5 of (Wilson 2009*b*). It is an entailment algorithm which checks whether or not $\alpha \succeq_\Gamma \beta$ by verifying that all weak orders generated by all cp-trees σ that satisfy Γ, agree with α being ordered before β as well (i.e., $\alpha \succeq_\sigma \beta$). The correctness of Algorithm 2 was stated in Theorem 1 in Section 5 in (Wilson 2009*b*). This algorithm looks for decisive sequence when checking if Γ implies $\alpha \succcurlyeq_\Gamma \beta$. The monotonicity property presented in Proposition 5 in (Wilson 2009*b*) allows Wilson's entailment algorithm to check the decisive sequence in polynomial time.

Given the outcomes us and us' and the set of input preferences Γ, the Entailment algorithm will determine whether Γ entails ψ (i.e., $us \succeq us'||\emptyset$). In other words, the algorithm checks whether us dominates us' or not.

In order to check if one outcome us dominates another outcome us' with respect to Γ, the algorithm iterates through subsets of variables in $\mathcal{Y}$ and checks if there does not exist a witness for the dominance not to hold (see Definition 15). First, it generates the set of all possible singleton subsets of variables in $\mathcal{Y}$. Then it navigates through those subsets to find out if there is some *decisive* subset of variables Y.

Algorithm 2 Does Γ entail ψ

input : Set of variables $Y_1, \ldots, Y_n$
A family $\mathcal{Y}$ of subsets of variables V which includes all singletons
A first outcome us
A second outcome us'
input : Set of variables $Y_1, \ldots, Y_n$
output: true if us dominates us'; false otherwise

```
12 for j ← 1 to n do
13     Let a_j be u restricted to Y_1 ∪ ..., Y_{j-1} (in particular, a_1 = ⊤);
       if there exists a set in 𝒴 which is pickable and decisive given a_j then
14         return false;
15     if there exists a set 𝒴 which is pickable given a_j then
16         let Y_j be any such set;
17     else
18         return true;
19 return true;
```

There is no particular order of subsets Y to be checked that is imposed by Wilson's algorithm. In fact, the monotonicity property, expressed by Proposition 5 in (Wilson 2009*b*), states, roughly speaking, that a set Y which is pickable remains pickable whatever the order according to which subsets Y are selected. This means that the search for a decisive sequence can be performed in a backtrack-free manner as we never need to undo the selection of a subset Y (already selected and checked).

The algorithm looks for a decisive sequence of subsets Y ($Y \subset \mathcal{Y}$). In (Wilson 2009*b*), the author defines a decisive sequence as a sequence of subsets Y that are *pickable*, this sequence ends by a *decisive* subset Y. Once found, the decisive sequence is considered as the proof that us does not dominate us'. In fact, Wilson shows in Proposition 4 of (Wilson 2009*b*) that a $\mathcal{Y}$-cp-tree σ_1 that satisfies Γ but does not agree with outcome us being preferred over outcome us' (i.e., $us \succcurlyeq_{\sigma_1} us$) corresponds to a decisive sequence. Then, if the algorithm finds a decisive subset Y, it declares that us does not dominate us'. Otherwise, us dominates us' as the algorithm is looking for a witness for non entailment but there was no proof for the existence of the $\mathcal{Y}$-cp-tree that does satisfy the set of preference statements Γ but does not agree with us being ordered before us'.

While iterating through the subsets $Y \subset \mathcal{Y}$, there are three possible cases:

- Y is *pickable* (and not decisive) so the algorithm extends assignment a with tuple $us(Y)$ (i.e., $a(Y) = us(Y)$) and eliminates Y from $\mathcal{Y}$.

- Y is *not pickable* so the algorithm eliminates it from $\mathcal{Y}$ with saving a copy of it in order to deal with afterwards, as soon as the assignment a is updated.
- Y is *decisive* so the algorithm concludes that us does not dominate us'.

Let us describe the entailment algorithm as a sequence of actions:

i) The algorithm generates $\mathcal{Y}$ and initializes the assignment $a = \top$.

ii) The algorithm navigates through all subsets Y of $\mathcal{Y}$. If Y is *decisive* then the algorithm stops and returns *False* (which says that us does not dominate us'). If Y is *pickable* then the algorithm updates assignment a, by extending it with assignment $u(Y)$. Then, the algorithm removes Y from $\mathcal{Y}$ and considers the very first subset that was considered as not pickable if there is any. In fact, this kind of subset is stored in a temporary memory zone that we call $temp_Y$ (e.g., a list). It looks through all variables in $temp_Y$ then it retrieves subsets from $\mathcal{Y}$ if there are any remaining subsets that were never checked (to be pickable or not). Every time assignment a is updated (i.e., extended with $us(Y)$), the algorithm first checks subsets Y in $temp_Y$. If Y is *pickable* then the algorithm updates assignment a and removes it from $temp_Y$. Otherwise (i.e., Y is *not pickable*), the next subset Y in $temp_Y$ is checked. When assignment a is updated, subsets Y, that were previously stored in $temp_Y$ as they were found *not pickable* in a previous iteration, are considered again (before the ones in $\mathcal{Y}$ that are not yet checked).

iii) The algorithm stops in two cases: 1) if it finds a *decisive* Y; 2) or if there is no *pickable* Y neither in $temp_Y$ nor in $\mathcal{Y}$. Indeed, the algorithm always updates $temp_Y$ with *not pickable* subsets Y when assignment a is updated (in case there is a subset Y which was found *pickable*). We can say that the algorithm returns *True* (which says that us dominates us') when assignment a is not updated anymore and all the remaining subsets Y were verified as *not pickable*.

3.6 Other Preferential Semantics

The problem of preference representation has been tackled in the literature with quite a different number of approaches. Given the incompleteness of available knowledge, we may need to use preferences and optimisation criteria to select the best candidate solutions according to both qualitative and quantitative measures.

Possibilistic logic is one formalism that may be used for encoding user preferences, since possibility measures can actually be viewed as rankings (on worlds or also objects) along an ordinal scale (Straccia 2008). One preliminary investigation of

the potentials of possibilistic logic in the representation and combination of preferences in decision analysis can be found in (Benferhat et al. 2001) which proposes a qualitative preference framework that allows the relative importance of preferences to be specified. When modeling preferences in possibilistic logic, the necessity measure κ associated with a classical formula ν in (ν, κ) is understood as the priority of ν rather than its certainty level (Benferhat et al. 2001). Possibilistic logic has been implemented in many ways, e.g., the POSLOG system (Dubois et al. 1990). The possibility measure was introduced by Zadeh (Zadeh 1999) and was greatly developed by Dubois, Prade and others (Dubois & Prade 1990). It deals well with default reasoning and counterfactual reasoning (Parsons 2006).

Ordered disjunctions are a recent, promising development in reasoning about preferences with logic programming which is subject of work by (Schaub & Wang 2001, Brewka 2002, Brewka, Niemelä & Syrjänen 2004, Brewka, Benferhat & Berre 2004), and others. Answer Set Programming (*ASP*) is a form of logic programming based on the answer set semantics (Gelfond & Lifschitz 1988*a*, Gelfond 2007), where solutions to a given problem are represented in terms of selected models (answer sets) of the corresponding logic program (Marek & Truszczynski 1998). Logic programming with ordered disjunction has been invented by Brewka (Brewka, Niemelä & Syrjänen 2004) as an extension of answer set programming to represent priority among literals and rules in an ASP program. Various forms of preferences have also been introduced in ASP (see (Delgrande et al. 2004)). Most of the proposed approaches are based on establishing priorities/preferences among rules.

In (Brewka 2002), the author introduces logic programs with ordered disjunction (denoted by *LPOD*) to represent preferences. It combines Qualitative Choice Logic (QCL) (Brewka, Benferhat & Berre 2002, Brewka, Benferhat & Berre 2004) and ASP. QCL is a propositional logic for representing alternative, ranked options for the solutions of the problem. QCL semantics is based on a preference relation among models. This preference framework is designed to represent preferences over alternatives and induces a complete pre-order over models. The semantics of LPOD is based on the notion of preferred answer sets.

In later papers (Brewka, Niemelä & Syrjänen 2002, Brewka, Niemelä & Syrjänen 2004), the authors present efficient techniques to enforce priorities and ordering relations among solutions of an ASP program. They introduce the notion of Pareto-preference and show that this criterion gives more intuitive results that the other criteria introduced in (Brewka 2002). The basic idea is to augment the syntax by some designated operator to form ordered disjunctions. Programs of this form (called logic programs with ordered disjunctions, or LPODs) can be evaluated in a standard way (presented in (Brewka 2002)) or with respect to different preferential semantics which take the occurrences of this new operator into account. A system called

Psmodels presented in (Brewka, Niemelä & Syrjänen 2004) serves as a computational engine for these semantics. In other related work (Brewka 2004*a*), Brewka proposes a language for answer set optimisation called *PLD*. The basic elements of PLD are rules which code context-dependent preferences over answer sets. More complex preference formulae are formed using different aggregation operators: sum, (ranked) set inclusion, (ranked) cardinality, Pareto, and lexicographic order.

A problem to be solved is represented as a LPOD and answer sets of the program are ranked, according to the degrees of satisfaction of ordered disjunctive rules. In this way a global ranking of answer sets is obtained. The following criteria have been proposed in (Brewka, Niemelä & Syrjänen 2002) to build this ranking: cardinality optimal criterion- maximizing a number of rules satisfied to the lowest degree, inclusion optimal criterion, based on set inclusion of the rules satisfied to the certain degree and Pareto optimal criterion favoring the answer set satisfying all ordered disjunctive rules not worse, than any other answer set does, and one rule strictly better.

Unfortunately, ordered disjunction appears to have problems with certain types of dynamic preferences, as well as to sometimes produce unintuitive results when there are conflicts among preferences in the program (Balduccini 2005).

In (Costantini & Formisano 2010), authors introduced Resourced ASP (denoted by *RASP*) so as to support declarative reasoning on consumption and production of resources. They have introduced the possibility of defining and using resources in ASP. In RASP, one can define resources with their amounts, where available resources can be used for producing other resources and the remaining amount, if any, can be used in a different way. In (Costantini & Formisano 2009), authors introduced *P-RASP* (RASP with Preferences) where it is possible to express preferences about which resources should be either consumed or produced. LPOD formalisms can be adapted to be able to solve RASP). Adaptation involves complex tasks. Along this line, instead of trying to adapt LPOD for application in RASP, authors in (Costantini & Formisano 2009) have chosen to provide an "autonomous" semantics which better reflects, in their opinion, the intuitive meaning that a programmer assigns to resources and quantities. They designed an approach to preferences specifically intended for the RASP context.

A preference criterion to compare answer sets is required in order to impose a preference order on the answer sets of an ASP program. Such a criterion should impose an order on the collection of answer sets by reflecting the preference degrees. Any criterion has to take into account that each rule determines a (partial) preference ordering on answer sets. In a sense, the criterion should aggregate/combine all local partial orders to obtain a global one. Fundamental techniques for combining preferences (seen as generic binary relations) can be found for instance in (Andréka

et al. 2002). Regarding the combination of preferences in logic programming, criteria are also given, for instance, in (Balduccini & Mellarkod 2003, Brewka 2004*a*, Brewka, Niemelä & Syrjänen 2004, Son & Pontelli 2006). Preference orderings in P-RASP are based on a subset of criteria among alternative possible choices.

A direct comparison between the preferential semantics of the languages described above and several others (e.g., (Kärger et al. 2008, Bouveret et al. 2009*a*)), and the semantics detailed in Section 3.4 can be a subject of future work.

3.7 Summary

Dominance testing represents a key operation when looking for the most preferred set of alternatives among a set of possible alternatives with regards to the user's preferences. Different formalisms have brought different preferences languages which show increasing expressiveness capabilities. Different dominance semantics and ways of computing dominance were explored in the literature, e.g., (Benferhat et al. 2001, Brewka, Niemelä & Syrjänen 2002, Brewka et al. 2003, Boutilier et al. 2004*a*, Boutilier et al. 2004*b*, Brewka, Benferhat & Berre 2004, Kärger et al. 2008, Costantini & Formisano 2009, Bouveret et al. 2009*a*, Wilson 2011). These semantics offer various ways of considering the dominance relation as we may need to tighten or extend the set of models that we consider in order to confirm dominance between two outcomes. Different ways of dominance computation were presented in the literature in an attempt to reduce the computational complexity of the dominance testing as it is a computationally hard problem in general with the standard CP-nets semantics (Goldsmith et al. 2005, Goldsmith et al. 2008). We presented the preference language and the dominance semantics and computation we are using in this thesis. We also presented other dominance approaches that adopt different semantics of the dominance relation. We described in detail the dominance testing algorithm that we have used in this dissertation.

4

Preferences Deduction for Conversational Recommender Systems

4.1 Introduction

In an era of overwhelming choices, *recommender systems* (RSs) are a new source of assistance, helping their users decide which goods, services or information to purchase or consume (Adomavicius & Tuzhilin 2005, Anand & Mobasher 2005, Bridge et al. 2006, Ricci et al. 2011*a*). RSs try to induce information about user preferences from data gathered either explicitly, for instance, in the form of product ratings, or implicitly by observing users' behaviour. They have been successfully used to recommend travel services, books, CD, financial services, insurance plans and news (Adomavicius & Tuzhilin 2005, Anand & Mobasher 2005, Bridge et al. 2006).

RSs aim at recommending the most suitable items to the user (Ricci et al. 2011*a*). However, it may happen that the recommended items proposed by the system do not match the users' needs as RSs might miss the users' preferences. Several cases of poor quality recommendations were described in (Zaslow 2002); the article also reported how affected users desperately tried to react and to change the deal and communicate their actual needs to the system but in vain. Examples of issues that might be raised are whether the system does understand why the user wants recommendations and whether recommenders have their own "personalities" to which users need to interact with, were discussed in (Zaslow 2002).

"Single-shot" RSs, which recommend a set of products to the user only once (i.e., in one shot), are one category of RSs that may experience some inappropriate recommendations since they are unlikely to guarantee that their first set of recommendations always satisfies the user. Figure 4.1 illustrates a single-shot recommendation scenario. Once the user's preferences collected (or elicited), the preference manager induces the preference model which is represented by ratings, features values, etc. Given a user inquiry (i.e., request) and the user's preferences, the request manager selects a set of recommendations (undominated options), retrieves the corresponding products from the database and presents them to the user. If one of the presented items suits the user, she selects and buys it. Otherwise, the user quits. Indeed, users are *rarely* satisfied with the first set of recommendations; they usually want to see more options and they exploit the initial recommendations to refine their preferences and articulate new requests (Bridge et al. 2006). Therefore, one approach to ensure that the system realizes what users need would be to generate a conversation between users and RSs as illustrated in Figure 4.2. This will be favorable for building or reinforcing the bridge between the two partners (i.e., the user and the RS). Figure 4.2 illustrates a conversational recommendation scenario. The difference with the single-shot recommendation scenario is that in the case where the user is not satisfied she can revise her request. Then, the preference manager captures the revision of the user request to update the elicited user's preference and so the user model. Therefore, given the updated user model plus the revised request, the request manager selects again a set of recommendations and retrieves their corresponding products from the database. If the user is satisfied with one of the items presented to her she selects and buys it. Otherwise, the user can revise her query and send it again to the system.

Starting initially with a set of products and initial preference information about the user, *conversational RSs* allow for such a conversation, and recognise that their users may be willing and able to provide more information on their constraints and preferences, over a short dialogue, thereby moving away from "single-shot" interaction. This is also an opportunity for the RSs to guide the user by asking questions, giving advice, displaying candidate products, and giving explanations (Reilly et al. 2004, McSherry 2005, Bridge et al. 2006, Ricci et al. 2006, Schmitt 2002*a*, Thompson et al. 2004*a*, Pu et al. 2006).

In Section 4.2 we describe in detail a kind of recommender system that inspired a major part of the work presented in this chapter. Then, we present the framework of preference dominance and its rationale in Section 4.3. This framework is illustrated through two instances that are described later in Section 4.4 and Section 4.5.

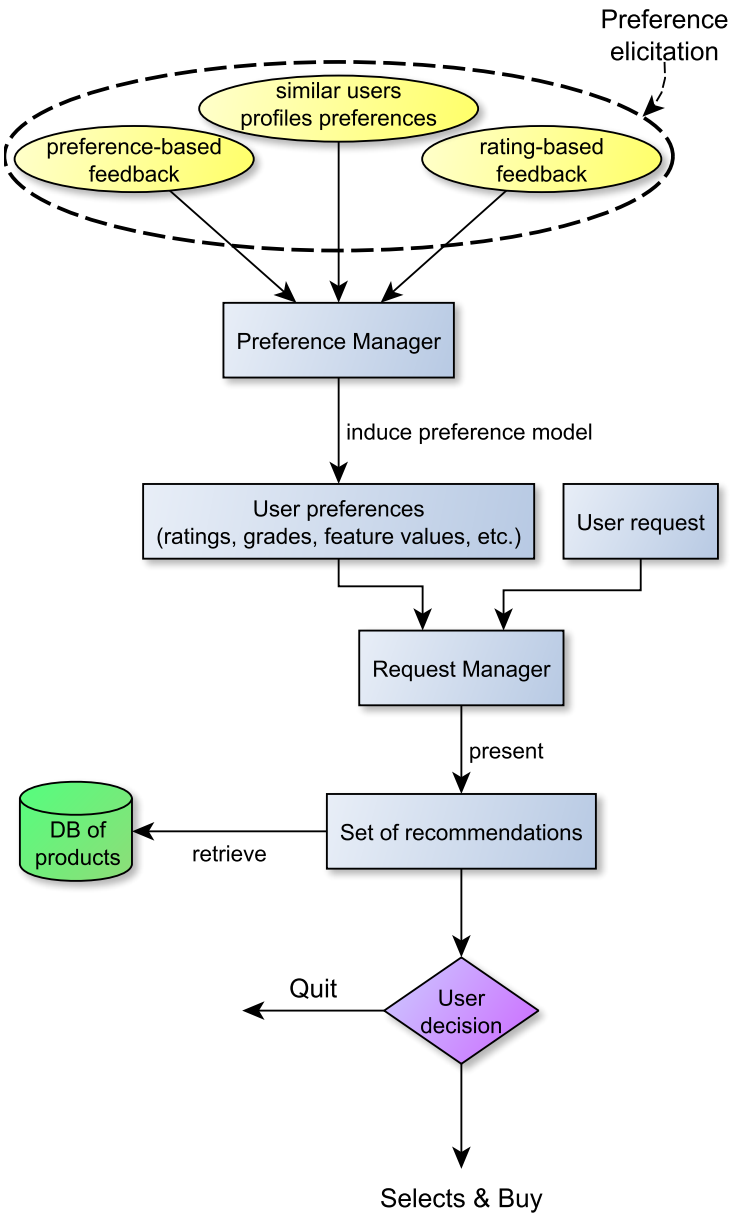


Figure 4.1: A single-shot recommendation scenario

One fundamental step in the process of recommending, which is preferences induction, is then presented in the two instances. The two instances were implemented and experimentally tested; experimentations and a comparative study are given in Section 4.7. Before concluding, we discuss in Section 4.9 possible extensions of the approaches that were used in this dissertation.

4.2 The Case Study: Information Recommendation

In order to use in practice the approaches that we proposed, we chose to integrate such approaches in one form of conversational RSs. This section gives a description

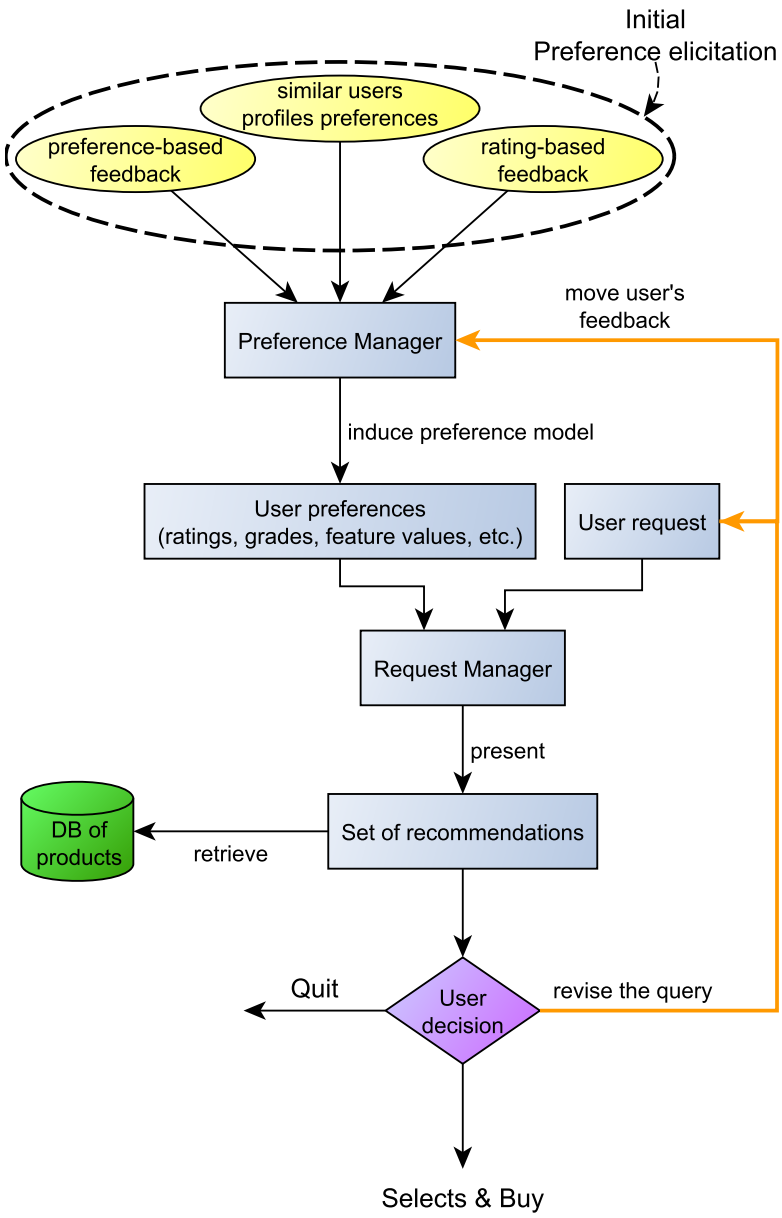


Figure 4.2: A conversational recommendation scenario

of the system that serves as a use case.

4.2.1 The advisor

Bridge and Ricci (Bridge & Ricci 2007) introduced a new kind of conversational recommendation called *Information Recommendation (IR)*. IR helps a user identify a suitable product, whose features are represented by Boolean variables, to purchase or consume. IR may be regarded as an instance of a kind of system in which the user repeatedly edits and resubmits a query, that she chooses from a set of potential queries suggested by the system, until she finds a product that she wants (Ricci et al. 2006, McSherry 2005). The potential queries are selected with regards

to the user's preferences that are collected from the user when she edits and submits queries. These potential queries are meant to best match the user's preferences which are induced from the user's selected query each time she has to select a query among those suggested by the system. The possible queries that the system may suggest to the user can be too large to be efficiently checked by the user. The user might select a query that results in a combination of features which is not included in any product in the database as not all queries are *satisfiable*. Then, the system will show only the satisfiable queries to ensure the user deals with available features combinations only. In fact, a query is said to be satisfiable if and only if there exists a product in the database which has all the boolean features in the query; otherwise, it is *unsatisfiable*. The user is not assumed to know the satisfiable queries beforehand.

The advice given to the user in each step of the dialogue can remain large enough to confuse the user and make the selection task more difficult. The RSs should reduce the number of queries shown to the user without losing the focus on the relevance the advice should have regarding the user's preferences. Then, these preferences have to be collected by the advisor during the interaction with the user. The system then infers preference relations from the user's previous contribution to the dialogue, such as her earlier queries. The system might, for instance, infer that the combination of features in the chosen query is more preferred than all the combinations of features that represent the non chosen queries. Having these preferences stored in a kind of user model, constructed dynamically and progressively updated as the dialogue continues, the system starts to be able to determine whether certain queries dominate others, to rank the next possible queries, and to suggest to the user those that are compatible with her preferences and compatible with the available products.

Let us suppose a user wants to book a hotel for a one-week holiday. She may want to be supported by an automated advisor. A "single-shot" RS will propose a set of products to the user once and for all. The system would have obtained the user's preferences either by capturing her feedback or looking for users' profiles that are considered similar to the user's profile. The user's preferences can for instance be obtained by a computationally intense offline phase that induces the user's preferences based on their opinions on items (Levandoski et al. 2012). Accordingly, the system looks for suitable items in the database and proposes to the user a set of recommendations. If the user is not satisfied then she can quit or send another request to the system. The system's response to the second request will be treated by the system independently from the system's answer to first request (see Figure 4.1).

However, this is not always the case as the user may not be willing or able to reveal all her preferences about products as she may not be very familiar with the available products (i.e., hotels) and their characteristics (e.g., piece of equipment in the hotel room). Her preferences would then be constructed while interacting with

the system.

A conversational RS engages the user in a dialogue during which the user is allowed to elaborate her requirements about the hotel she would like. A dialogue may be extended over several steps. During each step, the user supplies feedback on the items recommended by the system. This feedback will guide the selection of the set of items the system proposes in the next step. The dialogue comes to its end when the user is satisfied or terminates voluntarily the dialogue or there are no suitable items to be recommended. As a conversational RS, the advisor presented in (Bridge & Ricci 2007) suggests items, which are queries in this case (see Section 4.2.2 and 4.2.3) that help the user to efficiently search for products, which are hotels in the actual system. During each step of the dialogue when the user selects one query proposed by the advisor, the system will observe the chosen query and infer constraints about the user's model of preferences. Then constraints between two combinations of features will be added to a dynamic user model that the system constructs progressively. These constraints help the advisor deduce preference relations between queries and enable it to construct a ranking of these queries, rather than products. Then, the advisor delivers only optimal queries, regarding the user's preferences, to the user in the next step of the dialogue.

The advisor in IR gives freedom and flexibility to the user regarding the queries the user can perform at each step of the dialogue. This will steer the user smoothly towards the target.

After looking at the database, the advisor will propose queries that deal only with combinations of features available (i.e., included in at least one product in the database). The system will propose queries that lead to combinations of features that are included in at least one product in the database. This will dissuade the user from trying to request a combination of features in vain. Thus, the user will realize that she can satisfy a desire for other features. Consequently, the user may not give up and requests a hotel with features B and C (declared available by the system). Next, having seen the available queries, the user may want to replace one feature C by two other features D and E. The system observes the user's selection, learns and proposes more queries to encourage the user to target the most convenient hotel regarding her preferences, a part of which was learnt during the interaction.

We suggest for IR a reasoning framework where a set of models of the user is assumed, each with an associated preference ordering, along with a satisfaction relation between models and statements of constraints on preferences expressed in an appropriate language. Within this framework, given a set of constraints, the system infers that one product is preferred to another if this preference holds for all models satisfying the constraints. As the dialogue proceeds, the user's new actions reveal more constraints on her preference ordering. This can narrow the set of models of

the user. We describe this framework in more detail in Section 4.3. Two instances of this framework were developed and presented in Section 4.4 and Section 4.5.

4.2.2 The queries

In this chapter, we assume that the products are modeled with a collection of Boolean-valued features $V = \{F_1, \ldots, F_n\}$. The features are intended to relate to a set of products that the user is interested in choosing between; for example, in choosing a hotel room, one feature might be the availability of a swimming pool in the hotel.

Let us define a configuration α to be a mapping from $\{1, \ldots, n\}$ to $\{1, 0\}$. A configuration α can also be thought of as a selection of features: all features F_i such that $\alpha(i) = 1$. For convenience, we will write a configuration such as $(1, 0, 1) = \big(\alpha(1), \alpha(2), \alpha(3)\big)$ as $f_1 \bar{f}_2 f_3$.

Configurations can be thought of as queries over the set of features. If a user issues a query q and if $f_i \in q$, this means that the user is interested in products that have the i^{th} feature. A subset of the configurations correspond to products that are available to the user. In accordance with most Web-based product search systems, $f_i \notin q$ means only that the user has not (yet) declared any interest in feature F_i; it does not mean that the user wants products that lack the i^{th} feature. Therefore, for example, if q is $f_1 \bar{f}_2 f_3$, the user wants a product that has features f_1 and f_3 and is yet to say anything about f_2.

4.2.3 The dialogue

In the kind of system presented in (Bridge & Ricci 2007), the user submits an initial query, typically one that is quite under-specified: 'to test the water'. In our experiments (Section 4.7) we use an empty initial query. Let this query be denoted as the *current query*, q.

The RS does not know the user's preferences and does not ask about them. It may only infer them from the sequence of queries that the user submits. As the dialogue proceeds, the RS will infer constraints on the user's preferences and express them as statements in a language $\mathcal{L}$. We will denote the current set of statements by Φ. Initially Φ may contain a set of 'background' assumptions. In particular, we will want to express the idea that including a feature in a query is at least as good as not including it. Statements will be added to Φ as the dialogue proceeds. For example, if the user's query requests a certain feature, we may plausibly infer that this feature is more important than those not included in the query.

The interaction between the user and the RS proceeds as follows:

- The RS analyzes the current query q, with particular regard to differences between q and the queries the user might have submitted. The system induces some additional constraints on the user's preferences and adds corresponding statements to Φ.

- The RS generates a set of candidate next possible queries and prunes this set to those that are satisfiable and undominated (see below). It advises the user to confine her next query to this set.

- The user chooses and submits her next query. This becomes the new current query q. In the experiments reported in Section 4.7, we arrange that the user always chooses one of the queries that the system advises (although this might not be the case in practice).

The sequence of the three steps is repeated until the user is satisfied with q or the set of undominated, satisfiable candidates is empty, in which case as far as the RS is concerned q cannot be bettered. At this point, the user can request to see the products that satisfy q.

The goal of the RS is to give the advice that has the greatest value. We consider this to be that which minimises the total quantity of advice given and the dialogue length, while guiding the user to the best product.

During the second step, the RS computes the following three sets of queries:

- *Candidates:* Candidate queries are ones which are close, in a particular sense, to the current query. Each is a low-cost edit to the current query. For example, if $f_i \notin q$, the set of candidates will include the query that results from adding just feature f_i to q.

- *Satisfiables:* The RS should never include unsatisfiable queries in its advice: they make interaction length longer without leading the user to the best product. Hence, the system eliminates from *Candidates* those queries which are unsatisfiable; the remaining queries are called the *Satisfiables*.

- *Undominated*: The system could advise the user to confine her query to *Satisfiables*. However, this set can be large. Hence, the system eliminates from *Satisfiables* each query which is dominated by (i.e., worse than) some other member of *Satisfiables*; the remaining set of queries is called *Undominated*. The dominance relation is based on what is induced in the first step above. The rationale is to exclude from the system's advice queries that, on the basis of what the system has induced about the user's preferences, it thinks the user would regard as inferior.

In effect, the system's advice to the user is to confine her next query to a set which the system knows are satisfiable and believes the user is likely to try next (since, according to what the system has inferred about the user's preferences, they are ones that are not dominated by other satisfiable queries).

We will now explain how to obtain the three sets of queries.

Generating the candidates

Real user behaviour in query editing tends to proceed with modifications of limited 'reach'. Hence, following (Bridge & Ricci 2007), we define *Candidates* as the set of queries which we obtain by applying three editing operations, *Add*, *Switch* and *Trade*, to the current query. These operations are defined as follows. Given current query q and $F_i \notin q$, operation $Add(q, F_i)$ adds just feature F_i to q, giving the new query $q \cup \{F_i\}$, which we sometimes write as q^i. $Switch(q, F_i, F_j)$ where $F_i \in q, F_j \notin q, i \neq j$ discards F_i in favour of F_j, giving the new query $(q \setminus \{F_i\}) \cup \{F_j\}$. Finally, $Trade(q, F_i, F_j, F_k)$ where $F_i \in q, F_j \notin q, F_k \notin q, i \neq j, i \neq k, j \neq k$ discards feature F_i and introduces features F_j and F_k.

There are, of course, other edits that could be in this set of candidates, such as trading two features in q for three others not yet in q. It is possible that such an edit would turn out to be satisfiable and not dominated by any other edit. It is helpful, however, in a practical system to confine the recommender's advice to a set of readily understandable edits, particularly ones that the user herself is likely to contemplate. Other edits, such as deletion of a feature from q, are excluded because they would produce a new query which, although satisfiable if q is satisfiable, would be dominated by q itself. Finally, it should be noticed that, although $Trade(q, F_i, F_j, F_k) = Add(Switch(q, F_i, F_j), F_k)$, the $Trade$ operation is not redundant in IR. There are scenarios in which the recommender could advise the user to try the $Trade$ operator (if it results in a satisfiable query that is not dominated by any other edits to q) but could not advise the user to try the $Switch$ operator in the first place, as a precursor and then performing an Add; the latter advice would not be suitable in scenarios where the $Switch$ results in a query that, while satisfiable, is dominated by one of the other edits.

Checking satisfiability

As defined earlier, a query is satisfiable if and only if there exists a product which has all the features present in the query. If products are stored explicitly in a database, satisfiability of a candidate query can be checked by a scan of the database. For configurable products, where the set of products is represented as the set of solutions to a CSP, satisfiability of a candidate query can be checked by determining if the

CSP has solutions containing all the features in the query (which can be verified by checking satisfiability of an augmented CSP).

Checking for dominance

The final pruning of the satisfiable candidate queries is performed using one of the two instances of the framework for dominance of preferences that will be explained in Sections 4.3, 4.4 and 4.5. In either case, $q \in Satisfiables$ is pruned if it is strictly dominated, i.e., dominated according to relation $\succ_\Phi$, by $q' \in Satisfiables$.

4.2.4 The user

The advisor described in (Bridge & Ricci 2007) is evaluated through modeling interactions between the system and simulated users. The user is supposed to have her own true preferences that the advisor does not know. Those true preferences are represented by vectors of weights. The weights are related to product features; in our experiments they are randomly selected real numbers in the interval [0,1]. The system knows only about the constraints on the user's preferences that it infers each time the user sends a request. The system makes sure the user always chooses one of the queries that the system recommends even though this might not be the case in practice.

Three different types of simulated users were used in the experiments described in (Bridge & Ricci 2007): *optimizing*, *prioritizing* and *random*. In this dissertation, we are using only *optimizing users*: within a dialogue, they do not try queries that were already tried earlier in the dialogue; they are aware of their own preferences and never opt for the next query that would be inferior to the current one; and they take heed of all advice given, i.e., if the RS tells them to confine their next query to a certain set, then they do so; indeed they choose the best possible query from this set.

Figure 4.3 gives a model of the dialogue described above. In Figure 4.3, the system initially proposes a set of queries to the user who chooses one of them; the chosen query is called *the current query*. If the user is satisfied with the current query, the system retrieves a product that contains all features in the current query and shows it to the user so that she can select and buy it. In case the user is not satisfied with the current query, the user faces two decisions: to quit voluntarily (as she does not want to continue anymore) or to ask for more queries provided the system has more queries to be offered (otherwise the dialogue will end necessarily).

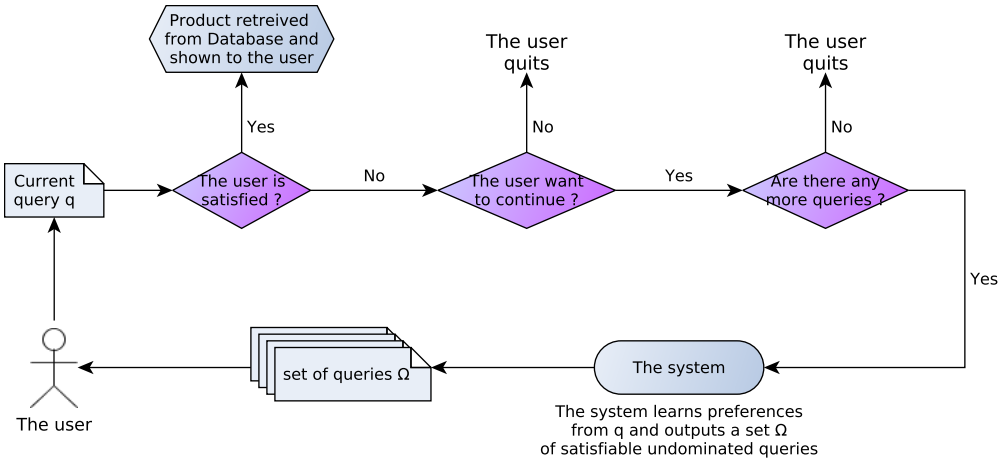


Figure 4.3: Interaction model of the user with the recommender.

4.3 A Framework for Preference Dominance

One major purpose of IR is to reduce irrelevant content and provide users with more pertinent information or product, in an attempt to offset information overload. One key operation is dominance : comparison between pairs of available options to find out which options are dominated and consequently eliminated by the system. An option q is said to be dominated if there exist at least another option q' that is preferred over q for all the orderings that satisfy the set of preferences that are collected from the user during the dialogue with the system.

RSs can be subtle and delicate to understand and control (Massa & Bhattacharjee 2004). Users often see RSs as a black box and are not aware of their working model (Denker et al. 2003). In fact, with current RSs it is tricky for the user to control the recommendation process so that if the RS starts to give poor quality recommendations, usually the user just stops using it (Zaslow 2002).

This section presents a framework for preference reasoning within RSs. This can be considered as a modest step towards bringing a kind of a comprehensive picture of the preference dominance engine which represents the core of item selection in RSs. Figure 4.6 summarizes the framework. All the details are analytically discussed through this section and also the following sections. The case study in this work (a particular kind of conversational RS described in section 4.2), implements this framework.

4.3.1 Ultimate goal: inference

Given a set Ω of possible configurations (as defined in Section 4.2.2), the system needs to know which configurations would be closer to what the user really wants. Allowing the user to focus only on recommended queries requires the system to

prune non optimal (i.e., undominated) queries that can distract the user and probably cause the user's web experience to be longer and thus unattractive (Konstan & Riedl 2012). This can deter the user from using the system again in the future.

Therefore, eliminating uninteresting or less desired (i.e., dominated) queries, with regards to the user preferences, within a large set of queries seems to be a vital process for RSs to succeed. This was confirmed and validated by experiences from fielded applications with conversational RSs described by (Felfernig et al. 2006, Felfernig & Gula 2006). The results, which were based on user questionnaires after using systems, showed that interactive recommenders help users to better steer themselves when being confronted with large sets of choices. In fact, in conversational RSs without pruning, the user is faced with a large number of subsequent queries. Eliminating "weak" items was adopted by early RSs and particularly conversational RSs. Conversational RSs offer the user mechanisms (e.g., checkbox or labels) to narrow down the large set of possible items to a smaller set of items that abide by specific conditions. These conditions involve several criteria inherent to the structure of products and which comply with the user's preferences.

One way to prune uninteresting queries is a pairwise comparison done by the system between the available queries. The advisor should propose only undominated queries. An undominated query is the one that is not dominated by any other query with respect to the user's preferences. Thus, we need to obtain a relation that points out the undominated configurations in Ω regarding information previously collected about the user's preferences.

The framework develops a preference dominance engine that will select the undominated options by the means of a dominance operator: a key test that will be potentially performed for a large number of options. Therefore, this test needs to be as efficient as possible. This framework also aims at constructing a sound theoretical basis for preference dominance engines.

4.3.2 Logical settings

The general idea is to provide a framework that allows other preference methods to be integrated easily. It also allows computations and comparison with other methods. One problem is that quite often the decision maker states preference statements in an informal language (e.g., natural language) which do not necessarily have a straightforward representation. It is therefore necessary to use appropriate languages for preference representation, taking into account the specific context where the constructed model is going to be used.

In general, inference denotes the process of assuming that certain statements about the world are true and deriving what follows for the truth of other statements.

Thus, in order to make non-trivial inferences regarding the actual user's preferences over configurations, this framework requires assumptions about the model of a user's preferences and how this model is said to be compatible with a set of preferences expressed in a given language.

The advisor associates a set of models for the user. For instance, a model can be a vector of weights or a cp-tree. An ordering is associated to each model. The user's preferences elicited from the user while interacting with the system are expressed as a set of constraints on preferences expressed in an appropriate language (e.g., CP-nets).

The system claims a query q dominates another query q' with respect to the user's preferences if and only if all the user models that are compatible (or satisfy) the users' preferences also prefer q over q'. A satisfaction relation associates a set of preferences statements with each model . A framework for preference dominance takes as input a set of available options and a set of preferences collected from the user during the dialogue, and delivers a set of undominated options (see Figure 4.6 in Section 4.6.2).

Example Let $V = \{F_1, F_2, F_3$ be the set of features and let the user be represented as a vector of weights $w = (w_1, w_2, w_3)$. The weights w_1, w_2 and w_3 correspond to F_1, F_2 and F_3 respectively. Let Φ be the set of user preference statements: $f_1\bar{f}_2f_3 \geq \bar{f}_1f_2f_3$, and $f_1f_2\bar{f}_3 \geq \bar{f}_1f_2f_3$. Let α be the configuration $f_1\bar{f}_2\bar{f}_3$, and let β be the configuration $\bar{f}_1f_2\bar{f}_3$. α dominates β if and only if all weights vectors w that satisfy the user's preferences also agree that the weight of α is greater or equal than the weight of β. We say that weights vector w satisfies the constraint $f_1\bar{f}_2f_3 \geq \bar{f}_1f_2f_3$ if and only if $w(f_1\bar{f}_2f_3) \geq w(\bar{f}_1f_2f_3)$, i.e., $w_1 + w_3 \geq w_2 + w_3$, which holds if and only if $w_1 \geq w_2$. By similar reasoning, w satisfies Φ if and only if $w_1 \geq w_2$ and $w_1 \geq w_3$. Also, w satisfies $\alpha \geq \beta$ if and only if $w_1 \geq w_2$.

We can see that any weights vector w which satisfies the first preference statements (i.e., $w_1 \geq w_2$) and the second one (i.e., $w_1 \geq w_3$) will necessarily comply with $\alpha \geq \beta$ (i.e., $w_1 \geq w_2$). The system will then confirm that α dominates β (i.e., $\alpha \geq \beta$).

The framework assumes :

- A set of models $\mathcal{M}$, each of which is intended to represent a possible user (or way the user could be). Associated with each $M \in \mathcal{M}$ is a total pre-order $\succcurlyeq_M$ on configurations, i.e., a reflexive, transitive and complete relation (so for all configurations α and β, we have either $\alpha \succcurlyeq_M \beta$ or $\beta \succcurlyeq_M \alpha$ or both).
- A formal language $\mathcal{L}$ whose statements express constraints on the user's preferences.

- A relation $\models$ between $\mathcal{M}$ and $\mathcal{L}$. For $M \in \mathcal{M}$ and $\varphi \in \mathcal{L}$, we interpret $M \models \varphi$ to mean that φ holds for the preferences of M.

Given the assumptions above, the framework gives necessary and sufficient conditions for a relation to be strictly dominance-compatible. Given a particular set $\Phi \subseteq \mathcal{L}$ of statements, we consider the orderings on configurations which hold for every model satisfying statements Φ. Formally, we can define the relation $\succcurlyeq_\Phi$ on configurations as follows: $\alpha \succcurlyeq_\Phi \beta$ if and only if $\alpha \succcurlyeq_M \beta$ for all M satisfying (every member of) Φ. It follows that $\succcurlyeq_\Phi$ is a pre-order (a reflexive and transitive relation) on configurations.

Now, $\alpha \succcurlyeq_\Phi \beta$ means that every user who agrees with Φ considers that configuration α is at least as desirable as configuration β (assuming this particular model of users). It is possible that we also have $\beta \succcurlyeq_\Phi \alpha$, in which case every user considers that α and β are equally desirable. We define the relation $\succ_\Phi$ to be the strict part of $\succcurlyeq$, so that $\alpha \succ_\Phi \beta$ if and only if $\alpha \succcurlyeq_\Phi \beta$ and $\beta \not\succcurlyeq_\Phi \alpha$. Relation $\succ_\Phi$ is irreflexive and transitive. We say that *Given* Φ, α *strictly dominates* β, if $\alpha \succ_\Phi \beta$, that is, if all users (represented by models M in $\mathcal{M}$) agreeing with Φ regard α as at least as preferable as β (i.e., $\alpha \succcurlyeq_M \beta$), and at least one such user regards α as strictly preferable to β (i.e., $\beta \not\succcurlyeq_M \alpha$).

4.3.3 Application

We present experimental evidence that verifies the ability of this framework to provide an efficient preference dominance engine that supplies optimal configurations to the conversational RSs. Indeed, two instances of the framework were developed in Sections 4.4 and 4.5. This shows the feasibility and usability of the framework in practice.

Consequently, we believe that this approach empowers the decision-maker with a generic framework, allowing to personalize and accommodate the dominance engine to the requirements of different users and applications. Thus, this framework lays the ground work for further general functionality for reasoning with preferences. This formalism allows comparison and analysis of different preferences approaches, which leads to a better understanding and convenient assessment of theoretical approaches for conversational RSs.

Ideally, a user can easily parameterize this framework, e.g., by choosing the preference language or more specifically the form of input preference statements. Although the rationale of the framework is simple and intuitive, the framework is still interesting and useful for the development of pragmatic preference reasoning formalisms that are completely generic and essentially independent of the concrete application at hand.

Therefore, this framework will pave the way for the integration of more preference reasoning engines and allow the user to parameterize the conversational RSs by specifying how the user model is represented (e.g., weights vector, cp-tree), which can also adapt automatically to the user and problem in context.

Two instances of this framework are developed and are presented in this chapter. The first, described in Section 4.4, is based on a simple quantitative preferences formalism, involving a sum of weights (one for each feature of the recommended products), with an associated language of linear inequalities. This is a very commonly used model for preference representation, specifically, in MAUT (Dyer 2005) described in Section 2.5.1 of Chapter 2. The second instance of the framework (Section 4.5) is a qualitative preference formalism, where models adopt a kind of generalised lexicographic order, and constraints are expressed as comparative preference statements in a language generalising CP-nets (Boutilier et al. 2004*a*).

4.4 Sum of weights-Model Approach

Additive models allow compact representation of a utility function where the degree of desirability of the user is expressed separately for each attribute by a real value (Balabanovic 1998, Shen 2007, Kim et al. 2011, Shen et al. 2005). For the reasons described in (Keeney & Raiffa 1993, Stewart 1996, Tsoukiàs 2008, Caballero et al. 2010), the additive model is usually used in many real decision-making problems.

4.4.1 Models

The set of models contains all vectors of weights $w = (w_1, \ldots, w_n)$, where w_i is a non-negative real number. w_i is the weight assigned to feature F_i. Given a weights vector w, the overall value $w(\alpha)$ of a configuration α is the sum of weights of the features included in α, i.e., $w(\alpha) = \sum_{i:\alpha(i)=1} w_i$, which also can be written as $\sum_i w_i\alpha(i)$. This is used to define the ordering on configurations. We define the preference relation $\succcurlyeq_w$ for model w by $\alpha \succcurlyeq_w \beta$ if and only if $w(\alpha) \geq w(\beta)$, i.e., if and only if $\sum_i w_i\big(\alpha(i) - \beta(i)\big) \geq 0$. Thus $\succcurlyeq_w$ is a total pre-order on configurations.

4.4.2 Constraint language

Constraints on the user's preferences are expressed as inequalities between sums of weights. The weights summed are related to the features included in the configurations that are shown to the user. For instance, when the user selects a configuration α from a set of three configurations proposed by the system (i.e., α, β, γ), the system

infers that the sum of weights of features in α (denoted by *w(α)*) is greater or equal than the two sums of weights *w(β)* and *w(γ)*.

The set of constraints will, at a later stage, determine whether a configuration α dominates another configuration β. In fact, during dominance computation the advisor is interested in all models that are compatible with these inequalities.

4.4.3 Dominance relation

Given a set of constraints on preferences (or preference statements) Φ, and two configurations α and β, we say $\alpha \succcurlyeq_\Phi \beta$ if and only if $\alpha \succcurlyeq_w \beta$ for all weight vectors w that comply with all preference statements in Φ.

Example. Let $V = \{F_1, F_2, F_3$ be the set of features and let the user be represented as a vector of weights $w = (w_1, w_2, w_3)$. The weights w_1, w_2 and w_3 correspond to F_1, F_2 and F_3 respectively. Let Φ be the set of preference statements: $f_1\bar{f}_2\bar{f}_3 \geq \bar{f}_1\bar{f}_2f_3$, and $f_1\bar{f}_2f_3 \geq f_1f_2\bar{f}_3$. Let α be the configuration $f_1\bar{f}_2\bar{f}_3$, and let β be the configuration $\bar{f}_1f_2\bar{f}_3$. α dominates β if and only if all weights vectors w that satisfy the user's preferences also agree that the weight of α is greater than the weight of β. Weights vector w complies with the preference statement $f_1\bar{f}_2\bar{f}_3 \geq \bar{f}_1\bar{f}_2f_3$ if and only if $w(f_1\bar{f}_2\bar{f}_3) \geq w(\bar{f}_1\bar{f}_2f_3)$, i.e., $w_1 \geq w_3$. w complies with the preference statement $f_1\bar{f}_2f_3 \geq f_1f_2\bar{f}_3$ if and only if $w(f_1\bar{f}_2f_3) \geq w(f_1f_2\bar{f}_3)$, i.e., $w_3 \geq w_2$. Therefore, w satisfies Φ if and only if $w_1 \geq w_3$ and $w_3 \geq w_2$. On the other hand, w satisfies $\alpha \succcurlyeq_w \beta$ if and only if $w_1 \geq w_2$. the statement $w_1 \geq w_2$ can be easily induced from $w_1 \geq w_3$ and $w_3 \geq w_2$. Therefore, any weight vector w that satisfies Φ necessarily satisfies $\alpha \succcurlyeq_w \beta$. The system will then conclude that α dominates β (i.e., $\alpha \succcurlyeq_\Phi \beta$).

4.4.4 Dominance computation

One assumption is that including a feature is always at least as good as not including that feature. This assumption is translated in this instance by the set of constraints $w_i \geq 0$, for $i = 1, \ldots, n$, which says that all feature weights are non negative. Thus, these statements are included in Φ as constraints on preferences at the beginning of the dialogue. While we go forward in the dialogue, additional constraints on preferences are induced and added to Φ. Given two configurations α and β, to show $\alpha \succcurlyeq_\Phi \beta$, the advisor needs to check whether Φ entails $\alpha \geq \beta$ with real-valued variables w_i. Φ entails $\alpha \geq \beta$ is equivalent to saying that all weights vectors that agree or satisfy all constraints in Φ, also agree with α being preferred over β, i.e., $\sum_i \big(\alpha(i) - \beta(i)\big)w_i \geq 0$. The lack of expressiveness of this quantitative approach is well known in utility theory (Chomicki 2002).

A standard form of linear programming can answer the question above. This is immediate since each constraint in Φ is represented as a linear inequality between linear combinations of weights (e.g., $w_1 + w_2 \geq w_1 + w_3$). For instance, we can have the two following linear inequalities: $c_1 : w_1 + w_3 \geq w_2 + w_3$ and $c_2 : w_1 \geq w_2 + w_3$. These inequalities might be induced when the user makes her choices among the queries that were suggested by the system. $\alpha \geq \beta$ is equivalent to the following inequality: $g : w_1 \geq w_2$ if α has only feature F_1 (i.e., $f_1\bar{f}_2\bar{f}_3$) and β has only feature F_2 (i.e., $\bar{f}_1 f_2\bar{f}_3$). A linear programming solver, which uses the Simplex algorithm (Brearley et al. 1975), can then check whether Φ entails $\sum\big(\alpha(i)-\beta(i)\big)w_i$. Technically, the solver checks whether or not c_1 and c_2 imply $w_1 \geq w_2$. In other words, the solver can check whether all weights vectors w whose values comply with c_1 and c_2 also comply with $w_1 \geq w_2$. An implementation of this using a linear programming solver is to express it as a linear optimisation problem. Define g_{min} to be the minimum value of $\sum_i\big(\alpha(i) - \beta(i)\big)w_i \geq 0$ subject to constraints Φ. It can be shown that $\alpha \succcurlyeq_\Phi \beta$ if and only Φ entails $\sum_i\big(\alpha(i) - \beta(i)\big)w_i \geq 0$ if and only if $g_{min} \geq 0$.

Example 1. Let Φ be the pair of statements: $f_1\bar{f}_2 f_3 \geq \bar{f}_1 f_2 f_3$, and $f_1 f_2\bar{f}_3 \geq \bar{f}_1 f_2 f_3$. Let α be the configuration $f_1\bar{f}_2\bar{f}_3$, and let β be the configuration $\bar{f}_1 f_2 f_3$. Weights vector w satisfies the constraint $f_1\bar{f}_2 f_3 \geq \bar{f}_1 f_2 f_3$ if and only if $w(f_1\bar{f}_2 f_3) \geq w(\bar{f}_1 f_2 f_3)$, i.e., $w_1 + w_3 \geq w_2 + w_3$, which holds if and only if $w_1 \geq w_2$. By similar reasoning, w satisfies Φ if and only if $w_1 \geq w_2$ and $w_1 \geq w_3$. Also, w satisfies $\alpha \geq \beta$ if and only if $w_1 \geq w_2 + w_3$. Thus Φ does not entail $\alpha \geq \beta$, so we do not have $\alpha \succcurlyeq^{sw}_\Phi \beta$, since, for example, weights vector w with $w_1 = 4$, $w_2 = 2$ and $w_3 = 3$ satisfies Φ but does not satisfy $\alpha \geq \beta$. □

Example 2. Suppose now that there are four features, and let Ψ be the pair of statements $f_1\bar{f}_2 f_3 f_4 \geq \bar{f}_1 f_2 f_3 f_4$ and $f_1 f_2\bar{f}_3\bar{f}_4 \geq f_1\bar{f}_2 f_3\bar{f}_4$. With the sum of weights semantics this implies $f_1 f_2\bar{f}_3 f_4 \geq \bar{f}_1 f_2 f_3 f_4$, since the first statement implies $w_1 \geq w_2$, and the second statement implies $w_2 \geq w_3$, which implies $w_1 \geq w_3$. Thus, we conclude that the third statement is satisfied. We therefore have $f_1 f_2\bar{f}_3 f_4 \succcurlyeq_\Phi \bar{f}_1 f_2 f_3 f_4$. In fact, we have strict dominance: $f_1 f_2\bar{f}_3 f_4 \succ_\Phi \bar{f}_1 f_2 f_3 f_4$ since we do not have $\bar{f}_1 f_2 f_3 f_4 \succcurlyeq_\Phi f_1 f_2\bar{f}_3 f_4$. □

4.5 CP-tree Model Approach

There are preferences that cannot be expressed through the sum of weights-based approach. One example is conditional preferences statements on the values of variables. These conditional preference dependence relationships are natural in many situations. Example 1 mentioned in Section 2.6.3 of Chapter 2 exhibits conditional preferences. Indeed, if the memory is equal to 1024MB, *UBUNTU* is preferred over *XP*. In case the memory is equal to 2048MB, *XP* is preferred over *UBUNTU*.

Comparative preference languages enable the framework to handle conditional preferences by expressing constraints on preferences. To our knowledge, this has not been used in IR or any other RSs before.

4.5.1 Models

In this approach, the user preferences are assumed to be modelled by a cp-tree presented in Section 3.4.2.1 of Chapter 3, through which she decides which of the given configurations is better. The cp-tree, one example of which is depicted in Figure 4.4, allows the user to have a total pre-order over a set of possible configurations. Unlike a weights vector-based user model, a user who is represented by a cp-tree explicitly and naturally handles conditional preferences. This is highlighted in the definition of cp-tree. A cp-tree applied to the context of this framework has the following shape and properties. Every node is associated with γ variables at most. A local value ordering associated with variables in each node depends on the values taken by the parents of that node. The assumption that having a feature is at least as good as not having that feature is also implemented in every cp-tree that we consider in this chapter.

As every cp-tree implements a kind of generalized lexicographic order, any two configurations α and β are compared as follows: α and β are first compared on the value they have for the set of variables associated with the most important node. If one of them has better value than the other then we conclude that the former configuration is definitely preferred over the other one regarding the user's model of preferences; the node is said to be *decisive*. If not then the two configurations are compared based on the next most important node which is determined regarding the value both configurations have for the set of variables associated with the previous node. This comparison continues through the following nodes in a lexicographic order until a decisive node is found; otherwise α and β are said to be equivalent. Let $M(\gamma)$ be the set of cp-trees with γ being the maximum number of variables associated with a node. Figure 4.4 represents an example of a cp-tree with $\gamma = 1$. Each node in the cp-tree depicted in Figure 4.4 is labeled with a variable (or feature). The root is labeled by the most important variable, F_2 in this example.

Each node is also associated with a preference ordering of the values of the variable. This local ordering in the case of the nodes in the example is $f_i \geq \bar{f}_i$, where f_i means F_i is included and $\bar{f}_i$ means that F_i is not included. This ordering captures the requirement that including a feature is never worse than not including it.

Two configurations α and β are first compared based on this most important variable. If they do not agree on this variable then the comparison is settled: in the example, if α contains feature F_2 and β does not, then α is better than β. This happens, for example, if α is $\overline{f_1} f_2 \overline{f_3}$ and β is $f_1 \overline{f_2} f_3$. Otherwise, α and β agree on the most important variable. The user may then have a next most important variable (labeling a child node); this can depend on the value assigned to the most important variable (signified by the value on the edge from parent to child). For this reason, this model allows *conditional preferences*. If there is no such next important variable, then α and β are considered equally preferable according to this cp-tree. Thus the cp-tree σ generates a total pre-order $\succcurlyeq_\sigma$ on outcomes.

Note that each node in the cp-tree in Figure 4.4 is associated with a single variable. In fact, we can allow a more general representation, where at most γ variables are associated with a node, along with a total pre-order over the assignments to that set of at most γ variables. Figure 4.5.1 shows an example of a cp-tree with $\gamma = 2$. In Figure 4.5.1, for instance, the root node is associated with the pair of variables $Y = \{F_2, F_3\}$ and the local ordering is then over assignments to Y, and is as follows, $f_2 f_3 \geq f_2 \bar{f}_3 \geq \bar{f}_2 f_3 \geq \bar{f}_2 \bar{f}_3$. The local ordering associated with a node associated with feature F_i, must then be $f_i \geq \bar{f}_i$, since including a feature is always at least as good as not including it. CP-trees are described in more detail in Section 3.4.2.1 of Chapter 3.

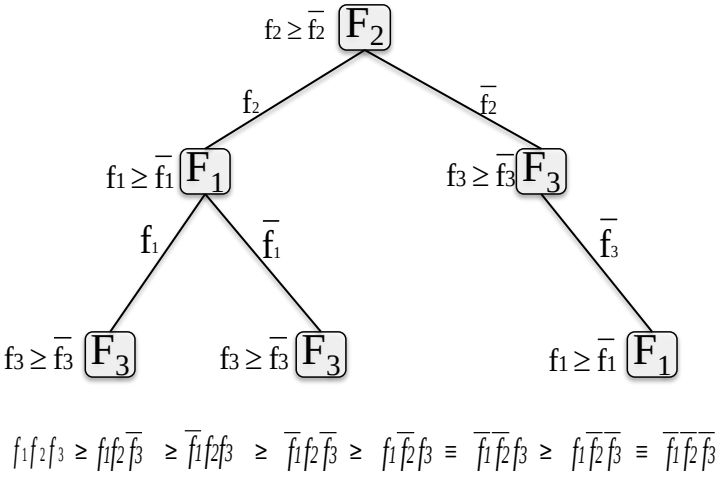


Figure 4.4: A cp-tree σ with $\gamma = 1$, along with its associated ordering $\succcurlyeq_\sigma$ on configurations

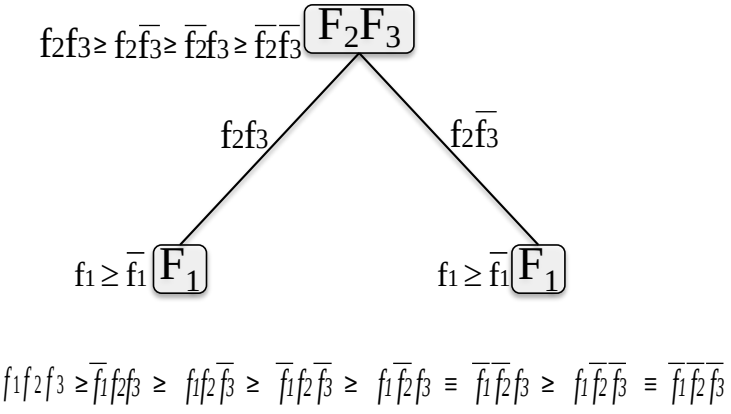


Figure 4.5: A cp-tree σ with $\gamma = 2$, along with its associated ordering $\succcurlyeq_\sigma$ on configurations

4.5.2 Constraint language

Comparative preference theories, which are presented in Section 3.3 in Chapter 3, are used in this instance to enable the advisor to handle conditional preferences while inducing constraints on user's preferences. We are specifically using comparative preference theories which allow the system to handle preferences statements φ that have the form $p \geq q \parallel T$, where P, Q and T are subsets of the set of features V, and p is an assignment to $P = Q$ (i.e., a function from P to $\{0, 1\}$), and q is an assignment to Q.

The statement φ expresses the preference of a partial configuration p over another partial configuration q with variables in T held constant. This statement also says that a model $\succcurlyeq$ satisfies $p \geq q \parallel T$ if and only if $\alpha \succcurlyeq \beta$ for all configurations α and β such that α extends p and β extends q with $\alpha(X) = \beta(X)$ for all $X \in T$.

As the constraint language in Section 4.4.2 does, this language allows to express $\alpha \geq \beta$ (i.e., configuration α is preferred over configuration β). This is possible when $P = Q = V$ and $T = \emptyset$.

Example Suppose that there are four binary features in $V = \{F_1, F_2, F_3, F_4\}$. and when the user selects a configuration $\alpha = f_1\bar{f}_2f_3f_4$ among a set of three configurations proposed by the system (i.e., $\alpha = f_1\bar{f}_2f_3f_4$, $\beta = \bar{f}_1f_2f_3f_4$, $\gamma = f_1f_2\bar{f}_3\bar{f}_4$), the system infers the set Ψ of preference statements $\Psi = \{f_1\bar{f}_2f_3f_4 \geq \bar{f}_1f_2f_3f_4$, $f_1\bar{f}_2f_3f_4 \geq f_1f_2\bar{f}_3\bar{f}_4\}$. The two preference statements (or constraints) in Ψ say the

combinations of features in α is better than both combinations of features in β and γ.

4.5.3 Dominance relation

Let α and β be two configurations. α dominates β (i.e., $\alpha \succcurlyeq_\Phi \beta$) if and only if all possible users that agree with the set of constraints in Φ, prefer α over β. A possible user in this instance is represented by one cp-tree σ among the set $\mathcal{M}(\gamma)$ of cp-trees for $\gamma = 1, 2$, or 3. Therefore, $\alpha \succcurlyeq_\Phi \beta$ holds if and only if every cp-tree σ in $\mathcal{M}(\gamma)$ that satisfies all constraints in Φ orders α before β ($\alpha \succcurlyeq_\sigma \beta$). In other words, $\alpha \succcurlyeq_\Phi \beta$ holds if every cp-tree σ in $\mathcal{M}(\gamma)$ that extends all constraints in Φ has α come before β or α and β are equivalent through σ (a cp-tree generates a total pre-order).

4.5.4 Dominance computation

The assumption stating that including a feature is always at least as good as not including it requires to include, in the set Φ, the statement $f_i \geq \overline{f_i} \parallel V \setminus \{F_i\}$ for each feature F_i, for $i = 1, \ldots, n$. This assumption is captured by the local ordering in the case of the nodes in a cp-tree that represent a user model.

A dominance testing algorithm is presented in (Wilson 2009*b*). As shown in the previous section and described in Section 3.4.2 of Chapter 3, $\alpha \succcurlyeq_\Phi \beta$ involves reasoning with cp-trees and the total pre-orders they generate. In Theorem 1 of (Wilson 2009*b*), the dominance algorithm can check, in polynomial time, whether $\alpha \succcurlyeq_\Phi \beta$ by verifying that all weak orders that satisfy Φ, agree with $\alpha \geq \beta$ as well.

Example 1 continued. With the cp-tree semantics, when $\gamma = 1$, the pair of statements Φ ($f_1\bar{f}_2f_3 \geq \bar{f}_1f_2f_3$, and $f_1f_2\bar{f}_3 \geq \bar{f}_1f_2f_3$) implies the preference statement $f_1\bar{f}_2\bar{f}_3 \geq \bar{f}_1f_2f_3$, so we have $f_1\bar{f}_2\bar{f}_3 \succcurlyeq_\Phi^{cp_1} \bar{f}_1f_2f_3$ (and indeed we have $f_1\bar{f}_2\bar{f}_3$ strictly dominates $\bar{f}_1f_2f_3$, i.e., $f_1\bar{f}_2\bar{f}_3 \succ_\Phi \bar{f}_1f_2f_3$). The reason is that, for any 1-cp-tree σ satisfying $f_1\bar{f}_2f_3 \geq \bar{f}_1f_2f_3$, the most important feature must be either F_1 or F_3. (If F_2 were the most important feature, then we would not have $f_1\bar{f}_2f_3 \succcurlyeq_\sigma \bar{f}_1f_2f_3$, because the local ordering is $f_2 \geq \bar{f}_2$, since the presence of a feature is never worse than its absence.) Similarly, if 1-cp-tree σ satisfies $f_1f_2\bar{f}_3 \geq \bar{f}_1f_2f_3$, then the most important feature must be either F_1 or F_2. Hence for any 1-cp-tree σ satisfying Φ, F_1 is the most important feature. The root node then determines the preference ordering of the pair of configurations $f_1\bar{f}_2\bar{f}_3$ and $\bar{f}_1f_2f_3$: since the local ordering of this node must be $f_1 \geq \bar{f}_1$, we have $f_1\bar{f}_2\bar{f}_3 \succeq_\sigma \bar{f}_1f_2f_3$. Hence we have $f_1\bar{f}_2\bar{f}_3 \succcurlyeq_\Phi \bar{f}_1f_2f_3$. The qualitative and lexicographic nature of the cp-tree semantics ensures this inference, in contrast with the numerical the sum of weights-based approach, which did not. □

Example 2 continued. Recall that Ψ is the pair of statements $f_1\bar{f}_2f_3f_4 \geq \bar{f}_1f_2f_3f_4$ and $f_1f_2\bar{f}_3\bar{f}_4 \geq f_1\bar{f}_2f_3\bar{f}_4$. In contrast with the sum of weights semantics, Ψ does not imply $f_1f_2\bar{f}_3f_4 \succcurlyeq_\Phi \bar{f}_1f_2f_3f_4$ with the cp-tree semantics. To show this we can construct a 1-cp-tree σ with F_4 as the most important (root node) variable, and where, given f_4, F_3 is more important than F_1 which is more important than F_2, and given $\bar{f}_4$, F_2 is more important than F_3 which is more important than F_1. σ then satisfies Ψ, but not $f_1f_2\bar{f}_3f_4 \geq \bar{f}_1f_2f_3f_4$.

A key issue here is that cp-trees can represent *conditional* preferences: the preferences can be different given f_4 from those given $\bar{f}_4$. In contrast, the sum of weights semantics assumes preferential independence, so preferences are not conditional at all, which is why the inference holds for the sum of weights semantics. □

The pair of examples show that the two preference dominance techniques are incomparable: $\succcurlyeq_\Phi$ with the cp-tree semantics can sometimes include preferences not included in $\succcurlyeq_\Phi$ with sum of weights semantics, and vice versa.

4.6 Induction of Constraints on Preferences Within the Framework

Having explained the two instances of the framework for dominance of preference, it remains to return to IR to explain what the system induces in the first step above denoted in Section 4.2.3, when it observes the user's queries. We explain this below for each of the two preference models.

4.6.1 Inducing constraints in the sum of weights model

$Add(q, F_i)$

> If the user has added feature F_i to query q, new query q^i is created which represents the current query q with the feature F_i added, then statements $q^i \geq q^j$ are induced for all $F_j \notin q$, $i \neq j$ unless $Add(q, F_j) = q^j$ is unsatisfiable. This assumes that the new query is preferred to other satisfiable queries that could have been generated by adding other features. This implies that the weight vector satisfies the linear inequality $w_i \geq w_j$. However, we do not infer $q^i \geq q^j$ in all cases. In particular, we do not infer it if $Add(q, F_j)$ is unsatisfiable. Users may have (incomplete) knowledge of which queries are unsatisfiable: if she knows a query is unsatisfiable, then she will not submit it. We 'play it safe': when q^j is unsatisfiable, in case the user knows this, we do not assume that the query that she does submit has higher weight than this

unsatisfiable query.

$Switch(q, F_i, F_j)$

If the user switches F_i for F_j, then we infer that $w_i \leq w_j$; and for all $f_k \notin q$ we can infer $w_j \geq w_k$ unless $Switch(q, F_i, F_k)$ is unsatisfiable. Indeed, switching F_i for F_j gives rise to a new query q^j_{-i} which represents the new query obtained when the user switches a feature F_i for another feature F_j, then statements $q^j_{-i} \geq q^k_{-i}$ are induced for all $f_k \notin q$, $k \neq i$ unless $Switch(q, F_i, F_k)$ is unsatisfiable. This assumes that the new query is preferred to other satisfiable queries that could have been generated by switching F_i for other features such as F_k. This implies that the weights vector satisfies the linear inequality $w_j \geq w_k$. However, we do not infer $q^j_{-i} \geq q^k_{-i}$ if $Switch(q, F_k, F_j)$ is unsatisfiable (because of the same reason given in the previous paragraph).

$Trade(q, F_i, F_j, F_k)$

$Trade(q, F_i, F_j, F_k)$ allows the user to switch the feature F_i for two other features F_j and F_k If the user trades F_i for F_j and F_k, then we infer that $w_i \leq w_j + w_k$; and for all $j', k' \notin q$ such that $\{j, k\} \neq \{j', k'\}$ and $j' \neq k'$, we infer $w_j + w_k \geq w_{j'} + w_{k'}$ unless $Trade(q, F_i, F_{j'}, F_{k'})$ is unsatisfiable.

In fact, trading F_i for F_j and F_k gives rise to a new query q^{jk}_{-i} which represents the new query obtained when the user switches a feature F_i for two other features F_j and F_k, then statements $q^{jk}_{-i} \geq q^{j'k'}_{-i}$ are induced for all $F'_j, F'_k \notin q$, $k \neq i$ unless $Trade(q, F_i, F_{j'}, F_{k'})$ is unsatisfiable. This assumes that the new query is preferred to other satisfiable queries that could have been generated by switching F_i for other pairs of features (i.e., F'_j, F'_k). This implies that the weights vector satisfies the linear inequality $w_j + w_k \geq w_{j'} + w_{k'}$. However, we do not infer $q^{jk}_{-i} \geq q^{j'k'}_{-i}$ if $Trade(q, F_i, F_{j'}, F_{k'})$ is unsatisfiable (because of the same reason given above).

Note that we have been quite conservative in what has been inferred. In the event of observing $Trade(q, F_i, F_j, F_k)$ for example, we might also have inferred that the selected $Trade$ is better than *all* other satisfiable $Trade$ operations, $Trade(q, F_{i'}, F_{j'}, F_{k'})$ for $\{i, j, k\} \neq \{i', j', k'\}$ instead of just other ways of trading f_i. Using the same reasoning, we might have inferred that a $Trade$ is better than all satisfiable Add and $Switch$ operations; and that a $Switch$ is better than all satisfiable Add operations.

We might make these inferences if we attribute ever greater rationality to the user. This chapter will not attribute these higher levels of rationality to the user, and hence we will infer only what we stated earlier. An important observation is that it is not a problem to our proposed methods if we assume that users are less rational than

they really are. The reason this does not pose a problem to our proposed methods is that assuming users are less rational than they really are, will result only in us making fewer deductions when observing their queries; it will not result in us drawing incorrect inferences. In fact, it is more adventurous to assume a fully rational user, who can really take the best query, since this will cause us to draw inferences that may be incorrect.

4.6.2 Inducing constraints in the cp-tree model

$Add(q, F_i)$

Again consider the situation where the user has chosen to add feature F_i rather than feature F_j (which is another feature different from F_i not present in q). For this model, there are alternative statements one might induce from this decision by the user. We consider two, each being a kind of counterpart for the constraint $w_i \geq w_j$ induced for the sum of weights-based approach. It is an advantage of the cp-tree model that it can express nuances that the sum of weights-based approach cannot.

- **Basic:** Let us consider q as the current query, let q^i be the current query q with the feature F_i added, and let q^j be q with the feature f_j added. A basic, somewhat conservative, approach is to just model the preference of feature F_i over feature F_j by the preference statement: $q^i \geq q^j || \emptyset$, i.e., $q^i \geq q^j$, which just expresses a preference for q^i over q^j.
- **Importance:** Alternatively, and less conservatively, we can induce $f_i \geq \overline{f_i} || V \setminus \{F_i, F_j\}$ which means the presence of feature F_i (i.e., f_i) is preferred over the absence of feature F_i (i.e., $\overline{f_i}$) whatever the case for feature F_j (i.e., feature F_j is absent or present in the query) all else being equal, which says that the eventual presence of the feature f_i is more important than the choice of F_j. Thus, regardless the state of the feature F_j in the query, the user will prefer F_i to be present in the query so that (if possible) this feature is included in the most suitable product.

Note too that in either case we ensure that the RSs 'plays it safe' when inducing preference statements, in the same way that we explained for the case of the sum of weights-based approach, by not inducing preferences over unsatisfiable queries. The second preference form (i.e., Importance) will very likely allow stronger inference (through a stronger dominance relation) than the first preference form (i.e., Basic) as the set of models (i.e., cp-trees) that comply with the induced preference statements in Importance form tends to be smaller (i.e., more restricted). This idea will be developed more in Section 4.9.1.

$Switch(q, F_i, F_j)$

If the user switches F_i for F_j then we can induce statements stating the superiority of the feature F_j over the feature F_i, and also over the features F_k that were not chosen instead of F_j.

- **Basic:** Let q be the current query, let q^j_{-i} be the current query q with the feature F_j added and the feature f_i removed, i.e., $Switch(q, F_i, F_j)$. We model the preference of feature F_j over feature F_i by the preference statement $q^j_{-i} \geq q||\emptyset$, i.e., $q^j_{-i} \geq q$. Similarly, we model the preference of feature F_j over feature F_k for any k such that q^k_{-i} is satisfiable by $q^j_{-i} \geq q^k_{-i}$.
- **Importance:** Here we induce the preference statements $f_j \geq \overline{f_j}||V \setminus \{F_i, F_j\}$, and $f_j \geq \overline{f_j}||V \setminus \{F_k, F_j\}$ for any k such that q^k_{-i} is satisfiable.

$Trade(q, F_i, F_j, F_k)$

If the user trades F_i for F_j and F_k then we can induce statements that assert the superiority of the combination of features F_j and F_k over the feature F_i, and also over the combinations of features F_m and F_n that were not chosen.

- **Basic:** Let q^{jk}_{-i} be the current query q with the features F_j and F_k added and the feature F_i removed, i.e., $Trade(q, F_i, F_j, F_k)$, and let q^{mn}_{-i} be q with the features F_m and F_n added and the feature F_i removed. We model the preference of the combination of features F_j and F_k over the feature F_i by the preference statement $q^{jk}_{-i} \geq q||\emptyset$, i.e., $q^{jk}_{-i} \geq q$. We model the preference of the combination of features F_j and F_k over the combination of features F_m and F_n, with q^{mn}_{-i} being satisfiable, by the preference statement $q^{jk}_{-i} \geq q^{mn}_{-i}$.
- **Importance:** We induce the preference statement $f_j f_k \geq \overline{f_j f_k}||V \setminus \{F_i, F_j, F_k\}$, which says that the presence or not of the combination of features F_j and F_k is more important than the choice of F_i. We also induce the statement $f_j f_k \geq \overline{f_j f_k}||V \setminus \{F_j, F_k, F_m, F_n\}$, for any $m, n \notin q$ that verify the conditions $\{j, k\} \neq \{m, n\}$ and $m \neq n$.

Figure 4.6 illustrates the framework for preference dominance instantiated with two different user models. When the user model is represented by weights vectors, preferences are collected as inequalities between sums of weights. For dominance computation between two configurations α and β, this instance needs to prove that $\alpha \succcurlyeq_w \beta$ for all weights vectors w that comply with all these inequalities. When the user model is represented by cp-trees, preferences are collected as preferences

between partial assignments, as described in Section 4.5.2. For dominance computation between two configurations α and β, this instance needs to prove that $\alpha \succcurlyeq_\sigma \beta$ for all cp-trees σ that comply with all the collected preferences.

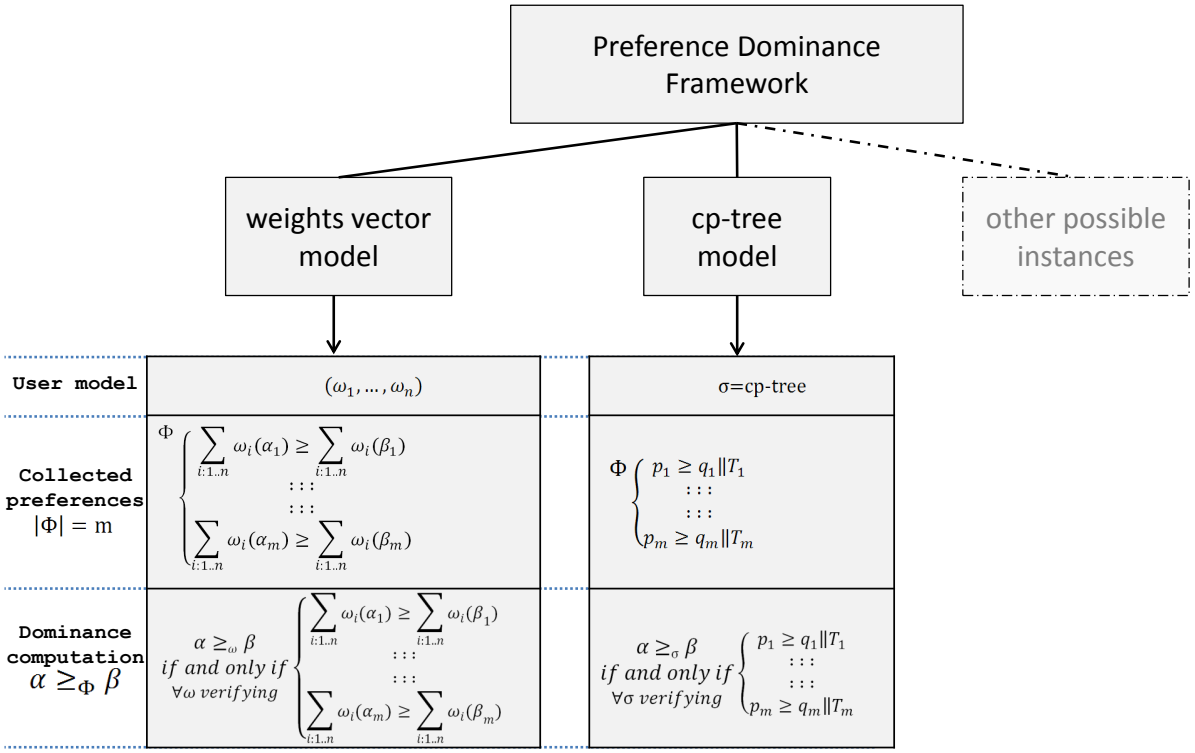


Figure 4.6: A preference dominance framework

4.7 Experimentation and Comparative Study

To compare the two instances described above, we need a methodology for the comparison to be as fair as possible. It would be useful to know more about the relation between dominance relation in the comparative preferences-based dominance and the utility-based dominance. For instance, we cannot confirm that the dominance in the first formalism implies the dominance in the formalism or vice versa. In such situations, it could be helpful to experimentally compare the two preference formalisms.

It is important that the preference dominance framework presented in Section 4.3 is implemented and tested. This section describes experiments to validate the two preference dominance approaches: the sum of weights-based approach described in Section 4.4 and the cp-tree model-based approach described in 4.5. By these experiments we aim at showing the feasibility and the applicability of the framework in

general and particularly the efficiency and suitability of the comparative preferences-based approach since this is the first time it is applied in the context of RS.

These experiments illustrate how a RS can exploit the expressiveness of comparative preferences and their relatively fast preference dominance engine.

In the remainder of this section, we provide detailed results of our experiments and comparative study. Section 4.7.1 gives different choices and settings adopted during the experiments. Section 4.7.2 covers the results of experiments and the comparative study of the simulated deployment of the two approaches presented in Section 4.4 and Section 4.5.

4.7.1 Settings

4.7.1.1 Offline experiments

We report experiments with simulated users. We validate the two instances of the preference dominance framework presented in Section 4.3 through simulations of user-system interactions. The ultimate evaluation and validation of the preference dominance approaches for conversational RSs should be performed online. However, experiments with real users cannot be used to extensively test alternative newly-deployed interaction control algorithms.

Indeed, some researchers pointed out the limitations of offline experiments and their evaluation mechanisms, whereas others argued that offline experiments are attractive because they allow comparing a wide range of approaches at an affordable cost (Shani & Gunawardana 2011). Besides, the quality of RSs cannot be directly measured because there are too many different objective functions as well (Jannach et al. 2011).

4.7.1.2 Products

We use two separate product databases that were retrieved from the Web, each describing hotels by their amenities expressed as Boolean features such as *airport shuttle*, *pets permitted*, *restaurant on-site*. Some details of the product databases are given in Table 4.1. Many hotels offer the same amenities, which explains the difference between the number of (physical) hotels and, from an amenities point of view, the number of (*distinct*) products. The Marriott-NY database records 9 features about 81 hotels; many offer the same amenities, and so there are 36 distinct products in the database. The Trentino-10 database records 10 features for 4056 hotels, of which 133 are distinct.

Table 4.1: Two databases of hotels

Name	Features	Hotels	Products
Marriott-NY	9	81	36
Trentino-10	10	4056	133

4.7.1.3 The initial query

In general, IR-based systems do not have to worry about the cold start problem (Good et al. 1999, Maltz & Ehrlich 1995, Park et al. 2006, Schein et al. 2002) as no user input is required to bootstrap the system. In (Bridge & Ricci 2007), the initial query in each dialogue was non-empty but small, and was randomly generated in a way that was compatible with the user's true preferences. In the dialogues that will be used in our experiments here, the initial query in each dialogue is empty. While this is less realistic, since real users usually make a few selections in Web search forms before first submitting, we found this to be the best way of ensuring that our comparisons of the two preference models are fairly performed.

4.7.1.4 User Modeling

Providing personalized recommendations to users requires modeling of their preferences, and needs. This information is referred to in the literature as the user model (Kobsa 2001, Fischer 2001). In order to evaluate algorithms offline, it is necessary to simulate the online process where the system makes recommendations to the user. This simulates the knowledge of how a user will make a selection and which recommendation a user will act upon. There are a number of ways of simulating users (Jannach et al. 2011).

User modeling is a cross-disciplinary research field that attempts to construct models of human behaviour within a specific computer environment which requires interaction with the system (Fischer 2001, Berkovsky et al. 2008). Some approaches of modeling user preferences have already been applied to RSs (Jannach et al. 2011). They mainly adopt techniques and methodologies from AI, Knowledge Engineering, or Data Mining, like ontological user profiling (Middleton et al. 2004), or from Statistics (Zukerman & Albrecht 2001). New ideas and approaches of User modeling (UM) appear throughout the literature and reveal that UM emerges as a significant functional means to enhance the performance of RSs (Berkovsky et al. 2009). Using advanced user models, we can execute simulations of users' interactions with the system, thus reducing the need for expensive user case studies and online testing (Ricci et al. 2011*a*).

We make assumptions concerning the behaviour of users, which could be regarded as a user model for a specific application. The simulated users in our experiments behave as described in Section 4.2.4. For a simulated user to make choices about which among the queries in the recommender's advice is the best one for it to submit next, the simulated user must be assigned a set of *true preferences*. In the frameworks described in Section 4.3, the user's true preferences can be modeled as a weights vector or a cp-tree.

The user's true preferences are represented either in the weights vector model by randomly generating weights vectors over product features or in the cp-tree model by randomly generating cp-trees over product features. The weights are related to product features; they are randomly selected real numbers in the interval [0,1]. The cp-trees representing the user's true preferences have the same structure and aspects as the cp-tree described in Section 3.4.2.1 in Chapter 3.

Every node N is associated a tuple <A_N, a_N, Y_N, $\succeq_N$>, where $A_N \subseteq V$ is a set of variables, $a_N \in \underline{A_N}$ is an assignment to those variables, $Y_N \subseteq \{V - A_N\}$ is a non-empty set of variables whose maximum number is equal to γ; $\succeq_N$ is a weak order on the set $\underline{Y_N}$ of values of Y_N which is not equal to the trivial full relation on $\underline{Y}$; so there exists some $y, y' \in \underline{Y}$ with $y \not\succeq_N y'$. A root node $N1$ is created first. Node $N1$ is associated a subset of variables $Y_{N1} \subseteq V$. $A_{N1} = \emptyset$. $\succeq_{N1}$ is generated randomly in such way that there exists some $y, y' \in \underline{Y_{N1}}$ with $y \not\succeq_{N1} y'$. Y_{N1} is instantiated with a different assignment in each of the node's children if the likeliness of the instantiation of that assignment (or value) is greater or equal to some threshold (e.g., threshold = 0.6). Once instantiated to some value $y \in \underline{Y_{N1}}$, this value is associated to some edge that links the current node (i.e., $N1$) to a next node $N2$ with a tuple <A_{N2}, a_{N2}, Y_{N2}, $\succeq_{N2}$>, where $A_{N2} = A_{N1} \cup Y_{N1}$ is a set of variables, $a_{N2} = y$ is an assignment to those variables, $Y_{N2} \subseteq \{V - A_{N2}\}$ is a non-empty set of variables that is chosen randomly among $\{V - A_{N2}\}$. The instantiation of Y_{N2} is generated randomly (in a similar way the previous node was instantiated). $\succeq_{N2}$ is also generated in the same way the local value ordering (i.e., $\succeq_{N1}$) is generated. All remaining nodes in the cp-tree are created in a similar fashion. A leaf node is reached when all variables in V are instantiated or when a no value was instantiated for a subset of variables associated with a node (e.g., the likeliness of all values associated with a node was below a specified threshold). Therefore, we obtain a cp-tree that represents a (simulated) user's true preferences.

Settings in the experiments arrange that the user's true preferences are known to the simulated user and used by that user for query selection, but they are not known to the RS which knows only about the constraints on the user's preferences that it infers each time the user sends a request.

The RS knows only what it induces about user preferences and adds to Φ when

observing user query behaviour. Those preferences are elicited under the assumption that the user's behaviour follows a kind of model: a weights vector over product features or a cp-tree whose nodes represent product features or subsets of product features.

If the true preferences are represented in the weights vector model, then recommenders that induce constraints on preferences in the sum of weights-based approach may have an advantage over recommenders that are using the cp-tree model, and vice versa. We talk about this aspect later in the analysis of results obtained from experiments in which we pair both ways of representing true preferences with recommenders that use both ways of representing induced preferences.

While we discuss relatively simple user models (for instance a vector of weights or a cp-tree), it is possible to suggest more complicated models for user behaviour (Mahmood & Ricci 2007). However the other side of the coin is that when the user model is inaccurate and too difficult to verify (e.g., when it is too complicated), we may optimize a system whose performance in simulation would not have a strong correlation with its performance in practice (Shani & Gunawardana 2011).

4.7.1.5 System runs

In the experiments, one RS uses the sum of weights model and so adopts the sum of weights-based approach (denoted by *Sum of Weights* in the tables showing the results of the experiments); six others use the cp-tree model and so adopt the comparative preferences-based approach, differing on which of the two alternative forms of preference statements they infer (Basic or Importance (denoted by *Comp. Prefs. Basic* and *Comp. Prefs. Importance* in the tables showing the results of the experiments)), and on their value for γ (1, 2 or 3). For each pairing of a user with a RS, we ran 500 simulated dialogues. In total then, we are reporting results for 2 databases $\times$ 2 ways of representing true preferences $\times$ 7 recommenders $\times$ 500 dialogues, which is 14,000 runs of the system. Each scenario has to be repeated several times in order to draw accurate and reliable conclusions. The Experiments design aim at inducing accurate and reliable conclusions.

4.7.1.6 The pruning

The use case considered in this dissertation, IR, is a conversational RS with the particularity that it does not prune products but queries, which lead to products in the database if satisfiable. The system will keep only those which are not dominated, by any other next possible query, regarding the user's preferences collected so far during the dialogue between the user and the system.

Hence, in the experiments we compare the pruning rates achieved by using the

sum of weights model with those achieved by the six RSs that use the cp-tree model. The pruning rate is defined as follows:

$$\textit{pruning rate} = \frac{|Satisfiables \setminus Undominated|}{|Satisfiables|} \times 100 \tag{4.1}$$

Equation 4.1 shows the extent to which an approach eliminates what it takes to be inferior satisfiable candidate queries from its advice. In general, the shorter the advice the better, as this reduces the set of options available to the user.

4.7.2 Comparative study

In order to perform an effective evaluation of the two preference dominance approaches within offline experiments, we select a group of evaluation criteria to assess the advisor. We identified influential success factors leading to user satisfaction behind the two approaches. The selected features, on which the approaches are compared, are meant to be appropriate for making choices between algorithms. We focus on some particular aspects and discuss them in the context of evaluating conversational RSs.

This section focuses on comparative studies, where two approaches are compared using several evaluation metrics. We review two types of experiments regarding the way the user's true preferences are represented, starting with the first set of experiments, where the user's true preferences are represented as a vector of weights, then the second set of experiments, where the user's true preferences are represented as a cp-tree. To analyze the particularities of each approach on the one hand, and the similarities and differences between the two approaches on the other hand, we computed the average over each set of runs, that is for each recommender and database run for a number of dialogues.

One feature brought by the first set of experiments is the ability to assess the behaviour of the two approaches when the user is depicted by a simple and commonly used user model, that is a weights vector. For instance, this eventually allows a fair comparison to be made between these two approaches and other preference dominance approaches for conversational RSs in the literature. The second set of experiments is gathering almost the same measures as the first set of experiments, with the user's true preferences modeled as a cp-tree (described in Section 3.4.2.1 of Chapter 3). These experiments gain their relevance from the fact that this is the first time user models are represented by cp-trees. They also give a flavour of the behaviour of users having such preferences structure.

These two groups of experiments give an additional proof of the robustness of experimental results and consequently reliable conclusions that can be drawn from

the analysis of these results.

4.7.2.1 Representing true preferences in the sum of weights model

In our first set of experiments, we set the user's true preferences by randomly generating weight vectors over product features. The pruning rates in this case are shown in Table 4.2. The table shows that, in most settings, the comparative preferences-based approach (i.e., Basic or Importance) is pruning non-optimal queries very slightly more than the sum of weights-based approach (the exceptions being Basic with $\gamma = 2$ or 3 when the pruning is very much less). For example, the cp-tree model using *Basic* preference statements and with $\gamma = 1$ eliminates 87.5% of satisfiable candidates in dialogues about the Marriott-NY database, whereas the sum of weights-based approach prunes 87.38%. The table also shows that, overall, with the comparative preference model, the amount of pruning increases as the preference statements induced become less conservative (from Basic to Importance). For example, in the *Trentino-10* part of Table 4.2, with $\gamma = 2$, pruning goes from 16.51% Basic to 87.57% Importance. (The very slight exception to this for the Trentino-10 $\gamma = 1$ case is probably due to random variation in the tie breaking.)

Table 4.2: The pruning rates (true preferences represented in sum of weights model)

	γ =1	γ =2	γ =3
Marriott-NY			
Comp. Prefs. Basic	87.50	14.48	12.65
Comp. Prefs. Importance	87.50	87.49	87.42
Sum of Weights		87.38	
Trentino-10			
Comp. Prefs. Basic	87.49	16.51	13.98
Comp. Prefs. Importance	87.42	87.57	86.72
Sum of Weights		85.72	

Furthermore, we can see that the parameter γ (the maximum number of variables that are associated with a node in a cp-tree) affects the degree of pruning. Specifically, as γ increases, the number of queries pruned tends to decrease. For example, in the *Marriott-NY* part of Table 4.2, with preference statements Importance, pruning goes from 87.50% ($\gamma = 1$) to 87.49% ($\gamma = 2$) to 87.42% ($\gamma = 3$). This is a reflection of the monotonicity with respect to γ observed above. (However, pruning deals with strict dominance, which is not necessarily monotonic with respect to γ, but will be very often, because of the monotonicity of dominance). The effect is especially marked in the Basic model where the pruning rate falls from nearly 90% to around

Table 4.3: The average number of steps per dialogue (true preferences represented in weights vector model)

	γ =1	γ =2	γ =3
Marriott-NY			
Comp. Prefs. Basic	6.004	5.998	6.000
Comp. Prefs. Importance	6.008	6.006	5.998
Sum of Weights		6.001	
Trentino-10			
Comp. Prefs. Basic	6.726	6.784	6.798
Comp. Prefs. Importance	6.748	6.756	6.790
Sum of Weights		6.790	

Table 4.4: The same final query rate (true preferences represented in weights vector model)

	γ =1	γ =2	γ =3
Marriott-NY			
Comp. Prefs. Basic	98.00	98.80	99.40
Comp. Prefs. Importance	99.00	98.80	99.60
Trentino-10			
Comp. Prefs. Basic	92.60	96.20	98.40
Comp. Prefs. Importance	92.00	93.00	97.40

16.5% or less. When γ is increased from 1 to 2, many queries of the *Trade* form become undominated in the Basic model, because of the more expressive preference relations which can be represented by cp-trees with $\gamma = 2$ (allowing more than one feature to be assigned at a node). With the stronger Importance preference form, these *Trade* queries are still dominated.

What is also of concern from a practical point of view is the average number of queries that the system gives, for example, the number of options available to the user; this is inversely related to the pruning rate. Except in the cases where pruning is very low (Basic with $\gamma = 2$ or 3), advice from the sum of weights-based recommenders are slightly longer than it is in the case of the comparative preferences-based recommenders, being around 10 for both datasets.

Table 4.3 shows that dialogue lengths are very similar in the case of all recommenders: around 6 steps on average for Marriott-NY, and around 6.8 steps for Trentino-10.

It is not enough to know that one approach prunes more than another, or gives shorter advice, or gives rise to shorter dialogues. If it were doing so at the expense of other factors, in particular the ability of the user to reach the best product, then

Table 4.5: The average shortfalls (true preferences represented in weights vector model)

	γ =1	γ =2	γ =3
Marriott-NY			
Comp. Prefs. Basic	0.0005	0.0000	0.0000
Comp. Prefs. Importance	0.0003	0.0001	0.0002
Sum of Weights		0.0000	
Trentino-10			
Comp. Prefs. Basic	0.0070	0.0010	0.0005
Comp. Prefs. Importance	0.0070	0.0050	0.0020
Sum of Weights		0.0005	

Table 4.6: The computation time (true preferences represented in weights vector model) in millisecond (ms)

	γ =1	γ =2	γ =3
Marriott-NY			
Comp. Prefs. Basic	46.10	1529.11	2937.20
Comp. Prefs. Importance	44.92	205.61	2466.01
Sum of Weights		59.73	
Trentino-10			
Comp. Prefs. Basic	195.70	6978.69	7493.94
Comp. Prefs. Importance	129.56	726.56	11053.50
Sum of Weights		289.53	

the extra pruning would be less valuable. We have measured the extent to which the final queries that the user reaches in a dialogue (and hence the final product that she might choose) agree across the different recommenders. We find (see Table 4.4) that the comparative preferences-based approaches (i.e., Basic and Importance) agree with the sum of weights-based approach between 92 and 99% of the time, and the more an approach prunes, the less this agreement is. For example, for Trentino-10, Basic $\gamma = 1$ agrees with sum of weights-based approach 92.6% of the time; this rises to 96.2% for $\gamma = 2$; and it falls to 92% for Importance $\gamma = 1$.

When true preferences are represented in the weights vector model, we can also measure the amount by which the utility of the product that the user ultimately chooses falls short of the utility of the best product that she could have reached, normalized by the difference between the products of highest and lowest utility: see Table 4.5. Unsurprisingly, these follow a similar pattern to the percentage agreements reported in the previous paragraph. The values are very close to zero, ranging from 0 to 0.007.

Table 4.7: The pruning rates (true preferences represented in cp-tree model)

	γ =1	γ =2	γ =3
Marriott-NY			
Comp. Prefs. Basic	85.77	14.28	14.28
Comp. Prefs. Importance	85.78	85.77	85.75
Sum of Weights		85.73	
Trentino-10			
Comp. Prefs. Basic	86.81	15.03	14.94
Comp. Prefs. Importance	86.81	86.79	85.93
Sum of Weights		85.15	

Table 4.6 shows the computation time taken by the different pruning implementations. When $\gamma = 1$, the time taken by the different pruning implementations is roughly similar—for example, around 0.2 seconds for a dialogue with the basic comparative preferences pruning for the Trentino dataset—with the sum of weights linear programming algorithm taking a little longer than the two others. We can see the computation time increasing very strongly with γ, from $\gamma = 1$ to $\gamma = 2$. For example, for the Basic-Trentino the time increases by more than 30 times from 195ms ($\gamma = 1$) to 6,978ms ($\gamma = 2$). This is partly due to the fact that the complexity of the dominance algorithm is exponential in γ.

Overall, for this experimental setup it seems that it is better to use the more restrictive set of models corresponding to $\gamma = 1$, at least for the Basic form, because it then generates much greater pruning (than the models with $\gamma = 2$ or 3), leading to manageable sets of options for the user, and is computationally cheaper. However, there may be situations (e.g., other datasets) where the more cautious reasoning corresponding to $\gamma = 2$ or 3 might pay off in terms of the final quality of solutions.

4.7.2.2 Representing true preferences in the cp-tree model

In our second set of experiments, the user's true preferences are set by randomly generating cp-trees over product features; more precisely, we use 1-cp-trees, i.e., with $\gamma = 1$. The pruning rate results in this case are shown in Table 4.7.

The results in Table 4.7 pattern in a very similar way to the ones illustrated in Table 4.2. For example, again, in most settings, the comparative preferences-based approach (i.e., Basic and Importance) is pruning non-optimal queries very slightly more than the sum of weights-based approach; also, with the comparative preference model, the amount of pruning increases as the preference statements induced become less conservative (from Basic to Importance); and, as γ increases, the number of queries pruned tends to decrease.

Table 4.8: The computation time (true preferences represented in cp-tree model) in millisecond (ms)

	γ =1	γ =2	γ =3
Marriott-NY			
Comp. Prefs. Basic	29.67	276.20	1076.61
Comp. Prefs. Importance	28.63	134.77	1776.30
Sum of Weights		39.35	
Trentino-10			
Comp. Prefs. Basic	38.52	1578.89	1780.70
Comp. Prefs. Importance	37.37	184.20	3086.39
Sum of Weights		55.72	

With users' true preferences represented as cp-trees, the pruning rate (Table 4.7) is roughly the same as when they are represented by weight vectors (Table 4.2), a little less for the $\gamma = 1$ case. The consequences of this for these experiments is slightly longer advice (around 13 queries on average for both datasets and in all settings except Basic with $\gamma = 2$ or 3, where pruning is very low) and shorter dialogues (around 3.7 steps for Marriott-NY and around 3.9 steps for Trentino-10). Table 4.8 shows that computation time in these settings shows a similar pattern as when the users' true preferences are represented by vectors of weights. Furthermore, in every dialogue, the product that the user ultimately chose was the optimal product for her true preferences (whereas we saw very small shortfalls in utility in the experiment described in the previous section). The dialogue involves features being added incrementally to the empty initial query—with the most important (according to the true cp-tree) first—until the optimal product is reached.

We were interested to see whether mis-matches between the ways in which true preferences and induced preferences are represented would have any effect. One might expect if induced preferences are represented in the same model as the true preferences, then they can capture more accurately the true preferences, resulting in greater pruning. But we are not seeing this in our experiments. We are seeing that, for most settings, irrespective of the way in which true preferences are represented, using the comparative preferences-based approach for induced preferences results in very slightly greater pruning.

4.8 Generalization for Non-Boolean Features

In Section 4.5.2, we present the constraint language that the system uses in the cp-tree model-based approach to induce preferences when it observes the user's queries. One preference form provided by this language, which is called *Importance*, allows

the system to infer the user preferences as comparative preferences that involve only binary features. In fact, when the user model is represented by cp-trees, preferences in *Importance* form are collected as preferences between partial assignments that represent combinations of binary feature values. These preference statements involve at most two feature values. We want to define similar preference statements which can involve more than two values for each feature. Hence, this kind of preference statements would be applicable for domains that deal with multi-valued features.

In this section, we assume that the products are modeled with a collection of *n* non-Boolean-valued features $V = \{F_1, \ldots, F_n\}$. These features are intended to relate to a set of products that the user is interested in choosing between. For example, if the product is a hotel, one feature might be the size of a swimming pool in the hotel (e.g., the swimming pool can be small, medium or large).

There are several approaches that the system can implement to induce the user preferences from the user's action or selection. A number of these approaches were presented in Section 2.10 in Chapter 2. In the present section we are interested in two among these approaches: *More Like This (MLT)* and *Partial More Like This (PMLT)*.

Example Let $V = \{F_1, F_2, F_3\}$ be the set of features having the following domains. $\underline{F_1} = \{f_{11}, f_{12}, f_{13}, f_{14}\}$, $\underline{F_2} = \{f_{21}, f_{22}, f_{23}\}$ and $\underline{F_3} = \{f_{31}, f_{32}, f_{33}\}$. Suppose we have 5 products to be presented to the user who has to choose only one product, denoted by *c*, among these products. Then, the system induces some user preferences with regards to the user's selection. The set of products to be shown are as follows: $p_1 = f_{12}f_{21}f_{33}$; $p_2 = f_{11}f_{21}f_{33}$; $p_3 = f_{13}f_{22}f_{33}$; $p_4 = f_{14}f_{23}f_{32}$ and $p_5 = f_{14}f_{22}f_{31}$.

The *MLT* approach states that any feature value f_{ij} that is contained in the chosen product *c* is preferred over any other feature value f_{ik} ($k \neq j$) of the feature *i*. For instance, if the user is shown the 5 products mentioned above and she chooses the product p_3 then the system will induce the following preference statements:

- $f_{13} \geq f_{12} || V \setminus \{F_1\}$;
- $f_{13} \geq f_{11} || V \setminus \{F_1\}$;
- $f_{13} \geq f_{14} || V \setminus \{F_1\}$;
- $f_{22} \geq f_{21} || V \setminus \{F_2\}$;
- $f_{22} \geq f_{23} || V \setminus \{F_2\}$;
- $f_{33} \geq f_{32} || V \setminus \{F_3\}$;
- $f_{33} \geq f_{31} || V \setminus \{F_3\}$.

The *PMLT* approach says that any feature value f_{ij} that is contained in the chosen product *c* is preferred over any other value f_{ik} ($k \neq j$) of the feature *i* provided the value f_{ik} is not contained in any rejected product. In this example, in case where the user chooses the product p_3 the system would induce the following preference statements:

- $f_{13} \geq f_{12} || V \setminus \{F_1\}$;
- $f_{13} \geq f_{11} || V \setminus \{F_1\}$;
- $f_{13} \geq f_{14} || V \setminus \{F_1\}$;

The set of preference statements induced with this approach is smaller than the one induced with the *MLT* approach as only the value f_{13} (assigned to the feature 1 in the chosen product) does not appear in any rejected product. But, it is the case for the values f_{22} and f_{33} (assigned to features 2 and 3 in the chosen product) since f_{22} appears in product p_5 and f_{33} appears in product p_1 and p_2.

In this section, we adopt similar approaches with comparative preference statements. We have derived three forms of preference statements the system that can induce when the user makes her selection. These preference forms are inspired from the the two approaches *MLT* and *PMLT*.

Let $V = \{F_1, F_2, F_3\}$ be a set of variables that represent features, *p* be a product, which is also denoted as $\{f_{p1}, f_{p2}, f_{p3}\}$ if it has the values f_{p1}, f_{p2} and f_{p3} for variables F_1, F_2 and F_3 respectively. Similarly a product *c* is also denoted as a combination of feature values $\{f_{c1}, \ldots, f_{cn}\}$ if it has the values $\{f_{c1}, \ldots, f_{cn}\}$ for variables in $\{F_1, \ldots, F_n\}$ respectively.

When the user chooses a product $c = \{f_{c1}, \ldots, f_{cn}\}$ and rejects another product $d = \{f_{d1}, \ldots, f_{dn}\}$ among a set of *k* non-dominated products that are shown to her, the system induces preference statements whose form depends on the following methods.

- **Basic:** We model the preference of the combination of feature values included in the chosen product *c* over the combination of feature values included in any rejected product *d* by the preference statement $\{f_{c1}, \ldots, f_{cn}\} \geq \{f_{d1}, \ldots, f_{dn}\} || \emptyset$.
- **Importance1:** We induce the preference statements $f_{ci} \geq f_{di} || V \setminus \{F_i\}$, for any value f_{di} assigned to feature F_i in a rejected product d.

  This states the superiority of every feature value taken by the chosen product *c* (i.e., f_{ci}) over any other value (of the same feature) that appears at least in one rejected product *d* (i.e., f_{di}).

Table 4.9: Recapitulative table of results (users as weights vectors)

	Prun	Time	NbStep	Fall
Basic	2.60	0.0293	5.25	9.23
Importance1	52.67	0.329	4.28	9.91
Importance2	0.018	0.035	5.33	9.18

- **Importance2:** Here we induce the preference statements $f_{ci} \geq x || V \setminus \{F_i\}$ for all values x of feature F_i that appear at least in one rejected product, for every value f_{ci} assigned to feature F_i in the chosen product c which is not present in any rejected product.

  This states the superiority of every feature value f_{ci} taken by the chosen product *c*, and which does not appear in any rejected product, over any other possible value x of the same feature that appears at least in one rejected product. The difference with *Importance1* form is that the preferred feature value needs to be present in the chosen product only.

We have generated random products with *n* variables having three values each. Having simulated the users' true preferences as vector of weights, we want the system to recommend suitable products to the user with regards to her preferences that are induced during a simulated user-system dialogue. Indeed, given a global set Ω of products, the system computes *k* non-dominated products and shows them to the user in addition to the product that the user selected in the previous step of the dialogue. When the user selects one product, the system induces preferences according to one of the above methods (i.e., *Basic*, *Importance1* and *Importance2*). If the user is not satisfied and there are still products in the global set of products then the system computes the next *k* non-dominated products and does the same as in the previous step. This dialogue continues until the user is satisfied (by either choosing the same product a number of times (e.g., 3 times) or she gets the product with the maximum possible utility with regards to her true preferences). The dialogue also ends when all products in Ω have been retrieved by the system.

We performed experiments with 1000 simulated users whose true preferences are represented by weights vectors. In Tables 4.9 and 4.10, we report averages (over 1000) of the following measures: the pruning rate (*Prun*), the running time (*Time*), the number of steps in a dialogue (*NbStep*) and the shortfalls (*Fall*). In Table 4.10, we inferred lexicographic models, which are described in Section 4.9.3, related to the methods above.

Table 4.9 shows *Importance1* (52.67) is pruning much more than *Importance2*

Table 4.10: Recapitulative table of results with lexicographic models (users as weights vectors)

	Prun	Time	NbStep	Fall
Basic	18.07	0.1533	4.96	9.43
Importance1	52.67	0.3327	4.28	9.91
Importance2	0.018	0.0343	5.33	9.18

(0.018) whose pruning is even weaker than *Basic* (2.60). Table 4.9 shows that *Importance2* does not prune that much compared to *Importance2* (0.018 vs 52.67). Besides, *Basic* is pruning very little compared to *Importance1* (2.60 vs 52.67) in less time (0.02ms vs 0.33ms). *Basic* communicates with the user within longer dialogues (5.25 vs 4.28) but with a bit larger shortfalls (9.23 vs 9.91) regarding *Importance2*.

Table 4.10 shows that only *Basic* with lexicographic models has significantly improved its pruning rate (18.07 vs 2.60). This is justified by the fact that the lexicographic version of *Basic* produces much stronger preference statements than those produced by *Basic* without applying lexicographic models. We can see, in Table 4.10, that the lexicographic inference does not affect the measures (e.g., pruning rate) for *Importance1* and *Importance2* as the preference statements adopted by these methods are very similar to the lexicographic model.

4.9 Other Kinds of Models of Preferences

This chapter focuses on two kinds of models of user preferences, one based on a weighted sum, the other based on comparative preferences. Depending on what the recommender is dealing with as products and the context of use of the system, other types of user models can be used in an attempt to tune some features of the set of user models then have kind of control on the capacity of pruning of the RSs. This section briefly discusses some other possible sets of models.

4.9.1 Larger sets of models

One way to get a large set of models, that represents possible ways the user can be is just to assume that the user's preferences are a total pre-order over possible configurations. This pattern of user models is adopted in the case of CP-nets for the standard inference for CP-nets described in Section 2.6.3 of Chapter 2. Then, the dominance relation in terms of the framework described in Section 4.3 is defined as follows: Let α and β be two configurations. α dominates β (i.e., $\alpha \succcurlyeq_\Phi \beta$) if and only if all "possible" users that agree with the set of constraints in Φ, prefer α over

β. A possible user in this instance is represented by a total pre-order. Therefore, $\alpha \succcurlyeq_\Phi \beta$ holds if and only if every total pre-order that satisfies all constraints in Φ orders α before β. In other words, $\alpha \succcurlyeq_\Phi \beta$ holds if every total pre-order that extends all constraints in Φ has α coming before β. When the number of models (i.e., total pre-orders) get larger, the dominance is more likely to be harder as there is a need for more models to comply with Φ and order α before β. Therefore, inference is more likely to be weaker.

Another way of modifying the size of the set of models in the approach presented in Section 4.5 is to tune the parameter γ in $\mathcal{M}(\gamma)$; when γ increases, $\mathcal{M}(\gamma)$ gets larger and so inference becomes weaker.

Multi-attribute decision theory (Keeney & Raiffa 1976) takes utility function representations of the preferences of a decision-maker and studies mathematical methods for decomposing these functions into weighted combinations of subfunctions corresponding to different attributes or sets of attributes (see for example (von Stengel 1988, Hansson 1989, Wellman & Doyle 1992)).

One instance is Generalised Additive Independence (GAI) (Fishburn 1967, Bacchus & Grove 1995): a compact model that represents utility as a sum of factors. Each factor can be a component weight function that depends on at most γ attributes. The user model in the sum of weights-based approach in this chapter corresponds to the case where $\gamma = 1$. For $\gamma = 2$, the set of models is larger and much more expressive. It will then lead to a substantially more conservative (weaker) dominance relation.

The size of the set of models has in general a direct impact on the strength of the pruning of a dominance relation: when the set of models gets much larger the pruning capacity tends to get smaller (i.e., weaker inference). This is because more models have to agree on both set of constraint Φ and $\alpha \succcurlyeq \beta$ (α and β are two items or configurations) at the same time which is more difficult to be true when the set of models gets larger. On the other hand, if the set of models gets much smaller then the pruning capacity does not necessarily get stronger because of eventual equivalences between items. These equivalences come from relatively frequent valid dominance relations between items. Due to the small size of models, it is more likely to find dominance between two items.

The size of the set of models has to be adjusted to be able to fit to the context of use of the dominance relation (e.g., type of application and device).

4.9.2 Towards a stronger pruning

Scenarios where we need a relatively small number of undominated solutions are not rare. A small number of options helps the decision-maker focus on what she probably needs the most. It is then interesting to look for ways to prune as many

dominated solutions as required by the context of use of the system in general. There are contexts of use which need the number of outcomes to be small (e.g., mobile devices, busy screens).

One simple idea is the combination of the two dominance approaches which can lead to more pruning. Items that are found to be undominated by one approach but dominated by the other approach can be eliminated.

We can consider lexicographic models that are included in the intersection of the two sets of models generated respectively by the first and second approaches described in Sections 4.4 and 4.5. The obtained set of models will be smaller than both initial sets of models which can increase the pruning rate as the inference tends to be stronger (i.e., more queries are dominated and so eliminated).

Lexicographic models can be adopted by cp-trees that implement the same feature ordering in all branches, and by weights vectors that select weights to implement the same lexicographic model (i.e., same most important features) as the one implemented by cp-trees: to assign large values for the most important features and very small values of the less important features.

For instance, given a problem with a five feature-domain and in $\gamma = 1$, one possible way to implement such lexicographic models is to consider F_1 as more than important than F_4 which is more important than the other features. This feature ordering should be implemented in all branches of cp-trees in $\mathcal{M}(1)$. In the second approach, we can assign 6 and 5 as values to the weights of F_1 and F_4 respectively then giving value 1 to the weights of the remaining features. Thus, any outcome α having feature F_1 dominates any other outcome β that does not have feature F_1 regarding the first approach and the second approach. For the first approach, all cp-trees in $\mathcal{M}(1)$ order α before β since they consider F_1 as the most important feature and the node which has F_1 is a decisive node that validates α is preferred to β regarding the current cp-tree.

4.9.3 Lexicographic inference

Let us consider preference statements of the form $p \geq q \parallel T$, where P, Q and T are subsets of the set of features V, p is an assignment to P (i.e., a function from P to $\{0, 1\}$), q is an assignment to Q, and $P = Q$. Informally, the statement $p \geq q \parallel T$ represents the following: p is preferred to q if T is held constant.

One way to implement lexicographic models is that, when the user chooses one configuration α instead of another configuration β, the system would want to extract two tuples t_α and t_β respectively from α and β such that $\forall X \in V$ for which $\alpha(X) \neq \beta(X)$, $t_\alpha(X) = \alpha(X)$ and $t_\beta(X) = \beta(X)$. Then, the system gathers all variables $X \in V$ for which $\alpha(X) = \beta(X)$ and adds them to the set T (i.e., $T = \{X \in$

Table 4.11: The pruning rates (true preferences represented in cp-tree model)

	γ = 1	γ = 2	γ = 3
Marriott-NY			
Comp. Prefs. Basic	86.17	12.397	12.397
Comp. Prefs. Lex	87.603	87.603	87.603
Comp. Prefs. Importance	87.603	87.603	87.603
Sum of Weights	87.438		

$V, \alpha(X) = \beta(X)\}$). Therefore, the system induces preference statement $t_\alpha \geq t_\beta \parallel T$.

We performed experiments in which we adopt the same setting as stated in Section 4.7.1 with the Marriott-NY database described in Section 4.7.1.2. The number of simulated users whose true preferences are represented by cp-trees is 500. In addition to this preference model (denoted by *Comp. Prefs. Lex*), we use the two forms of preference statements presented in Section 4.6 (i.e., Basic and Importance) as well as the sum of weights-based inference.

Table 4.11 shows the lexicographic model does not appear to be affected when γ increases while *Basic* and *Importance* do. Furthermore, with lexicographic inference, the product that the user ultimately chose was the optimal product for her true preferences (see Table 4.12). It was the case for the other forms of inference except *Basic* with $\gamma = 1$. Table 4.13 shows the computation time taken by the different pruning implementations. When $\gamma = 1$, the time taken for the lexicographic inference is a bit larger than than *Basic* (i.e., 37ms against 36ms) and also *Importance* (i.e., 37ms against 35ms). Lexicographic inference is still faster than the sum of weights-based inference (i.e., 37ms against 53ms). With $\gamma = 2$ and $\gamma = 3$, the time taken for the lexicographic inference looks close to the time achieved by *Importance* with a bit of delay. Lexicographic inference is faster than *Importance* inference with $\gamma = 2$ (i.e., 194ms against 1601ms) while *Importance* infers in a faster fashion than the former method with $\gamma = 3$ (i.e., 2407ms against 2444ms). Lexicographic inference is faster than the sum of weights only with $\gamma = 1$. The monotonicity effect of the pruning with regards to γ is not observed with these lexicographic models: as γ increases, the number of queries pruned is still the same.

4.9.4 Application to group recommender systems

A group recommender system is a recommender system aimed at generating a set of recommendations that have a common social welfare goal: to satisfy a group of users (instead of a single user), with potentially competing interests. The associated challenges include considering how to elicit and combine the preferences of

Table 4.12: The average normalized shortfalls (users as cp-trees)

	$\gamma = 1$	$\gamma = 2$	$\gamma = 3$
Marriott-NY			
Comp. Prefs. Basic	0.004	0.0	0.0
Comp. Prefs. Lex	0.0	0.0	0.0
Comp. Prefs. Importance	0.0	0.0	0.0
Sum of Weights	0.0		

Table 4.13: The execution time (users as cp-trees)

	$\gamma = 1$	$\gamma = 2$	$\gamma = 3$
Marriott-NY			
Comp. Prefs. Basic	36.54	1601.73	1682.25
Comp. Prefs. Lex	37.84	194.0	2444.4
Comp. Prefs. Importance	35.13	190.73	2407.19
Sum of Weights	53.187		

different users as they engage in simultaneous recommendation dialogs. Various group recommenders were introduced in the literature (for fitness centres, cities, televisions, vacation planning): PolyLens (O'Connor et al. 2001, *MovieLens* 2013), MusicFX (McCarthy & Anagnost 1998), Intrigue (Ardissono et al. 2003), TV program recommender (Yu et al. 2006), Travel Decision Forum (Jameson et al. 2004), Pocket Restaurant Finder (McCarthy 2002).

Voting is one of the most common used ways for users to manifest their preferences and negotiation is presented as the best method for management and resource sharing in a multi-agent environment (Popescu & Pu 2011). Ratings can come after users' interaction with the system or by interpreting her preferences. Once votes are submitted the recommender system should come up with a solution corresponding to the highest scored items. In more general perspective, a compatibility score is computed for each candidate and then a group compatibility score is computed. The option with the highest overall quality score is selected as a new recommendation.

Regarding social choice, investigations have been made regarding several objectives including: high expressive power, relative succinctness, low complexity, elicitation friendliness and cognitive appropriateness (Chevaleyre et al. 2008). The comparative preferences theories formalism is then one of the most appropriate preference formalisms in such context as it was shown that it can achieve all the objectives with a single user RS. Individual recommendations are generated using the user models, which are based on the user interaction with the system. However, in practice it is

not straightforward to model groups simply by modeling the aggregation of individual models. This raises the need for voting mechanisms from social choice theories to solve the comparative preferences theories aggregation problem.

One alternative is to gather the sets of comparatives preferences of the users and reason with them to arrive at an agreement about an overall solution. We might also need more sophisticated algorithms which would increase the satisfaction function for the entire group while managing the eventual inconsistencies that arise among the sets of preferences of the different users. In summary, there is a need for a theoretical framework on voting in group recommender systems that enable reasoning with comparative preferences theories.

4.9.5 Application to other forms of conversations in recommender systems

The comparative preferences-based formalisms developed for RSs can also be part of other intelligent query selection strategies to drive the elicitation process in the RSs. Critiquing is an interaction model that allows users to build their preferences by examining or reviewing examples shown to her by the system. The preference induction methods developed in this thesis can capture several preference nuances from the user's feedback for different forms of interaction. It can, for example, handle preferences the user would have between a pair of items that she might be asked to compare by the system.

These methods can generate preference statements stating the preference of any feature or combination of features that marks the selected item(s) over any other feature or combination of features that characterizes the rejected item(s) during an interaction stage or even a sequence of interaction stages. It would also be possible to redefine these rules for more expressive comparative preference languages so that a larger scope of problems with more kinds of input can be tackled using the pruning rules; the comparative preference language described in Section 3.3 in Chapter 3 is an example.

Example 6. *Let* $V = \{F_1, F_2, F_3\}$ *be a set of features whose domains are as follows.* $\underline{F_1} = \{f_1^1, f_1^2, f_1^3\}$, $\underline{F_2} = \{f_2^1, f_2^2\}$, $\underline{F_3} = \{f_3^1, f_3^2, f_3^3, f_3^4\}$. *Let us assume that the user is initially shown the two following products:* $f_1^1 f_2^1 f_3^2$ *and* $f_1^2 f_2^1 f_3^3$. *Then, the user chooses* $f_1^2 f_2^1 f_3^3$. *The system can induce preferences in several formats. For example, the system can induce the following preference statement:* $f_1^2 f_2^1 f_3^3 \geq f_1^1 f_2^1 f_3^2 \,||\emptyset$. *This is to express the preference of the features values combination included in the chosen product over the combination of values included in the rejected product.*

4.10 Conclusions

Although there has been a lot of effort in theoretical work produced on comparative preference formalisms in recent years, including award winning papers (Boutilier et al. 2004*a*, Koriche & Zanuttini 2009), development towards applications has lagged behind. On the other hand, RSs are gaining momentum in the e-commerce applications market to face the "information overload" problem. This progressively reveals an increasing need to enable those RSs with suitable preference dominance engines that are capable of efficient preference handling that support users in expressing their preferences with a minimum of burden during an automated process that includes user-system interaction.

In this chapter, we define a formalism for preference elicitation based on comparative preferences and integrate it into a conversational RS. To our knowledge, this type of comparative preference is deployed for the first time with RSs. We conduct different experiments and perform a comparative study between a first approach based on a sum of weights-based dominance relation and a second approach based on comparative preference theories dominance relation. Comparative preference theories were able to give freedom and flexibility to the system to handle the user's preferences by allowing the system to capture preference nuances and various forms of preferences without giving up the attractive computational properties of the preference dominance. The dominance algorithm described in Section 3.5 in Chapter 3 is proven to work efficiently for a range of comparative preference statements when checking dominance between two outcomes α and β.

The suitability and attractiveness of the comparative preference theories-based approach for RSs were verified and validated through several experiments that include repeated scenarios with simulated users represented by models that are based on different semantics (e.g., weights vector vs cp-tree).

5

Constrained Optimisation for Comparative Preferences

5.1 Introduction

Constrained optimisation looks for (feasible) solutions, regarding the constraints, that best meet the user's preferences. One goal is to assist users with cognitive tasks like configuring products for an online shopper, or scheduling a meeting for a busy executive. In such situations, the automated agent needs to balance the user's desires with hard and externally imposed constraints.

On the one hand, in daily life, people's expectations from decision support systems (DSSs) are getting higher and higher; they hope for more personalized and focused query handling. On the other hand, the preference representation approaches and preference reasoning engines are progressing towards more intuitiveness and compactness. Therefore, the integration of newly developed preference languages and reasoning approaches in DSSs paves the way for effective and intuitive information streams required by users. It will build the bridge between users and today's systems.

A decoupled approach that combines CSPs and conditional preference theories, for which a preference reasoning engine was developed, seems to be worth studying particularly when the user's preferences over an assignment to some subset of variables of the problem is usually conditioned on the assignments to other subsets of

variables. Then, we intend to develop decoupled COP approaches, using conditional preference theories (i.e., cp-theories), and taking a similar approach to (Boutilier et al. 2004*b*), in the sense that they are able to: 1) find the first non-dominated solution fast as the first solution met in the preference-based search tree is considered optimal and 2) find optimal solutions in anytime-working mode algorithm as the algorithm can stop at any point and returns the set of optimal solutions found so far. The solution set generated at that point will be a subset of the set of all optimal solutions.

In B&B, the bounding rule typically requires, at each node of the search tree, a test that checks some necessary conditions for the newly created branch to achieve an optimality level not worse than the so far best obtained level. In this chapter, in order to bound the horizon of the search space, we define pruning rules that check conditions, at each node of the search tree, to see whether the newly created subproblem no longer contributes to the solution of the global optimisation problem and hence can be discarded. In other words, the conditions check whether the partial assignment constructed so far at the current node cannot be extended to an optimal (i.e., non-dominated) solution. Optimality is referring to a (partially ordered) preference relation represented by a cp-theory described in Section 3.3.1 in Chapter 3. We are thus extending B&B for cp-theories. In Sections 5.3 and 5.4 we develop a first group of sufficient conditions that help the constrained optimisation algorithm prune the search space. Such pruning will certainly aid the search for optimal solutions, without jeopardizing any potentially meritorious solutions. One of the main advantages of pruning away subspaces from the search space is the reduction of the number of pairwise comparisons performed during the search.

Another way of avoiding uninteresting comparisons is to avoid involving any optimal solution that is unable to better (i.e., dominate) any extension (complete assignment) in a sub-space. In other words, an optimal solution α, which was already found, might be unable to dominate any outcome that extends the partial assignment obtained so far in the search. If we prove that this is the case then we do not need to involve α in any dominance check below the current node. This can be performed by our second category of pruning rules presented in Section 5.5 and which temporarily disengage any subset of optimal solutions below some node of the search tree from playing a role in the comparisons performed in the sub-space newly created. This pruning rule checks whether any optimal solution already found is unable to better any other possible assignment in the sub-space. If the condition is confirmed for a solution α, then α is no longer involved in eventual comparisons below the current node in the search tree.

Section 5.6 discusses an example of computer configuration which illustrates aspects of the cp-theories-based constrained optimisation. Issues encountered during

the implementation of the pruning rules are discussed in Section 5.7. In order to assess the efficiency of the developed pruning rules in practice, experiments were performed and results are discussed in Section 5.8.

5.2 Personalized Branch And Bound

In B & B, in order to personalize the bounding rule used to check the optimality of the search space below some node in the search tree, we define specific pruning rules that are based on some kind of preference relation. B&B can be personalized by the means of conditions to be tested at each node of the search tree. We describe the model of the search tree in Section 5.2.1 and present, in Section 5.2.2, the preference relation we are considering in the preference-based B&B presented in Section 5.2.3.

5.2.1 Model of the search tree

Let V be the set of variables associated with the problem. Variables are instantiated in a particular order in the search tree used to gather optimal solutions of a CSP, which is described in Section 2.11 of Chapter 2, augmented by a set of user preferences.

Similarly to a CSP search tree, described in Section 2.11.2 of Chapter 3, each node of the search tree is associated with a variable $X \in V$. At any node of a depth-first search algorithm, we have an associated partial assignment b to the variables $B \subseteq V$ that have already been instantiated, and we have the current domain $\mathcal{D}(X)$ of each variable X. We formalise this notion of a collection of domains as follows:

Definition 21. *A* collection of domains *is a function $\mathcal{D}$ on V such that $\mathcal{D}(X) \subseteq \underline{X}$ ($\underline{X}$ is the initial domain of X), so that $\mathcal{D}(X)$ is a set of possible values of X. For an outcome β, we say that β is* of *$\mathcal{D}$ if $\beta(X) \in \mathcal{D}(X)$ for all $X \in V$.*

When a variable $Y \in V \setminus B$ is instantiated to some value y, a new node N, considered as a child of the previously created node that is associated with the previously-instantiated variable's value, is created, the partial assignment b is extended with $Y = y$ and $\mathcal{D}(Y)$ is fixed to $\{y\}$ (i.e., $\mathcal{D}(Y) = \{y\}$). Domains $\mathcal{D}(X)$, for $X \in V \setminus B$, are then determined by constraint propagation (Bessière 2006) as elements of $\underline{X}$ that are inconsistent with b regarding the constraints in the CSP are eliminated (see Definition 2 in Section 2.11 of Chapter 2). In our experiments, we use arc consistency to prune away inconsistent values from $\mathcal{D}(X)$. Backtracking takes place when a complete assignment is created or there is no point in going deeper in the search tree, because there exists X such that $\mathcal{D}(X) = \emptyset$, or there is a proof that there will not be an optimal solution that can extend the current partial assignment. Thus, the process

backtracks to the parent node to assign an untried value of the variable associated with the parent node.

Let Ω be the set of optimal solutions found so far when the search reaches some node N. Let Ω' be the subset of Ω that will be the only solutions involved in any computation or comparison below the current node. Other solutions $\beta \in \Omega \setminus \Omega'$ are irrelevant for the comparisons that will be performed below node N. The state of the search at some node of the search tree is characterized by (b, B, $\mathcal{D}$, Ω, Ω').

5.2.2 Preference relation for optimisation

In order to select optimal solutions, we need to use a preference relation $\trianglerighteq$ between outcomes. In this chapter, we use the dominance relation described in Section 3.4.2 of Chapter 3. Thus, an outcome α dominates another outcome β ($\alpha \trianglerighteq \beta$) if and only if all the cp-trees with $\gamma = 1$ that satisfy the set of user preferences Γ order α before β (a cp-tree with $\gamma = 1$ corresponds to the pos-tree introduced in (Wilson 2006) and mentioned in Definition 12 Section 3.4.2.1 of Chapter 3). Γ is a set of preference statements expressed as a conditional preference theory (see Section 3.3.1 of Chapter 3). Let V be a set of variables and U and W be two subsets of V. Each preference statement is denoted by φ as follows: $u : x > x'[W]$, ($W \subseteq (V \setminus (U \cup X))$) meaning that given an assignment u for a set of variables U, we prefer value x to value x' for variable X, as long as variables outside of W are held equal.

5.2.3 Preference-based branch and bound

In the search tree, when branching, bounding requires checking whether a partial assignment b can be extended to an undominated (i.e., optimal) solution. In order to aid the search for optimal solutions, we are looking for new preference-based methods that find the way to discover dominated paths (regarding preferences) in the search tree, and so allow for pruning uninteresting sub-spaces from the search tree.

At some node N of the search tree, an optimal solution α might be able to dominate every β of $\mathcal{D}$. Thus, we say that α dominates $\mathcal{D}$. The other side of dominance is that α might not be able to dominate any β of $\mathcal{D}$. We then say that α non-dominates $\mathcal{D}$.

Definition 22. *Let α be an outcome and $\mathcal{D}$ be a collection of domains. We define:*

- α dominates $\mathcal{D}$ *if* $\alpha \trianglerighteq \beta$ *for all* β *of* $\mathcal{D}$.
- α non-dominates $\mathcal{D}$ *if for all* β *of* $\mathcal{D}$, $\alpha \not\trianglerighteq \beta$.

The definition suggested two ways of improving the optimisation process:

- In the case where $\mathcal{D}$ is proven to be dominated, there will be no point to exploring nodes in the search tree below the current node. We do not need to look for solutions which are dominated. In Section 5.3, we define sufficient conditions for α dominating $\mathcal{D}$ that can be efficiently checked. We call them *Dominance Pruning Rules*. For pruning even more sub-spaces in the search tree, we also use a stronger but unsound dominance relation (described in Section 5.4).

- In the case where α does not dominate any β of $\mathcal{D}$, bringing α into computations below the current node will only slow the search and increase the computational burden. Thus, we do not need to carry optimal solutions that are proven to be unable to dominate any possible solution that will be created in the not yet visited sub-space below node N. In Section 5.5, we define sufficient conditions for α non-dominating $\mathcal{D}$ that can be efficiently checked. We call them *Non-Dominance Pruning Rules*.

5.3 Dominance Pruning Rules

These pruning rules aim at checking sufficient conditions that might confirm, when satisfied, that one sub-space (i.e., all possible assignments that can be created in the sub-space) below a current node N is dominated by an optimal solution already found. Then, a backtrack is required. To define such conditions, we need to recall the notion of dominance between two outcomes. One way to look for such conditions is to consider pos-trees which correspond to cp-trees with $\gamma = 1$ (see Definition 12 Section 3.4.2.1 of Chapter 3). cp-trees are described in Section 3.4.2 of Chapter 3 and used in the definition of dominance in Section 3.5 of the same chapter. In fact, for α to dominate β we need to check that all pos-trees that satisfy Γ also order α before β. This can be ensured by checking that there is no pos-tree that satisfies Γ and strictly prefers β over α. Thus, the non existence of a pos-tree that witnesses a strict preference relation of one outcome β over another outcome α is sufficient to prove that α dominates β (that is, $\alpha \trianglerighteq \beta$).

Let σ be a pos-tree (a cp-tree with $\gamma = 1$). A node r is defined to be *decisive* for outcomes β and α if it is the deepest node (i.e., furthest from the root) which is both on the path to α and on the path to β (see Section 3.4.2.3 of Chapter 3). For all variables Y already instantiated in σ before reaching node r, we have $\alpha(Y) = \beta(Y)$. We also say that node r *decides* α and β. Node r decides which of the outcomes is preferred over the other regarding σ by comparing the two outcomes' values for the root node's associated variable Y_r (i.e., $\alpha(Y_r)$ and $\beta(Y_r)$) with respect to the ordering associated with node r. For instance, if $\alpha(Y_r) \succcurlyeq_r \beta(Y_r)$ then α is preferred over β.

Let y_i and y_j be two values from $\underline{Y}$. Relation $\sqsupset_a^Y$ on $\underline{Y}$ is defined to be the transitive closure of the set of pairs (y_i, y_j) over all preference statements φ written as $u_\varphi : x_\varphi > x'_\varphi[W_\varphi]$ in Γ such that u_φ is compatible with a, $x_\varphi = y_i$ and $x'_\varphi = y_j$ (see Example 7). (y_i, y_j) means y_i is preferred over y_j. Notice that for $a = \top$, all preference statements Φ where $X_\varphi = Y$ are concerned and $\sqsupset_\top^Y$ is the transitive closure of all pairs (x_φ, x'_φ). This is because $a = \top$ is compatible with any u_φ.

Proposition 1. *(Wilson 2006) Recall that each preference statement $\varphi \in \Gamma$ has the following form: $u_\varphi : x_\varphi > x'_\varphi[W_\varphi]$, $(W_\varphi \subseteq (V \setminus (U_\varphi \cup X_\varphi)))$. Then, the following pair of conditions are necessary and sufficient for a pos-tree σ to satisfy a cp-theory Γ.*

(1) For any $\varphi \in \Gamma$ and outcome α extending u_φ: on the path to α, X_φ appears before W_φ;

(2) For all nodes r in σ, $\succcurlyeq_r \supseteq \sqsupset_{a_r}^{Y_r}$

5.3.1 The root-dominates rule

Let $\mathcal{D}$ be a collection of domains and β be any outcome of $\mathcal{D}$. To prove that α dominates $\mathcal{D}$ (that is, $\alpha \trianglerighteq \beta$ for all β of $\mathcal{D}$), it is sufficient to present evidence that there is no pos-tree that attests β strictly dominates α, for all β of $\mathcal{D}$. If it is the case, then we have $\alpha \succcurlyeq_\sigma \beta$ for all σ satisfying Γ, which can be expressed as $\alpha \trianglerighteq \beta$, for all β of $\mathcal{D}$.

We look for this evidence in the root node of the pos-tree. In fact, when the root node in a pos-tree σ that satisfies Γ decides α dominates β then we can conclude that σ prefers α over β. If there are no pos-trees that satisfy Γ and also strictly prefer β over α, for any β of $\mathcal{D}$, then we can say that all pos-trees that satisfy Γ prefer α over β, for all β of $\mathcal{D}$. Then, we need to define sufficient conditions that guarantee the root node of every pos-tree that satisfies Γ decides α dominates β, for all β of $\mathcal{D}$.

There are two cases where the root node *r* reveals that β is not strictly preferred over α, for all β of $\mathcal{D}$:

(i) If α and β have the same value for the variable associated with the root node (i.e., Y_r), then *r* should have no children associated with the assignment $Y_r = \alpha(Y_r)$. This is guaranteed by having $\alpha(Y_r)$ and y are $\sqsupset_\top^Y$-equivalent for some $y \in \underline{Y_r} - \{\alpha(Y)\}$ (see Section 3.4.2.1 of Chapter 3);

(ii) if $\alpha(Y_r) \neq \beta(Y_r)$, then $\alpha(Y_r)$ should be better than the possible values that β can have for Y_r (i.e., $\alpha(Y_r) \succcurlyeq_r y$, for all $y \in \mathcal{D}(Y_r) - \{\alpha(Y)\}$).

Therefore, we say that α root-dominates a collection of domains $\mathcal{D}$ if: for all $Y \notin \bigcup_{\varphi \in \Gamma} W_\varphi$,

(i) $\alpha(Y) \sqsupset_{\top}^{Y} y$ for all $y \in \mathcal{D}(Y) - \{\alpha(Y)\}$;

(ii) if $\alpha(Y) \in \mathcal{D}(Y)$ then $\alpha(Y)$ and y are $\sqsupset_{\top}^{Y}$-equivalent for some $y \in \underline{Y} - \{\alpha(Y)\}$.

The following result states the soundness of the root-dominates rule.

Proposition 2. *If α root-dominates $\mathcal{D}$ then α dominates $\mathcal{D}$, i.e., $\alpha \trianglerighteq \beta$ for all β of $\mathcal{D}$.*

Proof: Let σ be any pos-tree satisfying Γ and let β be any element of $\mathcal{D}$. Let r be the root node of σ. Let us look into σ and focus more particularly on the root node r that decides α and β. Node r has an associated variable Y_r. Y_r cannot belong to W_φ of any preference statement φ satisfying Γ and written as $u_\varphi : x_\varphi > x'_\varphi\,[W_\varphi]$. This is because there will be at least one variable (X_φ) that should appear before variables in W_φ in σ (Proposition 1(2) in Section 5.3). There are two possibilities:

- $\alpha(Y_r) \neq \beta(Y_r)$: Then r decides α and β. Condition (i) implies $\alpha(Y) \sqsupset_{\top}^{Y} \beta(Y)$. This implies $\alpha(Y_r)$ and $\beta(Y_r)$. Thus, α and β differ on the root node r with α having better value on Y_r. Then, we can conclude $\alpha \succcurlyeq_\sigma \beta$.
- $\alpha(Y_r) = \beta(Y_r)$: This implies $\alpha(Y_r) \in \mathcal{D}(Y_r)$ (since $\beta(Y_r) \in \mathcal{D}(Y_r)$ as β is of $\mathcal{D}$), and so condition (ii) implies that $\alpha(Y)$ is $\geqslant$-equivalent to some other element of $\underline{Y}$. A root node with such equivalence cannot have children in σ (by Definition of a cp-tree - see Definition 11 in Section 3.4.2.1). Therefore, β is not strictly preferred over α by σ. Thus, since node r decides α and β so again $\alpha(Y_r) \succcurlyeq_r \beta(Y_r)$.

Then, we have shown that $\alpha \trianglerighteq \beta$ for all pos-trees σ that satisfy Γ. Thus, we can conclude that α dominates $\mathcal{D}$ as β was an arbitrary element of $\mathcal{D}$.

■

Example 7. *Let V be the set of variables $\{X, Y, Z\}$ with (initial) domains defined as follows: $\underline{X} = \{x_1, x_2, x_3, x_4\}$, $\underline{Y} = \{y_1, y_2\}$ and $\underline{Z} = \{z_1, z_2\}$. Let cp-theory Γ consist of the five statements $\top : x_1 > x_3$, $\top : x_2 > x_3$, and $\top : x_2 > x_4\ [\{Z\}]\ \ x_1 : y_1 > y_2$, and $x_2 : y_2 > y_1$.*

Suppose a CSP search tree is created and used to find optimal solutions (with respect to Γ). Suppose the variables are instantiated in the following order: X, Y then Z. This instantiation order is compatible with the user preferences, as should be the values' instantiation order. Let us assume that at some node of this search tree, we have a previously found optimal assignment $\alpha = x_2\, y_2\, z_2$ and the variables' domains are as follows. Let $\mathcal{D}(X) = \{x_3\}$, $\mathcal{D}(Y) = \{y_2\}$ and $\mathcal{D}(Z) = \{z_1\}$ with a partial assignment represented by a tuple $b = x_3\, y_2\, z_1$. We want to check whether α dominates the collection of domains $\mathcal{D}$ characterized by $\mathcal{D}(X)$, $\mathcal{D}(Y)$ and $\mathcal{D}(Z)$. If α root-dominates $\mathcal{D}$ then

we can avoid expanding the search tree and backtrack. We first look for all variables not in $\bigcup_{\varphi\in\Gamma} W_\varphi$. *We can see that* $\bigcup_{\varphi\in\Gamma} W_\varphi = \{Z\}$. *Now, we need to find out whether the sufficient conditions for* α *to root-dominate* $\mathcal{D}$ *are verified. For variable* X, *the preference statements in* Γ *show* $\sqsupset_\top^X = \{(x_1, x_3), (x_2, x_3), (x_2, x_4)\}$ *as only the three first statements concern the preference over the values of* X, $\sqsupset_\top^Y = \{(y_1, y_2), (y_2, y_1)\}$ *as only the fourth and fifth statements concern the preference over the values of* Y. *Thus, we can see that* $\alpha(X) = x_2 \sqsupset_\top^X x_3$. *For variable* Y, $\alpha(Y) = y_2 \in \mathcal{D}(Y)$, *and* $y_2 \sqsupset_\top^Y y_1 \sqsupset_\top^Y y_2$, *so* $\alpha(Y)$ *and* y_1 *are* $\sqsupset_\top^Y$*-equivalent. Hence,* α *root-dominates* $\mathcal{D}$.

Then, we backtrack and the collection of domains will be as follows. $\mathcal{D}(X) = \{x_3\}$, $\mathcal{D}(Y) = \{y_2\}$ *and* $\mathcal{D}(Z) = \{z_1, z_2\}$ *with* $b = x_3\, y_2$. *We can notice that only* $\mathcal{D}(Z)$ *changes but* $Z \notin \bigcup_{\varphi\in\Gamma} W_\varphi$. *So,* α *still root-dominates* $\mathcal{D}$.

The search backtracks again so that we have $\mathcal{D}(X) = \{x_3\}$, $\mathcal{D}(Y) = \{y_1, y_2\}$ *and* $\mathcal{D}(Z) = \{z_1, z_2\}$ *with* $b = x_3$. *Only* $\mathcal{D}(Y)$ *changes. The fourth and fifth statements in* Γ *show* $\alpha(Y) = y_2 \sqsupset_\top^Y y_1 \sqsupset_\top^Y y_2$, *which leads to confirm that* $\alpha(Y) = y_2$ *(*$y_2 \in \mathcal{D}(Y)$*) is* $\sqsupset_\top^Y$*-equivalent with another value* $y_1 \in \underline{Y} - \{\alpha(Y)\}$. *Thus,* α *root-dominates* $\mathcal{D}$ *again.*

■

5.3.2 The deciding-node dominance rule

Let α be an outcome and let $\mathcal{D}$ be a collection of domains. We can prove that there is no pos-tree that gives evidence of any β of $\mathcal{D}$ strictly dominating α by defining conditions that are sufficient to prove there is no node that *decides* α and β in any pos-tree σ that satisfies the cp-theory Γ (see Definition 13 in Chapter 3). Define S to be $\{X \in V : \mathcal{D}(X) \not\ni \alpha(X)\}$. These are the variables that α and β differ on for all β of $\mathcal{D}$. Define Ψ to be the set of all $\varphi \in \Gamma$ such that $X_\varphi \in S$ and u_φ is compatible with $\alpha(V - S)$. (u_φ is compatible with $\alpha(V - S)$ if and only if for all $X \in U_\varphi - S$, $\alpha(X) = u_\varphi(X)$.) Let $\alpha_* = \alpha(V - S)$. We will use the relation $\sqsupset_{\alpha_*}^Y$ equal to the transitive closure of all pairs (x_φ, x'_φ) such that $\varphi \in \Gamma$, $X_\varphi = Y$ and u_φ is compatible with α_*.

Definition 23. *We say that* α *deciding-node-dominates* $\mathcal{D}$ *if* $\alpha(Y) \sqsupset_{\alpha_*}^Y y$ *for all* $Y \notin \bigcup_{\varphi\in\Psi} W_\varphi$ *and for all* $y \in \mathcal{D}(Y) - \{\alpha(Y)\}$.

The following proposition states the soundness of the deciding-node-dominates rule.

Proposition 3. *If* α *deciding-node-dominates* $\mathcal{D}$ *then* α *dominates* $\mathcal{D}$.

Proof: Let σ be any pos-tree satisfying Γ and β any element of $\mathcal{D}$. Let us consider σ and focus more specifically on the node r that decides α and β. Node r has an

associated variable Y_r and a tuple $a \in \underline{A}$. Since *r* decides α and β, for all variables $X \in A$ instantiated in σ before instantiating Y_r, $\alpha(X) = \beta(X)$. This implies $A \cap S = \emptyset$ and hence $A \subseteq V - S$

Let us show that $Y \notin \bigcup_{\varphi \in \Psi} W_\varphi$. If $\varphi \in \Psi$ then $X_\varphi \in S$, and so $X_\varphi \notin A$. Thus, in σ, node r is met before any other node which is associated a variable $Z \in W_\varphi$ (Proposition 1(1) in Section 5.3). This shows $Y \notin \bigcup_{\varphi \in \Psi} W_\varphi$.

Now, let us show $\alpha(Y_r) \sqsupset_a^{Y_r} \beta(Y_r)$. Since α deciding-node dominates $\mathcal{D}$, we have $\alpha(Y_r) \sqsupset_{\alpha_*}^{Y_r} \beta(Y_r)$. We have $A \subseteq V - S$ and α extends a and so α_* extends a. This implies $\sqsupset_a^{Y_r}$ contains $\sqsupset_{\alpha_*}^{Y_r}$. This also means that $\alpha(Y_r) \sqsupset_{\alpha_*}^{Y_r} \beta(Y_r)$ implies $\alpha(Y_r) \sqsupset_a^{Y_r} \beta(Y_r)$. Proposition 1(2) in Section 5.3 implies $\alpha(Y_r) \geqslant_r \beta(Y_r)$ since $\alpha(Y_r) \sqsupset_a^{Y_r} \beta(Y_r)$. We then conclude $\alpha \succcurlyeq_\sigma \beta$, and hence $\alpha \trianglerighteq \beta$, as required. Since β was an arbitrary outcome of $\mathcal{D}$, we have α dominates D.

■

Example 8. *Consider again Example 7. Then $S = \{X\}$, α_* equals the partial assignment $y_2\, z_2$, and $\bigcup_{\varphi \in \Psi} W_\varphi = \{Z\}$. We have $\alpha(X) = x_2 \sqsupset_{\alpha_*}^X x_3$ showing that α deciding-node-dominates $\mathcal{D}$, and hence, by Proposition 3 in Section 5.3, α dominates $\mathcal{D}$.*

If we now remove statement $x_1 : y_1 > y_2$ from Γ we still have α deciding-node-dominates $\mathcal{D}$ but we no longer have α root-dominates $\mathcal{D}$. This is because we do not have anymore $\alpha(Y) = y_2 \sqsupset_\top^Y y_1 \sqsupset_\top^Y y_2$, and so $\alpha(Y)$ and y_1 are not anymore $\sqsupset_\top^Y$-equivalent. Thus, this does not satisfy the condition (ii) of root-dominates in Section 5.3.1.

Example 9. *Let V be the set of variables $\{X, Y, Z\}$ with (initial) domains defined as follows:*

Let $\underline{X} = \{x_1, x_2, x_3\}$, let $\underline{Y} = \{y_1, y_2\}$, and let $\underline{Z} = \{z_1, z_2, z_3\}$. Let Γ consist of: $z_1 : x_1 > x_2$, $z_1 : x_1 > x_3$, $z_2 : x_2 > x_1, x_1 : y_1 > y_2$, $x_2 : y_2 > y_1, x_1 : z_1 > z_2$, $x_2 : z_2 > z_1$ and $x_1 y_2 : z_1 > z_3$. Let $\alpha = x_1\, y_1\, z_1$. Let $\mathcal{D}(X) = \{x_2, x_3\}$, let $\mathcal{D}(Y) = \underline{Y} = \{y_1, y_2\}$, and let $\mathcal{D}(Z) = \underline{Z} = \{z_1, z_2, z_3\}$. Let us check if α does deciding-node-dominate $\mathcal{D}$. $\alpha = x_1\, y_1\, z_1$ and so $S = \{X\}$ as $\mathcal{D}(X) \not\ni \alpha(X) = x_1$. Thus, $\alpha_* = \alpha(\{Y, Z\}) = y_1 z_1$. $\Psi = \{z_1 : x_1 > x_2, z_1 : x_1 > x_3\}$. We can see $\bigcup_{\varphi \in \Psi} W_\varphi = \emptyset$. For variable X, $\alpha(X) = x_1 \sqsupset_{\alpha_*}^X x_2$ and $\alpha(X) = x_1 \sqsupset_{\alpha_*}^X x_3$. For variable Y, $\alpha(Y) = y_1 \sqsupset_{\alpha_*}^Y y_2$. For variable Z, $\alpha(Z) = z_1 \not\sqsupset_{\alpha_*}^Z z_3$. Therefore, we can conclude α does not deciding-node-dominate $\mathcal{D}$.

Let us check if α does root-dominate $\mathcal{D}$. As $\bigcup_{\varphi \in \Gamma} W_\varphi = \emptyset$, we check the two conditions in Section 5.3.1 for all variables in V. For variable X, $\alpha(X) = x_1 \sqsupset_\top^X x_2$ and $\alpha(X) = x_1 \sqsupset_\top^X x_3$. For variable Y, $\alpha(Y) = y_1 \sqsupset_\top^Y y_2 \sqsupset_\top^Y \alpha(Y)$. Then, $\alpha(Y) = y_1$ and y_2 are $\sqsupset_\top^Y$-equivalent. For variable Z, $\alpha(Z) = z_1 \sqsupset_\top^Z z_2 \sqsupset_\top^Z \alpha(Z)$ and $\alpha(Z) = z_1 \sqsupset_\top^Z z_3$. Then, $\alpha(Z)$ and z_2 are $\sqsupset_\top^Z$-equivalent and $\alpha(Z)$ is preferred over

z_3. Therefore, we can conclude α does root-dominates $\mathcal{D}$.

In the above example, α does not deciding-node-dominate $\mathcal{D}$ but α root-dominates $\mathcal{D}$, and so α dominates $\mathcal{D}$. Therefore, root-dominance and deciding-node-dominance are incomparable, and both are strictly stronger than dominance. ■

5.3.3 Projection-dominance condition

Let V be the set of all variables and t and t' be two tuples that can be partial or complete assignments. The dominance algorithm described in Section 3.5 of Chapter 3 is used to check whether or not t dominates t'. One example of use is introduced in Section 4.5 in Chapter 4 where complete tuples (e.g., queries) are compared using this dominance procedure. But, the procedure is defined to be able to compare partial tuples as well. Then, we can determine whether a tuple t dominates another tuple t' by simply using the dominance procedure. It aims at verifying whether every outcome that extends t dominates every outcome that extends t'.

Let b be an assignment to set of variables $B \subset V$, and let $\mathcal{D}$ be a collection of domains such that $\mathcal{D}(X) = \{b(X)\}$ for $X \in B$. Let α be an optimal outcome that was already found. Let $\alpha(B)$ be the projection of α on B, which means α restricted to B. The test presented above in this section can be useful at some node of the search tree (when looking for optimal outcomes) as we can check whether $\alpha(B)$ dominates b, which means that we can check whether every assignment that extends $\alpha(B)$ dominates every assignment that extends b. If the dominance procedure checks and confirms that $\alpha(B)$ dominates b, then all outcomes that extend b are dominated (by all outcomes that extend $\alpha(B)$). It follows that the condition $(*)$ below is a sufficient condition for: α dominates $\mathcal{D}$.

$(*)$ $\delta \trianglerighteq \beta$ for all outcomes $\delta \in \underline{V}$ agreeing with α on B (i.e., $\delta(B) = \alpha(B)$), and all $\beta \in \underline{V}$ extending b (i.e., $\beta(B) = b$).

Thus, the condition is sufficient to have every outcome, whose projection to B is $\alpha(B)$, dominates every outcome whose projection to B is b. Condition $(*)$ can be determined directly using the polynomial algorithm presented in Section 3.5 of Chapter 3 by checking if $\Gamma \models_{\mathcal{Y}} \psi$ where $\mathcal{Y}$ is the set of singleton subsets of V, and ψ is the preference statement written as $\alpha(B) > b \parallel \emptyset$.

5.4 p(Γ): A sufficient Condition for Dominance based on Unsound Dominance Relation

In order to prune the search tree, we can use an unsound dominance relation described in Section 2.2.3 in (Wilson 2011). It is stronger than the dominance relation from which we derive the sufficient conditions in Section 5.3. We will be able to prune more and we will obtain relatively less optimal solutions in a noticeably shorter time. This can be convenient for applications in which few (or not necessarily a large number of) optimal solutions are needed in a quick response time. For instance, in the case of a configurator used by a salesperson in a business-to-business (B2B) manufacturer which sells complex products (financial services, etc.), the number of available products can be large (Veron et al. 1999, Lilien & Grewal 2012). The salesperson wants to give the customer a quick look at a sample of optimal products which meet the customer's preferences. Thus, the salesperson collects the user's preferences and generates some optimal products as samples to be considered by the client. Besides, mobile systems are providing personalized and focused recommendation to mobile users and so the number of optimal solutions needs to be relatively small to adapt to the physical constraints of the mobile devices (Ricci 2011).

Computation of this dominance relation is easier than the computation of the previous dominance relation (we use to derive the sufficient conditions in Section 5.3). In order to achieve pruning in the search tree, we will derive a sufficient condition for dominance based on such an unsound dominance relation. We will then obtain relatively less optimal solutions in relatively shorter time than the sound pruning rules described in Section 5.3.

Let α and β be two outcomes and let $\mathcal{D}$ be a collection of domains. Define $\Delta(\alpha, \beta)$ to be $\{X \in V : \alpha(X) \neq \beta(X)\}$. $\Delta(\alpha, \beta)$ represents all the variables $X \in V$ that α and β differ on. In Section 6.3 of (Wilson 2011), the author presented a polynomial preference relation that is denoted by $\gg_\Gamma$ and based on a dependency graph between variables in V regarding the set of user preferences (i.e., a graph whose nodes are variables).

Let Γ be a cp-theory representing the user preferences of the form $u_\varphi : x_\varphi > x'_\varphi[W_\varphi]$, and $\alpha \in \underline{V}$ be an assignment to a set of variables V. For $P, Q \subseteq V$, define $P \to Q$ to be the set of edges $\{(X, Y) : X \in P, Y \in Q\}$. In Section 2.2.3 and Definition 15 of (Wilson 2011), the author presented a relation denoted by $G(\Gamma)$ which contains sets of edges $U_\varphi \to X_\varphi$ and $X_\varphi \to W_\varphi$ for all $\varphi \in \Gamma$. Γ is said to be fully acyclic if $G(\Gamma)$ is acyclic. The author also introduced a directed graph $G_\alpha(\Gamma)$ (abbreviated to G_α) on V to consist of the set $U_\varphi \to \{X_\varphi\} \cup W_\varphi$ of edges for all $\varphi \in \Gamma$, also the set $X_\varphi \to W_\varphi$ of edges for all $\varphi \in \Gamma$ such that α extends u_φ. The author also defined an order $\rhd_\alpha(\Gamma)$ on V (abbreviated to $\rhd_\alpha$) to be the transitive

closure of $G_\alpha(\Gamma)$.

Therefore, in order to compare two outcomes α and β, the author considers the set of variables $X \in \Delta(\alpha, \beta)$ which are not dominated by any other variable with respect to $\rhd_a$. The author states that Y $\in \Delta(\alpha, \beta)$ is $\rhd_a$-undominated if Y has no parent in $\Delta(\alpha, \beta)$ with respect to the order $\rhd_a$, i.e., there does not exist Z $\in \Delta(\alpha, \beta)$ with Z $\rhd_a$ Y. This set of $\rhd_a$-undominated variables is called $\Theta(\alpha, \beta)$. We can write Definition 12 in Section 5.2 of (Wilson 2011) in short as follows:

Definition 24. *$\alpha \gg_\Gamma \beta$ holds if and only if $\alpha(X) \sqsupseteq_\alpha^X \beta(X)$ for all $X \in \Theta(\alpha, \beta)$.*

Let $\Delta(\alpha, \mathcal{D})$ be the set of variables $X \in V$ such that $\mathcal{D}(X) \not\ni \alpha(X)$. Let $\Theta(\alpha, \mathcal{D})$ be the set of variables $X \in \Delta(\alpha, \mathcal{D})$ that are $\rhd_a$-undominated.

We say that α *p(Γ)-dominates* collection of domains $\mathcal{D}$ if: for all $X \in \Theta(\alpha, \mathcal{D})$, we have $\alpha(X) \sqsupseteq_\alpha^X x'$, for all $x' \in \mathcal{D}(X) - \{\alpha(X)\}$.

Wilson stated in Proposition 23 in (Wilson 2011) that $\gg_\Gamma$ is a strict partial order containing $\trianglerighteq$ and so is an upper approximation for the preference relation $\trianglerighteq$. Then, we can see that the set of optimal solutions generated using the relation $\gg_\Gamma$ is a subset of the optimal solutions generated using the relation $\trianglerighteq$.

Example 10. *Consider again Example 7 in Section 5.3.1. In the example, $\alpha = x_2\, y_2\, z_2$. G_α includes (X, Y) since Γ contains $x_2 : y_2 > y_1$ (α extends x_2). G_α also includes (X, Z) since Γ contains $\top : x_2 > x_4$ $[\{Z\}]$. Then $\rhd_\alpha = \{(X, Y), (X, Z)\}$. In this example, X is the only variable which is not $\rhd_\alpha$-dominated by any other variable (i.e., Y or Z). We have $\alpha(X) = x_2$ and $\mathcal{D}(X) = \{x_3\}$. The preference statement : $\top : x_2 > x_3$ ensures $\alpha(X) = x_2$ is better than the remaining value in the current domain of X (i.e., x_3) (regarding $\sqsupseteq_\alpha^X$). Then we can say α p(Γ)-dominates collection of domains $\mathcal{D}$.*

5.5 Non-Dominance Pruning Rules

Let $\mathcal{D}$ be a collection of domains. Suppose, in the depth-first search, we already found an optimal outcome α and b is the current partial assignment (associated with a particular node of the search tree) and that $\mathcal{D}$ is the collection of domains of the set of variables not yet instantiated and compatible with b. If we can show that α will not dominate any of the outcomes that extend b then we do not need to involve α any further in any dominance check in descendants of the current node. Therefore, we can think about sufficient conditions that ensure an outcome α is not able to dominate any assignment that extends b. Then, we can save dominance checks that involve α below the current node in the search tree.

Definition 25. α root non-dominates $\mathcal{D}$ *if there exists $Y \notin \bigcup_{\varphi \in \Gamma} W_\varphi$ such that $\mathcal{D}(Y) \not\ni \alpha(Y)$ and, for all $y \in \mathcal{D}(Y)$, $\alpha(Y) \not\sqsupseteq_\top^Y y$.*

Proof: Let β any element of $\mathcal{D}$. We will construct a pos-tree σ (see Definition 12 Section 3.4.2.1 of Chapter 3) satisfying Γ and such that $\alpha \not\succcurlyeq_\sigma \beta$. Suppose there exists $Y \notin \bigcup_{\varphi\in\Gamma} W_\varphi$ such that $\mathcal{D}(Y) \not\ni \alpha(Y)$ and $\alpha(Y) \not\rhd^Y_\top \beta(Y)$. We can define a pos-tree σ with just a root node r with associated variable Y and local ordering $\geqslant$ with $\alpha(Y) \not\geqslant \beta(Y)$ and extending $\sqsupset^Y_\top$. Then, $\alpha(Y) \not\geqslant \beta(Y)$ for root node r implies σ does not order α before β ($\alpha \not\succcurlyeq_\sigma \beta$). $Y \notin \bigcup_{\varphi\in\Gamma} W_\varphi$ ensures the condition (1) in Proposition 1 in Section 5.3 is valid. As $Y \notin \bigcup_{\varphi\in\Gamma} W_\varphi$ and the local ordering of node r which extends $\sqsupset^Y_\top$ we can say that σ satisfies Γ according to Proposition 1 in Section 5.3.

Then, we have a pos-tree σ which satisfies Γ but does not order α before β ($\alpha \not\succcurlyeq_\sigma \beta$). This shows that $\alpha \not\trianglerighteq \beta$. Thus, we can conclude that α root non-dominates $\mathcal{D}$ as β was an arbitrary element of $\mathcal{D}$.

Proposition 4. *If α root non-dominates $\mathcal{D}$ then α non-dominates $\mathcal{D}$, i.e., for all β of $\mathcal{D}$, we have $\alpha \not\trianglerighteq \beta$.*

Example 11. *Let Γ be as in Example 7 in Section 5.3.1, let γ be the outcome $x_1\, y_2\, z_2$, and we define $\mathcal{D}'$ by $\mathcal{D}'(X) = \{x_4\}$, $\mathcal{D}'(Y) = \underline{Y}$ and $\mathcal{D}'(Z) = \underline{Z}$. Then γ root non-dominates $\mathcal{D}'$, because $X \notin \bigcup_{\varphi\in\Gamma} W_\varphi = \{Z\}$, and $\mathcal{D}'(X) \not\ni \gamma(X) = x_1$, and $\gamma(X) \not\rhd^X_\top x_4$.* ∎

Our experiments show that the application of this rule while searching optimal outcomes can sometimes save a lot of time. It does, however, make the implementation a little more complicated: instead of a single set of undominated solutions (which could be treated as a global variable), the set of undominated solutions that we are currently interested in depends on the node in the search tree.

5.6 Example: Computer Configuration Problem

It has been shown that supporting users of product configuration systems with recommendations can lead to a higher satisfaction with the configuration process (Mandl, Felfernig & Tiihonen 2011).

A product can be a complex combination of components and relations. Looking for preferred products is a challenging task. Today, configuration problems solving has enormous potential of applications such as in the computer industry (e.g., PC configuration), telecommunication industry (e.g., configuration of switching systems) (Mandl, Felfernig & Tiihonen 2011), or automotive industry (e.g., car sales configuration) (McDermott 1982, Searls & Norton. 1990, Axling & Haridi. 1994, Mittal & Falkenhainer 1996, Barker et al. 1989). The use of the configuration tools is

increasingly being challenged by the growing need to find the way to communicate and involve the user's preferences in the search process of optimal products. Several configurators have been recognized as suitable tools to help customers configure complex products according to their requirements (Fleischanderl et al. 1998, Sabin & Weigel 1998, Stumptner 1997).

The conditional preference theories that we use in this chapter might be used in real world domains (e.g., configuration and recommendation of financial services, etc.) (Felfernig & Burke 2008, Felfernig, Friedrich, Jannach & Zanker 2007, Felfernig, Isak, Szabo & Zachar 2007) with preserving quite attractive computational properties.

Constraints and preferences-based configuration technologies can rely on the pruning rules described in Section 5.3 and 5.5, in order to get rid of non-optimal options. This will offer expressiveness and computational capabilities to these technologies.

5.6.1 Computer configuration problem

Let us suppose we have a configurator which manages the part of the sales system that enables customers to choose different components and options when making a purchase of a computer. When customers request a computer with certain specifications the configurator will return optimal instances. Instances here are combinations of computer components that are manufactured and make up the final configuration. The configuration problem can be considered as an instance of a CSP where the final solution is a list of instances (or variables instantiations) from the domain of products that do not violate constraints.

The CSP can be augmented by a set of preferences in order to get only the most preferred instances. In order to define the best configuration we define an order over the configuration space. It is assumed that every decision maker (i.e., customer) has her own order. In fact we will generate (implicitly) a preference order (from user inputs) to form a partial order that holds for the user's preferences order over the configuration space. A configuration $c \in C$ is said to be optimal if there does not exist another configuration $c' \in C$ such that c' dominates c w.r.t the generated preference relation.

5.6.2 Example

The configuration problem is defined using a CSP where we define the model with a set of variables *V* and constraints over the variables. Let V be a subset of components for the computer: *Operating System(OS)*, *Memory(M)* (unit=megabyte), *Central Processing Unit(CPU)*, *Monitor(Mon)* (unit=inch) and *Graphics Card(GC)*. V={*OS*, *M*, *CPU*,

Table 5.1: Computer components.

Variables	Values
Operating System (OS)	{*XP, UBUNTU11.4 (UB)*}
Memory (M)	$\{512, 1024, 2048\}$
Central Processing Unit (CPU)	{*Pentium(P), AMD_Athlon(A)*}
Monitor (Mon)	$\{17, 19\}$
Graphics Card (GC)	{*ATI_Radeon_HD5000(AT), Nvidia_GeForce_MX440(NV)*}

Mon, *GC*}.

The problem is subject to the following constraints:

If $OS = XP$ then $M \geq 1024$ (since *XP* requires a fair deal of memory to work conveniently).

If $GC = AT$ then $M \geq 1024$ (it is a requirement of the constructor of the graphics card).

If $GC = NV$ then $Mon \leq 17$ (the *NV* card cannot support some resolutions of a large screen).

If $GC = AT$ then $OS \neq UB$ (the driver of the *AT* card is not available for *UB*).

The user has preferences, which are expressed over partial or complete combinations of computer components, are presented below.

$2048 \geq 1024$ $[\emptyset]$

$1024 \geq 512$ $[\emptyset]$

$AT \geq NV$ $[\emptyset]$

M=1024 : $XP \geq UB$ $[\{Mon, GC\}]$

M=2048 : $XP \geq UB$ $[\{Mon, GC\}]$

M=512 : $UB \geq XP$ $[\emptyset]$

$P \geq A$ $[\emptyset]$

GC=AT:19 ≥ 17 $[\emptyset]$

Her preferences are justified as follows:

- For memory (M), the greater the memory the better.
- For the graphics card, *ATI_Radeon_HD5000* is preferred over *Nvidia_GeForce_MX440* as it is faster in a wide range of games using the latest drivers.
- Given the memory is equal to 512MB, *UBUNTU (UB)* is preferred over *XP* because *UBUNTU (UB)* can be personalized and optimized in order to operate quite conveniently even with small memory which is not the case for *XP* which needs at least 1024MB of memory. Otherwise, *XP* is more preferred

than *UBUNTU (UB)* whatever the values of the monitor (*Mon*) and the processor (*graphics card (GC)*) everything else being equal. This is because the user does not want to go for *UBUNTU (UB)* just because of the graphics card or the monitor values are better or worse since she is not doing sophisticated computations or graphics.

- In matters of processors the user prefers *Pentium (P)* over *AMD_Athlon (A)*, everything else being equal, because of its solid public image built over years.
- A bigger screen (i.e., 19 inches) is better than a small screen (ie., 17 inches) if the graphics card is as powerful as *ATI_Radeon (AT)*. This is not the case when the graphics card is *Nvidia_GeForce_MX440 (NV)* as it will be quite difficult for this type of card to support a particular kind of resolution when the screen is 19 inches.

We want to find the optimal solutions for this problem. We first applied only the basic constraint optimisation approach described in 2.12.4 of Chapter 2, and we obtain the search tree in Figure 5.1 which shows 19 solutions (i.e., configurations). Then, we applied the deciding-node-dominance pruning rule and we obtained the search tree in Figure 5.2 which shows one optimal solution α = $\{M = 2048, OS = XP, GC = AT, Mon = 19, CPU = P\}$. α deciding-node-dominates the remaining (possible) configurations.

Let us consider β = $\{M = 2048, OS = XP, GC = AT, Mon = 19, CPU = A\}$ (regarded as a non-optimal configuration according to the deciding-node-dominance rule). After assigning the value P to the variable CPU in order to obtain α, the search backtracks to look for another optimal configuration. At that point of the search tree, the collection of domains $\mathcal{D}$ is as follows: $\mathcal{D}(M) = \{2048\}$, $\mathcal{D}(OS) = \{XP\}$, $\mathcal{D}(GC) = \{NV\}$, $\mathcal{D}(Mon) = \{19\}$ and $\mathcal{D}(CPU) = \{A\}$.

As S = $\{X \in V : \mathcal{D}(X) \not\ni \alpha(X)\}$, S will then contain only the variable *CPU* (i.e., $S = \{$*CPU*$\}$).

Then, the set of preferences of concern is $\Psi = \{P \geq A[\emptyset]\}$.

Besides, we have $\bigcup_{\varphi \in \Psi} W_\varphi = \emptyset$ and $\alpha(CPU) = P \sqsupset_{\alpha_*}^{CPU} A$.

According to Definition 23 in Section 5.3.2, α deciding-node-dominates $\mathcal{D}$ as $\alpha(Y) \sqsupset_{\alpha_*}^{Y} y$ for all $Y \notin \bigcup_{\varphi \in \Psi} W_\varphi$ and for all $y \in \mathcal{D}(Y) - \{\alpha(Y)\}$. Then, β is shown to be dominated by α (i.e., β is a non-optimal configuration).

α is also proven to deciding-node-dominate subtrees of the search tree. One example is the following subtree. In the part of the search tree where the variable M is assigned the value 2048 and the variable OS is assigned the value UB, the collection of domains $\mathcal{D}$ is as follows: $\mathcal{D}(M) = \{2048\}$, $\mathcal{D}(OS) = \{UB\}$, $\mathcal{D}(GC) = \{NV\}$, $\mathcal{D}(Mon) = \{17\}$ and $\mathcal{D}(CPU) = \{P, A\}$. As S = $\{X \in V : \mathcal{D}(X) \not\ni \alpha(X)\}$, S

contains three variables which are OS, GC and Mon (i.e., S = $\{OS, GC, Mon\}$), α_* is equal to $\{M = 2048, CPU = P\}$ and the set of preferences we deal with is $\Psi = \{M = 2048 : XP \geq UB\ [\{Mon, GC\}];\ AT \geq NV\ [\emptyset];\ GC = AT : 19 \geq 17\ [\emptyset]\}$ (according to Definition 23 in Section 5.3.2).

Then, $\bigcup_{\varphi \in \Psi} W_\varphi = \{Mon, GC\}$.

Besides, we have $\alpha(CPU) = P \sqsupseteq_{\alpha_*}^{CPU} A$ and $\alpha(OS) = XP \sqsupseteq_{\alpha_*}^{OS} UB$.

Notice that $\alpha(M) = \mathcal{D}(M) = \{2048\}$ and so $\mathcal{D}(M) \ni \alpha(M)$.

According to the same definition, α *deciding-node-dominates* $\mathcal{D}$ as $\alpha(Y) \sqsupseteq_{\alpha_*}^{Y} y$ for all $Y \notin \bigcup_{\varphi \in \Psi} W_\varphi$ and for all $y \in \mathcal{D}(Y) - \{\alpha(Y)\}$.

The search will backtrack till the root node where the collection of domains $\mathcal{D}$ is as follows: $\mathcal{D}(M) = \underline{M} - \{2048\}$, $\mathcal{D}(OS) = \underline{OS}$, $\mathcal{D}(GC) = \underline{GC}$, $\mathcal{D}(Mon) = \underline{Mon}$ and $\mathcal{D}(CPU) = \underline{CPU}$.

As S = $\{X \in V\ :\ \mathcal{D}(X) \not\ni \alpha(X)\}$, S will then contain only the variable M (i.e., S = $\{M\}$) as $\alpha(M)$ = 2048 and $\mathcal{D}(M) = \underline{M} - \{2048\}$. Then, the set of preferences of concern is Ψ = $\{2048 \geq 1024\ [\emptyset];\ 1024 \geq 512\ [\emptyset]\}$, and so $\bigcup_{\varphi \in \Psi} W_\varphi = \emptyset$. α_* is equal to the following assignment $\{OS = XP, GC = AT, Mon = 19, CPU = A\}$

According to Definition 23 in Section 5.3.2, α deciding-node-dominates $\mathcal{D}$ if $\alpha(Y) \sqsupseteq_{\alpha_*}^{Y} y$ for all $Y \notin \bigcup_{\varphi \in \Psi} W_\varphi$ and for all $y \in \mathcal{D}(Y) - \{\alpha(Y)\}$.

Let us look at all variables and check whether the above condition is true. We have $\alpha(CPU) = P \sqsupseteq_{\alpha_*}^{CPU} A$ as $P \geq A\ [\emptyset]$. $\alpha(Mon) = 19 \sqsupseteq_{\alpha_*}^{Mon} 17$ as GC=AT:$19 \geq 17\ [\emptyset]$. $\alpha(GC) = AT \sqsupseteq_{\alpha_*}^{GC} NV$ as $AT \geq NV\ [\emptyset]$. $\alpha(OS) = XP \sqsupseteq_{\alpha_*}^{OS} UB$ as $M = 2048 : XP \geq UB\ [\{Mon, GC\}]$. $\alpha(M) = 2048 \sqsupseteq_{\alpha_*}^{M} 1024$ and $\alpha(M) = 2048 \sqsupseteq_{\alpha_*}^{M} 512$ as $2048 \geq 1024\ [\emptyset]$ and $1024 \geq 512\ [\emptyset]$.

Hence, we can see that $\alpha(Y) \sqsupseteq_{\alpha_*}^{Y} y$ for all $Y \notin \bigcup_{\varphi \in \Psi} W_\varphi$ and for all $y \in \mathcal{D}(Y) - \{\alpha(Y)\}$. Then, we conclude that α deciding-node-dominates $\mathcal{D}$ and we can prune the subtrees whose root nodes correspond to a partial assignment equal to $\{M = 1024\}$ and $\{M = 512\}$ respectively. The search tree, after pruning, is shown in Figure 5.2.

When we compare the number of visited nodes in the two search trees in Figure 5.1 and 5.2, we can see the deciding-node-dominance rule was able to prune significantly the search tree (i.e., 5 nodes vs 42 nodes).

5.7 Implementation Issues

During the implementation of the approaches described above, we had to deal with a number of issues, which are discussed in the following paragraphs.

In the basic constraint optimisation approach described in 2.12.4, the optimal solutions found during search are usually stored in a global set Ω. This set of solutions, whose size keeps increasing monotonically, is used to check whether a newly found

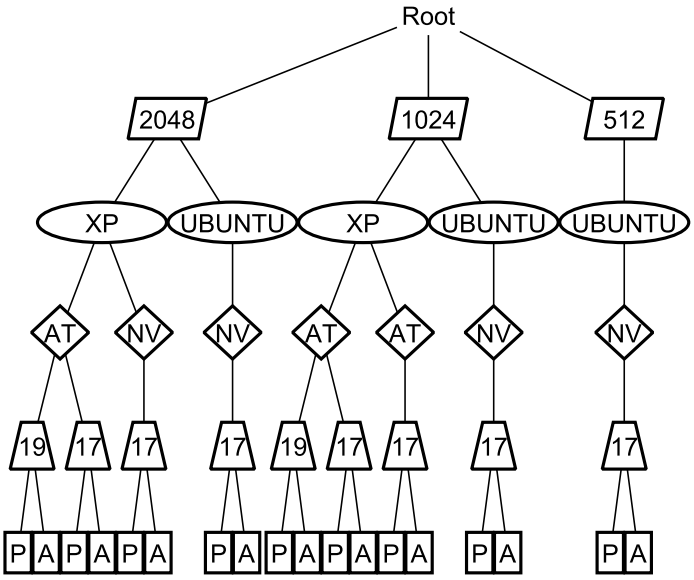


Figure 5.1: Search trees using basic approach

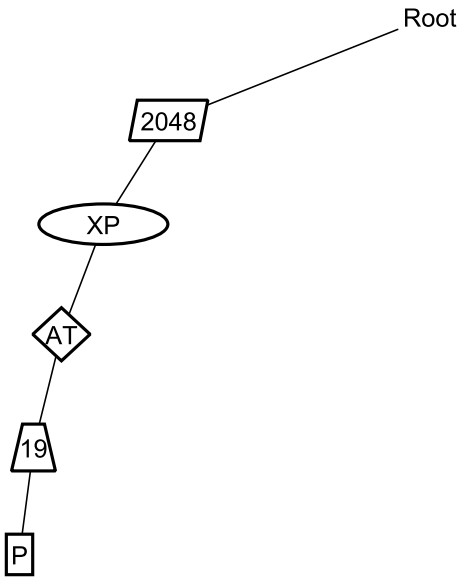


Figure 5.2: Search trees using deciding-node-dominance rule

solution β or a subtree not yet explored is dominated by at least one solution that belongs to Ω. This kind of dominance was described in Section 5.3 and 5.4. We notice that, at some node N in the search tree, there will probably be a number of solutions that will not be useful with regards to dominance below node N since they cannot dominate any of the solutions that will be found below N. Ignoring those irrelevant solutions at a given level of the search tree can therefore save a significant number of dominance checks and can consequently save time which will be shown and verified in the experiments. These solutions can be eliminated by using the pruning rule

described in Section 5.5. The remaining solutions are considered as relevant optimal solutions associated with node *N*. One challenge is to find out how to update and bring into play the associated set of relevant optimal solutions for each node in the search tree. In Section 5.7.1, we describe the structure and characteristics of the search tree used to find undominated solutions. The way undominated solutions are transferred between nodes in the search tree is described in Section 5.7.2. In Section 5.7.3, we will describe the procedure used to remove irrelevant solutions. In Section 5.7.4, we will sketch the procedure for testing dominance when navigating the search tree. Then, we will outline the procedure that scans the search tree for optimal solutions in Section 5.7.5.

5.7.1 The search tree

The backtracking search tree generated by a CSP solver can be depicted as a directed rooted tree with the root at the top and leaves at the bottom. To remove infeasible paths, we enforce arc consistency (Sabin & Freuder 1994). This will remove inconsistent values from domains $\mathcal{D}(X)$ for uninstantiated variables X. When searching for solutions, the search branches on unassigned variables and instantiates their unpruned values. The search backtracks when it reaches a dead end (a path in the search tree that cannot be extended to any optimal solution) or all variables were instantiated. Children nodes are created in between from the top down to the bottom linked through directed edges so that every node has a unique parent (except the root). When all variables are instantiated, the resulting (complete) assignment is associated with a node that we call an *end node*. Thus, a leaf node at the bottom of the search tree can be a node associated with a partial assignment or an *end node* which is associated with a complete assignment.

Solutions (i.e., complete assignments) are created when all variables are instantiated. Variables and values are instantiated in an order compatible with the user's preferences (see Proposition 1 in Section 5.3). This will prevent any newly found solution β to be bettered by any other solution found later during the search. In fact, the algorithm proceeds by assigning values to the variables in a top-down manner according to the variable ordering shown in the user's preferences (see Proposition 1(1) in Section 5.3).

The values for a variable *X* are considered according to the preferential ordering induced by the assignment made to the variables that are instantiated before *X* in the search tree and whose values determine the ordering of the values of *X*. Let V be a set of variables and Γ be the set of user preferences. Let U_φ and W_φ be two subsets of V. If there is a preference statement $\varphi \in \Gamma$ which has the following form:

$u_\varphi : x_\varphi > x'_\varphi[W_\varphi]$, ($W_\varphi \subseteq (V \setminus (U_\varphi \cup X_\varphi))$), then variables in U_φ are instantiated in the search tree before X_φ which is instantiated before variables in W_φ. According to φ, if the assignment already made in the search tree is compatible with u_φ, then x_φ should be instantiated before x'_φ (x_φ and x'_φ are assumed to be consistent with regards to the CSP).

This ordering is not necessarily as efficient as the commonly-used variable ordering heuristics from the perspective of solving a CSP; it is indeed effectively random regarding the CSP solving. Further, it is well known that imposing an ordering on the splitting heuristic, used for searching solutions, may lead to a significant degradation in the performances of the solver used for finding solutions (Järvisalo et al. 2005, Järvisalo & Junttila 2009). Although this is not the case for several applications, see, e.g., (Giunchiglia et al. 1998, Maratea et al. 2010) in the context of satisfiability planning (Kautz & Selman 1992) and Answer Set Programming (Gelfond & Lifschitz 1988*b*, Gelfond & Lifschitz 1991).

However, such an ordering gives the algorithm the anytime property. In fact, this ordering ensures that a solution that is already found (previously generated) is guaranteed to be optimal: it will not be dominated by a solution that will be found later during the search. Thus, the algorithm can stop anytime, being sure the set of solutions obtained so far (till the moment it stops) is a subset of the set of all optimal solutions (obtained if the algorithm terminates the execution normally). Without this property, we would have to maintain a potentially large set of candidate solutions. Section 4 of (Boerkoel et al. 2010) discussed ways of realising the performance benefits of a heuristic variable ordering while preserving the early termination properties of the CP-net-based ordering.

In the search tree, each node *N* is associated with a variable *V(N)* and has its current domain $\mathcal{D}(V(N))$ (which we abbreviate to $\mathcal{D}(N)$) filled with consistent values regarding constraint propagation and as presented in Definition 2 in Section 2.11 of Chapter 2. Every time the associated variable is instantiated with a value $v \in \mathcal{D}(N)$, a new branch in the search tree is generated at the top of which node *M* is created. *N* will then become the parent of *M*. A partial assignment *A(M)* (i.e., A_v in the CPOptimizer procedure) is created at node *M*. It corresponds to the assignments made on the path from the root to node *M*. The search process tries to extend this partial assignment incrementally by instantiating the uninstantiated variables. *N* will have a number of children equal to the number of values in $\mathcal{D}(N)$ after instantiating variable V(N). Every child node (e.g., node *M*) will become the root of a subtree ST_v which is created below *N* if at least one of its values is consistent regarding constraint propagation (e.g., the instantiation of one value satisfies all constraints). There will be search for solutions in ST_v. Each time all variables are instantiated (at each *end node*), a new solution is recorded. After exploring each subtree of *N* we backtrack

to *N* and we assign the next unassigned value (if any) to *V(N)*. Then, we add the set of solutions that were found after creating node *N* to the set of solutions that were found before creating node *N*.

Example 12. *Let V be the set of variables $\{X_1, X_2, X_3\}$ with (initial) domains defined as follows: $\underline{X_1} = \{x_{11}, x_{12}\}$, $\underline{X_2} = \{x_{21}, x_{22}\}$ and $\underline{X_3} = \{x_{31}, x_{32}\}$. The set of variables have their domains constrained as the following pairs of values are not allowed in a solution: (x_{21}, x_{32}), (x_{12}, x_{31}). The user's preferences are represented by a cp-theory Γ which consists of the five statements: $\top : x_{22} > x_{21}[\emptyset]$, $x_{22} : x_{12} > x_{11}[\emptyset]$, $x_{21} : x_{11} > x_{12}[\emptyset]$, $x_{11} : x_{32} > x_{31}[\emptyset]$, $x_{12} : x_{31} > x_{32}[\emptyset]$.*

The corresponding search tree with a variable instantiation ordering compatible with the user's preferences (i.e., Γ) is depicted in Figure 5.7.1.

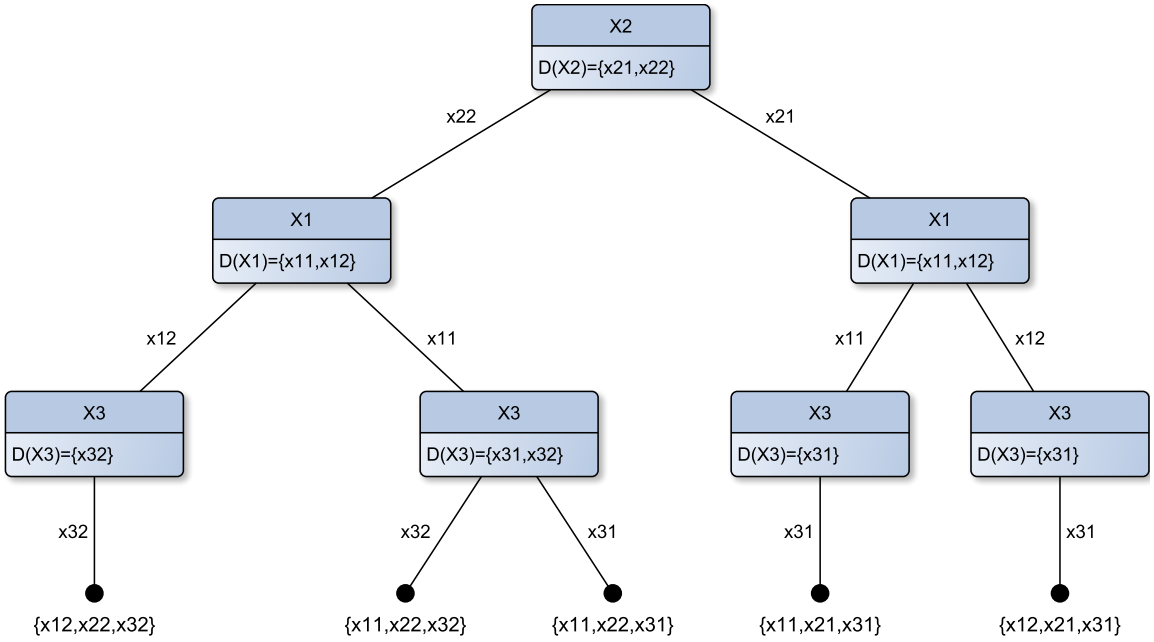


Figure 5.3: Example of a search tree

5.7.2 Relevant undominated solutions

Let N be a node in the search tree with an associated variable $V(N)$ whose domain contains two values $v1$ and $v2$ which are consistent with regards to constraint propagation. When the two values $v1$ and $v2$ are instantiated in the search tree, two subtrees ST_{v1} and ST_{v2} are created. The root nodes of these subtrees are the two child nodes of N whose names are $C1$ and $C2$ respectively. Let node P be the parent of node N in the search tree.

Define *S(N)* to be the set of solutions found so far at some node N of the search tree. If we prove, under some conditions, that some solution $\alpha \in$ *S(N)* cannot

dominate any of the solutions that will eventually be generated when exploring a subtree (e.g., ST_{v1}) below N, then it is better not to involve α in the dominance checks that will be performed in that subtree. Solution α is then said to be *ineffective*. This can save a significant number of dominance checks. In order to implement the idea above, we need to eliminate the irrelevant (or ineffective) solutions at each point of the search tree. Therefore, node N will have an appropriate set of solutions which excludes ineffective ones. We call this set of solutions the *Relevant Undominated Solutions (RUS)* (see Figure 5.4).

When created, node N receives the set of inherited undominated solutions (denoted by $IRUS_p$) from its parent node P. $IRUS_p$ may include solutions that are ineffective below node N. That is why we need to filter out the ineffective solutions from $IRUS_p$ in order to transfer them to the child nodes of node N. This is performed by the *Reduce* procedure described in Section 5.7.3. Thus, Node N obtains RUS_N that is sent to its child node $C1$. Once the subtree ST_{v1} is explored, the search backtracks to node N in order to instantiate the next consistent value (i.e., $v2$). The set of undominated solutions that were found when exploring ST_{v1} is denoted by New_{C1} and is transferred to node N. New_{C1} might contain undominated solutions that are ineffective in the next subtree created when instantiating $v2$ (i.e., ST_{v2}). Therefore, node N adds New_{C1} to $IRUS_p$ in one set of solutions from which it extracts the relevant undominated solutions using the *Reduce* procedure. RUS_N is updated and sent to the second child node $C2$ which does the same as $C1$ and sends the newly found solutions denoted by New_{C2} to node N. Once all uninstantiated and consistent values (i.e., $v1$ and $v2$) are instantiated, node N gathers all undominated solutions newly found (i.e., New_{C1} and New_{C2}) in the set New_N and sends New_N to its parent node P. Thus, between any parent node P and child node N there is an exchange of two sets of undominated solutions: $IRUS_p$ is sent by node P to N and New_N is sent by N to P. Figure 5.4 illustrates the dialogue described above between nodes in the search tree during which there is a transfer of undominated solutions. The elimination of ineffective solutions is carried out by the *Reduce* procedure called in the *CPOptimizer* algorithm.

5.7.3 The *Reduce* procedure

Given a set of solutions and a collection of domains $\mathcal{D}$, this procedure eliminates any solution that is proven to be ineffective. A solution α is said to be ineffective if and only if α non-dominates $\mathcal{D}$ as described in Definition 22 in Section 5.2.1. To prove that a solution is ineffective, the *Reduce* procedure applies the non-dominance rule

described in Section 5.5. Then it returns all remaining solutions that were deemed not to be ineffective (i.e., *RUS*). Once the ineffective solutions are eliminated, the dominance test can be performed in order to check whether $\mathcal{D}$ is dominated.

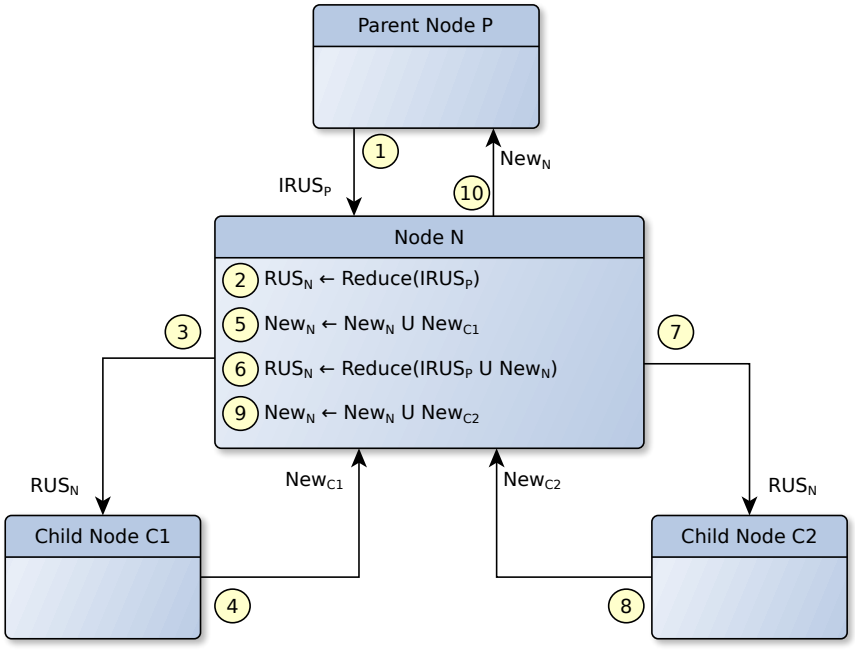


Figure 5.4: Model of transferring the set of undominated solutions between nodes in the search tree

Figure 5.4 shows how a given node N in the search tree transfers and exchanges undominated solutions with its parent node P and its children nodes (i.e., $C1$ and $C2$).

5.7.4 The dominanceTest procedure

Given a partial assignment *b*, the collection of current domains $\mathcal{D}$ and a set of relevant undominated solutions *RUS*, this procedure will attempt to prove that there exists a solution $\alpha \in$ *RUS* that dominates all outcomes that extend *b* according to Definition 22 given in Section 5.2.1. This can be achieved by applying one or a combination of the pruning rules described in Section 5.3, 5.4 and 5.5 for each solution $\alpha \in$ *RUS* until it finds dominance or all solutions in *RUS* have been tried. It will return *true* if it is proven that there is at least one solution α which dominates $\mathcal{D}$.

5.7.5 The CPOptimizer procedure

The navigation in the search tree is ensured by the *CPOptimizer* procedure. At node *N*, when we obtain the solutions newly generated (i.e., New(*N*)), from earlier values

assigned to variable *V(N)*, and the set of undominated solutions already gathered (RUS) we call the *Reduce* procedure. Then, the Reduce procedure takes the union of the two sets (i.e., $New \cup RUS$) and returns the current relevant undominated solutions. Once we obtain the set of *Relevant Undominated Solutions* (RUS), we apply one or a combination of the dominance pruning rules through the procedure *dominanceTest*. It returns *true* when there is at least one solution β already found which dominates the subtree not yet explored below the node *M*. *M* is the child node of the current node after assigning the value *v*. It if returns *false* then we continue exploring the subtree below *M*. Then, CPOptimizer communicates *RUS(N)* to the next child node of *N* that will be created by assigning the next unpruned value (if any) to variable *V(N)*.

At any node *N* of the search tree we call CPOptimizer with the following parameters: X, A, $IRUS$ and $\mathcal{D}(X)$. There is a loop over the values *v* in the current domain $\mathcal{D}(x)$. Each time the variable X is assigned one value $v \in \mathcal{D}(x)$ which extends $A(N)$ to be A_v. The Reduce procedure applies the non-dominance rule so that we obtain the set of relevant solutions gathered so far (i.e., RUS). Procedure *dominanceTest* checks whether there exists at least one relevant undominated solution which dominates all extensions of $A(N)$ (i.e., A_v). If there is no dominance (i.e., isDominated=*false*) then a CPOptimizer procedure is called for a child node. This procedure will have as parameters the next variable according to the variable ordering (i.e., $Next(X)$), the current partial assignment (i.e., A_v), *RUS(N)* and $\mathcal{D}(Next(X))$. It will return all solutions found in subtree ST_v to the calling environment: CPOptimizer called for node *N*.

The calling environment then collects all solutions that were found in subtree ST_v and adds them to the set of newly discovered solutions of node *N* (i.e., New(*N*)). In the case where there is no dominance and *N* is an *end node* (checked by *isEndNode*) we only add the new solution to New(*N*). CPOptimizer procedure called with node *N* will return the undominated solutions recently found below *N* (i.e., New(*N*)) to the parent node of *N*.

5.8 Experimental Testing

5.8.1 Experimental setup

We performed experiments with four families of cp-theories and several sets of binary CSP instances. The CSPs were generated using Christian Bessiere's random uniform CSP generator (`www.lirmm.fr/~bessiere/generator.html`). Experiments

Algorithm 3 CPOptimizer

Input : Variable X
Partial assignment A
Set of inherited relevant solutions *IRUS*
$\mathcal{D}(X)$
Output: Set of newly found undominated solutions *New*

20 $New \leftarrow \emptyset$;

21 **foreach** *value* $v \in D(X)$ **do**
22 $A_v \leftarrow A \cup \{X = v\}$;
$RUS \leftarrow$ Reduce($IRUS \cup New$);
$isDominated \leftarrow$ dominanceTest(A_v , *RUS*);
if *isDominated=false* **then**
23 **if** *isEndNode=true* **then**
24 return A_v ;
25 **else**
26 childNew $\leftarrow$ CPOptimizer(*Next*(*X*), A_v, *RUS*);
$New \leftarrow New \cup$ childNew;

were run as a single thread on Dual Quad Core Xeon CPU, running Linux 2.6.25 x64, with overall 11.76 GB of RAM, and processor speed 2.66 GHz. Solving a CSP involves searching through possible variable assignments, and pruning assignments that violate constraints, to find one or more assignments satisfying all constraints. In these experiments, we enforce arc consistency (Bessière 2006) to generate the collection of domains $\mathcal{D}$ at each node of the search tree. During the search, backtracking occurs when the domain of any variable becomes empty, since there cannot then be any solution extending the partial assignment b at the current node in the search tree.

The conditional preferences impose restrictions on the variable orderings that can be used in the search tree (related to the condition (1) of Proposition 1 in Section 5.3), which much reduces the potential benefit of a dynamic variable ordering. It gains the "anytime" property in return.

For P, Q , define $P \rightarrow Q$ to be the set of edges $\{(X, Y) : X \in P, Y \in Q\}$. Let Γ be the cp-theory. Every preference statement φ has the form $u_\varphi : x_\varphi > x'_\varphi[W_\varphi]$ which means that given an assignment u_φ for a set of variables U_φ, we prefer value x_φ to x'_φ for variable X_φ, as long as variables outside of W_φ are held equal. In Section 2.2.3 and Definition 15 of (Wilson 2011), the author presented a relation denoted by $G(\Gamma)$ which contains sets of edges $U_\varphi \rightarrow X_\varphi$ and $X_\varphi \rightarrow W_\varphi$ for all $\varphi \in \Gamma$.

For the sake of simplicity, we used a fixed variable ordering (which is possible since in the experiments we used only fully acyclic cp-theories (Wilson 2011), including acyclic CP-nets).

Random Generation of Preferences: We consider four families of cp-theories, CP-nets (*CPnet*), a form of lexicographic preferences (*Lex*), a family with varying W component (*Rand-W*) as preference statements have the form the form $u : x > x'[W]$, and CP-nets with local total orderings (*CPn-to*). These are generated as follows. We order the variables V as $X_1, \ldots, X_n$. For each variable X_i we randomly choose the parents set U_i to be a subset of cardinality 0, 1 or 2 of $\{X_1, \ldots, X_{i-1}\}$. For the *CP-net* and *CPn-to* families, we set $W_i = \emptyset$. For the *Lex* family we set $W_i = \{X_{i+1}, \ldots, X_n\}$. For random-$W$ (*Rand-W*) problems we define W_i to be a random subset of $\{X_{i+1}, \ldots, X_n\}$. Then, for each assignment u to U_i, we randomly choose an ordering $x^1, \ldots, x^m$ of the domain of X_i (so we will usually have different orderings for various u). We randomly choose a number of pairs (x^j, x^k) with $j < k$, except for the *CPn-to* family when we include all pairs (x^j, x^{j+1}), for $j = 1, \ldots, |\underline{X_i}| - 1$. For each of these pairs we include the corresponding statement $u : x^j > x^k\ [W_i]$ in the cp-theory Γ. Algorithm 4 shows how preferences are generated through a sequence of steps.

Algorithm 4 PreferenceGenerator

input : Variables $X_1, \ldots, X_n$
output: A set of preferences Γ

```
27 Γ ← ∅  for i ← 1 to n do
28     Choose a set U_i of variables, where U_i is a subset of {X_1, ..., X_{i-1}} and U_i has
       cardinality 0, 1, or 2
29     switch Preference format do
30         case CP-nets or CPn-to
31             W_i ←∅
32         case Rand-W
33             W_i ← random subset of {X_{i+1}, ..., X_n}
34         case Lex
35             W_i ← {X_{i+1}, ..., X_n}
36     for assignment u to U_i do
37         Randomly choose a permutation x_1, x_2, ..., x_k of the domain of X_i  if CPn-to
           then
38             for j ← 1 to k − 1 do
39                 Γ ←Γ ∪ {u : x_l > x_{l+1}[W_i]}
40         else
41             Choose some random integer r between 1 and 0.5 * k^2  for l ← 1 to r do
42                 Choose two random numbers between 1 and k, Let h be the smaller
                   of the two and let j be the bigger  if h ≠ j then
43                     Γ ←Γ ∪ {u : x_h > x_j[W_i]}
```

We consider 12 versions of the algorithm. They differ according to whether they

use root-dominance (labelled **r** in the tables) which is presented in Section 5.3.1, deciding node-dominance (**d**) which is introduced in Section 5.3.2, the projection-dominance condition (**p**) presented in Section 5.3.3, **p(Γ)** dominance (presented in Section 5.4), or the root non-dominance condition (**n**) (presented in Section 5.5). These are compared against the **Basic** algorithm (Section 5.2) which uses none of these additional pruning methods. We also consider some combinations of the methods.

We report the following measures: running time (*Time*), number of visited nodes (*#nd*), number of dominance checks at *end nodes* (*chk*). These measures give an accurate idea about the effect the different pruning rules and their combinations have on avoiding uninteresting subspaces in the search tree. We performed three groups of experiments.

In this experiential setup, we used Algorithm 5 to generate a number of CSP instances whose number of solutions, denoted by *NumberSolutions*, is around a target number (*targetSolutions*) (e.g., 500, 1000, 10,000). Algorithm 5 generates a number k of CSPs whose effective number of solutions (denoted by *NumberSolutions* in the algorithm) differs from *targetSolutions* by at most *p%*. *p* is the maximum percentage of *targetSolutions* that the difference between *NumberSolutions* and *targetSolutions* is allowed to be equal to. In our experiments we set *p* equal to 10. To ensure a CSP instance has a number of solutions that is around *targetSolutions*, we need to generate a CSP instance with an expected number of solutions that is around *targetSolutions* (i.e., they differ by *p%* at most). The expected number of solutions (*expectedSolutions*) for a random CSP can be computed using the number of variables (*n*), the domain size of variables (*m*), the number of constraints (*c*), and the number of incompatible value pairs in each constraint (*i*) by the following formula: *expectedSolutions* = $m^n * (1 - i/m^2)^c$ (Smith 2001). For instance, if a CSP has the following parameters n=10, m=4, c=24 and i=4, then *expectedSolutions* $\approx$ 1052. The number of constraints (*c*) and the number of incompatible value pairs in each constraint *i* are determined in order to have *expectedSolutions* around *targetSolutions*. Two procedures are used in Algorithm 5: *CSPCreate* and *CSPSolve*. The procedure *CSPCreate* creates a random CSP instance based on the parameters n, m, c and i using Christian Bessiere's random uniform CSP generator (`www.lirmm.fr/~bessiere/generator.html`). The procedure *CSPSolve* solves and returns the number of solutions of the CSP instance given as input.

Once *expectedSolutions* is computed, Algorithm 5 creates CSP instances, solves them and keeps only those whose number of solutions differs from *targetSolutions* (i.e., 1000) by $p\%$ at most. Algorithm 5 outputs k (e.g., 50) CSP instances that are stored in *CSPInstances*. This set of CSPs is used in the experiments.

The first group of experiments involves CSPs based on ten four-valued variables (see Table 5.2). Table 5.2 shows comparisons between all the methods for CSPs with

Algorithm 5 CSPGenerator

Input : Number of variables n
Domain size of variables m
Number of constraints c
Number of incompatible value pairs in each constraint i
Target number of solutions *targetSolutions*
Percentage of allowed gap between *targetSolutions* and effective number of solutions p
Number of instances required k
Output: *CSPInstances*

```
44 CSPInstances ← ∅
45 for j ← 1 to k do
46     repeat
47         CSPInstance ← CSPCreate(n, m, i)
48         NumberSolutions ← CSPSolve(CSPInstance)
49     until |NumberSolutions - targetSolutions| ≤ (p / 100) * targetSolutions;
50     CSPInstances ← CSPInstances ∪ {CSPInstance}
```

around 500 solutions. This group is intended to evaluate ten versions of the algorithm. Evaluating the pruning rules and their combinations involves comparing each rule to the basic approach and to other rules. This will help assess the added value of each of these rules with regards to the basic approach in particular and the other rules as well.

Table 5.2 shows, for each preference family and method, the number of optimal solutions (*#sol*), the Central Processing Unit (CPU) running time (*Time*), the number of visited nodes (*#nd*) and the number of dominance checks at *end nodes* (*#chk*).

Table 5.2 gives the results of experiments with 50 CSP instances whose average number of solutions is around 500. The measures shown are averaged over 50 CSP instances.

The same experiments' results are shown in separate tables for each preference family: *CPnet* (see Table 5.4), *Rand-W* (see Table 5.5), *Lex* (see Table 5.6) and *CPn-to* (see Table 5.7). We just focus on a fewer number of methods mentioned in Table 5.2 in addition to **p(**Γ**)** (presented in Section 5.4) and **p(**Γ**)+n**, which is a combination between **p(**Γ**)** and root non-dominance (**n**). In fact, the number of optimal solutions found by **p(**Γ**)** and **p(**Γ**)+n** is not necessarily the same as the number of optimal solutions found by the other sound methods (e.g., **Basic**, **r**, **d**, **n**) and the different combinations of these methods. In these tables, we mention the number of optimal solutions, denoted by *#sol*, for each method and preference family, as well as other additional measures: *ratio*, *SDTime* and *frac-sol*. *ratio* denotes the average time (in ms) per solution found. The smaller this value is the better. The standard deviation from the average CPU time (*SDTime*) gives an idea of how much the time achieved

by a method for one CSP instance differs from the mean time achieved over 50 instances. *frac-sol* denotes the fraction of optimal solutions found by a method with regards to the number of optimal solutions found by the Basic method. It informs the reader about the proportion of solutions found by a method and so the proportion of solutions that a method missed (e.g., p(Γ) missed 62% of the total number of solutions found by Basic method). These measures give the opportunity to analyse and compare the most relevant sound methods (i.e., **Basic**, **r**, **d**, **n**) and their combinations (i.e., **r+d**, **r+d+n**) with the unsound method (i.e., **p(Γ)**) and its combination with the root non-dominance (i.e., **p(Γ)+n**).

The second group of experiments involves 50 CSP instances with ten four-valued variables (see Table 5.3). Table 5.3 concerns two groups of CSPs, where, we show results for two of the best combinations: (1) root-dominance (**r**) combined with deciding node-dominance (**d**), this combination of methods is denoted by **r+d**; and (2) root-dominance (**r**) combined with deciding node-dominance (**d**) and root non-dominance (**n**), this combination of methods is denoted by **r+d+n**.

These two families of CSPs differ on the number of feasible solutions: 1)the first subset has an average number of solutions around 2000. The second subset has an average number of 10000 solutions. Only running time is reported for this group of experiments. The computation time is revealed to be clearly dependant on the number of solutions of the CSP.

One of the goals of these experiments is to check that the improvement made by the two best combinations of pruning methods regarding the basic method still has the same amplitude with large number of solutions.

The third group of experiments aims at assessing the performance of the **Basic** method and the two combinations of methods **r+d** and **r+d+n** by computing the running time when the number of variables varies within a range. In these experiments, we have 50 CSP instances with n three-valued variables. When n varies from 10 to 40 in steps of 5, the average number of solutions of the randomly generated CSP instances was approximately 1000. We generated performance curves that show the average running time as a function of n (the number of variables). Results are shown in Figure 5.5.

The second and third set of experiments both focus on two of the most promising combinations, **r+d** and **r+d+n**.

All figures in the tables and graphs are the mean over 50 random instances.

5.8.2 Discussion of results

The experimental results confirmed that no optimal solutions were lost by the additional pruning methods (as implied theoretically by Proposition 2 in Section 5.3.1,

Table 5.2: Mean number of optimal solutions for each preference family, and CPU time (ms), number of visited nodes and number of dominance checks at *end nodes* for each preference family and each method. The CSPs averaged around 500 solutions.

	CP-nets			**Rand-W**			**Lex**			**CPn-to**		
# opt:	87.74			35.74			25.18			13.56		
Methods	*Time*	*#nd*	*chk*	*Time*	*#nd*	*chk*	*Time*	*#nd*	*chk*	*Time*	*#nd*	*chk*
Basic	7826	1173	22430	1941	1173	7247	1199	1173	5260	2169	1173	2248
r	9672	1172	22421	2532	881	7001	1408	740	4749	2394	1148	2227
d	3576	536	7956	536	192	1465	196	95	625	176	148	148
r+d	3582	536	7956	534	191	1465	196	95	625	173	148	148
p	34037	1173	89680	3391	35	1230	658	95	2855	10871	1173	12745
n	775	1173	979	802	1173	2155	508	1173	1568	853	1173	560
r+n	836	1172	978	967	881	2012	591	740	1375	897	1148	545
d+n	244	536	288	219	192	501	97	95	244	44	148	13
r+d+n	243	536	288	220	191	500	95	97	244	42	148	13
p+n	4929	1173	6136	1636	288	4331	318	95	960	4944	1173	4170

Table 5.3: Mean number of optimal solutions for each preference family, and running times (ms) for each family and each method.

	CPnet	**Rand-W**	**Lex**	**CPn-to**
10 vars, 4 values, Mean 1993 solutions				
# opt	221.2	73.0	39.5	16.4
Basic	62608	14711	6496	13728
r+d	31998	3204	673	651
r+d+n	2164	1557	509	445
10 vars, 4 values, Mean 9910 solutions				
# opt	364.8	204.5	133.8	6.5
Basic	564733	183710	110303	29285
r+d	278583	28666	8482	358
r+d+n	18623	14595	5307	352

Table 5.4: CPU time (Time), mean number of optimal solutions (#sol), number of visited nodes (#nd), ratio, standard deviation from the average time (SDTime) and frac-sol for **CP-nets family** and each method. The CSPs averaged around 500 solutions.

Methods	*Time*	*#sol*	*#nd*	*ratio*	*SDTime*	*frac-sol*
Basic	7826	87.74	1173.62	89.19	6600	1
r+d	3582	87.74	536.78	40.82	2623	1
p(Γ)	369	35.5	143.74	10.38	498.5	0.38
r+d+n	243	87.74	536.78	2.77	244.6	1
p(Γ)+n	32	35.5	143.74	0.90	35.9	0.38

Table 5.5: Mean number of optimal solutions for each preference family, and CPU time (ms), the ratio (Time/#sol), the frac-sol (#sol/#sol for Basic) and SDTime (standard deviation from the average time) for **Rand-W family** and each method. The CSPs averaged around 500 solutions.

Methods	*Time*	*#sol*	*#nd*	*ratio*	*SDTime*	*frac-sol*
Basic	1941	35.74	1173.62	54.31	2371.4	1
r+d	534	35.74	191.28	14.94	662.1	1
p(Γ)	129	20.5	83.98	94.9	198.5	0.60
r+d+n	220	25.18	191.28	6.17	303.5	1
p(Γ)+n	63	20.5	84.34	3.06	94.7	0.60

Table 5.6: Mean number of optimal solutions for each preference family, and CPU time (ms), the ratio (Time/#sol), the frac-sol (#sol/#sol for Basic) and SDTime (standard deviation from the average time) for **Lex family** and each method. The CSPs averaged around 500 solutions.

Methods	*Time*	*#sol*	*#nd*	*ratio*	*SDTime*	*frac-sol*
Basic	1199	25.18	1173.62	47.64	1445.2	1
r+d	196	25.18	95.12	7.78	317.6	1
p(Γ)	180	25.18	95.12	6.35	299.8	1
r+d+n	97	25.18	95.12	6.17	148.7	1
p(Γ)+n	95	25.18	95.12	3.78	145.1	1

Table 5.7: Mean number of optimal solutions for each preference family, and CPU time (ms), the ratio (Time/#sol), the frac-sol (#sol/#sol for Basic) and SDTime (standard deviation from the average time) for **CPn-to family** and each method. The CSPs averaged around 500 solutions.

Methods	*Time*	*#sol*	*#nd*	*ratio*	*SDTime*	*frac-sol*
Basic	2169	13.56	1173.62	160	1427.4	1
r+d	173	13.56	148.14	12.78	139.1	1
p(Γ)	11	3.5	34.46	3.35	19.4	0.28
r+d+n	42	13.56	148.14	3.14	45.4	1
p(Γ)+n	5	3.5	34.46	1.59	9.4	0.28

Proposition 3 in Section 5.3.2 and Proposition 4 in Section 5.5). For each preference family, the number of optimal solutions is the same for all methods, except p(Γ), as these methods are complete.

Analyzing the results in Table 5.2, Table 5.3, Table 5.4, Table 5.5, Table 5.6, Table 5.7 and Figure 5.5 leads to the following general conclusions about the comparisons between the different versions of the algorithm.

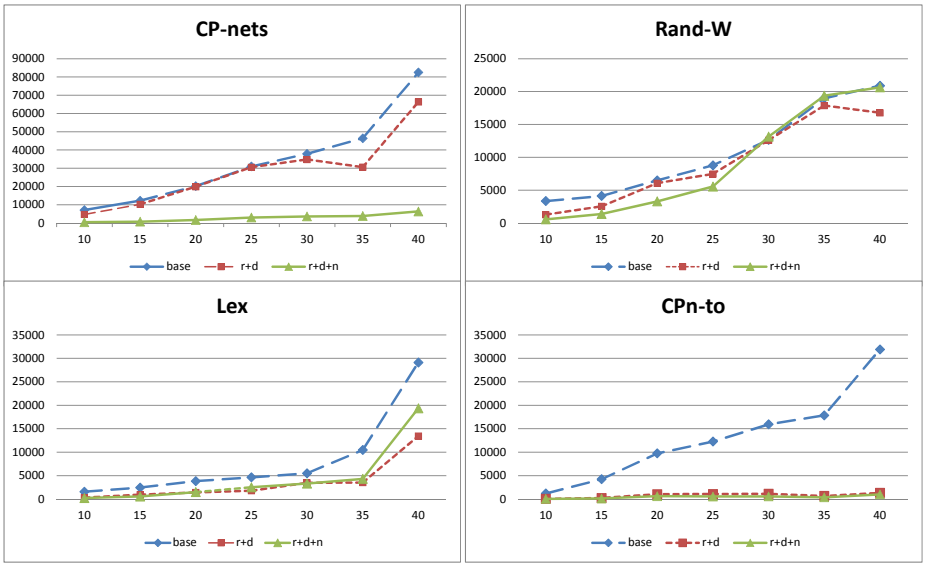


Figure 5.5: Running time(ms) for each family of preferences for $n = 10, 15, \ldots, 40$ variables (having 3 values each). Each CSP has approximately 1000 solutions.

The deciding-node-dominates rule (**d**) appears to be much the most effective pruning method among the dominance pruning rules (i.e., **r**, **d**, **p** and p(Γ)). With this rule the number of visited nodes, and the number of dominance checks, is reduced significantly with regards to the basic algorithm. It also cuts down the number of relatively expensive dominance checks performed at *end nodes*. It can make an order of magnitude improvement over the Basic approach. For instance, Table 5.2 shows method **d** run for 176 ms to enumerate optimal solutions found by **Basic** approach in 2169 ms for *CPn-to* preference family.

The root-dominates rule (**r**) is not that efficient in pruning nodes. We can notice in Table 5.2 that for *Rand-W* preference family, for instance, the average running time achieved by method **r** is equal to 2532 ms while the **Basic** approach spends 1941 ms in average to find the same set of optimal solutions. Method **r** is slow as it prunes only 292 (i.e., 1173-881) nodes in the search tree comparing to **d** which prunes 981 (i.e., 1173-192) nodes to gather all optimal solutions in 536 ms for *Rand-W* family for which **d** performed 1465 dominance checks against 7001 done by **r**. In fact, the application of **r** dominance condition tests at all nodes was detrimental as these tests did not significantly reduce the size of the search tree.

Root-dominates (**r**) can be slightly useful when used in conjunction with the deciding-node-dominates rule (**d**) to set the combination (**r+d**). Indeed, it can sometimes be effective when the deciding-node-dominates rule (**d**) is unable to prune more irrelevant solutions. In Table 5.2, we can see a very slight improvement of the running time achieved by (**r+d**) against (**d**) for *CPn-to*: 173 ms **r+d** against 176 by **d**.

The projection-dominates rule (**p**) was not effective. With **p**, there was no pruning for *CPnet* and *CPn-to* while it prunes significantly the search tree for *Rand-W* (35 vs 1173 for **Basic** approach) and *Lex* (95 vs 1173 for **Basic** approach). The pruning capability did not speed up **p** because of the costliness of the **p** dominance condition test that was applied at all nodes and not only *end nodes* (see Table 5.2).

The root non-dominance condition can improve the performance of the algorithm considerably, especially for the *CPnet* and *CPn-to* families, since it can substantially reduce the number of dominance tests. For instance, for the *CPnet* family, **n** can make an order of magnitude improvement over the **Basic** approach (775ms vs 7826ms). This rule is capable of bringing down the running time achieved by most of the versions of the algorithm.

For the *CPnet* family, the **p(**Γ**)** pruning rule finds only 38% of the total number of solutions obtained by the **Basic** approach (see Table 5.4) while p(Γ) finds all the optimal solutions for *Lex* family (see Table 5.6). For Lex family, p(Γ) finds a solution every period of time that lasts less than 4 milliseconds. We can see that **r+d+n** is able to find solutions faster than the other methods except (p(Γ)+n) for all preference families. Tables 5.4, 5.5, 5.6 and 5.7 also show that the **r+d+n** method is the fastest to find optimal solutions among the complete methods.

In Figure 5.5, the results show that the computation time does not increase very strongly with the number of variables. (By the way, it turns out that for each preference family, the mean number of optimal solutions does not vary markedly with n, being centred on around $160, 90, 60$ and 9 for the CP-net, Rand-W, Lex and CPn-to families, respectively.) For the Rand-W family, the new algorithms do not perform much (if at all) better than the basic algorithm. For the Lex family, the two new algorithms are often twice as fast as the basic one. For the *CPnet* family, the **r+d** combination is only slightly better than the basic algorithm, but performs excellently on the *CPn-to* family with mostly an order of magnitude improvement, as does the **r+d+n** algorithm, which also shows more than an order of magnitude strong speed up for the CP-nets family.

5.8.3 Synthesis of discussion

The experimental results have shown the relevance of the developed pruning rules for constrained optimisation for comparative preferences. Repeated tests proved these methods were able to improve the performance of the basic algorithm by an order of magnitude improvement for, at least, the two CP-nets families.

5.9 Conclusion

In this chapter, we proposed new methods for finding non-dominated feasible solutions with respect to a set of comparative preferences. These pruning rules were derived from the user preferences in an attempt to guide constrained optimisation algorithms to find optimal solutions for COPs where preferences can be expressed as a set of comparative preferences. To make the presentation concrete, we applied the new methods to random CSPs with different sizes. The resulting pruning techniques have shown appreciable initial promise for solving constrained optimisation for comparative preferences.

One of the main research questions that was set forth in this chapter was to investigate mechanisms to integrate cp-theories to optimisation algorithms, which produce more efficient algorithms than the state-of-the-art.

This work also aims at supporting decoupled COP approaches that blend reasoning about preferences with reasoning about feasibility. This is done by providing and testing in practice a new COP approach that brings conditional preference theories to support preference representation and reasoning while relying on CSP solving methods to look for feasible paths in the search tree.

6 Conclusion

In this chapter we discuss what has been achieved in this thesis and we describe a number of possible directions we would like to pursue in the future.

6.1 Summary

The need for effective technologies to help web users locate items (information or products) is increasing as the amount of information on the web is growing. User preferences play an important role in both preference-based recommender systems (RSs) and constraint optimisation-based systems. In this thesis, we have made contributions in both directions.

We considered preference elicitation in conversational RSs for which pruning non-optimal options is one of the main tasks. This type of RSs provides challenges for more elaborate formalisms that handle the user preferences while conversing with her. This thesis presents the motivation and the description of a comparative preferences-based approach for conversational RSs. These RSs infer user preferences from the user actions (e.g., to choose or reject an option). Then, they use a comparative preferences-based reasoning engine to select the optimal options to be shown to the user, amongst which the user chooses the option that fits her needs the best. We show that comparative preference theories offers the system the capability of capturing and dealing with multiple preference nuances and diverse forms of preferences without sacrificing the attractive computational properties of the preference

dominance.

The applicability of the comparative preferences-based approach for RSs were established via several experiments that we have performed. These same experiments have shown that the comparative preferences-based dominance algorithm is proven to work efficiently for a range of comparative preference statements when checking dominance between two outcomes α and β.

We also considered constraint optimisation problems (COPs) where optimality is checked with regards to a set of comparative preferences. In order to be easily applicable to COPs for which preferences are expressed as comparative preferences, the B & B technique was extended for comparative preferences. In fact, we developed pruning rules and appended them to the B & B technique. This helps the corresponding constrained optimisation algorithm identify and avoid paths that lead to non-optimal solutions. These rules are based on and derived from the user preferences expressed as comparative preference statements. They aim at reducing the number of pairwise comparisons performed during the search while guiding constrained optimisation algorithms to find optimal solutions. When applied with a set of random binary CSPs and a set of comparative preferences, these rules showed a promising pruning capability when solving COPs for comparative preferences. Our experimental results have given evidence that the use of these pruning rules can make a major difference to the efficiency of constrained optimisation for particular comparative preference families including CP-nets.

6.2 Future directions

There are several future directions which will be worth investigating. The results of this thesis provide a motivation to continue exploring searching for formalisms that can handle and reason with preferences in RSs and other applications that involve COPs.

More formalisms for preference elicitation in conversational RSs: From the AI perspective, preference elicitation presents a bottleneck for designing automated decision aids ranging from critical financial, medical, and logistics domains to RSs or automated software configuration (Braziunas 2006). Therefore, designing effective preference elicitation techniques is an important problem facing AI. As a continuation of the work done in this thesis, we intend to look for more elaborate and intuitive preference elicitation formalisms that we can prove to be efficient in practice with conversational RSs. These formalisms will adapt with the different dialogue strategies the conversational RSs go through.

In this dissertation, we have studied comparative preference representations that

involve multi-valued features where the user model is represented by a vector of weights. We need to go further beyond the preference forms we have already tested to look for more suitable ways of expressing the user's preferences over multi-valued features in a RS context.

Tuning the size of the set of user models: In general, the goal of RSs is to show to the user a set of alternatives that lead her to the best possible alternatives. To do this, RSs usually tend to adjust, as much as needed, the set of alternatives to those that are not dominated by any other alternative and whose number is adapted to the context of use of the system, based on the user's preferences (e.g., comparisons between pairs of alternatives). For instance, if the reduced alternative set is too big for the decision maker to make a choice or the device is too small to conveniently display this set of alternatives, then the system should find a way to adapt (e.g., reduce) the set of non-dominated alternatives, and continue the process as long as is necessary for optimal alternative selection. In case the system opts for narrowing the size of the set of alternatives to be shown, the pruning needs to be stronger. In a different case, the reduced alternative set might be too small and the user would need too many iterations in a conversational RS to get what she wants. Then, allowing more options to be shown is more convenient and so the pruning needs to be weaker.

In the context of RSs, we are planning to deeply study the size of the set of user models that is used in the dominance verification. In fact, the size of the set of user models affects the dominance relation in the sense that the pruning capability can be somewhat controlled (e.g., stronger or weaker) when tuning the size of the set of user models. We also plan to investigate more applications of RSs and study how the dominance relation adapts with regards to the circumstances of use of RSs. We are going to implement different ways of controlling the dominance relation. The combination of the two dominance approaches described in Chapter 4 would be one approach to start with.

Real-world applications of the comparative preferences-based pruning rules: There is also room for future research in the area of COPs whose preferences are expressed as comparative preferences. The pruning rules defined in this thesis were tested with only randomly generated problems. We can look for applications of the pruning rules we have defined in this thesis. This will be one way to validate the claimed advantage of a number of these pruning rules in practice with real-world constrained optimisation problems, e.g., design engineering (D'Ambrosio & Birmingham 1995).

Experimentations performed in Chapter 5 have shown that the non-dominance rule can drastically reduce the number of ineffective dominance checks below a node

in the search tree, and so the running time. This pruning rule is likely to have the same (positive) effect when applied to similar problems where assignments are ordered through a partial preference relation. Thus, we can broaden the set of real-world applications that can benefit from this pruning rule.

Dynamic variable orderings in the search tree: The search trees we use in Chapter 5 are generated by instantiating variables in an order that is compatible with the conditional preferences. First, static variable orderings can be considered as a limitation of the proposed pruning methods. To amend this limitation, we could develop an approach that uses dynamic variable orderings, making use of the consistency conditions from Section 6 of (Wilson 2011). Secondly, when the set of preferences is inconsistent (i.e., the induced order is not acyclic), there would not be a compatible search tree. In this case, there is no guarantee that a solution α that was already found in the search tree, and included in the current set of solutions K, will not be dominated by another solution β that was found later. Hence, we need to check also if β dominates α, as well as if α dominates β. This implies that an outcome $\alpha \in K$ can be dominated by a newly found solution β and so be eliminated from K. Thus, K is not monotonically increasing as it can lose elements as well as gain them.

Application of the pruning rules in RSs: Suitable comparative preferences-based techniques were developed for RSs and COPs. Some techniques can be adapted and used in both contexts: RSs and COPs. For instance, the pruning rules developed for COPs can also be applied for RSs. In this dissertation we considered the case where the set Ω of outcomes is expressed as the solutions of a CSP. In the context of RSs, the set of outcomes can represent a set of available products, for example, and might be listed explicitly. The new constrained optimisation algorithms developed in this dissertation apply here as well. Thus, we can define dynamic variable and value orderings that determine a search tree compatible with a set of the comparative preferences; this search tree can be used to explore Ω (which is then implicitly being expressed as a decision tree), and find the optimal ones, using, as in Chapter 5, the dominance and non-dominance pruning rules to prune the search tree and reduce the dominance checks. The final result would be the set of optimal products selected with regards to the user preferences.

In conclusion, comparative preferences theories appear to be a promising concept that can be integrated not only in RSs but also in real-world applications that aim at finding the optimal alternative with regards to the user's preferences (e.g., in configuration or in planning problems). We are looking forward to new ideas about

how to exploit the polynomial dominance related to the comparative preferences, with real-world problems, in order to speed up the selection process without giving up on the intuitive and easy to elicit aspects of this type of preference statement.

Acronyms

AI Artificial Intelligence.

APA Adaptive Place Advisor.

B & B Branch and Bound.

BNs Bayesian Networks.

CBR Case-Based Reasoning.

CD Compact Disks.

COPs constrained optimization problems.

CP-nets Conditional Preference Networks.

CPT conditional preference table.

CPU Central Processing Unit.

CSP constraint satisfaction problem.

CSP constraint satisfaction problem.

DSSs decision support systems.

GAI generalized additive independence.

IR Information Recommendation.

MAUT multi-attribute utility theory.

PC Personal Computer.

QCL Qualitative Choice Logic.

RS Recommender System.

RSs Recommender Systems.

TCP-nets Tradeoff-Conditional Preference Networks.

UM User modeling.

References

A. Caprara, P. Toth, D. V. & Fischetti, M. (1998), 'Modeling and solving the crew rostering problem', *Operations Research* **46**, 820–830.

Adomavicius, G. & Tuzhilin, A. (2005), 'Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions', *IEEE Transactions on Knowledge and Data Engineering* **17**(6), 734–749.

Agarwal, P. K. & Sharir, M. (1998), 'Efficient algorithms for geometric optimization', *ACM Computing Surveys* **30**(4), 412–458.

Alexander, S. H. (1977), *The New Science of Management Decision*, Prentice Hall PTR, Upper Saddle River, NJ, USA.

Amazon (2012), `http://www.amazon.com` (last accessed 3rd May, 2013).

Anand, S. S. & Mobasher, B. (2005), Intelligent techniques for web personalization, *in* S. Anand & B. Mobasher, eds, 'Intelligent Techniques for Web Personalization, IJCAI 2003 Workshop, ITWP 2003, Acapulco, Mexico, August 11, 2003, Revised Selected Papers', Springer, pp. 1–36.

Andréka, H., Ryan, M. & Schobbens, P.-Y. (2002), 'Operators and laws for combining preference relations', *Journal of Logic and Computation* **12**(1), 13–53.

Apt, K. R., Rossi, F. & Venable, K. B. (2005), 'CP-nets and nash equilibria', *CoRR* .

Ardissono, L., Goy, A., Petrone, G., Segnan, M. & Torasso, P. (2003), 'Intrigue: Personalized recommendation of tourist attractions for desktop and hand held devices', *Applied Artificial Intelligence* **17**(8-9), 687–714.

Axling, T. & Haridi., S. (1994), 'A tool for developing interactive configuration applications', *Journal of Logic Programming* **19**, 658–679.

Bacchus, F. & Grove, A. J. (1995), Graphical models for preference and utility, *in* P. Besnard & S. Hanks, eds, 'Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-95), Montreal, Quebec, Canada, August 18-20, 1995', Morgan Kaufmann, pp. 3–10.

Balabanovic, M. (1998), 'Exploring versus exploiting when learning user models for text recommendation', *User Modeling and User-Adapted Interaction* **8**, 71–102.

Balduccini, M. (2005), Answer set based deesign of highly autonomous, rational agents, PhD thesis, Texas Tech University.

Balduccini, M. & Mellarkod, V. S. (2003), Cr-prolog with ordered disjunction, *in* M. D. Vos & A. Provetti, eds, 'Answer Set Programming', Vol. 78 of *CEUR Workshop Proceedings*, CEUR-WS.org.

Barker, V. E., O'Connor, D. E., Bachant, J. & Soloway, E. (1989), 'Expert systems for configuration at digital: Xcon and beyond', *Communication ACM* **32**(3), 298–318.

Barták, R. (2012), 'Constraint online guide', `http://ktiml.mff.cuni.cz/~bartak/constraints/` (last accessed 3rd May, 2013).

Bell, D. E., Raiffa, H. & Tversky, A. (1988), *Decision making: Descriptive, normative and prescriptive interactions*, Cambridge University Press.

Benferhat, S., Cayrol, C., Dubois, D., Lang, J. & Prade, H. (1993), Inconsistency management and prioritized syntax-based entailment, *in* R. Bajcsy, ed., 'IJCAI', Morgan Kaufmann, pp. 640–647.

Benferhat, S., Dubois, D. & Prade, H. (2001), 'Towards a possibilistic logic handling of preferences', *Applied Intelligence* **14**(3), 303–317.

Berkovsky, S., Kuflik, T. & Ricci, F. (2008), 'Mediation of user models for enhanced personalization in recommender systems', *User Modeling and User-Adapted Interaction* **18**(3), 245–286.

Berkovsky, S., Kuflik, T. & Ricci, F. (2009), 'Cross-representation mediation of user models', *User Modeling and User-Adapted Interaction* **19**(1-2), 35–63.

Bessière, C. (2006), Constraint propagation, *in* F. Rossi, P. van Beek & T.Walsh, eds, 'Handbook of Constraint Programming', Elsevier.

Bessière, C., Zanuttini, B. & Fernandez, C. (2004), Measuring search trees, *in* 'Proceedings of Proceedings of the sixteenth Eureopean Conference on Artificial Intelligence(ECAI-04) Workshop on Modelling and Solving Problems with Constraints', IOS Press, pp. 31–40.

Bettman, J., Luce, M. & Payne, J. (1998), 'Constructive consumer choice processes', *Journal of Consumer Research* **25**, 187–217.

Bienvenu, M., Fritz, C. & McIlraith, S. A. (2006), Planning with qualitative temporal preferences, *in* 'Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR-06)', Lake District, UK, pp. 134–144.

Bienvenu, M., Fritz, C. & McIlraith, S. A. (2011), 'Specifying and computing preferred plans', *Artificial Intelligence* **175**(7-8), 1308–1345.

Bienvenu, M., Lang, J. & Wilson, N. (2010), From preference logics to preference languages, and back, *in* Lin et al. (2010).

Binshtok, M., Brafman, R. I., Shimony, S. E., Mani, A. & Boutilier, C. (2007), Computing optimal subsets, in *AAAI* (*Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada* 2007), pp. 1231–1236.

Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G. & Fargier, H. (1999), 'Semiring-based csps and valued csps: Frameworks, properties, and comparison', *Constraints* **4**(3), 199–240.

Bitner, J. R. & Reingold, E. M. (1975), 'Backtrack programming techniques', *Communications ACM* **18**(11), 651–656.

Biundo, S., Myers, K. L. & Rajan, K., eds (2005), *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, AAAI.

Blanco, H., Ricci, F. & Bridge, D. (2012*a*), Conversational query revision with a finite user profiles model, *in* G. Amati, C. Carpineto & G. Semeraro, eds, 'Procs. of the Third Italian Information Retrieval Workshop (IIR)', Vol. 835 of *CEUR Workshop Proceedings*, pp. 77–88.

Blanco, H., Ricci, F. & Bridge, D. (2012*b*), Recommending personalized query revisions, *in* M. de Gemmis et al., ed., 'Procs. of Decisions@RecSys: The Workshop on Human Decision Making in Recommender Systems (Workshop Programme of the Sixth ACM Conference on Recommender Systems)'.

Blum, A. L. & Furst, M. L. (1995), 'Fast planning through planning graph analysis', *Artificial Intelligence* **90**(1), 1636–1642.

Boerkoel, J. C., Durfee, E. H. & Purrington, K. (2010), Generalized solution techniques for preference-based constrained optimization with (cp)-nets, *in* W. van der Hoek,

G. A. Kaminka, Y. Lespérance, M. Luck & S. Sen, eds, 'AAMAS', IFAAMAS, pp. 291–298.

Bonzon, E., Lagasquie-Schiex, M.-C., Lang, J. & Zanuttini, B. (2009), 'Compact preference representation and boolean games', *Autonomous Agents and Multi-Agent Systems* **18**(1), 1–35.

Boschetti, M. A., Maniezzo, V., Roffilli, M. & Röhler, A. B. (2009), Matheuristics: Optimization, simulation and control, *in* 'Proceedings of the 6th International Workshop on Hybrid Metaheuristics', HM '09, Springer-Verlag, Berlin, Heidelberg, pp. 171–177.

Boubekeur, F., Boughanem, M. & Tamine-Lechani, L. (2006), Towards flexible information retrieval based on cp-nets, *in* 'FQAS', pp. 222–231.

Boutilier, C., Bacchus, F. & Brafman, R. I. (2001), UCP-networks: A directed graphical representation of conditional utilities, *in* J. S. Breese & D. Koller, eds, 'UAI', Morgan Kaufmann, pp. 56–64.

Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H. & Poole, D. (2004*a*), 'CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements', *Journal of Artificial Intelligence Research* **21**, 135–191.

Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H. & Poole, D. (2004*b*), 'Preference-based constrained optimization with CP-nets', *Computational Intelligence* **20**(2), 137–157.

Boutilier, C., Brafman, R. I., Geib, C. W. & Poole, D. (1997), A constraint-based approach to preference elicitation and decision making, *in* 'AAAI Spring Symposium on Qualitative Decision Theory', Stanford.

Boutilier, C., Brafman, R. I., Hoos, H. H. & Poole, D. (1999), Reasoning with conditional ceteris paribus preference statements, *in* K. B. Laskey & H. Prade, eds, 'UAI', Morgan Kaufmann, pp. 71–80.

Boutilier, C., ed. (2009), *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*.

Bouveret, S., Endriss, U. & Lang, J. (2009*a*), Conditional importance networks: A graphical language for representing ordinal, monotonic preferences over sets of goods, *in* Boutilier (2009), pp. 67–72.

Bouveret, S., Endriss, U. & Lang, J. (2009*b*), Conditional importance networks: A graphical language for representing ordinal, monotonic preferences over sets of goods, *in* Boutilier (2009), pp. 67–72.

Bouyssou, D., Marchant, T., Pirlot, M., Perny, P., Tsoukiàs, A. & Vincke, P. (2000), *Evaluation and decision models: a critical perspective*, Kluwer Academic, Dordrecht.

Bouyssou, D., Marchant, T., Pirlot, M., Tsoukiὰ, A. & Vincke, P. (2006), *Evaluation and decision models with multiple criteria: Stepping stones for the analyst*, International Series in Operations Research and Management Science, Volume 86, 1st edn, Springer, Boston.

Brafman, R. I. & Chernyavsky, Y. (2005), Planning with goal preferences and constraints, *in* Biundo et al. (2005), pp. 182–191.

Brafman, R. I. & Dimopoulos, Y. (2004*a*), 'Extended semantics and optimization algorithms for CP-networks', *Computational Intelligence* **20**(2), 218–245.

Brafman, R. I. & Dimopoulos, Y. (2004*b*), 'Extended semantics and optimization algorithms for CP-networks', *Computational Intelligence* **20**, 2004.

Brafman, R. I. & Domshlak, C. (2002), Introducing variable importance tradeoffs into CP-nets, *in* A. Darwiche & N. Friedman, eds, 'UAI', Morgan Kaufmann, pp. 69–76.

Brafman, R. I. & Domshlak, C. (2008), 'Graphically structured value-function compilation', *Artificial Intelligence* **172**(2-3), 325–349.

Brafman, R. I. & Domshlak, C. (2009), 'Preference handling - an introductory tutorial', *AI Magazine* **30**(1), 58–86.

Brafman, R. I., Domshlak, C. & Shimony, S. E. (2003), 'TCP-nets - introducing variable importance tradeoffs into CP-nets'.

Brafman, R. I., Domshlak, C. & Shimony, S. E. (2006), 'On graphical modeling of preference and importance', *Journal of Artificial Intelligence Research* **25**, 389–424.

Brafman, R. I., Domshlak, C., Shimony, S. E. & Silver, Y. (2006), Preferences over sets, *in* 'AAAI', AAAI Press, pp. 1101–1106.

Braziunas, D. (2006), Computational approaches to preference elicitation, Technical report, Department of Computer Science,University of Toronto.

Brearley, A. L., Mitra, G. & Williams, H. P. (1975), 'Analysis of mathematical programming problems prior to applying the simplex algorithm', *Mathematical Programming* **8**, 54–83.

Brélaz, D. (1979), 'New methods to color the vertices of a graph', *Communication ACM* **22**(4), 251–256.

Brewka, G. (2002), Logic programming with ordered disjunction, *in* R. Dechter & R. S. Sutton, eds, 'AAAI/IAAI', AAAI Press / The MIT Press, pp. 100–105.

Brewka, G. (2004*a*), Complex preferences for answer set optimization, *in* D. Dubois, C. A. Welty & M.-A. Williams, eds, 'KR', AAAI Press, pp. 213–223.

Brewka, G. (2004*b*), A rank based description language for qualitative preferences, *in* de Mántaras & Saitta (2004), pp. 303–307.

Brewka, G., Benferhat, S. & Berre, D. L. (2002), Qualitative choice logic, *in* Fensel et al. (2002), pp. 158–169.

Brewka, G., Benferhat, S. & Berre, D. L. (2004), 'Qualitative choice logic', *Artificial Intelligence* **157**(1-2), 203–237.

Brewka, G., Coradeschi, S., Perini, A. & Traverso, P., eds (2006), *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*, Vol. 141 of *Frontiers in Artificial Intelligence and Applications*, IOS Press.

Brewka, G., Niemelä, I. & Syrjänen, T. (2002), Implementing ordered disjunction using answer set solvers for normal programs, *in* S. Flesca, S. Greco, N. Leone & G. Ianni, eds, 'JELIA', Vol. 2424 of *Lecture Notes in Computer Science*, Springer, pp. 444–455.

Brewka, G., Niemelä, I. & Syrjänen, T. (2004), 'Logic programs with ordered disjunction', *Computational Intelligence* **20**(2), 335–357.

Brewka, G., Niemelä, I. & Truszczynski, M. (2003), Answer set optimization, *in* Gottlob & Walsh (2003), pp. 867–872.

Bridge, D. G. (2002), Towards conversational recommender systems: A dialogue grammar approach, *in* Craw & Preece (2002), pp. 9–22.

Bridge, D. G., Göker, M. H., McGinty, L. & Smyth, B. (2005), 'Case-based recommender systems', *The Knowledge Engineering Review* **20**(3), 315–320.

Bridge, D. G. & Ricci, F. (2007), Supporting product selection with query editing recommendations, *in* J. A. Konstan, J. Riedl & B. Smyth, eds, 'RecSys', ACM, pp. 65–72.

Bridge, D., Göker, M., McGinty, L. & Smyth, B. (2006), 'Case-based recommender systems', *The Knowledge Engineering review* **20**(3), 315–320.

Brush, A. B., Krumm, J. & Scott, J. (2010), Exploring end user preferences for location obfuscation, location-based services, and the value of location, *in* 'Proceedings of the 12th ACM international conference on Ubiquitous computing', Ubicomp 10, ACM, New York, NY, USA, pp. 95–104.

Brusilovsky, P., Kobsa, A. & Nejdl, W., eds (2007), *The Adaptive Web, Methods and Strategies of Web Personalization*, Vol. 4321 of *Lecture Notes in Computer Science*, Springer.

Burke, R. (2000), Knowledge-based recommender systems, *in* 'Encyclopedia of Library and Information Systems', Vol. 69.

Burke, R. D. (1999), The wasabi personal shopper: A case-based recommender system, *in* Hendler & Subramanian (1999), pp. 844–849.

Burke, R. D. (2002), 'Hybrid recommender systems: Survey and experiments', *User Modeling and User-Adapted Interaction* **12**(4), 331–370.

Burke, R. D. (2004), Hybrid recommender systems with case-based components, *in* Funk & González-Calero (2004), pp. 91–105.

Burke, R. D. (2007), Hybrid web recommender systems, *in* Brusilovsky et al. (2007), pp. 377–408.

Burke, R. D., Hammond, K. J. & Young, B. C. (1996), Knowledge-based navigation of complex information spaces, *in* W. J. Clancey & D. S. Weld, eds, 'AAAI/IAAI, Vol. 1', AAAI Press / The MIT Press, pp. 462–468.

Burke, R. D., Hammond, K. J. & Young, B. C. (1997*a*), 'The findme approach to assisted browsing', *IEEE Expert* **12**(4), 32–40.

Burke, R. D., Hammond, K. J. & Young, B. C. (1997*b*), 'The findme approach to assisted browsing', *IEEE Expert* **12**(4), 32–40.

C., F. P. (1968), 'Utility theory', *Management Science* **14**(5), 335–378.

Caballero, A. M., Martín, A. J., Garcia, E. A. A. & Díaz, A. M. (2010), Ranking alternatives on the basis of a dominance intensity measure, *in* 'Proceedings of the 15th IFIP Working Group 8.3 International Conference, DSS 2010.', IOS Press, Amsterdam, Paises Bajos.

Carey, M. J. & Schneider, D. A., eds (1995), *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, ACM Press.

Castell, T., Cayrol, C., Cayrol, M. & Berre, D. L. (1996), Using the davis and putnam procedure for an efficient computation of preferred models, *in* 'In ECAI'96', pp. 350–354.

Cavada, D., Ricci, F. & Venturini, A. (2003), Interactive trip planning with trip@dvise, *in* M. Rauterberg, M. Menozzi & J. Wesson, eds, 'INTERACT', IOS Press.

Chajewska, U., Koller, D. & Parr, R. (2000), Making rational decisions using adaptive utility elicitation, *in* 'Proceedings of the National Conference on Artificial Intelligence (AAAI-00)', pp. 363–369.

Chen, L. & Pu, P. (2004), Survey of preference elicitation methods, *in* 'Technical Report IC/200467'.

Chen, L. & Pu, P. (2009), 'Interaction design guidelines on critiquing-based recommender systems', *User Modeling and User-Adapted Interaction* **19**(3), 167–206.

Chen, L. & Pu, P. (2010), 'Experiments on the preference-based organization interface in recommender systems', *ACM Transactions on Computer-Human Interaction* **17**(1).

Chevaleyre, Y., Dunne, P. E., Endriss, U., Lang, J., Lemaître, M., Maudet, N., Padget, J. A., Phelps, S., Rodríguez-Aguilar, J. A. & Sousa, P. (2006), 'Issues in multiagent resource allocation', *Informatica (Slovenia)* **30**(1), 3–31.

Chevaleyre, Y., Endriss, U., Lang, J. & Maudet, N. (2008), 'Preference handling in combinatorial domains: From ai to social choice', *AI Magazine* **29**(4), 37–46.

Chinneck, J. W. (2007), *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, 1st edn, Springer Publishing Company, Incorporated.

Chomicki, J. (2002), Querying with intrinsic preferences, *in* C. S. Jensen, K. G. Jeffery, J. Pokorný, S. Saltenis, E. Bertino, K. Böhm & M. Jarke, eds, 'EDBT', Vol. 2287 of *Lecture Notes in Computer Science*, Springer, pp. 34–51.

Ciaccia, P. (2007), Querying databases with incomplete cp-nets, *in* 'Multidisciplinary Workshop on Advances in Preference Handling (MPref-06)'.

Clarke, G. & Wright, J. (1964), 'Scheduling of vehicles from a central depot to a number of delivery points', *Operations Research* **12**, 568–581.

Costantini, S. & Formisano, A. (2009), 'Modeling preferences and conditional preferences on resource consumption and production in asp', *Journal of Algorithms* **64**(1), 3–15.

Costantini, S. & Formisano, A. (2010), 'Answer set programming with resources', *Journal of Logic and Computation* **20**(2), 533–571.

Craw, S. & Preece, A. D., eds (2002), *Advances in Case-Based Reasoning, 6th European Conference, ECCBR 2002 Aberdeen, Scotland, UK, September 4-7, 2002, Proceedings*, Vol. 2416 of *Lecture Notes in Computer Science*, Springer.

D'Ambrosio, J. G. & Birmingham, W. P. (1995), 'Preference-directed design', *Journal for Artificial Intelligence in Engineering Design* **9**, 219–230.

de Mántaras, R. L. & Saitta, L., eds (2004), *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, IOS Press.

Dechter, R. (2003), *Constraint processing*, Elsevier Morgan Kaufmann, chapter constraint Optimization, pp. 395–396.

Dechter, R. & Dechter, A. (1988), Belief maintenance in dynamic constraint networks, *in* H. E. Shrobe, T. M. Mitchell & R. G. Smith, eds, 'AAAI', AAAI Press / The MIT Press, pp. 37–42.

Dechter, R. & Pearl, J. (1987), 'Network-based heuristics for constraint-satisfaction problems', *Artificial Intelligence* **34**(1), 1–38.

Delgrande, J. P., Schaub, T. & Tompits, H. (2007), 'A general framework for expressing preferences in causal reasoning and planning', *Journal of Logic and Computation* **17**(5), 871–907.

Delgrande, J. P., Schaub, T., Tompits, H. & Wang, K. (2004), 'A classification and survey of preference handling approaches in nonmonotonic reasoning', *Computational Intelligence* **20**(2), 308–334.

Denker, G., Kagal, L., Finin, T. W., Paolucci, M. & Sycara, K. P. (2003), Security for daml web services: Annotation and matchmaking, *in* D. Fensel, K. P. Sycara & J. Mylopoulos, eds, 'International Semantic Web Conference', Vol. 2870 of *Lecture Notes in Computer Science*, Springer, pp. 335–350.

Do, M. B. & Kambhampati, S. (2001), 'Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP', *Artificial Intelligence* **132**(2), 151–182.

Domshlak, C. (2008), *Proceedings of Preferences and Similarities*, 1 edn, Springer Publishing Company, Incorporated, chapter A snapshot on reasoning with qualitative preference statements in AI, pp. 265–282.

Domshlak, C. & Brafman, R. I. (2002), CP-nets: Reasoning and consistency testing, *in* Fensel et al. (2002), pp. 121–132.

Domshlak, C., Brafman, R. I. & Shimony, S. E. (2001), Preference-based configuration of web page content, *in* Nebel (2001), pp. 1451–1456.

Domshlak, C., Hüllermeier, E., Kaci, S. & Prade, H. (2011), 'Preferences in ai: An overview', *Artificial Intelligence* **175**(7-8), 1037–1052.

Domshlak, C., Prestwich, S. D., Rossi, F., Venable, K. B. & Walsh, T. (2006), 'Hard and soft constraints for reasoning about qualitative conditional preferences', *Journal of Heuristics* **12**(4-5), 263–285.

Dongen, M. V. & Lecoutre, C. (2010), *Constraint Propagation and Implementation*, ISTE, pp. 83–104.

Dorigo, M. & Caro, G. D. (1999), *in* D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli & K. V. Price, eds, 'New ideas in optimization', McGraw-Hill Ltd., UK, Maidenhead, UK, England, chapter The ant colony optimization meta-heuristic, pp. 11–32.

Dorigo, M., Caro, G. D. & Gambardella, L. M. (1999), 'Ant algorithms for discrete optimization', *Artificial Life* **5**(2), 137–172.

Dorigo, M., Maniezzo, V. & Colorni, A. (1996), 'The Ant System: Optimization by a colony of cooperating agents', *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics* **26**(1), 29–41.

Doyle, J. & McGeachie, M. (2003), Exercising qualitative control in autonomous adaptive survivable systems, *in* 'Proceedings of the 2nd international conference on Self-adaptive software: applications', IWSAS'01, Springer-Verlag, Berlin, Heidelberg, pp. 158–170.

Doyle, J., Shoham, Y. & Wellman, M. P. (1991), A logic of relative desire (preliminary report), *in* Z. Ras, ed., 'Proceedings of the International Symposium on Methodologies for Intelligent Systems', Springer-Verlag.

Doyle, J. & Thomason, R. (1999), 'Background to qualitative decision theory', *AI Magazine* **20**, 55–68.

Doyle, M. & Cunningham, P. (2000), A dynamic approach to reducing dialog in on-line decision guides, *in* 'Proceedings of the European Workshop on Case-Based Reasoning (EWCBR-00)', Springer, pp. 49–60.

Dubois, D., Fargier, H., Prade, H. & Perny, P. (2002), 'Qualitative decision theory: from savage's axioms to nonmonotonic reasoning', *Journal of the ACM* **49**(4), 455–495.

Dubois, D., Lang, J. & Prade, H. (1990), POSLOG, an inference system based on possibilistic logic, *in* 'Proceedings of the North American Fuzzy Information Processing Society Conference (NAFIPS'90), Toronto, Canada, 06/06/90-08/06/90'.

Dubois, D. & Prade, H. (1990), *in* G. Shafer & J. Pearl, eds, 'Readings in uncertain reasoning', Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, chapter An introduction to possibilistic and fuzzy logics, pp. 742–761.

Dushnik, B. & Miller, E. (1941), 'Partially ordered sets', *American Journal of Mathematics* **63**, 600–610.

Dyer, J. S. (2005), Maut - multiattribute utility theory, *in* J. Figueira, S. Greco & M. Ehrgott, eds, 'Multiple Criteria Decision Analysis - State of the Art Surveys', Springer International Series in Operations Research and Management Science Volume 76.

ebay (2012), `http://www.ebay.com` (last accessed 3rd May, 2013).

Edelkamp, S. & Schródl, S. (2012), *Heuristic search. Theory and applications.*, Amsterdam: Elsevier/Morgan Kaufmann.

Endres, M. & Kiessling, W. (2006), Transformation of TCP-Net queries into preference database queries, *in* '2nd Multidisciplinary Workshop on Advances in Preference Handling (MPref-06)'.

Endriss, U. (2011), Logic and social choice theory, Technical report, Institute for Logic Language and Computation University of Amsterdam.

Engel, Y. (2008), Structured Preference Representation and Multiattribute Auctions, PhD thesis, University of Michigan.

Engel, Y. & Wellman, M. P. (2008), 'CUI networks: A graphical representation for conditional utility independence', *Journal of Artificial Intelligence Research* **31**, 83–112.

Engel, Y. & Wellman, M. P. (2010), 'Multiattribute auctions based on generalized additive independence', *Journal Artificial Intelligence Research* **37**, 479–525.

expedia (2012), `http://www.expedia.com` (last accessed 3rd May, 2013).

Faltings, B., Pu, P., Torrens, M. & Viappiani, P. (2004), Designing example-critiquing interaction, *in* J. Vanderdonckt, N. J. Nunes & C. Rich, eds, 'IUI', ACM, pp. 22–29.

Faltings, B., Torrens, M. & Pu, P. (2004), 'Solution generation with qualitative models of preferences', *Computational Intelligence* **20**(2), 246–263.

Farquhar, P. (1984), 'Utility assessment methods', *Management Science* **30**(11), 1283–1300.

Felfernig, A. & Burke, R. D. (2008), Constraint-based recommender systems: technologies and research issues, *in* D. Fensel & H. Werthner, eds, 'ICEC', Vol. 342 of *ACM International Conference Proceeding Series*, ACM, pp. 1–10.

Felfernig, A., Friedrich, G., Jannach, D. & Zanker, M. (2007), 'An integrated environment for the development of knowledge-based recommender applications', *International Journal of Electronic Commerce* **11**, 11–34.

Felfernig, A. & Gula, B. (2006), An empirical study on consumer behavior in the interaction with knowledge-based recommender applications, *in* 'CEC/EEE', IEEE Computer Society, p. 37.

Felfernig, A., Isak, K. & Russ, C. (2006), Knowledge-based recommendation: Technologies and experiences from projects, *in* Brewka et al. (2006), pp. 632–636.

Felfernig, A., Isak, K., Szabo, K. & Zachar, P. (2007), The vita financial services sales support environment, in *AAAI* (*Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada* 2007), pp. 1692–1699.

Fensel, D., Giunchiglia, F., McGuinness, D. L. & Williams, M.-A., eds (2002), *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002*, Morgan Kaufmann.

Figueira, J., Greco, S. & Ehrgott, M. (2005), *Multiple Criteria Decision Analysis - State of the Art Surveys*, Springer International Series in Operations Research and Management Science Volume 76.

Fischer, G. (2001), 'User modeling in human-computer interaction', *User Modeling and User-Adapted Interaction* **11**(1-2), 65–86.

Fishburn, P. C. (1967), 'Interdependence and additivity in multivariate, unidimensional expected utility theory', *International Economic Review* **8**(3).

Fishburn, P. C. (1970), *Utility Theory For Decision Making*, Wiley.

Fishburn, P. C. (1974), 'Lexicographic orders, utilities, and decision rules: A survey', *Management Science* **20**(11), 1442–1471.

Fishburn, P. C. (1999), 'Preference structures and their numerical representations', *Theoretical Computer Science* **217**(2), 359–383.

Fleischanderl, G., Friedrich, G. E., Haselböck, A., Schreiner, H. & Stumptner, M. (1998), 'Configuring large systems using generative constraint satisfaction', *IEEE Intelligent Systems* **13**(4), 59–68.

Fox, D. & Gomes, C. P., eds (2008), *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, AAAI Press.

Freuder, E. C., Heffernan, R., Wallace, R. J. & Wilson, N. (2010), 'Lexicographically-ordered constraint satisfaction problems', *Constraints* **15**(1), 1–28.

Funk, P. & González-Calero, P. A., eds (2004), *Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004, Madrid, Spain, August 30 - September 2, 2004, Proceedings*, Vol. 3155 of *Lecture Notes in Computer Science*, Springer.

Gavanelli, M. & Pini, M. S. (2008*a*), Fcp-nets: extending constrained cp-nets with objective functions, *in* 'Constraint Solving and Constraint Logic Programming (ERCIM)'.

Gavanelli, M. & Pini, M. S. (2008*b*), FCP-Nets: extending constrained CP-Nets with objective functions, *in* A. Oddi, A. Cesta, F. Fages, N. Policella & F. Rossi, eds, 'Annual ERCIM Workshop on Constraint Solving and Constraint Logic Programming', ERCIM, ISTC-CNR, Institute for Cognitive Science and Technology, Rome, Italy.

Ge, M., Delgado-Battenfeld, C. & Jannach, D. (2010), Beyond accuracy: evaluating recommender systems by coverage and serendipity, *in* X. Amatriain, M. Torrens, P. Resnick & M. Zanker, eds, 'RecSys', ACM, pp. 257–260.

Gelfond, M. (2007), Answer sets, *in* 'Handbook of Knowledge Representation', number 7, Elsevier Science, pp. 73–105.

Gelfond, M. & Lifschitz, V. (1988*a*), The stable model semantics for logic programming, *in* R. Kowalski & K. Bowen, eds, 'Proceedings of the 5th Intl. Conference and Symposium on Logic Programming', MIT Press, pp. 1070–1080.

Gelfond, M. & Lifschitz, V. (1988*b*), The stable model semantics for logic programming, *in* 'ICLP/SLP', MIT Press, pp. 1070–1080.

Gelfond, M. & Lifschitz, V. (1991), 'Classical negation in logic programs and disjunctive databases', *New Generation Computing* **9**, 365–385.

Gigerenzer, G. & Goldstein, D. G. (1996), 'Reasoning the fast and frugal way: Models of bounded rationality', *Psychological Review* **103**, 650–669.

Giunchiglia, E. & Maratea, M. (2011), 'Introducing preferences in planning as satisfiability', *Journal of Logic and Computation* **21**(2), 205–229.

Giunchiglia, E., Massarotto, A. & Sebastiani, R. (1998), Act, and the rest will follow: Exploiting determinism in planning as satisfiability, *in* J. Mostow & C. Rich, eds, 'AAAI/IAAI', AAAI Press / The MIT Press, pp. 948–953.

Glover, F. (1986), 'Future paths for integer programming and links to artificial intelligence', *Computers and Operations Research* **13**(5), 533–549.

Glover, F. & Laguna, M. (1997), *Tabu Search*, Kluwer Academic Publishers, Norwell, MA, USA.

Goker, M. H. & Thompson, C. A. (2000), 'Personalized conversational case-based recommendation'.

Goldsmith, J., Lang, J., Truszczynski, M. & Wilson, N. (2005), The computational complexity of dominance and consistency in CP-nets, *in* L. P. Kaelbling & A. Saffiotti, eds, 'IJCAI', Professional Book Center, pp. 144–149.

Goldsmith, J., Lang, J., Truszczyński, M. & Wilson, N. (2008), 'The computational complexity of dominance and consistency in CP-nets', *Journal of Artificial Intelligence Research* **33**, 403–432.

Gonzales, C., Perny, P. & Queiroz, S. (2008), Preference aggregation with graphical utility models, *in* 'Proceedings of the National Conference on Artificial Intelligence (AAAI-08)', pp. 1037–1042.

Good, N., Schafer, J. B., Konstan, J. A., Borchers, A., Sarwar, B. M., Herlocker, J. L. & Riedl, J. (1999), Combining collaborative filtering with personal agents for better recommendations, *in* Hendler & Subramanian (1999), pp. 439–446.

Gottlob, G. & Walsh, T., eds (2003), *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, Morgan Kaufmann.

Hammond, K. J., Burke, R. D. & Schmitt, K. (1994), A case-based approach to knowledge navigation, *in* 'KDD Workshop', pp. 383–394.

Hansson, S. O. (1989), 'A new semantical approach to the logic of preference', *Humanities Social Sciences and Law* **31**(1), 1–42.

Hansson, S. O. (2001), 'Preference logic', *Handbook of Philosophical Logic* **8**.

Hansson, S. O. & Grüne-Yanoff, T. (2011), Preferences, *in* E. N. Zalta, ed., 'The Stanford Encyclopedia of Philosophy', fall 2011 edn.

Haralick, R. M. & Elliott, G. L. (1980), 'Increasing tree search efficiency for constraint satisfaction problems', *Artificial Intelligence* **14**, 263–313.

Häubl, G. & Murray, K. B. (2001), Recommending or persuading? the impact of a shopping agent's algorithm on user behavior, *in* 'ACM Conference on Electronic Commerce', ACM, pp. 163–170.

Hendler, J. & Subramanian, D., eds (1999), *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA*, AAAI Press / The MIT Press.

Hill, W. C., Stead, L., Rosenstein, M. & Furnas, G. W. (1995), Recommending and evaluating choices in a virtual community of use, *in* Katz et al. (1995), pp. 194–201.

Hoare, C. A. R. (1989), *Essays in Computing Science*, Prentice-Hall.

Hoos, H. H. & Stützle, T. (2004), *Stochastic Local Search: Foundations & Applications*, Elsevier / Morgan Kaufmann.

Howard, R. A. & Matheson, J. E. (2005), 'Influence diagram retrospective', *Decision Analysis* **2**(3), 144–147.

Hurley, R. J. & Tewksbury, D. (2012), 'News aggregation and content differences in online cancer news', *Journal of Broadcasting & Electronic Media* **56**(1), 132–149.

Jameson, A., Baldes, S. & Kleinbauer, T. (2004), Two methods for enhancing mutual awareness in a group recommender system, *in* M. F. Costabile, ed., 'AVI', ACM Press, pp. 447–449.

Jannach, D., Zanker, M., Felfernig, A. & Friedrich, G. (2011), *Recommender Systems: An Introduction*, 1 edn, Cambridge University Press.

Jannach, D., Zanker, M., Jessenitschnig, M. & Seidler, O. (2007), Developing a conversational travel advisor with advisor suite, *in* M. Sigala, L. Mich & J. Murphy, eds, 'ENTER', Springer, pp. 43–52.

Järvisalo, M. & Junttila, T. A. (2009), 'Limitations of restricted branching in clause learning', *Constraints* **14**(3), 325–356.

Järvisalo, M., Junttila, T. A. & Niemelä, I. (2005), 'Unrestricted vs restricted cut in a tableau method for boolean circuits', *Annals of Mathematics and Artificial Intelligence* **44**(4), 373–399.

Jin, H., Han, H. & Somenzi, F. (2005), Efficient conflict analysis for finding all satisfying assignments of a boolean circuit, *in* N. Halbwachs & L. D. Zuck, eds, 'TACAS', Vol. 3440 of *Lecture Notes in Computer Science*, Springer, pp. 287–300.

Jin, H. & Somenzi, F. (2005), Prime clauses for fast enumeration of satisfying assignments to boolean circuits, *in* W. H. J. Jr., G. Martin & A. B. Kahng, eds, 'DAC', ACM, pp. 750–753.

Joseph, R.-R., Chan, P., Hiroux, M. & Weil, G. (2007), 'Decision-support with preference constraints', *European Journal of Operational Research* **177**(3), 1469–1494.

Junker, U. (2001), Preference programming for configuration, *in* 'Proceedings of Workshop on Configuration (IJCAI-01)', Seattle, pp. 50–56.

Kaci, S. (2011), *Working with Preferences: Less Is More*, Cognitive Technologies, Springer.

Kadioglu, S. (2012), Efficient Search Procedures for Solving Combinatorial Problems, PhD thesis, Brown University.

Kahneman, D. & Tversky, A. (1979), 'Prospect theory: An analysis of decision under risk', *Econometrica* **47**, 263–291.

Kärger, P., Lopes, N., Olmedilla, D. & Polleres, A. (2008), Towards logic programs with ordered and unordered disjunction, *in* 'Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2008), 24th International Conference on Logic Programming (ICLP 2008)', Udine, Italy.

Katz, I. R., Mack, R. L., Marks, L., Rosson, M. B. & Nielsen, J., eds (1995), *Human Factors in Computing Systems, CHI '95 Conference Proceedings, Denver, Colorado, USA, May 7-11, 1995*, ACM/Addison-Wesley.

Kautz, H. & Selman, B. (1992), Planning as satisfiability, *in* 'IN ECAI-92', Wiley, pp. 359–363.

Keeney, R. (1982), 'Decision analysis: An overview', *Operations Research* **30**(5), 803–838.

Keeney, R. L. (1992), *Value-Focused Thinking: A Path to Creative Decision making*, Harvard University Press.

Keeney, R. L. & Raiffa, H. (1976), *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*, Cambridge University Press.

Keeney, R. L. & Raiffa, H. (1993), *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*, Cambridge.

Kießling, W. (2002), Foundations of preferences in database systems, *in* 'VLDB', Morgan Kaufmann, pp. 311–322.

Kim, H.-N., Alkhaldi, A., Saddik, A. E. & Jo, G.-S. (2011), 'Collaborative user modeling with user-generated tags for social recommender systems', *Expert Systems with Applications* **38**(7), 8488 – 8496.

Kobsa, A. (2001), 'Generic user modeling systems', *User Modeling and User-Adapted Interaction* **11**(1-2), 49–63.

Kondrak, G. & van Beek, P. (1997), 'A theoretical evaluation of selected backtracking algorithms', *Artificial Intelligence* **89**(1-2), 365–387.

Konstan, J. A. & Riedl, J. (2012), 'Recommender systems: from algorithms to user experience', *User Modeling and User-Adapted Interaction* **22**(1-2), 101–123.

Koopman, B. (1956), 'Fallacies in operations research', *Operations Research* **3**, 422–426.

Koriche, F. & Zanuttini, B. (2009), Learning conditional preference networks with queries, *in* Boutilier (2009), pp. 1930–1935.

Kristensen, L. M., Westergaard, M. & Nørgaard, P. C. (2005), Model-based prototyping of an interoperability protocol for mobile ad-hoc networks, *in* J. Romijn, G. Smith & J. van de Pol, eds, 'IFM', Vol. 3771 of *Lecture Notes in Computer Science*, Springer, pp. 266–286.

Lang, J. (2010), Graphical representation of ordinal preferences: Languages and applications, *in* M. Croitoru, S. Ferré & D. Lukose, eds, 'ICCS', Vol. 6208 of *Lecture Notes in Computer Science*, Springer, pp. 3–9.

Lang, J. & Xia, L. (2009), 'Sequential composition of voting rules in multi-issue domains', *Mathematical Social Sciences* **57**(3), 304–324.

Lecoutre, C. (2009), *Constraint Networks: Techniques and Algorithms*, Wiley-IEEE Press.

Levandoski, J. J., Sarwat, M., Mokbel, M. F. & Ekstrand, M. D. (2012), Recstore: an extensible and adaptive framework for online recommender queries inside the database engine, *in* E. A. Rundensteiner, V. Markl, I. Manolescu, S. Amer-Yahia, F. Naumann & I. Ari, eds, 'EDBT', ACM, pp. 86–96.

Li, M., Vo, Q. B. & Kowalczyk, R. (2010), An efficient approach for ordering outcomes and making social choices with CP-nets, *in* J. Li, ed., 'Australasian Conference on Artificial Intelligence', Vol. 6464 of *Lecture Notes in Computer Science*, Springer, pp. 375–384.

Lieberman, H. (1997), Autonomous interface agents, *in* S. Pemberton, ed., 'CHI', ACM/Addison-Wesley, pp. 67–74.

Lilien, G. L. & Grewal, R. (2012), *Handbook of Business to Business Marketing*, Edward Elgar Publishing.

Lin, F., Sattler, U. & Truszczynski, M., eds (2010), *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*, AAAI Press.

Linden, G., Hanks, S. & Lesh, N. (1997), Interactive assessment of user preference models: The automated travel assistant, *in* 'In Proceedings of the International Conference on User Modeling', Springer, pp. 67–78.

Lops, P., de Gemmis, M. & Semeraro, G. (2011), Content-based recommender systems: State of the art and trends, *in* Ricci et al. (2011*b*), pp. 73–105.

Lorenzi, F. & Ricci, F. (2003), Case-based recommender systems: A unifying view, *in* B. Mobasher & S. S. Anand, eds, 'ITWP', Vol. 3169 of *Lecture Notes in Computer Science*, Springer, pp. 89–113.

Luce, R. (1956), 'Semiorders and a theory of utility discrimination', *Econometrica* **24**, 178–191.

Mackworth, A. K. (1977), On reading sketch maps, *in* R. Reddy, ed., 'IJCAI', William Kaufmann, pp. 598–606.

Maes, P. (1994), 'Agents that reduce work and information overload', *Communication ACM* **37**(7), 30–40.

Mahmood, T. (2009), Learning Adapted Interaction Strategies in Conversational Recommender Systems, PhD thesis, DIT - University of Trento.

Mahmood, T. & Ricci, F. (2007), Learning and adaptivity in interactive recommender systems, *in* M. L. Gini, R. J. Kauffman, D. Sarppo, C. Dellarocas & F. Dignum, eds, 'ICEC', Vol. 258 of *ACM International Conference Proceeding Series*, ACM, pp. 75–84.

Mahmood, T. & Ricci, F. (2009), Improving recommender systems with adaptive conversational strategies, *in* 'Hypertext', ACM, pp. 73–82.

Mahmood, T., Ricci, F. & Venturini, A. (2009), Learning adaptive recommendation strategies for online travel planning, *in* W. Höpken, U. Gretzel & R. Law, eds, 'ENTER', Springer, pp. 149–160.

Mahmood, T., Ricci, F., Venturini, A. & Höpken, W. (2008), Adaptive recommender systems for travel planning, *in* P. O'Connor, W. Höpken & U. Gretzel, eds, 'ENTER', Springer, pp. 1–11.

Maltz, D. & Ehrlich, K. (1995), Pointing the way: Active collaborative filtering, *in* Katz et al. (1995), pp. 202–209.

Mandl, M., Felfernig, A., Teppan, E. & Schubert, M. (2011), 'Consumer decision making in knowledge-based recommendation', *Journal of Intelligent Information Systems* **37**(1), 1–22.

Mandl, M., Felfernig, A. & Tiihonen, J. (2011), Evaluating design alternatives for feature recommendations in configuration systems, *in* B. Hofreiter, E. Dubois, K.-J. Lin, T. Setzer, C. Godart, E. Proper & L. Bodenstaff, eds, 'CEC', IEEE, pp. 34–41.

Maratea, M., Ricca, F. & Veltri, P. (2010), Dlv^{mc}: Enhanced model checking in dlv, *in* T. Janhunen & I. Niemelä, eds, 'JELIA', Vol. 6341 of *Lecture Notes in Computer Science*, Springer, pp. 365–368.

Marek, V. W. & Truszczynski, M. (1998), 'Stable models and an alternative logic programming paradigm', *CoRR* **cs.LO/9809032**.

Massa, P. & Bhattacharjee, B. (2004), Using trust in recommender systems: An experimental analysis, *in* C. D. Jensen, S. Poslad & T. Dimitrakos, eds, 'iTrust', Vol. 2995 of *Lecture Notes in Computer Science*, Springer, pp. 221–235.

McCarthy, J. F. (2002), Pocket restaurant finder: A situated recommender systems for groups, *in* 'Proceeding of Workshop on Mobile Ad-Hoc Communication at the 2002 ACM Conference on Human Factors in Computer Systems'.

McCarthy, J. F. & Anagnost, T. D. (1998), Musicfx: An arbiter of group preferences for computer aupported collaborative workouts, *in* S. E. Poltrock & J. Grudin, eds, 'CSCW', ACM, pp. 363–372.

McCarthy, K., McGinty, L., Smyth, B. & Reilly, J. (2005), A live-user evaluation of incremental dynamic critiquing, *in* H. Muñoz-Avila & F. Ricci, eds, 'ICCBR', Vol. 3620 of *Lecture Notes in Computer Science*, Springer, pp. 339–352.

McCarthy, K., Reilly, J., McGinty, L. & Smyth, B. (2004), On the dynamic generation of compound critiques in conversational recommender systems, *in* P. D. Bra & W. Nejdl, eds, 'AH', Vol. 3137 of *Lecture Notes in Computer Science*, Springer, pp. 176–184.

McCarthy, K., Reilly, J., McGinty, L. & Smyth, B. (2005), Experiments in dynamic critiquing, *in* R. S. Amant, J. Riedl & A. Jameson, eds, 'IUI', ACM, pp. 175–182.

McDermott, J. (1982), 'R1: a rule-based configurer of computer systems', *Artificial Intelligence* **19**, 39–88.

McGinty, L. & Reilly, J. (2011), On the evolution of critiquing recommenders, *in* Ricci et al. (2011*b*), pp. 419–453.

McGinty, L. & Smyth, B. (2002*a*), Comparison-based recommendation, *in* Craw & Preece (2002), pp. 575–589.

McGinty, L. & Smyth, B. (2002*b*), Evaluating preference-based feedback in recommender systems, *in* 'AICS', pp. 209–214.

McGinty, L. & Smyth, B. (2003), Tweaking critiquing, *in* 'Proceedings of the Workshop on Personalization and Web Techniques, Workshop Program at the International Joint Conference on Artificial Intelligence', pp. 20–27.

McGinty, L. & Smyth, B. (2006), 'Adaptive selection: An analysis of critiquing and preference-based feedback in conversational recommender systems', *International Journal of Electronic Commerce* **11**(2), 35–57.

McGuinness, D. L. & Ferguson, G., eds (2004), *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, AAAI Press / The MIT Press.

McMillan, K. L. (1992), Symbolic model checking: an approach to the state explosion problem, PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA. UMI Order No. GAX92-24209.

McSherry, D. (2002), 'A generalised approach to similarity-based retrieval in recommender systems', *Artificial Intelligence Review* **18**(3-4), 309–341.

McSherry, D. (2003), Similarity and compromise, *in* K. D. Ashley & D. G. Bridge, eds, 'ICCBR', Vol. 2689 of *Lecture Notes in Computer Science*, Springer, pp. 291–305.

McSherry, D. (2005), 'Retrieval failure and recovery in recommender systems', *Artificial Intelligence Review* **24**(3-4), 319–338.

McSherry, D. & Aha, D. W. (2007), Mixed-initiative relaxation of constraints in critiquing dialogues, *in* R. Weber & M. M. Richter, eds, 'ICCBR', Vol. 4626 of *Lecture Notes in Computer Science*, Springer, pp. 107–121.

Meisels, A., Shimony, S. E. & Solotorevsky, G. (1997), Bayes networks for estimating the number of solutions to a CSP, *in* B. Kuipers & B. L. Webber, eds, 'AAAI/IAAI', AAAI Press / The MIT Press, pp. 179–184.

Menzel, W. (1998), 'Constraint satisfaction for robust parsing of spoken language', *Journal of Experimental and Theoretical Artificial Intelligence* **10**(1), 77–89.

Middleton, S. E., Shadbolt, N. R. & De Roure, D. C. (2004), 'Ontological user profiling in recommender systems', *ACM Transactions on Information Systems* **22**(1), 54–88.

Miguel, I. & Shen, Q. (2001*a*), 'Solution techniques for constraint satisfaction problems: Advanced approaches', *Artificial Intelligence Review* **15**(4), 269–293.

Miguel, I. & Shen, Q. (2001*b*), 'Solution techniques for constraint satisfaction problems: Foundations', *Artificial Intelligence Review* **15**(4), 243–267.

Mindolin, D. & Chomicki, J. (2007), Hierarchical CP-networks, *in* '3rd Multidisciplinary Workshop on Advances in Preference Handling (MPref-06)'.

Mirzadeh, N. & Ricci, F. (2007), 'Cooperative query rewriting for decision making support and recommender systems', *Applied Artificial Intelligence* **21**(10), 895–932.

Mittal, S. & Falkenhainer, B. (1996), A logic-based description of configuration: the constructive problem solving approach, *in* 'In AAAI Fall Symposium', pp. 11–118.

Montanari, U. (1974), Optimization methods in image processing, *in* 'IFIP Congress', pp. 727–732.

Moreno, M. N., Segrera, S., López, V. F., noz, M. D. M. & Ángel Luis Sánchez (2011), Mining semantic data for solving first-rater and cold-start problems in recommender systems, *in* 'Proceedings of the 15th Symposium on International Database Engineering & Applications', IDEAS '11, ACM, New York, NY, USA, pp. 256–257.

Movielens (2012), `http://www.movielens.org/` (last accessed 3rd May, 2013).

MovieLens (2013), `http://movielens.umn.edu/main` (last accessed 3rd May, 2013).

Mukhtar, H., Belaïd, D. & Bernard, G. (2011), 'Dynamic user task composition based on user preferences', *TAAS* **6**(1), 4.

Mura, P. L. & Shoham, Y. (1999), Expected utility networks, *in* 'UAI', pp. 366–373.

Nebel, B., ed. (2001), *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, Morgan Kaufmann.

netflix (2012), `http://www.netflix.com/` (last accessed 3rd May, 2013).

Neumaier, A. (2004), Complete search in continuous global optimization and constraint satisfaction, *in* A. I. ed, ed., 'Acta Numerica 2004', Cambridge University Press.

Neumann, J. V. & Morgenstern, O. (1944), *Theory of Games and Economic Behavior*, Princeton University Press, Princeton.

Nguyen, H. & Haddawy, P. (1999), The decision-theoretic interactive video advisor, *in* 'UAI', pp. 494–501.

O'Connor, M., Cosley, D., Konstan, J. A. & Riedl, J. (2001), Polylens: a recommender system for groups of users, *in* 'Proceedings of the seventh conference on European Conference on Computer Supported Cooperative Work', ECSCW'01, Kluwer Academic Publishers, Norwell, MA, USA, pp. 199–218.

Ostanello, A. & Tsoukiàs, A. (1993), 'An explicative model of public interorganizational interactions', *European Journal of Operational Research* **70**, 67–82.

Oster, Z. J., Santhanam, G. R. & Basu, S. (2011), Automating analysis of qualitative preferences in goal-oriented requirements engineering, *in* P. Alexander, C. S. Pasareanu & J. G. Hosking, eds, 'ASE', IEEE, pp. 448–451.

Öztürk, M., Tsoukiàs, A. & Vincke, P. (2004), Preference modelling, *in* G. Bosi, R. I. Brafman, J. Chomicki & W. Kießling, eds, 'Preferences', Vol. 04271 of *Dagstuhl Seminar Proceedings*, IBFI, Schloss Dagstuhl, Germany.

P. Journee, P. P. & Vanderpooten, D. (1998), 'A multicriteria methodology for the verification of arms control agreements in europe', *Foundations of Computing and Decision Sciences* **23**, 64–85.

Park, S.-T., Pennock, D., Madani, O., Good, N. & DeCoste, D. (2006), Naïve filterbots for robust cold-start recommendations, *in* T. Eliassi-Rad, L. H. Ungar, M. Craven & D. Gunopulos, eds, 'KDD', ACM, pp. 699–705.

Parker, R. & Rardin, R. (1988), *Discrete Optimization*, Acad. Press.

Parsons, S. (2006), 'Reasoning about uncertainty by joseph halpern, mit press', *Knowledge Engineering Review* **21**(3), 290–291.

Payne, J., Bettman, J. & Johnson, J. (1992), 'Behavioral decision research: a constructive processing perspective', *Annual Review of Psychology* **43**, 87–131.

Payne, J. W., Bettman, J. R. & Johnson, E. J. (1993), *The adaptive decision maker.*, Cambridge University Press, New York, NY.

Pazzani, M. J. (1999), 'A framework for collaborative, content-based and demographic filtering', *Artificial Intelligence Review* **13**(5-6), 393–408.

Pazzani, M. J. & Billsus, D. (2007), Content-based recommendation systems, *in* Brusilovsky et al. (2007), pp. 325–341.

Pearl, J. (1988), 'Embracing causality in default reasoning', *Artificial Intelligence* **35**(2), 259–271.

Pearl, J. & Paz, A. (1986), Graphoids: Graph-based logic for reasoning about relevance relations or when would x tell you more about y if you already know z?, *in* 'ECAI', pp. 357–363.

Peintner, B., Moffitt, M. D. & Pollack, M. E. (2005), Solving over-constrained disjunctive temporal problems with preferences, *in* Biundo et al. (2005), pp. 202–211.

Perron, L. & Trick, M. A., eds (2008), *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008, Paris, France, May 20-23, 2008, Proceedings*, Vol. 5015 of *Lecture Notes in Computer Science*, Springer.

Pham, D. T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S. & Zaidi, M. (2006), The bees algorithm, a novel tool for complex optimisation problems, *in* 'Proceedings of the 2nd International Virtual Conference on Intelligent Production Machines and Systems (IPROMS 2006)', IPROMS 2006, pp. 454–459.

Pirlot, M. (1996), 'General local search methods', *European Journal of Operational Research* **92**(3), 493–511.

Polya, G. (1971), *How to Solve It: A New Aspect of Mathematical Method*, 2nd edn, Princeton University Press.

Poole, D. (1992), Decision-theoretic defaults, *in* 'In Proceedings of the Ninth Biennial Conference of the Canadian Society for Computational Studies of Intelligence', Morgan Kaufmann, San Francisco, pp. 190–197.

Popescu, G. & Pu, P. (2011), Group recommender systems as a voting problem, *in* 'EPFL HCI Technical report, Feburary 2011'.

Prestwich, S. D., Rossi, F., Venable, K. B. & Walsh, T. (2005), Constraint-based preferential optimization, *in* M. M. Veloso & S. Kambhampati, eds, 'AAAI', AAAI Press / The MIT Press, pp. 461–466.

Prestwich, S., Rossi, F., Venable, K. B. & Walsh, T. (2004), Constrained CP-nets, *in* 'Proceedings of CSCLP'04'.

Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada (2007), AAAI Press.

Pu, P. & Chen, L. (2005), Integrating tradeoff support in product search tools for e-commerce sites, *in* J. Riedl, M. J. Kearns & M. K. Reiter, eds, 'ACM Conference on Electronic Commerce', ACM, pp. 269–278.

Pu, P., Chen, L. & Kumar, P. (2008), 'Evaluating product search and recommender systems for e-commerce environments', *Electronic Commerce Research* **8**(1-2), 1–27.

Pu, P. & Faltings, B. (2000), Enriching buyerséxperiences: the smartclient approach, *in* T. Turner & G. Szwillus, eds, 'CHI', ACM, pp. 289–296.

Pu, P. & Faltings, B. (2004), 'Decision tradeoff using example-critiquing and constraint programming', *Constraints* **9**(4), 289–310.

Pu, P., Faltings, B., Chen, L., Zhang, J. & Viappiani, P. (2011), Usability guidelines for product recommenders based on example critiquing research, *in* Ricci et al. (2011*b*), pp. 511–545.

Pu, P., Viappiani, P. & Faltings, B. (2006), Increasing user decision accuracy using suggestions, *in* R. E. Grinter, T. Rodden, P. M. Aoki, E. Cutrell, R. Jeffries & G. M. Olson, eds, 'CHI', ACM, pp. 121–130.

Purrington, K. & Durfee, E. H. (2007), Making social choices from individuals' CP-nets, *in* E. H. Durfee, M. Yokoo, M. N. Huhns & O. Shehory, eds, 'AAMAS', IFAAMAS, p. 179.

Purrington, K. & Durfee, E. H. (2008), NP-completeness of outcome optimization for partial CP-nets, *in* Fox & Gomes (2008), pp. 1826–1827.

Rafter, R., Bradley, K. & Smyth, B. (2000), Automated collaborative filtering applications for online recruitment services, *in* P. Brusilovsky, O. Stock & C. Strapparava, eds, 'AH', Vol. 1892 of *Lecture Notes in Computer Science*, Springer, pp. 363–368.

Reilly, J., McCarthy, K., McGinty, L. & Smyth, B. (2004), Dynamic critiquing, *in* Funk & González-Calero (2004), pp. 763–777.

Reilly, J., McCarthy, K., McGinty, L. & Smyth, B. (2005), 'Incremental critiquing', *Knowledge Based Systems* **18**(4-5), 143–151.

Reilly, J., Zhang, J., McGinty, L., Pu, P. & Smyth, B. (2007), A comparison of two compound critiquing systems, *in* D. N. Chin, M. X. Zhou, T. A. Lau & A. R. Puerta, eds, 'IUI', ACM, pp. 317–320.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P. & Riedl, J. (1994), Grouplens: An open architecture for collaborative filtering of netnews, *in* 'CSCW', pp. 175–186.

Resnick, P. & Varian, H. R. (1997), 'Recommender systems - introduction to the special section', *Communication ACM* **40**(3), 56–58.

Ricci, F. (2011), 'Mobile recommender systems', *International Journal of Information Technology and Tourism* **12**(3).

Ricci, F., Arslan, B., Mirzadeh, N. & Venturini, A. (2002), Itr: A case-based travel advisory system, *in* Craw & Preece (2002), pp. 613–627.

Ricci, F., Cavada, D., Mirzadeh, N. & Venturini, A. (2006), Case-based travel recommendations, *in* D. R. Fesenmaier et al., eds, 'Destination Recommendation Systems: Behavioural Foundations and Applications', CABI, pp. 67–93.

Ricci, F., Mirzadeh, N. & Venturini, A. (2002), Intelligent query management in a mediator architecture, *in* '1st International IEEE Symposium on Intelligent Systems'.

Ricci, F. & Nguyen, Q. N. (2005), 'Critique-based mobile recommender systems', *ÖGAI Journal* **24**(4).

Ricci, F., Rokach, L., Shapira, B. & Kantor, P. B., eds (2011*a*), *Recommender Systems Handbook*, Springer.

Ricci, F., Rokach, L., Shapira, B. & Kantor, P. B., eds (2011*b*), *Recommender Systems Handbook*, Springer.

Rosa, E. D., Giunchiglia, E. & Maratea, M. (2010), 'Solving satisfiability problems with preferences', *Constraints* **15**(4), 485–515.

Rossi, F. (2005), Preference reasoning, *in* P. van Beek, ed., 'CP', Vol. 3709 of *Lecture Notes in Computer Science*, Springer, pp. 9–12.

Rossi, F., van Beek, P. & Walsh, T. (2006), *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*, Elsevier Science Inc., New York, NY, USA.

Rossi, F., Venable, K. B. & Walsh, T. (2004), mCP-nets: Representing and reasoning with preferences of multiple agents, *in* McGuinness & Ferguson (2004), pp. 729–734.

Rossi, F., Venable, K. B. & Walsh, T. (2008), 'Preferences in constraint satisfaction and optimization', *AI Magazine* **29**(4), 58–68.

Rossi, F., Venable, K. B. & Walsh, T. (2011), *A Short Introduction to Preferences: Between Artificial Intelligence and Social Choice*, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers.

Roubens, M. (1989), 'Some properties of choice functions based on valued binary relations', *European Journal of Operational Research* **40**(3), 309 – 321.

Roy, B. (1991), 'The outranking approach and the foundations of electre methods', *Theory and Decision* **31**, 49–73.

Roy, B. (1996), *Multicriteria Methodology for Decision Aiding*, Kluwer Academic, Dordrecht.

Russell, S. J., Norvig, P., Candy, J. F., Malik, J. M. & Edwards, D. D. (1996), *Artificial intelligence: a modern approach*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Saaty, T. L. (1980), *The Analytic Hierarchy Process: Planning, Priority Setting,Resource Allocation*, McGraw-Hill, New York.

Sabin, D. & Freuder, E. C. (1994), Contradicting conventional wisdom in constraint satisfaction, *in* 'Proceedings of the International Workshop on Principles and Practice of Constraint Programming', PPCP '94, Springer-Verlag, London, UK, UK, pp. 10–20.

Sabin, D. & Weigel, R. (1998), 'Product configuration frameworks-a survey', *IEEE Intelligent Systems* **13**, 42–49.

Sadeh, N. M. & Fox, M. S. (2003), 'Hybrid solving for CSP', *ALP newsletter* **6**.

Salton, G. & McGill, M. (1983), *Introduction to Modern Information Retrieval*, McGraw-Hill Book Company.

Santhanam, G. R., Basu, S. & Honavar, V. (2008), TCP-Compose* - a TCP-Net based algorithm for efficient composition of web services using qualitative preferences, *in* A. Bouguettaya, I. Krüger & T. Margaria, eds, 'ICSOC', Vol. 5364 of *Lecture Notes in Computer Science*, pp. 453–467.

Santhanam, G. R., Basu, S. & Honavar, V. (2010*a*), Dominance testing via model checking, *in* M. Fox & D. Poole, eds, 'AAAI', AAAI Press.

Santhanam, G. R., Basu, S. & Honavar, V. (2010*b*), Efficient dominance testing for unconditional preferences, *in* Lin et al. (2010).

Santhanam, G. R., Basu, S. & Honavar, V. (2011), 'Representing and reasoning with qualitative preferences for compositional systems', *Journal of Artificial Intelligence Research* **42**, 211–274.

Sardiña, S. & Shapiro, S. (2003), Rational action in agent programs with prioritized goals, *in* 'AAMAS', ACM, pp. 417–424.

Sarwar, B., Karypis, G., Konstan, J. & Reidl, J. (2001), Item-based collaborative filtering recommendation algorithms, *in* 'Proceedings of the international conference on World Wide Web', WWW 01, ACM, New York, NY, USA, pp. 285–295.

Savage, L. J. (1954), *Foundations of Statistics*, New York Wiley.

Schafer, J. B., Konstan, J. A. & Riedl, J. (2001), 'E-commerce recommendation applications', *Data Mining and Knowledge Discovery* **5**(1/2), 115–153.

Schaub, T. & Wang, K. (2001), A comparative study of logic programs with preference, *in* Nebel (2001), pp. 597–602.

Schclar, A., Tsikinovsky, A., Rokach, L., Meisels, A. & Antwarg, L. (2009), Ensemble methods for improving the performance of neighborhood-based collaborative filtering, *in* L. D. Bergman, A. Tuzhilin, R. D. Burke, A. Felfernig & L. Schmidt-Thieme, eds, 'RecSys', ACM, pp. 261–264.

Schein, A. I., Popescul, A., Ungar, L. H. & Pennock, D. M. (2002), Methods and metrics for cold-start recommendations, *in* 'SIGIR', ACM, pp. 253–260.

Schiex, T. (1992), Possibilistic constraint satisfaction problems or "how to handle soft constraints?", *in* D. Dubois & M. P. Wellman, eds, 'UAI', Morgan Kaufmann, pp. 268–275.

Schiex, T., Fargier, H. & Verfaillie, G. (1995), Valued constraint satisfaction problems: Hard and easy problems, *in* 'Proceedings of the International Joint Conference on Artificial Intelligence 1995 (IJCAI-95)', pp. 631–639.

Schmitt, S. (2002*a*), '*simVar*: A similarity-influenced question selection criterion for e-sales dialogs', *Artificial Intelligence Review* **18**, 195–221.

Schmitt, S. (2002*b*), 'simvar: A similarity-influenced question selection criterion for e-sales dialogs', *Artificial Intelligence Review* **18**(3-4), 195–221.

Scott, D. & Suppes., P. (1958*a*), 'Foundational aspects of theories of measurement', *Journal of Symbolic Logic* pp. 113–128.

Scott, D. & Suppes, P. (1958*b*), 'Foundational aspects of theories of measurement', *Journal of Symbolic Logic* **23**, 113–128.

Searls, D. & Norton., L. (1990), 'Logic-based configuration with a semantic network', *Journal of Logic Programming* **19**, 53–73.

Serna, M., Trevisan, L. & Xhafa, F. (2005), 'The approximability of non-boolean satisfiability problems and restricted integer programming', *Theoretical Computer Science* **332**(1-3), 123–139.

Shani, G. & Gunawardana, A. (2011), Evaluating recommendation systems, *in* 'Recommender Systems Handbook', pp. 257–297.

Shardanand, U. & Maes, P. (1995), Social information filtering: Algorithms for automating "word of mouth", *in* Katz et al. (1995), pp. 210–217.

Shen, X. (2007), User-centered adaptive information retrieval, PhD thesis, Champaign, IL, USA.

Shen, X., Tan, B. & Zhai, C. (2005), Implicit user modeling for personalized search, *in* O. Herzog, H.-J. Schek, N. Fuhr, A. Chowdhury & W. Teiken, eds, 'CIKM', ACM, pp. 824–831.

Shimazu, H. (2001), Expertclerk: Navigating shoppers buying process with the combination of asking and proposing, *in* Nebel (2001), pp. 1443–1450.

Shimazu, H. (2002), 'Expertclerk: A conversational case-based reasoning tool for developing salesclerk agents in e-commerce webshops', *Artificial Intelligence Review* **18**(3-4), 223–244.

Shimazu, H., Shibata, A. & Nihei, K. (2001), 'Expertguide: A conversational case-based reasoning tool for developing mentors in knowledge spaces', *Applied Intelligence* **14**(1), 33–48.

Smith, B. M. (2001), 'Constructing an asymptotic phase transition in random binary constraint satisfaction problems', *Theoretical Computer Science* **265**(1-2), 265–283.

Smyth, B. (2007), *The Adaptive Web*, Vol. 4321, Springer, chapter Case-Based Recommendation, pp. 342–376.

Smyth, B. & Cotter, P. (1999), Surfing the digital wave, *in* K.-D. Althoff, R. Bergmann & K. Branting, eds, 'ICCBR', Vol. 1650 of *Lecture Notes in Computer Science*, Springer, pp. 561–571.

Smyth, B. & Cotter, P. (2000), 'Enabling technologies: a personalized television listings service', *Communication ACM* **43**(8), 107–111.

Smyth, B. & McGinty, L. (2003), The power of suggestion, *in* Gottlob & Walsh (2003), pp. 127–132.

Son, T. C. & Pontelli, E. (2006), 'Planning with preferences using logic programming', *TPLP* **6**(5), 559–607.

Sprecher, A. (2002), 'Network decomposition techniques for resource-constrained project scheduling', *The Journal of the Operational Research Society* **53**(4).

Stahl, A. (2002), Defining similarity measures: Top-down vs. bottom-up, *in* Craw & Preece (2002), pp. 406–420.

Stahl, A. (2006), Combining case-based and similarity-based product recommendation, *in* T. Roth-Berghofer, M. H. Göker & H. A. Güvenir, eds, 'ECCBR', Vol. 4106 of *Lecture Notes in Computer Science*, Springer, pp. 355–369.

Stefanidis, K., Koutrika, G. & Pitoura, E. (2011), 'A survey on representation, composition and application of preferences in database systems', *ACM Transactions on Database Systems* **36**(3), 19.

Stewart, T. J. (1996), 'Robustness of Additive Value Function Methods in MCDM', *Journal of Multi-Criteria Decision Analysis* **5**(4), 301–309.

Stolze, M. (2000), 'Soft navigation in electronic product catalogs', *International Journal on Digital Libraries* **3**(1), 60–66.

Straccia, U. (2008), Reasoning web, Springer-Verlag, Berlin, Heidelberg, chapter Managing Uncertainty and Vagueness in Description Logics, Logic Programs and Description Logic Programs, pp. 54–103.

Stumptner, M. (1997), 'An overview of knowledge-based configuration', *AI Communications* pp. 111–125.

Sycara, K. P., Roth, S. P., Sadeh, N. M. & Fox, M. S. (1991), 'Resource allocation in distributed factory scheduling', *IEEE Expert* **6**(1), 29–40.

Thompson, C. A., Göker, M. H. & Langley, P. (2004*a*), 'A personalized system for conversational recommendations', *Journal Artificial Intelligence Research* **21**, 393–428.

Thompson, C. A., Göker, M. & Langley, P. (2004*b*), 'A personalized system for conversational recommendations', *Journal of Artificial Intelligence Research* **21**, 393–428.

Tintarev, N. & Masthoff, J. (2011), Designing and evaluating explanations for recommender systems, *in* 'Recommender Systems Handbook', pp. 479–510.

Topaloglu, S. & Ozkarahan, I. (2004), Comparison of different variable and value order strategies for the optimum solution of a single machine scheduling problem with sequence-dependent setups., *in* C. Aykanat, T. Dayar & I. Korpeoglu, eds, 'ISCIS', Vol. 3280 of *Lecture Notes in Computer Science*, Springer, pp. 996–1005.

Torrens, M., Faltings, B. & Pu, P. (2002), 'Smartclients: Constraint satisfaction as a paradigm for scaleable intelligent information systems', *Constraints* **7**(1), 49–69.

Tou, F. N., Williams, M. D., Fikes, R., Jr., D. A. H. & Malone, T. W. (1982), Rabbit: An intelligent database assistant, *in* D. L. Waltz, ed., 'AAAI', AAAI Press, pp. 314–318.

Tsoukiàs, A. (2008), 'From decision theory to decision aiding methodology', *European Journal of Operational Research* **187**(1), 138–161.

Venturini, A. & Ricci, F. (2006), Aplying trip@dvice recommendation technology to www.visiteurope.com, *in* Brewka et al. (2006), pp. 607–611.

Veron, M., Fargier, H. & Aldanondo, M. (1999), From CSP to configuration problems., *in* 'Proceedings of the AAAI'99 Worshop on Configuration , Orlando (Floride), 18/07/99', AAAI Press, pp. 101–106.

Vig, J., Sen, S. & Riedl, J. (2011), Navigating the tag genome, *in* P. Pu, M. J. Pazzani, E. André & D. Riecken, eds, 'IUI', ACM, pp. 93–102.

Vincke, P. (1992), *Multicriteria Decision-Aid*, J. Wiley, New York.

Vion, J. (2006), Csp4j: a black-box csp solving api for java, *in* 'Proceeding of the 2nd International CSP Solver Competition', pp. 75–88.

Visser, W., Hindriks, K. V. & Jonker, C. M. (2009), Argumentation-based preference modelling with incomplete information, *in* 'CLIMA', pp. 141–157.

von Stengel, B. (1988), 'Decomposition of multiattribute expected-utility functions', *Annals of Operations Research* **16**, 161–183.

von Winterfeldt, D. & Edwards, W. (1996), *Decision Analysis and Behavioural Research*, Cambridge University Press.

Voss, S., H.Osman, I. & Roucairol, C., eds (1999), *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, Norwell, MA, USA.

Wakker, P. & Deneffe, D. (1996), 'Eliciting von neumann-morgenstern utilities when probabilities are distorted or unknown', *Management Science* **42**(8), 1131–1150.

Wallace, R. J. & Wilson, N. (2009), 'Conditional lexicographic orders in constraint satisfaction problems', *Annals OR* **171**(1), 3–25.

Warnestal, P. (2007), Dialogue behavior management in conversational recommender systems, PhD thesis, Linkopings universitet.

Wellman, M. P. & Doyle, J. (1991), Preferential semantics for goals, *in* T. L. Dean & K. McKeown, eds, 'AAAI', AAAI Press / The MIT Press, pp. 698–703.

Wellman, M. P. & Doyle, J. (1992), Modular utility representation for decision-theoretic planning, *in* 'Proceedings of the first international conference on Artificial intelligence planning systems', Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 236–242.

Wellman, M. P. & Doyle, J. (1994), Representing preferences as ceteris paribus comparatives, *in* 'AAAI Spring Symposium on Decision Theoretic Planning', pp. 69–75.

Wilson, N. (2004*a*), Consistency and constrained optimisation for conditional preferences, *in* de Mántaras & Saitta (2004), pp. 888–894.

Wilson, N. (2004*b*), Extending CP-nets with stronger conditional preference statements, *in* McGuinness & Ferguson (2004), pp. 735–741.

Wilson, N. (2006), An efficient upper approximation for conditional preference, *in* Brewka et al. (2006), pp. 472–476.

Wilson, N. (2009*a*), An efficient deduction mechanism for expressive comparative preference languages, *in* Boutilier (2009), pp. 961–966.

Wilson, N. (2009*b*), Efficient inference for expressive comparative preference languages, *in* Boutilier (2009), pp. 961–966.

Wilson, N. (2011), 'Computational techniques for a simple theory of conditional preferences', *Artificial Intelligence* **175**(7-8), 1053–1091.

Woronowicz, E. & Zalewska, A. (2004), 'Properties of binary relations', *Journal of Formalized Mathmatics* **1**.

Xia, L., Conitzer, V. & Lang, J. (2008), Voting on multiattribute domains with cyclic preferential dependencies, *in* Fox & Gomes (2008), pp. 202–207.

Xia, L., Lang, J. & Ying, M. (2007), Strongly decomposable voting rules on multiattribute domains, in *AAAI* (*Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada* 2007), pp. 776–781.

Xu, Z. (2007), 'Multiple-attribute group decision making with different formats of preference information on attributes', *IEEE Transactions on Systems, Man, and Cybernetics, Part B* **37**(6), 1500–1511.

Yaman, F., Walsh, T. J., Littman, M. L. & desJardins, M. (2011), 'Democratic approximation of lexicographic preference models', *Artificial Intelligence* **175**(7-8), 1290–1307.

Yu, Z., Zhou, X., Hao, Y. & Gu, J. (2006), 'Tv program recommendation for multiple viewers based on user profile merging', *User Model. User-Adapt. Interact.* **16**(1), 63–82.

Zadeh, L. A. (1999), 'Fuzzy sets as a basis for a theory of possibility', *Fuzzy Sets and Systems* **100**(supp.), 9–34.

Zaslow, J. (2002), 'If tivo thinks you are gay, here's how to set it straight', *The Wall Street Journal* .

Zins, A. H., Bauernfeind, U., Missier, F. D., Mitsche, N., Ricci, F., Rumetshofer, H. & Schaumlechner, E. (2004), Prototype testing for a destination recommender system: steps, procedures and implications, *in* 'Proceedings of the International Conference on Information and Communication Technologies in Travel and Tourism ENTER-04'.

Zins, A. H., Bauernfeind, U., Missier, F. D. & Rumetshofer, H. (2004), An experimental usability test for different destination recommender systems, *in* 'Proceedings of the International Conference on Information and Communication Technologies in Travel and Tourism ENTER-04'.

Zukerman, I. & Albrecht, D. W. (2001), 'Predictive statistical models for user modeling.', *User Modeling and User-Adapted Interaction* **11**(1-2), 5–18.