SCALABLE INTELLIGENT ELECTRONIC CATALOGS

THÈSE N^{O} 2690 (2002)

PRÉSENTÉE À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

SECTION D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Marc TORRENS ARNAL

Ingénieur informaticien, Universitat Politècnica de Catalunya, Espagne
et de nationalité espagnole

acceptée sur proposition du jury:

Prof. B. Faltings, directeur de thèse
Prof. P. Meseguer, rapporteur
Dr P. Pu Faltings, rapporteur
Dr M. Stolze, rapporteur
Prof. A. Wegmann, rapporteur

Lausanne, EPFL
2003

En memòria de l'*abuelo*, per la seva sensibilitat i tots els bons records que ens ha deixat.

A tu, Laura

Abstract

The world today is full of information systems which make huge quantities of information available. This incredible amount of information is clearly overwhelming Internet end-users. As a consequence, intelligent tools to identify worthwhile information are needed, in order to fully assist people in finding the right information. Moreover, most systems are ultimately used, not just to provide information, but also to solve problems.

Encouraged by the growing popular success of Internet and the enormous business potential of electronic commerce, e-catalogs have been consolidated as one of the most relevant types of information systems. Nearly all currently available electronic catalogs are offering tools for extracting product information based on key-attribute filtering methods. The most advanced electronic catalogs are implemented as recommender systems using collaborative filtering techniques.

This dissertation focuses on strategies for coping with the difficulty of building intelligent catalogs which fully support the user in his purchase decision-making process, while maintaining the scalability of the whole system. The contributions of this thesis lie on a mixed-initiative system which is inspired by observations on traditional commerce activities. Such a conversational model consists basically of a dialog between the customer and the system, where the user criticizes proposed products and the catalog suggests new products accordingly.

Constraint satisfaction techniques are analyzed in order to provide a uniform framework for modeling electronic catalogs for configurable products. Within the same framework, user preferences and optimization constraints are also easily modeled. Searching strategies for proposing the adequate products according to criteria are described in detail.

Another dimension of this dissertation faces the problem of scalability, *i.e.*, the problem of supporting hundreds, or thousands of users simultaneously using intelligent electronic catalogs. Traditional wisdom would presume that in order to provide full assistance to users in complex tasks, the business logic of the system must be complex, thus preventing scalability. *SmartClient* is a software architectural model that uses constraint satisfaction problems for representing solution spaces, instead of traditional models which represent

solution spaces by collections of single solutions. This main idea is supported by the fact that constraint solvers are extreme in their compactness and simplicity, while providing sophisticated business logic. Different *SmartClient* architecture configurations are provided for different uses and architectural requirements.

In order to illustrate the use of constraint satisfaction techniques for complex electronic catalogs with the *SmartClient* architecture, a commercial Internet-based application for travel planning, called *reality*, has been successfully developed. Travel planning is a particularly appropriate domain for validating the results of this research, since travel information is dynamic, travel planning problems are combinatorial, and moreover, complex user preferences and optimization constraints must be taken into consideration.

Résumé

Le monde d'aujourd'hui est rempli de systèmes d'information qui rendent disponibles d'énormes quantités de données. Cet incroyable quantité d'information surpasse clairement l'utilisateur d'Internet. C'est pourquoi des outils intelligents permettant d'identifier les informations pertinentes sont nécessaire pour pouvoir aider les personnes à trouver l' information adéquate. De plus, la plupart des systèmes sont finalement utilisés non seulement pour trouver de l'information mais également pour résoudre des problèmes.

Encouragés par le succès croissant d'Internet et l'énorme potentiel du commerce électronique, les catalogues électroniques sont devenus un des types de systèmes d'information les plus incontournables. La plupart de ceux-ci permettent de rechercher des produits grâce à des méthodes de filtrages basés sur leurs attributs. Les catalogues électroniques les plus évolués sont conçus comme des systèmes de recommandation utilisant des techniques de filtrage coopératif.

Cette thèse se concentre sur les stratégies pour faire face à la difficulté de construire des catalogues intelligents qui supportent l'utilisateur dans son processus de décision d'achat tout en préservant la scalabilité du système. Les contributions de cette thèse se reposent sur modèle d'interaction inspiré par l'observation du commerce traditionnel. Ce modèle consiste d'un dialogue entre le client et le système, où l'utilisateur critique les produits proposés et le catalogue suggère de nouveaux produits en conséquence.

Les techniques de satisfaction de contraintes sont analysées afin de fournir un cadre uniforme pour la modélisation de catalogues électroniques de produits configurables. A l'intérieur de ce cadre, les préférences d'utilisateur ainsi que les contraintes d'optimisation sont aussi facilement modélisées. Des stratégies de recherche pour proposer les produits conformes à ces préférences et contraintes sont décrites en détails.

Un autre aspect de cette thèse fait face au problème de scalabilité, c'est-à-dire au problème de supporter des centaines, ou milliers, d'utilisateurs simultanés des catalogues électroniques. La sagesse traditionnelle présumerait que, pour fournir une assistance complète à l'utilisateur pour des tâches complexes, la logique du système doit être complexe donc empêchant toute scalabilité. *SmartClient* est un modèle d'architecture logicielle qui utilise les problèmes de satisfaction de contraintes pour représenter les espaces de solu-

tions à la place des modèles traditionnels qui les représentent à l'aide de collections de solutions isolées. La principale idée est soutenue par le fait que les algorithmes de résolution par contraintes sont extrêmement compacts et simples tout en fournissant une logique sophistiquée. Différentes configurations de l'architecture *SmartClient* sont données pour différentes utilisations et spécifications d'architecture.

Afin d'illustrer l'utilisation des techniques de satisfaction de contraintes pour des catalogues électroniques complexes utilisant l'architecture *SmartClient*, une application commerciale pour la planification de voyages, appelée *reality*, a été développée avec succès. La planification de voyages est un domaine particulièrement approprié pour la validation des résultats de cette recherche car l'information est dynamique, les problèmes de planification sont combinatoires et, de plus, de complexes préférences d'utilisateur et des contraintes d'optimisation doivent être prises en considération.

Acknowledgements

Firstly, I wish to address my sincerest thanks to my Ph.D. advisor Professor Boi Faltings who gave me the opportunity to work at the Artificial Intelligence Laboratory (LIA). He was constantly assisting me with new research directions and interesting ideas. I would like to also thank him for his efforts in leading the creation of Iconomic Systems S.A. which allowed us to commercially develop the concepts of this thesis. Actually, this work and Iconomic Systems S.A. were always strongly related, and Boi contributed in both projects in a very decisive manner.

I am very grateful to Professors Pedro Meseguer, Pearl Pu, Markus Stolze and Alain Wegmann, the thesis committee, for their careful reading of my thesis and the many valuable comments I received in the thesis defense.

I would like to thank Christian Frei who gave me pertinent and very detailed comments on the thesis report which contributed to a significant improvement of the final result. Without any doubt, the constraint-based engine that he implemented during his stay at Iconomic Systems S.A. was of a great help for this thesis elaboration. Patrick Hertzog was very supportive during the writing phase of the thesis, providing: many detailed comments on almost every chapter, comments on all figures, the reality scenario of the Appendix A, and a lot of help (even the computer I used for the exam was his notebook!). I'm very thankful to Sarah Ordoño for her efforts in ironing the English of the whole thesis report! Jean-Cédric Chappelier and Loic Samson also provided me with many useful comments.

The starting point of my Swiss adventure dates back to summer 1997 and is located in the campus of the *Universitat Politècnica de Catalunya* (UPC), more specifically in the office of Professor Ulises Cortés. He proposed me to carry out my diploma work at the EPFL, against my first preference which was some far away university in the north of Finland. Ulises, thank you for your smart advice on that! We shared (and will share) many great moments in Barcelona and Lausanne, nothing to do with computer science but restaurants, home-made Mexican cooking, philosophical discussions, aged Tequila tasting...

Throughout my almost 4 years at the LIA I had a very exciting and interesting working experience. One of the factors for that is the cosmopolitan population of the LIA: I counted more than 15 different nationalities! I learned a lot with Rainer Weigel who introduced me

to the insights of constraint satisfaction technology while being my diploma work advisor. I really enjoyed the teaching experiences I had with Martin Rajman and Boi Faltings. Many thanks also to all the inhabitants of the LIA for great coffee breaks with so many political, cultural and philosophical conversations!

This thesis would never be as it is without Iconomic Systems S.A. which commercialized *reality*[1], the product demonstrating the concepts of this thesis work. Iconomic Systems S.A. was founded by Boi Faltings, Pearl Pu, Christian Frei and myself. A technology start-up company is a very challenging and unforgettable experience implying many people, in our case: Boi (leading the adventure), Andreas Günthard (the big boss who sold the company in despite of the catastrophic general economic situation), Pearl (strong contributions on human-computer interaction and business aspects), Chris (magnificent work with the constraint-based engine!), Sebastian Gerlach (wow! what a graphical world map!), Loic Samson (our architecture guru), David Nemeshazy (our great all-in-one computer scientist), Patrick Hertzog (our usability guru), Denis Lalanne (Pat's usability guru) and Mattia Bosco (my `UT` teammate). I'm also grateful to the executive management of i:FAO A.G.[2] to allow us to continue with our project.

Merci beaucoup à tous les amis que j'ai rencontrés en Suisse! Ça serait trop long d'en faire la liste mais vous savez qui vous êtes!

Arpofito l'ocasió per agrair tot el suport que sempre he rebut i les visites a Lasuanne dels meus amics de sempre a Barcelona: Víctor, Rebeka, Òscar, Marta i Genís.

Gràcies a tota la meva família. Als meus avis i *abuelos* per la seva estimació i dedicació. Al Josep Ma., pels seus valuosos consells i per l'ajuda que m'ha brindat en moments complicats. A la Míriam, molt més que una germana, la meva millor amiga, moltes gràcies per tot! Al meu pare per descobrir-me l'interessant món de la ciència, sense dubtes la base d'aquest treball. I per últim, a la meva mare, simplement per ensenyar-me el més difícil d'aprendre, el què les universitats mai podran ensenyar: l'art de saber viure i estimar! Infinites gràcies per tots els teus esforços durant tots aquests anys.

Laura, finalment ho hem fet! Sense tu, tot això no seria el què és. T'agraeixo tot el teu incondicional suport. M'has ajudat molt més del què et penses... Gràcies!

[1] Firstly, it was called *SmartTravel*, then *IsyTravel*, and finally *reality*.
[2] Iconomic Systems S.A. was acquired by i:FAO A.G. in 2001.

Contents

Abstract **i**

Résumé **iii**

Acknowledgements **v**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Information systems . . . 2
1.2.1 Electronic commerce . . . 2
1.2.2 Electronic catalogs . . . 4
1.2.3 Configuration . . . 7
1.2.4 Personalization . . . 11
1.2.5 Recommender systems . . . 12
1.3 Complex electronic catalogs . . . 14
1.3.1 Examples in the industry . . . 15
1.4 Contributions . . . 17
1.5 Overview of the dissertation . . . 18

2 Personalization **21**
2.1 Introduction . . . 21
2.2 A mixed-initiative system for electronic catalogs . . . 24
2.2.1 Conversational model . . . 25
2.2.2 Preference elicitation . . . 25
2.2.2.1 Preferences in any order at any time . . . 26
2.2.2.2 Posting preferences . . . 26
2.2.2.3 Retracting and modifying preferences . . . 27
2.2.2.4 Contextual preferences . . . 27
2.2.3 Converging to satisfactory solutions . . . 27
2.3 Requirements for online interaction . . . 28
2.4 Related Work . . . 28
2.4.1 Personalization by mixed-initiative systems . . . 29
2.4.2 Personalization by user profiles . . . 29
2.5 Summary . . . 30

3 Modeling Electronic Catalogs as CSP **31**
3.1 Introduction . . . 31
3.2 Constraint satisfaction problems (CSPs) . . . 32
3.2.1 Notational conventions . . . 32
3.2.2 Classical CSPs definitions . . . 33
3.2.3 Extending classical CSPs . . . 38
3.3 Encoding configurable electronic catalogs as CSPs . . . 40
3.3.1 Variables and values . . . 41
3.3.2 Constraints, preferences and optimization . . . 41
3.3.3 Solution space . . . 43
3.3.4 An alternative CSP model . . . 43
3.4 Extended CSP frameworks for electronic catalogs . . . 45
3.5 Constraint satisfaction for configurable catalogs . . . 47
3.6 Optimal solutions to MCOPs . . . 48
3.7 The quantitative approach . . . 49
3.7.1 Order of feasible solutions . . . 50
3.7.2 User preferences and optimization criteria . . . 51
3.8 The qualitative constraint combination approach . . . 54
3.8.1 Order of feasible solutions . . . 55
3.8.2 Pareto optimality . . . 56
3.8.3 User's preferences and optimization criteria . . . 60
3.8.4 Pareto-optimal solutions on the convex hull . . . 60
3.8.5 What are the k best solutions? . . . 61
3.8.5.1 Too few Pareto-optimal solutions . . . 63
3.8.5.2 Too many Pareto-optimal solutions . . . 64
3.8.5.3 Totally ordering for Pareto-optimal solutions . . . 64
3.9 An example: photo equipments . . . 64
3.10 Summary . . . 68

4 Searching Methods for Electronic Catalogs as CSP **71**
4.1 Introduction . . . 71
4.2 The quantitative approach . . . 71
4.2.1 Depth First Branch and Bound (DFBB) algorithm . . . 72
4.2.2 Prospective strategies . . . 77
4.2.2.1 Partial Forward Checking . . . 77
4.2.2.2 Preprocessing look-ahead techniques . . . 81
4.2.3 Search ordering heuristics . . . 82
4.2.3.1 Static and dynamic ordering . . . 82
4.2.3.2 Variable ordering . . . 83
4.2.3.3 Value ordering . . . 84
4.2.4 Retrospective strategies . . . 84
4.2.4.1 Backjumping . . . 84
4.2.4.2 Backmarking . . . 85

4.2.5 Improving initial upper bound . . . 85
4.2.6 A note on unary constraints . . . 86
4.2.7 A note on non-binary constraints . . . 86
4.2.8 DFBB as an anytime algorithm . . . 87
4.2.9 Soundness, completeness and complexity . . . 87
4.3 The qualitative constraint combination approach . . . 88
4.3.1 Methods for approximating pareto optimal solutions . . . 89
4.3.1.1 Simple method . . . 90
4.3.1.2 Iterative method . . . 92
4.3.2 Soundness, completeness and complexity . . . 93
4.3.2.1 Complexity . . . 94
4.3.3 Evaluation . . . 95
4.3.3.1 Random MCOP generation . . . 95
4.3.3.2 Results on random problems . . . 96
4.3.4 Solutions on the convex hull . . . 99
4.4 Converging to satisfactory solutions . . . 99
4.4.1 The quantitative approach . . . 100
4.4.2 The qualitative constraint combination approach . . . 102
4.4.3 A mixed approach . . . 102
4.5 Related work . . . 103
4.5.1 The quantitative approach . . . 103
4.5.1.1 Systematic search . . . 103
4.5.1.2 Local search . . . 103
4.5.2 The qualitative constraint combination approach . . . 105
4.5.2.1 Systematic search algorithms . . . 106
4.5.2.2 Methods for finding *supported* Pareto-optimal solutions . . . 106
4.5.2.3 Local Search methods . . . 109
4.5.2.4 Genetic algorithms . . . 110
4.5.2.5 Pareto-optimal solutions on the convex-hull . . . 111
4.6 Summary . . . 112

5 The *SmartClient* Architecture 113
5.1 Motivation . . . 113
5.2 Software architectures . . . 114
5.2.1 Definitions . . . 114
5.2.2 Client-server architectures . . . 115
5.2.3 Stateful vs stateless servers . . . 115
5.2.4 Layered architectures . . . 116
5.2.4.1 Two-tier architectures . . . 117
5.2.4.2 Three-tier architectures . . . 117
5.2.5 Topics of interest . . . 118
5.3 Standard architectures for electronic catalogs . . . 118
5.3.1 Purchase decision making process . . . 119

5.3.1.1 On-step configuration . . . 120
5.3.1.2 Sequential configuration . . . 120
5.3.2 Stateless server vs stateful server . . . 122
5.3.2.1 Client-stateless-server . . . 122
5.3.2.2 Client-stateful-server . . . 122
5.3.2.3 Trade-off between stateless-servers and stateful-servers . . 122
5.4 The *SmartClient* architecture . . . 123
5.4.1 *SmartClient* main concepts . . . 124
5.4.2 Representing solution spaces . . . 124
5.4.3 *SmartClient* architecture configurations . . . 126
5.4.3.1 Stand-alone application clients . . . 127
5.4.3.2 Remote application clients . . . 128
5.4.3.3 Front-end clients . . . 129
5.5 The scalability of the *SmartClient* concept . . . 130
5.5.1 Conventional approach . . . 131
5.5.2 Basic *SmartClient* configuration . . . 131
5.6 Summary . . . 132

6 The Travel Planning Problem 133
6.1 Motivation . . . 133
6.2 The traditional travel industry . . . 133
6.2.1 Travel supply chain . . . 134
6.2.2 Global Distribution Systems (GDS) . . . 134
6.3 The internet travel industry . . . 136
6.3.1 Main GDS/CRS functionalities . . . 138
6.3.1.1 `AirAvailability` request . . . 138
6.3.1.2 `PriceItinerary` request . . . 139
6.3.1.3 `BestBuy` request . . . 139
6.3.2 Standard Internet travel planning systems . . . 139
6.3.2.1 One-step configuration . . . 140
6.3.2.2 Sequential configuration . . . 140
6.4 The problem of planning travels . . . 141
6.4.1 Statement and definitions . . . 141
6.5 Modeling the travel problem as a CSP . . . 145
6.5.1 Variables and values . . . 145
6.5.2 Configuration rules . . . 146
6.5.3 User preferences . . . 146
6.5.3.1 Soft and crisp constraints . . . 146
6.5.3.2 Soft and flexible constraints . . . 147
6.5.3.3 Soft constraint weights . . . 148
6.5.3.4 Contextual soft constraints . . . 148
6.5.4 Optimization criteria . . . 148
6.5.5 Note on using fares . . . 149

6.5.6 Note on using `BestBuy` request . . . 150
6.6 An alternative CSP modeling approach . . . 150
6.7 Related work . . . 151
6.8 Summary . . . 152

7 A Commercial Application for Travel Planning: *reality* 153
7.1 Context . . . 153
7.2 Architecture . . . 154
7.2.1 Gathering information on the server side . . . 154
7.2.2 Travel planning on the client side . . . 155
7.2.3 Some implementation facts . . . 155
7.3 Mixed-initiative system . . . 156
7.3.1 Posting, modifying and retracting preferences . . . 156
7.3.2 Showing violations . . . 157
7.3.3 Solution displays . . . 157

8 Conclusions 161
8.1 Scope . . . 161
8.2 Contributions . . . 162
8.2.1 Mixed-initiative system for electronic catalogs . . . 162
8.2.2 Modeling and searching strategies for electronic catalogs . . . 162
8.2.3 *SmartClient* architecture . . . 163
8.2.4 Validation with a commercial application in the travel domain . . . 163
8.3 Limitations . . . 164
8.4 Further research . . . 165
8.5 Conclusion . . . 167

A Using *reality*: a scenario 169
A.1 The scenario using *reality* . . . 169
A.2 The scenario using a standard travel planning tool . . . 171
A.3 Comparison between *reality* and *cytric v7* . . . 172

B Solving classical Constraint Satisfaction Problems 185
B.1 Systematic search algorithms . . . 185
B.1.1 Generate and Test . . . 185
B.1.2 Chronological Backtracking . . . 186
B.2 Local consistency techniques . . . 186
B.3 Look-back strategies . . . 187
B.3.1 Backjumping . . . 187
B.3.2 Graph-based Backjumping . . . 188
B.3.3 Conflict-directed Backjumping . . . 188
B.3.4 Backmarking . . . 188
B.4 Look-ahead strategies . . . 189
B.4.1 Forward Checking . . . 189

B.4.2 Maintaining Arc Consistency 190
B.5 Variable and value ordering heuristics 190
B.5.1 Variable ordering 190
B.5.1.1 Static Variable Ordering 190
B.5.1.2 Dynamic Variable Ordering 191
B.5.2 Value ordering 192
B.6 Stochastic and heuristic algorithms 192
B.6.1 Basic local search schema: Hill-climbing 192
B.6.2 Min-conflicts 193
B.6.3 Random walk 193
B.6.4 Connectionist approach 193
B.6.5 Guided Local Search and Fast Local Search 193
B.6.6 Tabu search 194
B.6.7 Simulated Annealing 194
B.6.8 Genetic Algorithms 194

C Extended Constraint Satisfaction Problems 197
C.1 Constraint hierarchies 197
C.2 Partial constraint satisfaction problems 198
C.3 Soft constraint satisfaction problems 198
C.3.1 Fuzzy CSPs (FCSP) 198
C.3.2 Probabilistic CSPs 199
C.3.3 Possibilistic CSPs 199
C.3.4 Weighted CSPs (WCSP) 200
C.3.5 Lexicographic CSPs 200
C.4 Semiring-based constraint satisfaction problems (SCSP) 200
C.5 Valued constraint satisfaction problems (VCSP) 202
C.6 Valuations for constraints and tuples 203
C.7 Main properties of SCSP and VCSP 204
C.8 Casting CSP frameworks into CSP meta-frameworks 205

D Java Constraint Library 207
D.0.1 Historical background of the JCL 207
D.1 Java Constraint Library 208
D.1.1 The constraint library 208
D.1.2 The graphical user interface 210
D.2 Choice Constraint Language (CCL) 210
D.3 JCL with CCL 211

Bibliography 213

Index 235

Curriculum Vitae and Publications 239

List of Figures

1.1 Basic components involved in electronic catalogs within a 3-tier architecture. The *presentation layer* provides graphical user interfaces between the user and the system. The *business logic layer* processes the data from the presentation layer to the persistence layer and vice-versa. The *persistence layer* is used to store the data of the catalog and provides mechanisms for updating and retrieving data. 5

1.2 Complex electronic catalogs require to handle configuration and personalization in order to support the user in the selection decision making process. 15

1.3 Outline of the thesis and dependencies of chapters. 20

2.1 Example of an interaction between the buyer and the seller in the travel domain. The conversation supports the user to find out the flight that best fits his needs and preferences. Observation 1 and Observation 2 are illustrated in this dialog. 22

2.2 The goal of a trading conversation between the seller and the buyer is to narrow down the initial available product space to a reduced set of satisfactory products. 22

2.3 Example about the level of vagueness of initial requests for photo lenses to take animal pictures. The level of expertise of the customer influences the precision of the initial request. The customer's level of expertise indicates the customer's needed support in the purchase decision making process. . . 23

2.4 Conversational model for electronic catalogs. In the main query, the potential customer specifies his rough idea about the product. Then, a conversation takes place in order to converge to a satisfactory product. The conversation mainly consists in a loop where the buyer criticizes, and the seller proposes items according to the user's critiques. 24

3.1 The crossword puzzle structure of Example 3.1 with a solution. 36

3.2 The CSP graph representing the crossword puzzle of Example 3.1. 37

3.3 The search tree associated to the search space of the Example 3.1. For simplicity, only the branches containing a solution are expanded. The search tree of this example has 216 (6^3) leaves. Leaves represent all the possible total assignments. The nodes being part of the solution are displayed in bold. 37

3.4 The constraint's hierarchy related to any electronic configurable catalog. Hard (configuration) constraints are more important than soft constraints which are more important than optimization constraints. A constraint type *dominates* another constraint type if constraints belonging to the first type have more important valuations than the second type. 42

3.5 Example of solutions in a CSP with two soft constraints. The two coordinates show the values indicating the degrees to which criteria, horizontal and vertical, are violated. Solutions of the problem strongly depend on the CSP framework. Possible orderings of these solutions are presented in Table 3.1. 46

3.6 Subfigure (a) is an example of a valuation function for a preference normalized in the range $[0,1]$. If we assume that the problem has op optimization constraints, then the valuation of all the optimization constraints lies within the range $[0,op]$. Subfigure (b) shows a possible way of transforming the preference valuation function in a discrete function with k values. Subfigure (c) plots the valuation function for the same preference rescaled to the range $[0,k \cdot op]$. 54

3.7 Example of Pareto-optimal solutions in a MCOP with two preference criteria. The two coordinates show the values indicating the degrees to which criteria, horizontal and vertical, are violated. Each solution dominates solutions in a rectangle. The solutions which are not dominated (1, 3, 4 and 6) are Pareto-optimal. 57

3.8 The two coordinates show the values indicating the degrees to which criteria, horizontal and vertical, are violated. Pareto-optimal solutions (1, 3, 4 and 6) are shown. The constraint weight vector associated to 4 different users are depicted as vectors (`user 1`, `user 2`, `user 3` and `user 4`). The projections of each Pareto-optimal solution to the different user profile vectors indicate to what extent the solution satisfies the criteria for the associated user. . . 61

3.9 The two coordinates show the values indicating the degrees to which criteria, horizontal and vertical, are violated. Pareto-optimal solutions (1, 3, 4 and 6) are shown. The solutions on the convex-hull are linked by a double line. Pareto-optimal solution 3 is not optimal for any user profile because it is not on the convex-hull. 62

4.1 Number of Pareto-optimal solutions depending on the number of soft constraints for random generated problems with 5 variables, 10 values per domain and 20% of hard unary/binary constraint density. The results, in average, are shown for problems with hard tightness of 20%, 40%, 60% and 80%. The total number of solutions in average for these problem topologies are presented in Table 4.1. 97

4.2 Number of Pareto-optimal solutions in percentage of the total number of solutions depending on the number of soft constraints for random generated problems with 5 variables, 10 values per domain and 20% of hard unary/binary constraint density. The results, in average, are shown for problems with hard tightness of 20%, 40%, 60% and 80%. 98

4.3 Pareto-optimal solutions found by the different proposed methods (in %). Methods are applied to 50 randomly generated problems with 10 variables, 10 values per domain, 40% of density of hard unary/binary constraints with 40% of hard tightness and 6 criteria (soft constraints). The number of total computed solutions for each method varies from 30 to 530 in steps of 100. . 99

4.4 Pareto-optimal solutions found by the different proposed methods (in %). Methods are applied to 50 randomly generated problems with 10 variables, 10 values per domain, 40% of density of hard unary/binary constraints with 40% of hard tightness and 6 criteria (soft constraints). The number of total computed solutions, for each method, varies from 30 to 1,030 in steps of 1,000. 100

4.5 Number of Pareto-optimal solutions found by the different proposed methods with respect to the computing time. For this plot, the problems have 10 variables, 10 values per domain with 40% of hard unary/binary constraints with 40% of hard tightness and 6 criteria (soft constraints). 101

4.6 The optimality (valuation normalized on the range [0..1]) of the best 1,000 solutions. The valuations are computed on average for 20 randomly generated problems with 10 variables, 10 values per variable, 40% of density of hard unary/binary constraints with 40% of hard tightness. The number of criteria have been stated to 3, 5, 7, 9, and 12. 102

4.7 The optimality (valuation normalized on the range [0..1]) of the best 1,000 solutions. The valuations are computed on average for 20 randomly generated problems with 10 variables, 10 values per variable, 10% of density of hard unary/binary constraints with 40% of hard tightness. The number of criteria have been stated to 3, 5, 7, 9, and 12. 103

5.1 Basic client-server architecture. Clients simultaneously access services offered by a server through a computer network. 115

5.2 The standard client-server architecture for electronic catalogs. Lines between the client side and the server side show the interactions, *i.e.*, data transfers through the computer network. Almost all the processing is done on the server side, whilst the client side is only used for rendering results. . 119

5.3 One-step decision making process for configurable products in standard electronic catalog architectures. A loop implying the server side and the client side allow the user to browse through product configurations until he finds the right product. 120

5.4 Sequential decision making process for configurable products with n components, in standard electronic catalog architectures. A loop implying the server side and the client side allows the user to configure the product step by step, *i.e.*, at each step a component of the product is selected. When the user realizes he made a mistake in selecting a choice for component i, the server proposes again the choices for that component. 121
5.5 Decision making process for configurable products in *SmartClient* architectures. Most of the business logic is carried out on the client side, allowing the user to benefit from having a mixed-interactive system as explained in Chapter 2. 125

6.1 The travel industry supply chain (from [110]). Users purchase travel products offered by suppliers through the services managed by physical or online booking processes. Distribution systems provide the required travel data to physical or online booking services. 137
6.2 Combinatorial explosion in travel planning. 142

7.1 *reality* architecture: a *SmartClient* architecture for travel planning. 154
7.2 Parallel coordinates to express preferences. 157
7.3 Menus and other graphical widgets to express preferences. 158
7.4 Showing violations on the attributes of an itinerary solution. 159

A.1 By using the graphical world map in *reality*, the user can decide what airports are the most appropriate for his itinerary. On top of that, the user is also able to take a look at the current weather in the airports. 173
A.2 Defining the main query. The user enters the dates and locations of the first leg of his trip.. Note that the *My Choices* panel contains previous booked itineraries. 173
A.3 Defining the main query. The user enters the dates and locations of the second leg of his trip. 174
A.4 Defining the main query. The user enters the dates and locations of the third leg of his trip. 174
A.5 The user receives the first solutions proposed by the system according to the main query. Since the initial query was very vague, it defines more than 3 million different alternatives. 175
A.6 After looking at the first solution, the user criticizes it by posting preferences. Preferred airlines: `AA`, `UA`, and `LX`. Preferred class of service: business, and disliked class of service: economy restricted. 175
A.7 The user prefers to arrive at the first destination before 5 p.m. Then, the first leg of the itinerary proposed by the system satisfies him, and decides to keep it by clicking on the check box on the right. 176
A.8 The user does not like the aircraft of the second leg proposed by the system (a `Boeing 737`), so he indicates this preference. Then, the user receives a proposal for the second leg with an `Airbus A320`. 176

A.9 In the received itinerary, the second leg connects via Minneapolis. Since in winter, Minneapolis is very cold and delays are frequent, the user decides to force the system to propose flights connecting via Houston. The new proposed second leg connects now via Houston, but contains flights with `Boeing 737`. However, the user thinks that it is a good trade-off and decides to keep it as is. 177
A.10 The user does not like to fly with `AirFrance`, so he posts a preference for avoiding `AirFrance`. 177
A.11 Finally, the solution satisfies the user and decides to continue the process. . 178
A.12 The selected itinerary has been priced and fare rules are shown to the user. 178
A.13 Defining the itinerary of the scenario with *cytric v7*. 179
A.14 Results for the flights of the first leg of the itinerary with *cytric v7*. 180
A.15 Results for the flights of the second leg of the itinerary with *cytric v7*. . . . 181
A.16 Results for the flights of the third leg of the itinerary with *cytric v7*. 182
A.17 The selected itinerary has been priced and details are shown with *cytric v7*. 183

D.1 The components of the JCL environment. The JCL is composed by two main elements: the constraint library and the graphical user interface. Java applications can directly use the constraint library or the GUI provided by the JCL Shell. Users can also directly build, save, and solve CSPs through a web browser using the JCL Shell. 209
D.2 JCL and CCL provide the needed tools to build multi-agent applications for solving choice problems. In this example, only 4 agents are considered. . 211

List of Tables

3.1 Solving algorithms of different CSP frameworks may give different results for the same problem. The solutions to the problem are displayed in Figure 3.5. 46

3.2 Individual users with the same criteria would choose different flights because they give different weights to their criteria. The last solution is dominated, thus it would never be rationally preferred for any user profile. 58

3.3 Domain sizes for each type of variable in a CSP for modeling a photo equipment. 66

3.4 num. of b, num. of l and num. of f indicate the number of bodies, lenses and flashes respectively in the problem. The search space for different instances of the CSP modeling photo equipments. Different instances correspond to different sets of components to be selected in the catalog. 66

4.1 The number of of solutions in average for generated problems with 5 variables, 10 values per variable and 20% of hard unary/binary constraint density. 96

4.2 The number of of solutions, in average, under the 0.5 of the quality range of the 1,000 best solutions for problems with different number of criteria. . 101

5.1 Trade-off between efficiency and scalability regarding client-stateless-server and client-stateful-server architectures for electronic catalogs. 123

5.2 *SmartClient* architecture configurations derived from considering different types of clients. 127

5.3 Conventional approach vs. *SmartClient* approach. The data of the table is calculated considering that the user is interested in all possible combinations of flights that match the initial query of the example mentioned before. . . 131

6.1 GDS markets in late 1990s (from [110]). Market share numbers do not equal 100% because not all systems were evaluated. 136

6.2 GDS `AirAvailability` request example: from London to Rio de Janeiro on the 19/10/2002. 138

6.3 GDS `AirAvailability` response example: from London to Rio de Janeiro on the 19/10/2002. 139

A.1 Comparison of the functionality provided by *reality* which are missing in standard travel planning tools. 172

C.1 Specifications $(E, +, \times, \mathbf{0}, \mathbf{1})$ of SCSP for different soft CSPs frameworks. . . 205
C.2 Specifications $(E, \succ, \circledast, \top, \bot)$ of VCSP for different soft CSPs frameworks. . 205

List of Algorithms

1 Depth First Branch and Bound (DFBB) algorithm to find the best solution to a WCOP. 75
2 Depth First Branch and Bound (DFBB) algorithm to find the k best solutions to a WCOP. 76
3 Partial Forward Checking (PFC) algorithm to find the best solution to a WCOP. 81
4 Method for approximating the Pareto-optimal set of a MCOP by using a single WCOP. 91
5 Method for filtering a set of solutions to get a Pareto-optimal set. 92
6 Weighted-sums method for approximating the Pareto-optimal set of a MCOP. The collection of p weight vectors $\mathcal{W}$ are generated to give an adequate distribution of solutions. 93

Chapter 1

Introduction

A weekday edition of the New York Times contains more information than the average person was likely to come across in a lifetime in seventeenth-century England.

Information Anxiety, Richard Saul Wurman.
New York, Doubleday, 1989.

1.1 Motivation

The world today is full of information systems which make huge quantities of information available. A good example is the travel domain, where information systems accessible through the Internet provide information about schedules, fares and availability of almost any means of transport throughout world.

The first generation of information systems provided simple database access facilities such as SQL[1] which allow a user to access specific information. The current generation provides some intelligence for locating the right information, for example by searching for flights at a certain fare or with certain schedule constraints.

However, most information systems are ultimately used to not just provide information, but to *solve problems*. Thus, we believe that the next generation of intelligent information systems should provide explicit support for the problem-solving activities that a user carries out with them. For example, a travel information system should assist the user plan an entire trip according to constraints and preferences, and not just give information about certain airlines schedules.

One dimension of this new generation is the integration of various information systems into a uniform framework using agents. Such integration is appearing for example in shopping robots, or in the integrated travel information system designed by Siemens as a demonstration within the FIPA[2] consortium.

Another dimension is to provide explicit problem-solving capabilities: help with configuring a complete solution, possibly consisting of many parts. For example, a travel planning system would *configure* an entire trip with matching outbound and inbound

[1] SQL stands for Structured Query Language.

[2] Foundation of Intelligent Physical Agents (FIPA) is a non-profit organization aimed at producing standards for the interoperation of heterogeneous software agents. `http://www.fipa.org`.

flights, ground connections, and so on. An insurance planner would configure a suitable insurance package from offers of different companies with different parameters. Such problem solvers will be essential to support people deal with the complexity of the information provided by the information servers.

Scalability is another major issue in this kind of application, since web servers often have to deal with hundreds of users at the same time. Traditional wisdom would say that in order to make *intelligent* tools, it will have to include a lot of complex code and data, *i.e.*, complex software. Thus, having complex software to better support the user apparently increases the difficulty of building scalable architectures.

1.2 Information systems

Recently, it has been estimated that there are 200 million host servers, and 820 million internet users[3]. Clearly, this incredible growth of available information through Internet entails an **overload of information**. Thus, **intelligent tools** to process information to identify worthwhile information are needed.

overload of information $\longrightarrow$ *intelligent* tools

These tools exist for different purposes, ranging from e-mail filters to web search engines. Several techniques for extracting and filtering the right information are widely applied to assist the user. However, in complex domains, more sophisticated techniques to fully support the user are required. This dissertation focuses on a specific kind of information system in the area of electronic commerce, namely electronic catalogs.

1.2.1 Electronic commerce

Traditional commerce can be defined as the set of activities involved in selling and buying goods or services. Analogously, electronic commerce (also called e-commerce) is the electronic counterpart of traditional commerce, *i.e.*, the activities are carried out in some electronic environment, usually through Internet. Actually, the goals of any commerce activity are the same in its traditional form than in its electronic form. The main differences between these commerce modes are the nature of the stakeholders and the interactions among them:

Traditional commerce activities are done directly between human individuals or organizations. The selling and buying activities involve humans without any automatic computerized process. Therefore, traditional commerce activities are achieved by means of **human interactions**.

[3]Data provided by Netsizer of Telcordia Technologies on August 2002, `http://www.netsizer.com` and `http://www.telcordia.com`. These estimations are done by samplings of 150,000 randomly generated IP addresses each day.

Electronic commerce activities are carried out within computer mediated environments. This thesis focuses on electronic commerce systems where the seller is replaced by a software entity[4]. The user is a human being who interacts with the selling software entity. Hence, the e-commerce activity implies **human-computer interactions**. Note that e-commerce systems could also involve buyer software entities with human sellers. Even more, one could imagine e-commerce systems where the whole process is done through software entities. However, this thesis deals with electronic commerce software where only the seller is implemented within a software entity, which is currently the most commonly used form of electronic commerce in the industry.

Electronic mediated commerce arises a broad range of new issues, involving many different disciplines:

- **Security** is an important issue in electronic commerce, for both the buyer and the seller. It involves mechanisms to guarantee that the buyer gets the goods he pays for, and the seller receives the money for the goods. Security in electronic commerce also deals with preventing the hacking of the system, especially in the electronic transaction phase (electronic payments).
- **Trust reputation** faces the difficulties that users encounter to identify to what extent a seller in an e-commerce environment can be trusted. Sometimes, it is difficult for the user to identify the organization behind an e-commerce system.
- **Law** applied to the paradigm of electronic commerce emerges as a new discipline to cover a new way of trading goods and services world-wide electronically.
- **Payment mechanisms** are needed to accomplish transactions electronically in a secure way. Besides the payment security issue, new electronic payment mechanisms arise in the area of electronic transactions.
- **Advertising** can be adapted to increase its efficiency into new electronic forms. For instance, electronic environments allow organizations much more targeted and cost effective marketing.
- **Ontologies** are needed to build electronic market places where several organizations or individuals share and use heterogeneous sources of information.
- **Intermediaries** have to deal with new challenges in electronic commerce. It is believed that e-commerce is changing the way people exchange goods and services, thus the role of intermediaries has been adapted according to these changes.
- **Back-office management** is also changing for those traditional commerce organizations that are combining traditional with electronic commerce or completely migrating to e-commerce.

[4] A *software entity* is also called a *software agent* by some authors. However, since software agents are studied in a specific topic in computer science and this work is not directly related to it, the term *software entity* will be preferred in the reminder of the thesis.

- **Negotiation** can be done (semi) automatically by means of software agents. Negotiation techniques have received a lot of attention from the research community, leading to new e-commerce possibilities such as electronic auctions or shopping robots.
- **Personalization** is about adapting an e-commerce system to each user or potential buyer. Personalized electronic commerce systems allow organizations to build electronic commerce platforms that are able to propose personalized contents and interfaces depending on user profiles.
- **On-line catalogs** can be seen as tools for supporting the buyer to find the best product according to his preferences.

Note that commerce activities handle many different types of products or services (items in general). Depending on different properties, one can distinguish the following types of items:

- **Tangible** goods are traded almost daily by all of us. These are items that can be physically examined. Intangible goods are normally traded by means of contracts between the buyer and the seller. Some examples of intangible items are insurance policies, flight tickets, services, and so on.
- **Configurable** goods are formed by several components. Usually, these components must satisfy a set of configuration or compatibility rules. Thus, these items accept a degree of personalization in the sense that buyers can build their own product (customization). For instance, consider a holiday package which is composed by a flight ticket, a hotel reservation and a rental car booking. Clearly, travel agencies accept variances of such packages under certain compatibility rules. Other examples of configurable goods are houses which can be personalized in many ways, cars which have many available options, insurances which can be adapted to specific needs, and so forth. Note that unconfigurable items will be also called *atomic*.

1.2.2 Electronic catalogs

An electronic catalog[5] is defined here[6] as an information system that provides access to a **collection of product descriptions**[7] that an organization wants to offer. The items of the catalog are identified by a set of **attributes**, and the whole set of items define the **product space**.

Electronic catalogs can be implemented in numerous ways. The most widely accepted architecture, called 3-tier layer architecture, involves the layers and components shown in Figure 1.1:

[5]Electronic catalogs are called by many different terms, for instance e-catalogs, on-line catalogs, electronic shops, or internet stores.

[6]To our knowledge, no formal definition of electronic catalog exists in the research community, thus our definition is given to fit the topic of this thesis.

[7]Note that the term *item* will be referred to as *product* and vice-versa indifferently throughout this thesis.

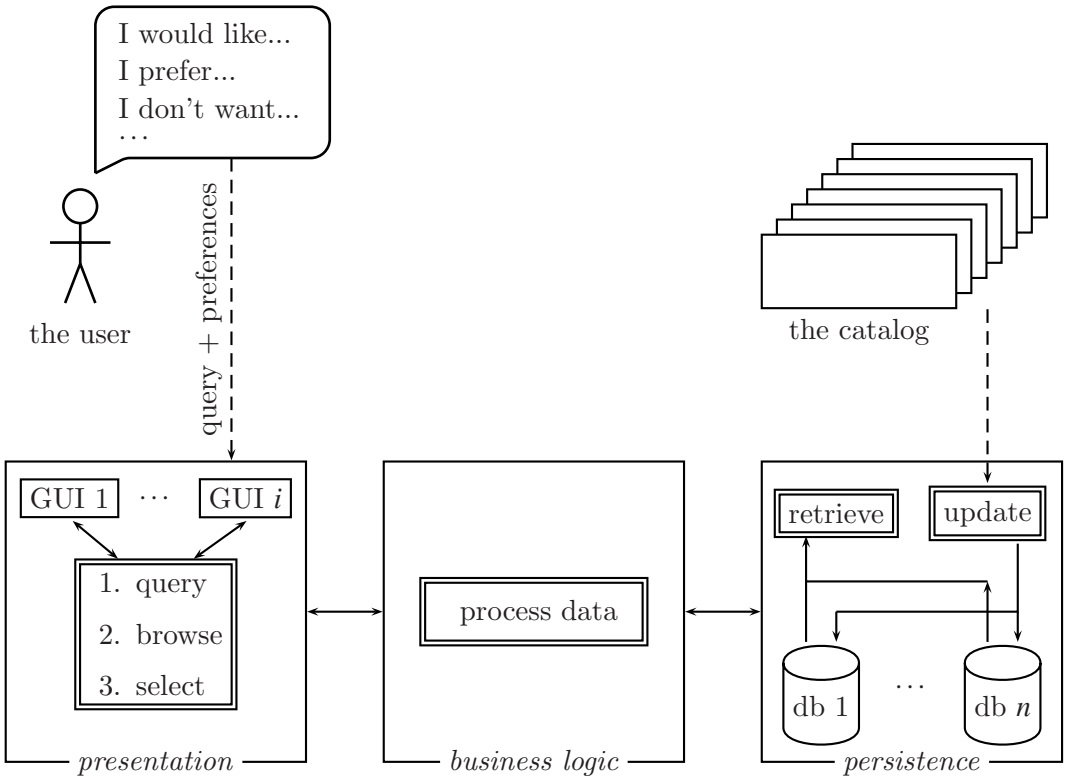


Figure 1.1: Basic components involved in electronic catalogs within a 3-tier architecture. The *presentation layer* provides graphical user interfaces between the user and the system. The *business logic layer* processes the data from the presentation layer to the persistence layer and vice-versa. The *persistence layer* is used to store the data of the catalog and provides mechanisms for updating and retrieving data.

- **Persistence layer** organizes and stores the products of the catalog. It can also contain other information, for instance user profiles which keep relevant information about users. In general, the persistence layer can be composed by several internal or external databases or other information systems. Basically, the persistence layer handles mechanisms for:
  - **Updating** data. Catalogs in general are dynamic since organizations change their offers by adding or removing items, adjusting prices and so on. Thus, these changes must be reflected accordingly in the persistence layer.
  - **Retrieving** data. This mechanism allows the system to retrieve information from the databases or other information systems and it can be implemented in a more or less sophisticated manner. The simplest method would be to directly access specific products by some attributes. More sophisticated methods exist for accessing several products that match a set of criteria, for example, in the case of a database, by means of SQL queries.
- **Business logic layer** processes data between the persistence layer and the presen-

tation layer, in both directions:

- **presentation layer → persistence layer**: this component takes the user requests and performs the appropriate queries to the *retrieving component* in the persistence layer in order to access the information he is looking for. This process could imply to split the user request into several queries for the different databases or information systems of the persistence layer. In the case the user expresses preferences about the product he would like, this process should adapt the queries, taking these preferences into account. Another approach for handling user's preferences would be to deal with them after getting the information, and in such case this process would take place in the following component (persistence layer → presentation layer).
- **persistence layer → presentation layer**: this component gets product information from the *retrieve* component of the persistence layer to serve user requests. For configurable products, this process should take care of *building* the products satisfying the compatibility rules. It could also rank the products by some criteria, for example according to the user preferences. In general, a mechanism could exist to transform the data from the persistence layer into an appropriate format for the presentation layer. In this way, the information can be shifted to different types of graphical user interfaces such as dedicated applications, web browsers, PDAs[8], mobile phones, and so forth.

Other mechanisms can be implemented in this layer. For example, some organizations are interested in producing statistics on the users' requests or profiles. This information could then be used to adapt their marketing strategy.

- **Presentation layer** is responsible for the interaction with the user by means of Graphical User Interfaces (GUI). Eventually, the presentation layer can contain several GUIs to support different interaction modes, *e.g.*, dedicated application, web browser, PDA, mobile phone, and so forth. It is able, in general, to provide mechanisms for handling the following steps:

  1. **Querying**: the user states his main request by specifying the type of product he is looking for. Preferences can also be expressed within this step. The query information is sent to the business logic layer.
  2. **Browsing**: the user browses through the product space provided by the business logic layer. By browsing the product space, the user is able to evaluate the different proposed alternatives.
  3. **Selecting**: the user selects the product he prefers the most, in order to conclude the purchase decision making process. The selected item is then sent to the business logic layer to be processed.

[8] Personal Digital Assistants.

When the 3-tier layers are implemented separately, *i.e.*, the code belonging to each layer is clearly separated, the resulting architecture is called a 3-tier architecture. Nowadays, it is widely accepted that the best architecture for information systems is a 3-tier architecture. The independence of the three tier layers ease the maintenance of such systems. Usually, the communication between the presentation layer and the business layer is done through Internet, but it is not a requirement.

Many different types of electronic catalogs are currently available, see for example the classification made by Spiller and Lohse in [227]. They identify the characteristics of Internet retail stores and compare a large set of representative real electronic catalogs in the industry.

1.2.3 Configuration

As a consequence of the increasing demand for customized products, the industry is offering more and more configurable products in contrast to the *old* model based on mass-production [186]. Faltings and Freuder in [60] wrote that Henry Ford reportedly said that his customers could have their cars in any color as long as it was black. Today, no car company could survive with such a narrow range of choices. The main advantages of applying configuration or customization to many industries can be summarized as follows:

1. **Better serve customers**. Nowadays, people are used to a much wider range of products available in the market than ever before, and thus, configurable products are required to better serve customers, by satisfying their specific needs and preferences.

2. **Differentiate from competitors**. In the current highly globalized market, organizations try to differentiate their offer from the competitors. Clearly, having products that can be customized can be a relevant factor in order to achieve such differentiation.

3. **Decrease production costs**. In many industries, the process of manufacturing separate components and then assembling them together, is much more cost effective than directly manufacturing different customized products. For example, in the car industry, many components are built separately and cars are then customized with such components in an *on-line* demand basis.

4. **Avoid wrong customer orders**. Wrong customer orders can be avoided by having in place configurator software. These order mistakes can be produced by the seller or the buyer. For example, the seller can promise delivery dates that cannot be accomplished because the vendor has made some mistake in the delivery planning. The customer can also make mistakes in his orders by, for instance, evaluating price/performance trade-offs in a wrong way or by misunderstanding his specific preferences.

5. **Better understand product engineering phases**. Manufacturing complex products is a cumbersome task that can be eased by applying configuration techniques.

In this way, organizations improve their knowledge about the product engineering process.

As a consequence of the aforementioned advantages, configuration task plays an important role in many commercial activities. In [78], Freuder wrote that according to Charles Carson, companies lose "2-3% of revenue in rework and penalty costs due to errors made in the initial product configuration"[9]. Moreover, in [78], several companies identified the usefulness of constraint technology[10] for the configuration problem applied to the business process:

- Lucent Technologies, Bruce Ambler: "Once the resources required and offered by each component are specified, the constraint-resolution process can select and balance the resources needed to satisfy a customer need".
- Triology Development Group, David Franke: "Configuration knowledge can significantly improve business-process efficiency in the obvious way, by reducing errors in customer orders".
- ILOG SA, Daniel Mailharro and Jean-François Puget: "Constraint programming lets us program the solution-search procedure and integrate domain-expert knowledge as search strategies for each decision of the configuration task. Constraint programming also lets us define optimization algorithms that answer the actual need for *intelligent* configurators to be able to minimize one or several criteria, such as the price of the built artifact or the number of components used".
- Concentra Corp., Bob Phillips: "The complexity of selling and servicing a wide range of products in a highly competitive global market requires that there be a controlling architecture for product knowledge throughout the enterprise. We can realize this controlling architecture through a combination of technologies (..., constraint technology) that will let everyone share product knowledge to improve customer satisfaction".

The configuration problem A configurable product is composed by several **components**. Each component can be selected from a set of **choices**. Choices for components can be combined together in predefined ways which are stipulated by **configuration rules**. The **configuration problem** can be informally defined as the task of selecting and arranging choices for a set of given components in a way that the configuration rules are satisfied. In general a configuration problem involves two tasks [204]:

1. Domain knowledge description: The knowledge for a configuration task comprises the type of components with their choices, and the configuration rules.
2. Problem specification: A configuration problem is specified by the components that must be present in a solution.

[9] PC AI, January-February 1996.
[10] Constraint satisfaction techniques are introduced later, in Chapter 3.

For example, consider the problem of configuring a personal computer (PC)[11]. The domain knowledge can be described by:

- the type of components with the description of their choices, for example:
  - CPU: Pentium 1.7, 1.8, 2.0, 2.2 GHz, Celeron 1.3, 1.4 GHz, $\cdots$,
  - RAM memory: 64MB, 128MB, $\cdots$,
  - motherboard: number of slots, memory type, $\cdots$,
  - graphic card: speed, resolution, $\cdots$,
  - monitor: resolution, size, $\cdots$,
  - keyboard: standard, wireless, country, $\cdots$,
  - mouse: standard, wireless, optic, $\cdots$,
  - printer: B&W, color, fax, photo, laser, $\cdots$,
  - scanner: paper, photo, film, $\cdots$,

  $\vdots$

- configuration rules between some type of components, for example:
  - CPU $\leftrightarrow$ RAM memory,
  - CPU $\leftrightarrow$ motherboard,
  - graphic card $\leftrightarrow$ monitor,

  $\vdots$

Following the above example, a configuration problem for a PC can be formulated as "I would like a PC with at least 256MB of RAM, very fast, without a keyboard or mouse, but with a color printer. My budget is around 2,000 CHF". Such a problem would imply to select a choice from each of the demanded components in the problem: a motherboard, a processor unit, RAM memory, a monitor, and a printer. Besides the configuration rules of the domain knowledge, the problem has to consider the configuration rules coming from the user request (256MB of RAM, fast processor unit and color printer). Many configurator software are publicly available on the web, see for example the web site of Dell[12].

A configuration problem can have several solutions which are called feasible or valid configurations. In the previous example, there could be many different PC configurations satisfying the configuration rules from the domain knowledge and from the user request.

[11] The purpose of this example is to illustrate the concepts related to configuration, but it is not a complete and rigorous example about configuring PCs.

[12] Dell, `http://www.dell.com`.

Approaches for the configuration problem Configuration problems have been studied for many years in the areas of operational research and artificial intelligence. In [205], Sabin and Weigel reviewed the main product configuration frameworks in the following way:

- **Ruled-based reasoning** or expert systems are based on *production rules* of type `if` *condition* `then` *consequence* that express

  1. the actions that must be performed to obtain a valid configuration, and
  2. when an action can appropriately occur in relation to other actions.

  The knowledge base are the facts that are known to be true together with the rules. Then, the solving procedure, called *inference procedure*, chains the rules in a forward or backward manner (respectively, forward chaining and backward chaining) taking into account the facts in the knowledge base. The knowledge base then is modified by new facts, hopefully containing a solution (valid configuration). The main drawback of this approach is that maintaining knowledge bases for complex domains is extremely difficult and cumbersome.

- **Model-based reasoning** appeared to overcome the maintenance difficulty of ruled-based reasoning systems:

  – **Logic-based approaches** are based on *description logic* (DL). DL models deal with three types of elements: *individuals* (objects), *concepts* (set of individuals) and *roles* (binary relations between individuals). The main inference mechanism is *subsumption* which is the decision whether one concept is more general than another. DL frameworks can support configuration tasks in two ways:

    1. to provide support to other configuration engines by means of efficient and organized taxonomy of objects, and
    2. solve the entire configuration problem.

    Entirely solving the configuration problem with DL implies to guide the user through some key questions to make initial selections. Then, through more questions to the user, the system guides the process to complete valid configurations.

  – **Resource-based approach** transforms the configuration problem into a producer consumer model, where abstract resources represent how the components interact (configuration rules). The goal is then formulated as finding a set of components that bring the overall set of resources in a balanced state, in which all demands are fulfilled. The solving method is a generic backtracking algorithm where the starting point is an initial configuration. At each step, the algorithm selects a resource which is not yet balanced and chooses another component that can be assigned to that resource. In a dead-end situation, it backtracks to the previous chosen resource.

– **Constraint-based approach** was the first attempt to define general-purpose configuration engines (see [173]). Configuration problems can be naturally described as *constraint satisfaction problems*[13] (CSP) where variables are the components of the product and constraints define the configuration rules. Because the mapping between functional roles and the set of components available is typically many-to-many, the configuration task is dynamic in nature. This observation yields to an extension to the CSPs into *dynamic* CSPs [172, 111]. Sabin and Freuder proposed in [204] a different constraint satisfaction framework for configuration called *composite constraint satisfaction*. In this framework, variables represent complete sub-problems. This allows an increased representational power (organization of components by means of aggregation and classification) and efficiency solving methods (specific CSP consistency algorithms).

  Constraint satisfaction techniques are used throughout this thesis and the suggested modeling and solving techniques for electronic catalogs are described in detail in Chapter 3 and Chapter 4. Furthermore, Appendix B reviews the main classical constraint satisfaction solving strategies.

- **Case-based reasoning** (CBR) is based on a base of cases which represents valid product configurations. This case base can be initially built by hand with valid configurations and be enlarged with new configurations, as long as the system finds more valid product configurations. The solving approach consists in identifying the most similar case to the one to be solved. Then, the system adapts this similar case in order to satisfy the specific configuration problem requirements. The idea behind CBR relies on the assumption that similar problems have similar solutions. The steps of the overall process are 1) input customer requirements, 2) retrieve a configuration of a similar problem from the case base, 3) adapt the case to the new problem, and 4) store the new configuration to the base case.

In [205], the reader can find many references to the literature with respect to the above described models for product configuration.

Configuration for electronic catalogs Clearly, when electronic catalogs offer configurable products, configuration techniques are needed. As it will be shown in Chapter 3 and Chapter 4, constraint satisfaction techniques are very well suited for modeling complex electronic catalogs.

1.2.4 Personalization

Personalization of information systems can be defined in numerous ways. From a marketing point of view, the Personalization Consortium[14] defined personalization as the combination

[13] Formal definition of CSP is given in Chapter 3.

[14] The Personalization Consortium is an international advocacy group formed to promote the development and use of responsible one-to-one marketing technology and practices on the World Wide Web. For more information, see `http://www.personalization.org`.

of information technology with marketing practices to provide:

- better services to the customer by anticipating needs,
- efficient and satisfactory interactions for both parties, and
- relationships that encourage the customer to return for subsequent purchases.

Actually, in a marketing sense, personalization is a common activity in traditional commerce. Human vendors are used to anticipate concrete customers' needs, and to adapt their behavior depending on specific customers.

Regarding electronic forms of commerce, it is believed that business activities are changing from the model where organizations search customers for their products to the model where organizations search products for their customers[15]. Richard Danzel[16] said that the key issue for a virtual shop is to constantly change to be adapted to every individual customer. In this direction, Jeff Bezos, CEO of Amazon, said "if I have 2 million customers on the Web, I should have 2 million stores on the Web".

Often, a distinction between customization and personalization is done[17]. Customization is more related to systems where the user can explicitly select certain options, *i.e.*, the user is active in the process. On the other hand, personalization is often referred to systems which *guess* the customer's needs by implicit information, *i.e.*, the user is more passive whilst the system is more active. Throughout this document, the term *personalization* will be preferred than the term *customization* and no distinction will be made among both terms.

From a technology point of view, personalization is about applying techniques to information systems for adapting interfaces and contents to particular users. Such personalization techniques can mainly use two different types of data: explicit and implicit. Explicit data comes from direct interactions with the user, for example, input forms, questionnaires, profile editors and so on. Implicit data comes, for example, from the user behavior when browsing through Internet. Techniques such as collaborative filtering and case-based reasoning, among others, are used to provide personalization to information systems.

Note that configuration can be seen as a technique for achieving personalization. Actually, an electronic catalog that is able to configure their products according to user's preferences is serving the customer in a personalized manner.

1.2.5 Recommender systems

Recommender systems[18] can be seen as information systems that use personalization techniques to *recommend* the right information or item, to the right user and at the right time.

[15] Don Peppers from Peppers and Rogers Group Consulting, paraphrased in [113].

[16] Amazon, `http://www.amazon.com`.

[17] Jakob Nielsen's Alertbox for October 4, 1998: Personalization is Over-Rated, `http://www.useit.com/alertbox/981004.html`.

[18] Also called personalized systems.

See [197] for a brief introduction on recommender systems. Following this informal definition, personalized electronic catalogs can be also seen as recommendation systems. Recommender systems can be implemented using numerous techniques. For example, the tutorial about recommender systems given by Jameson, Konstan and Riedl [124] enumerates the following AI techniques: case-based reasoning, content-based methods, demographically based methods, hybrid algorithms, utility-based methods, and knowledge-based methods.

Recommender systems arise naturally in the area of electronic commerce, where they can assist the user to find the product that fits his needs. This is especially relevant since customers are often faced with an overwhelming selection of items. Nowadays, recommender systems are deployed on many different sites, serving millions of customers. When applying recommender systems' techniques to electronic commerce, the following enhancements have been identified by Schafer *et al.* in [210]:

- **Finding the right product.** By means of recommender systems, users are assisted in their purchase decision making process. In this way, users are not directly faced with the problem of selecting the best product among a huge amount of available items.
- **Cross-selling.** Recommender systems can improve cross-selling by suggesting additional products to users. For example, in the checkout process, the system could suggest products related to the items which are already selected in the shopping basket of the user.
- **Loyalty.** Recommender systems offer an added-value with respect to the relationship between the seller organization and the buyer. This added-value can increase the loyalty of users to the system.

As described by Schafer *et al.* in [210], different types of recommendations are identified. Recommender systems use one (or a mix) of the following recommendation types:

Non-personalized recommendations are independent of each user, thus all users get exactly the same recommendations. They are usually made by rankings that other users have made. For example, Amazon[19] and E!Online[20] recommend products (books and movies respectively) upon what the other customers have said about them. Another example of non-personalization recommendation is eBay[21] where users rank other users.

Attribute-based recommendations or *preference-based item recommendations* are based on syntactic attributes of the products. Customers enter the desired properties of the products, and the system proposes the available items according to the customer preferences. For instance, the electronic commerce site of FNAC[22] is based on hierarchical categories of products from where customers can browse and select

[19] `http://www.amazon.com`.
[20] `http://movies.eonline.com`.
[21] `http://www.ebay.com`.
[22] `http://www.fnac.com`.

properties of the wished items. Stolze and Rjaibi explains attribute-based recommendation applied to electronic catalogs in [232]. They also describe an XML[23] framework for representing simple value functions that allow developers easily implement preference-based recommendation for e-catalogs. Another approach pointed out in [64] proposes to recommend products for non-expert customers in a needs-oriented way, instead of in a featured-oriented way.

Item-to-item correlation based recommender systems use a set of products the customer has expressed interest in for recommending other related products. For example, in Amazon, when a user has selected a book, other similar or related books are automatically suggested.

People-to-people correlation or *collaborative filtering* is based on the correlation between the customer and other customers that have already used the system. It is based on the principle that similar customers purchase similar products. For instance, Album Advisor from CDnow Online[24] infers customer opinions upon the user profile a customer sets up. Collaborative filtering has been successful in research, see for example the GroupLens project[25] described in [135]. O'Connor *et al.* in [179] describes a recommender system called PolyLens which recommends items for groups of users, rather than for individuals. In the domain of job finders, Bradley *et al.* [25] use case-based reasoning for recommending job positions to the users. In [112], the authors explore how to implement explanation interfaces of recommendations done by collaborative filtering engines. They argue that providing explanations about recommendations, the customer acceptance can significantly increase.

Sarwar *et al.* analyze in [208] different recommendation algorithms and evaluate them within the context of electronic commerce.

Our approach for personalizing electronic catalogs, from a user-system interaction point of view, is based on a **mixed-initiative system** and it is described in detail in Chapter 2. From a recommender systems point of view, our approach is based on **attribute-based recommendations**, see Chapter 3 and Chapter 4 for more details on this aspect.

1.3 Complex electronic catalogs

This thesis focuses on modeling and solving complex electronic catalogs in order to support the user in the purchase decision making process. Complex electronic catalogs have the following two characteristics:

- **Configuration**. Complex electronic catalogs contain items that result from a configuration process. A configurable product or service is not just an *atomic* item but a set of compatible components. *Configuration rules* define which components can be put together to form a *feasible* or *valid* product.

[23] XML stands for eXtended Markup Language.
[24] `http://www.cdnow.com`.
[25] `http://www.cs.umn.edu/Research/GroupLens`.

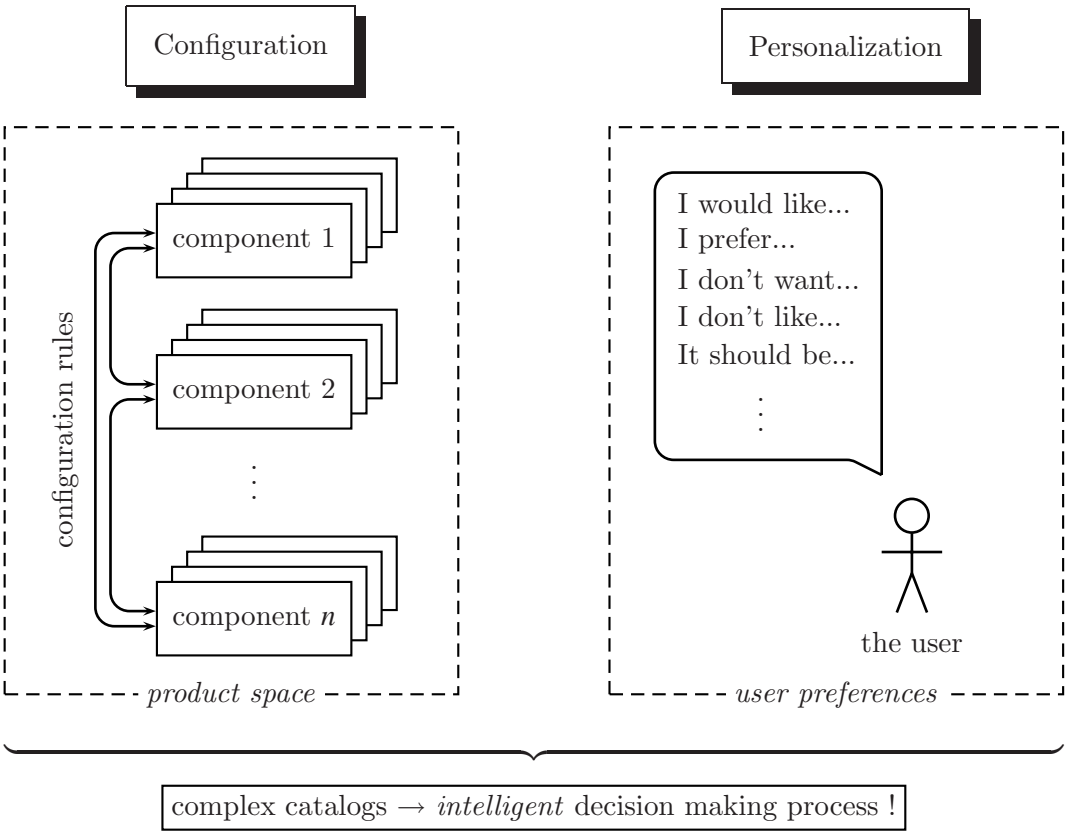


Figure 1.2: Complex electronic catalogs require to handle configuration and personalization in order to support the user in the selection decision making process.

- **Personalization**. A key issue in complex electronic catalogs is how to personalize the contents of the catalog. In this sense, complex electronic catalogs should consider different user profiles (user's preferences) being able to support him in the purchase decision making process.

In spite of the above characteristics of complexity, this work deals with catalogs which are *huge*, *i.e.*, that handle a huge set of items. One could imagine complex catalogs which only contain few items, and in such a case, the catalog can be modeled as a simple catalog where the products are precomputed off-line. On the other hand, however, modeling and solving large complex catalogs clearly require more sophisticated techniques in order to support the user in the purchase decision making process.

Figure 1.2 shows graphically the concepts of configuration and personalization applied to electronic catalogs.

1.3.1 Examples in the industry

Some relevant domains where complex catalogs arise are:

Travel industry was one of the earliest industries to go online, and is one of the most important areas of electronic commerce in terms of market volume [110]. Nowadays,

users can book flights together with hotels and cars easily on the web, see for example Travelocity[26], Expedia[27], or Orbitz[28]. However, these booking engines do not provide sufficiently intelligent tools to support the user. Most of the on-line booking engines use interfaces to GDSs[29] to retrieve the travel information according to specific user's needs (locations and dates). Then, the user can browse through all the options to evaluate the alternatives and choose the one that best fits his/her preferences. The most advanced booking engines allow the user to express some preferences about preferred airlines, time schedules or aircraft types. In such systems, a simple attribute-based filtering process ranks the options according to the user's preferences. For example, ITA Software[30] allows users to express preferences and rank the options by a specified attribute. In most cases, on-line booking engines present the available options to the user in the form of a flat list. Therefore, current on-line booking engines are far away from the quality of service that traditional travel agencies offer in terms of supporting the user in the purchase decision making process.

Insurance industry is present in numerous different services such as car insurances, life insurances and house insurances. Insurance policies are very complex because they are highly customizable. Very often, insurance companies offer a huge range of products, and in addition to that, they can be personalized with many options. Encoding and processing all the available options and insurance policies is not feasible using traditional software. In this area, much more intelligent tools than the ones used currently are required, in order to provide the necessary support for the user.

Financial industry has a lot of potential regarding complex catalogs. For example, one could imagine the management of stock exchange portfolios. Users of such systems would state their preferences on the type of industry to invest in, the investment risk, the period of the investment, and so on. Then, a intelligent process would recommend a concrete portfolio. Other domains in the financial industry where complex catalogs arise are: credits or saving plans.

Employment market is prominent in the Internet by means of electronic job banks, also called job finders[31]. The concept is based on the fact that there are people looking for jobs and companies looking for employees. Job banks allow companies and unemployed people to be connected, but again, the support these systems currently offer to users, companies and people looking for a job, could be improved with more sophisticated techniques. Usually, these systems only allow to browse job positions

[26] `http://www.travelocity.com`.

[27] `http://www.expedia.com`.

[28] `http://www.orbitz.com`. It is relevant to note that the technology behind Orbitz has been developed by ITASoftware, `http://www.itasoftware.com`, which uses advanced constraint satisfaction techniques.

[29] GDS stands for Global Distribution System. GDSs are information systems which distribute electronically travel information, such as fares or seat availability in flights.

[30] `http://www.itasoftware.com`.

[31] See for example, `http://www.jobfinder.ie`.

by some general criteria such as industry domain, or type of job position.

Logistic cargo industry is another complex industry where electronic catalogs could add value to the traditional environments. For instance, merchandise transport companies could offer their services to companies that need transportation services with specific needs. Clearly, many preferences and configuration rules appear to find out the best way of transporting merchandise. Examples of constraints are: specific means of transport regarding the type of merchandise, constraints for optimizing routes, or schedule constraints ensuring the delivery on time to the correct destination.

Telecommunication industry could offer electronic catalogs for bandwidth allocation. The users of these catalogs would be Internet providers or companies that need some quality of service for their networks. This is a very specialized industry, and much work has been done to attack this problem. For example, the reader is referred to the thesis of Christian Frei [75] for static bandwidth allocation and the thesis of Steven Willmott [263] for the dynamic case.

Electronic catalogs for the above industries offer configurable and intangible goods. Complex catalogs also exist for tangible goods, consider for example, home electronic goods such home theaters, complete photo equipments, or customized computers.

Chapter 6 describes how to model complex electronic catalogs with a concrete example on the travel industry. Furthermore, Chapter 7 shows a commercial application, called *reality*, for air travel planning which is based on the concepts of this thesis. Appendix A illustrates how *reality*works with a concrete scenario with the corresponding screenshots of the software.

1.4 Contributions

The main contributions of the thesis can be briefly summarized as follows.

Mixed-initiative system for electronic catalogs based on a conversational model where the user criticizes product proposals provided by the catalog, the goal being to converge to a reduced set of satisfactory solutions. This user interaction model is inspired by the traditional conversational model between the customer and the seller. Such mixed-initiative system is detailed in Chapter 2.

Modeling and searching strategies for complex electronic catalogs. An uniform framework for modeling electronic catalogs using constraint satisfaction problems is proposed in Chapter 3. Such constraint-based model supports configuration rules, user preferences and optimization constraints. Two different approaches to generate solutions (products) are analyzed. The first approach deals with linear combinations of constraint violations, while the second approach deals with the concept of Pareto-optimality. Solving strategies for both models are described in Chapter 4.

***SmartClient* architecture** faces the problem of scalability for complex electronic catalogs. *SmartClient* architecture is an architectural model that uses constraint satisfaction problems for representing solution spaces, instead of traditional models which represent solutions spaces by collections of single solutions. Different configurations of the *SmartClient* architecture are provided for different architecutal requirements. This software architecture is described in detail in Chapter 5.

Validation of the main thesis contributions is provided through Chapter 6 and Chapter 7. The validation of the main contributions of this work has been done by applying them to a concrete application domain. The chosen domain is the travel planning because its complexity and relevance in the electronic commerce area. Chapter 6 describes the problem of travel planning in detail, and Chapter 7 illustrates a commercial application for travel planning that uses the concepts provided in this thesis.

1.5 Overview of the dissertation

In Figure 1.3 the organization of this thesis is shown. This chapter has introduced the main basic concepts and challenges of this work which focuses on electronic catalogs. In Chapter 2 a model for personalizing electronic catalogs is proposed. This model suggests how to support the user in the purchase decision making process in complex electronic catalogs, *i.e.*, in catalogs for configurable products. After proposing an user interaction model to deal with catalogs, Chapter 3 proposes to encode complex catalogs by means of constraint-based techniques. Such a constraint-based model yields to the need of considering solving algorithms, described in Chapter 4, that allow to search for products in complex catalogs by taking into account the preferences of the user. In order to really exploit the ideas presented in previous chapters in a networked environment, a software architectural model is described in Chapter 5. Finally, the ideas of this thesis are illustrated in Chapter 6 describing the problem of planning travels. A commercial application that puts in practice the ideas of this work within the example of planning travels is showed in Chapter 7.
The contents of this thesis can be summarized as follows:

Chapter 1: Introduction introduces the topics of the thesis. It gives the needed notions about electronic commerce and electronic catalogs, and it introduces the complexity of e-catalogs for handling configuration and personalization. Some domains in the industry where complex catalogs are needed to support users are identified. Finally, the main contributions of the thesis are summarized.

Chapter 2: Personalization suggests a mixed-initiative system for electronic catalogs to support the user in the decision making process of choosing the product that best fits his needs and preferences. The model is based on the dialog that takes place in any traditional commerce conversation involving a seller organization and a customer.

Chapter 3: Modeling Electronic Catalogs gives a modeling framework for electronic catalogs based on constraint satisfaction techniques. It analyzes two different approaches: the quantitative approach and the qualitative constraint combination approach. The first approach is based on weighted constraint satisfaction problems and the second one deals with the Pareto optimality concept.

Chapter 4: Solving Electronic Catalogs explores different problem solving approaches, taking into account the modeling frameworks of Chapter 3 and the mixed-initiative based system proposed in Chapter 2.

Chapter 5: The *SmartClient* Architecture describes an architecture for electronic catalogs where the main processing is done on the client side. *SmartClient* architecture is designed for supporting personalization as is described in Chapter 2. The proposed architecture is analyzed and compared with standard client-server architectures for electronic catalogs.

Chapter 6: The Travel Planning Problem gives a complete description of an example of complex catalogs in the travel industry.

Chapter 7: A Commercial Application for Travel Planning: *reality* validates the techniques suggested throughout this dissertation. A commercial application for planning air travels which uses the techniques of this thesis, is described in detail.

Chapter 8: Conclusions provides a brief summary of the thesis work and its contributions. Limitations are also discussed and future research directions are proposed.

The appendixes are given to support the contents of this thesis:

Appendix A: Using *reality*: a scenario shows by means of a collection of screenshots how *reality*works. The purpose of this chapter is to illustrate graphically some of the concepts of this thesis applied to a commercial application for planning air trips.

Appendix B: Solving classical Constraint Satisfaction Problems reviews the main solving techniques for classical constraint satisfaction problems.

Appendix C: Extended Constraint Satisfaction Problems exposes the main extensions to the classical CSP model.

Appendix D: Java Constraint Library describes a library for solving classical CSPs. The library is written in Java and can be used to examine CSP techniques over the Internet.

This document ends with bibliography used and an index of the main key words of the dissertation.

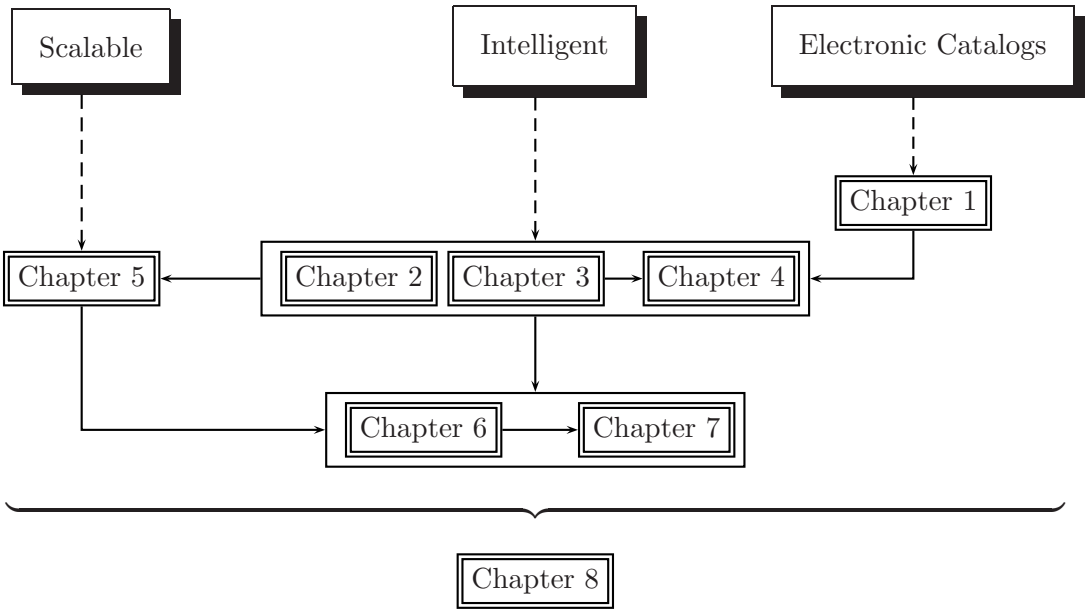


Figure 1.3: Outline of the thesis and dependencies of chapters.

Chapter 2

Personalization

The time to stop talking is when the other person nods his head affirmatively but says nothing.

Anonymous.

2.1 Introduction

Personalization can be defined as the capacity to adapt contents and interfaces of information systems to the particular user's needs. Actually, traditional commerce activities often incorporate more or less sophisticated capacities for personalization.

In traditional commerce, complex trading activities are carried out by means of conversations between the involved parties, namely: the seller and the buyer. For example, in the travel domain one could have the conversation illustrated in Figure 2.1. These trading dialogs are sequences of proposals from the buyer and critiques of such proposals made by the customer. Each time the customer criticizes some attributes of the proposed items, the seller tries to change or adapt his last proposals to better fit the customer particular needs. The goal of these trading dialogs is to reduce the product space and thus converge to a satisfactory small set of products as depicted in Figure 2.2. At the end, the user decides which is his best option among the products that result from the dialog. These kind of dialogs are especially relevant within complex domains, such as travel, insurance, banking and so forth. Without these trading conversations, few people could be able to decide about, for instance, optimal travel combinations, their best insurance policy, or a well-balanced investment portfolio. Customers are used to having support from experts (such as travel agents, insurance vendors or investment managers) in order to be assisted in the purchase decision making process.

From the customer's point of view, in the context of trading dialogs within complex domains, the following two observations can easily be made:

Observation 1: Initial rough idea about the product.
Very often, when people decide to purchase a good which is significantly complex, they

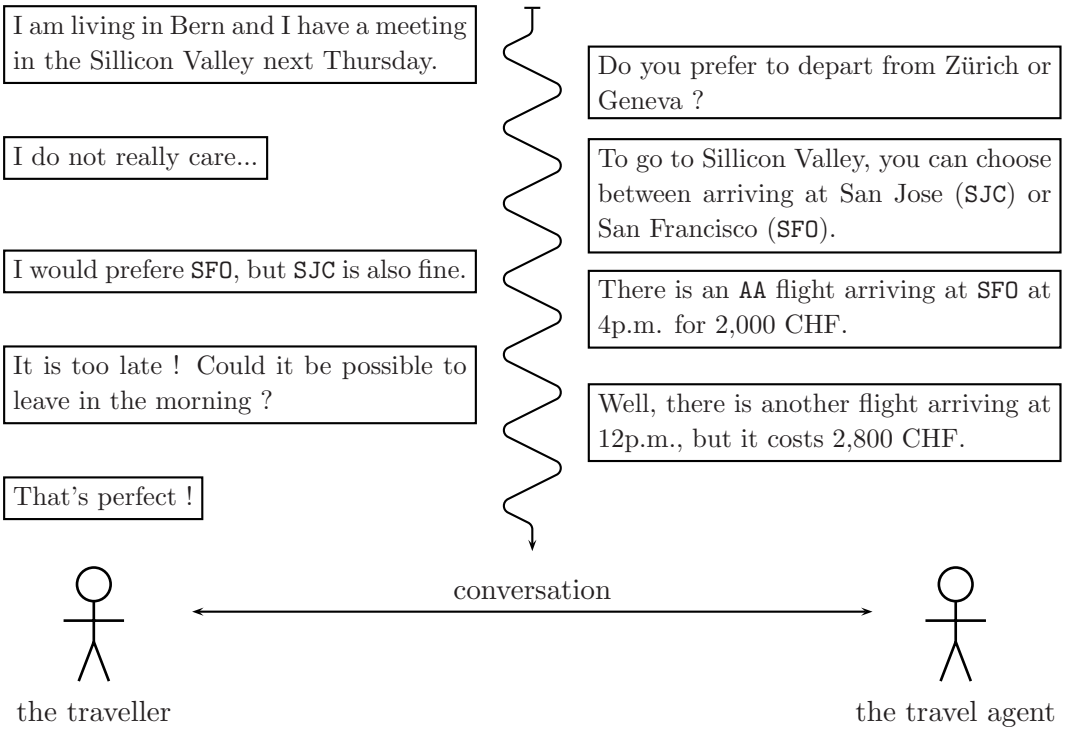


Figure 2.1: Example of an interaction between the buyer and the seller in the travel domain. The conversation supports the user to find out the flight that best fits his needs and preferences. Observation 1 and Observation 2 are illustrated in this dialog.

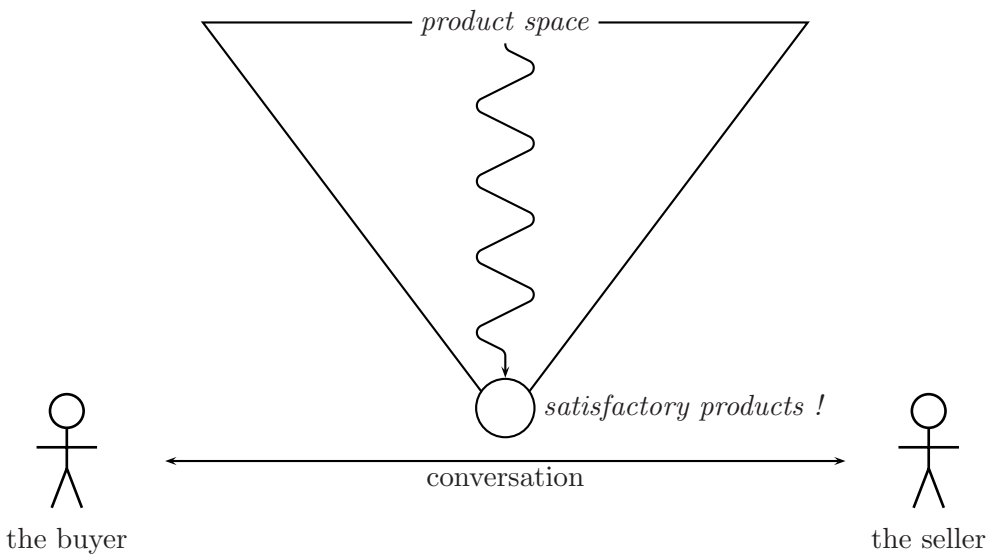


Figure 2.2: The goal of a trading conversation between the seller and the buyer is to narrow down the initial available product space to a reduced set of satisfactory products.

have only a very rough idea about their criteria or preferences. For example, in the travel

expertise

level of expertise	initial request
professional	`Sigma 70-200mm 2.8 EX`
amateur expert	covering 70-200mm and around 2.8 of aperture
amateur	large focal length and wide aperture
neophyte	pictures of animals

support

Figure 2.3: Example about the level of vagueness of initial requests for photo lenses to take animal pictures. The level of expertise of the customer influences the precision of the initial request. The customer's level of expertise indicates the customer's needed support in the purchase decision making process.

domain, one would specify to the travel agent "I would like to go to Barcelona for one day in the middle of next week" rather than "I would like to book the flights `IB4493 GVA→BCN` and `IB8936 BCN→GVA` on March 27th, in business class, at the fare of 1,479 CHF".

Another example of this observation, in the photo equipment domain, is that a customer would ask for a lenses for making pictures of wild animals rather than directly asking for a concrete product, for instance a `Sigma 70-200mm EX 2.8`. In general, the level of vagueness of the customer's initial request strongly depends on his level of expertise in the domain. Figure 2.3 shows this concept, in the aforementioned example about photo lenses for wild animal photography.

The justification of this observation can be done through the following question: "how can people concretely specify the product they are looking for if they are not really experts in the product domain and if they do not know the available offers for that product ?". The answer could be *with difficulty*, and it depends on the level of expertise of the customer[1].

Observation 2: Criteria are mainly discovered by reacting to samples.
As a consequence of Observation 1, people in general do not state their preferences up-front because initially they only have a rough idea of the product they would like. Hence, the question that precedes this observation is: "how and when people state their preferences about products in complex domains ?".

Usually, criteria about the product the customer would like to purchase are specified during the dialog with the seller. Furthermore, people express naturally their criteria reacting to concrete samples. For example, in Figure 2.1, the user reacts to the first concrete proposal of the travel agent by asking for an earlier flight. In this way, people are used to express their criteria incrementally as long as they react to product samples.

In some cases, expert customers know precisely what product they need, even in complex domains. In these situations, they do not need assistance from expert vendors. This

[1]This work focuses on intelligent tools for supporting the user in electronic shopping activities within complex domains. Obviously, really expert users do not need such assisting tools, since they can directly order concrete items.

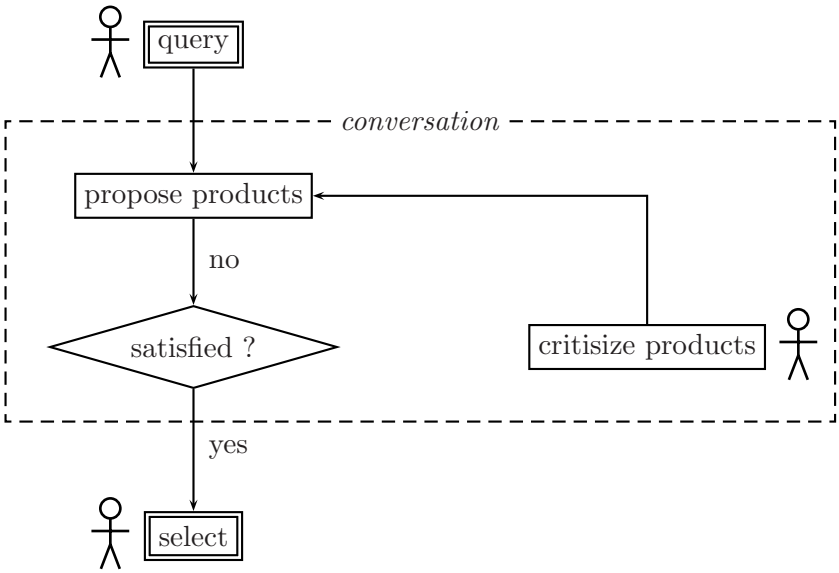


Figure 2.4: Conversational model for electronic catalogs. In the main query, the potential customer specifies his rough idea about the product. Then, a conversation takes place in order to converge to a satisfactory product. The conversation mainly consists in a loop where the buyer criticizes, and the seller proposes items according to the user's critiques.

situation can be translated in the area of electronic catalogs where expert customers would not need intelligent tools, which are the focus of this thesis. However, note that in some domains, even expert users need to be assisted because available products dynamically change. This remark can be easily illustrated in the travel industry, where available seats on flights, ticket fares or special offers constantly change.

This chapter suggests a *human-computer interaction* model for supporting the user in complex electronic catalogs. The model is based on the interactions used in traditional commerce relationships, more concretely on the aforementioned Observation 1 and Observation 2.

2.2 A mixed-initiative system for electronic catalogs

A mixed-initiative system [118] can be defined as a system where a software entity[2] and a human user collaborate to achieve a common goal. In electronic commerce, the collaboration takes place between the potential customer and the electronic catalog tool, the goal being to find the product that best fits the particular criteria and needs of the user.

[2] A *software entity* could also be called *software agent*, but in the remainder of the thesis the term software entity is preferred. This is because software agents are not directly related to the topics of this thesis and they are studied in a concrete field of computer science.

2.2.1 Conversational model

Our approach is based on a mixed-initiative collaboration between the user and the software entity [192] (Figure 2.4). This collaboration is done through the following steps:

1. **Main query:** According to Observation 1, the user has only a rough idea about the product. Therefore, at this first step only a initial vague query is entered into the system. Some preferences can also be entered at this step, but the key point is that they are not necessarily required.

2. **Conversation within a mixed-initiative system:** After entering the main query which specifies the product in a high level, the conversation between the user and the system begins. Such a conversation is a loop involving the two following steps:

   - **The reasoning engine** proposes solutions according to the user's criteria. The first time that the engine proposes solutions, user preferences are not available yet. Then, optimization criteria, which are built upon *common sense*, are used by the reasoning engine to propose *optimal* solutions. Chapter 3 and Chapter 4 describe the framework and the solving strategies respectively for such a reasoning engine based on constraint satisfaction techniques.
   - **The user** evaluates the alternatives by browsing the product space suggested by the reasoning engine. Properties of the proposed solutions can be criticized by means of preference elicitation Section 2.2.2.

   This interaction loop finishes once the user has found the product that really fits his criteria. As indicated in [222], if there is nothing to criticize, they have already found what they were looking for. However, note that in many cases, perfect solutions do not exist, and the customer is forced to make trade-offs among several unsatisfied criteria. In many situations, the criterion about price is faced to other preferences or qualities of items. For instance, it is quite evident that a car that costs 4 times more than another, will be more powerful, and/or safer and/or more luxurious.

3. **The selection process:** Once a satisfactory product has been encountered, the user can select it in order to proceed with the purchase process[3].

2.2.2 Preference elicitation

A typical user has many constraints that are not stated up front as discussed in Observation 1. The user becomes aware of these only when proposed solutions violate them as pointed out by Observation 2. Actually, preferences arise in a natural manner from the critique the user makes to concrete examples. In the following subsections, the way users elicitate their preferences in our mixed-initiative system, is explained.

[3]The processes taking place after the purchase decision has been made, are out of the scope of this thesis.

2.2.2.1 Preferences in any order at any time

In many electronic catalogs, users are forced to enter their criteria in a fixed order and in a specific sequence in time. Such fixed schemas do not allow the simulation of the metaphor of traditional commerce interactions. In our model, users are able to enter preferences in any order and at any time [193]. This can be done by using constraint satisfaction reasoning engines, as described in Chapter 3 and Chapter 4.

2.2.2.2 Posting preferences

Posting preferences on attributes (representing properties of the products) is the way users express their critiques to the examples suggested by the system. Such critiques can be done either *positively* or *negatively*:

- **Positive critiques** are done via preferences on attributes of the product, enforcing the reasoning engine to propose products satisfying such properties.

- **Negative critiques** are also done via preferences on attributes of the product, enforcing the reasoning engine to avoid proposals with such properties.

For instance, consider a user planning air travel with an electronic booking platform. The user receives, from the system, 5 different alternative itineraries. One alternative is operated by `Swiss` whilst the other 4 are operated by `Lufthansa`. If the user really prefers to fly with `Swiss`, he must be able to indicate that the 5 alternatives he is looking at should be operated by `Swiss`. This could be done by a positive critique on the airline attribute. Such a critique could be formulated textually as "the airlines should be `Swiss`". Now consider that, one of the suggested itineraries includes a flight with a `Boeing 737`, and he really does not like this aircraft. He must be able to negatively criticize this option by making a negative critique on the aircraft type attribute. This negative critique could be expressed in words as "I do not want to fly with `Boeing 737`".

It is worth remarking that positive and negative critiques are complementary and have different usages. Following the above example about airlines (4 alternatives operated by `Lufthansa` and one by `Swiss`), the following critiques are not equivalent:

1. positive critique: "I want to fly with `Swiss`"

2. negative critique: "I do not want to fly with `Lufthansa`"

The first critique would force the reasoning engine to prioritize itineraries operated by `Swiss`, whilst the second critique would force the reasoning engine to discard itineraries operated by `Lufthansa`.

It seems therefore natural that users who are experts in the domain of the product, make positive critiques, while neophyte users are more likely to make negative critiques.

2.2.2.3 Retracting and modifying preferences

Retracting and modifying preferences are the mechanisms that users can use to correct previous critiques. These mechanisms are especially important, because it could happen that a user enters a criterion that he thinks will be appropriate for narrowing down the product space, but the system suggests unexpected items. Since attributes of products are correlated[4], changing one property may involve changing other properties that were satisfactory to the user. Considering such situations, the system must be able to allow the user to retract or modify previously entered criteria.

2.2.2.4 Contextual preferences

Contextual preferences are preferences with a condition of the form:
`if` *condition* `then` *preference*. Contexts or conditions for contextual preferences are boolean functions on some attributes of a solution, indicating if the preference must be applied or not[5].

The reason for considering contextual preferences is that they naturally appear in many situations. For example, let us consider an example in the travel domain. The user in the example is living in Yverdon, and is able to depart either from Zürich or Geneva. Zürich is 2 hours by train from Yverdon, and Geneva 1 hour. He really does not want to wake-up before 8 a.m., thus our user would like to express this requirement, however, as he lives in Yverdon, the necessary time to get to Geneva airport or Zürich airport is different. Such preference can be easily and naturally expressed with two following contextual preferences:

1. `if departAirport = Geneve then departTime > 9:30 a.m.`, and
2. `if departAirport = Zürich then departTime > 10:30 a.m.`

2.2.3 Converging to satisfactory solutions

As in the case of traditional trading conversations, the goal of posting preferences is to converge to a small set of satisfactory products. As customers are used to traditional trading dialogs, they can easily use an *electronic* metaphor of such methodology. Converging to a reduced set of satisfactory solutions can be done in the two following different manners:

1. By **ranking** the set of solutions, the user has the perception that solutions are converging to the satisfactory ones. The purpose of this alternative is that, as long as the user posts constraints, the first ranked solutions become more and more satisfactory, while the last ranked solutions become less and less satisfactory. In

[4]Correlations among different properties of items are mainly expressed in our approach by configuration rules, user preferences and optimization constrains. See Chapter 3 and Chapter 4 for the whole framework dealing with configuration rules, user preferences and optimization criteria.

[5]Contextual preferences can be modeled as normal soft constraints (described in Chapter 3). Note that when the context of a preference does not apply, the constraint is completely satisfied. One could perfectly implement contextual constraints by considering the context in the valuation function of the constraint. Nevertheless, in some applications, it can be useful to implement contexts as real boolean functions to activate the associated constraints

other words, the quality (the ranking function) of the suggested solutions becomes more and more disperse as long as the number of the user's preferences increase.

2. By **reducing** the set of solutions, the user clearly has the perception that the solutions proposed by the system are converging to a hopefully satisfactory set of solutions.

One could also think about combining the two aforementioned approaches. In this manner, the set of shown solutions would be reduced and also ranked as long as the user express his criteria.

Solving techniques for converging to satisfactory solutions according to user's preferences are described in detail in Chapter 4. Concretely, in Section 4.4 the two ways of converging to satisfactory solutions (by ranking and by reducing) are discussed.

2.3 Requirements for online interaction

The mixed-initiative approach described above requires an online interaction between the user and the system. In this context, online interaction requires the capacity of the system to process user's requests in a short time. In this way, the traditional trading conversation can be really *simulated* through the system. For achieving such a quick online interaction, the following requirements must be fulfilled:

- **Reasoning engine** must be powerful enough to process the user's criteria quickly. For complex domains with configuration, this implies the use of *advanced* problem solving techniques. Chapter 3 and Chapter 4 suggest the use of constraint satisfaction techniques to deal with the problem of finding *optimal* products according to the user's particular preferences in a *reasonable* computation time.

- **Architecture** must allow the system simulate a conversation with the user. In order to achieve this goal, the accesses to the server should be minimized, and the processing must be done on the client side as much as possible. *SmartClient* is an architectural model that meets this requirement, especially when combined with constraint satisfaction reasoning. On top of that, *SmartClient* architectures allow to build scalable systems. This architecture is detailed in Chapter 5.

2.4 Related Work

Some work has been done by the research community about similar approaches to the one presented in this chapter. On the other hand, most of the current available electronic catalogs are based on user profiles. In the following, related work to our approach is highlighted and the major drawbacks of using user profiles are outlined.

2.4.1 Personalization by mixed-initiative systems

In human-computer interaction (HCI) area, some work has been done on conversational models where the user expresses criteria by criticizing proposed examples. For instance, see [153] for an example on the travel domain or [222] for a tool to find apartments. These approaches are similar to the one described in this chapter. Nevertheless, our mixed-initiative system is reinforced with the usage of constraint satisfaction techniques (Chapter 3 and Chapter 4) together with the *SmartClient* architecture (Chapter 5).

2.4.2 Personalization by user profiles

Most of the current electronic catalogs available through Internet propose interfaces to allow the user to enter his user profile when using the system for the first time. However, using user profiles implies certain important drawbacks, namely:

- **Criteria stated up-front**. It is difficult to state all the criteria up front, specially in complex domains (see Observation 1). Koenemann *et al.* [132] observed this aspect with a usability study with an interactive retrieval information system.

- **Criteria stated forever**. Criteria of a user profile is not applicable in the same way for different situations, because criteria change. Often, preferences are different for the same person for different purchases. For example, in the travel domain, the same user profile is not applicable in the case the user plans a business trip or a holiday trip. Sometimes criteria are different for the same person for different purchases. Stolze and Strobel [233] propose an e-commerce approach where a user might assume multiple shopping roles depending on the specific needs.

- **Privacy**. User profiles are stored in databases which are owned by private organizations. This fact yields to the issue of personal data privacy. For more details, see for example [139].

- **User effort**. In general, user profiles are acquired by means of online forms or questionnaires. From the users' point of view, many of these forms demand a considerable user effort and time.

In many electronic catalogs, recommendations are made taking into account the criteria in the user profile and the request the user expresses by means of a questionnaire. Then, according to the user profile and the specific request, the system returns a list (usually ranked) of the proposed items. If the user does not like any of the proposed products, he has to enter a new request and start the process from scratch, *i.e.*, no conversation exists between the electronic catalog and the customer.

Stolze and Ströbel [234] describes a framework for adaptive interviewing the end-user in order to build an utility-based decision tree to determine the set of products that match the customer's and seller's requirements. This approach can be seen as a dialog to drive the customer through the right products without static profile or questionaires, but a

dynamicaly created dialog. At the end of the interview, the products optimized according to the utility-based decision tree are proposed to the user.

General approaches to e-commerce buying processes have been described for example in [216] and [104].

2.5 Summary

A mixed-initiative system for personalization applied to electronic commerce has been described. Electronic catalogs following this mixed-initiative model can be seen as recommender systems[6] based on attribute-based recommendations. Effectively, in our model recommendations are made based on preferences that users express on syntactic attributes or properties of the items they are looking for.

The approach is based on the traditional way of trading with complex products, *i.e.*, through a conversation between the seller and the customer. This conversation is used as a metaphor for our mixed-initiative system. The needed steps for simulating traditional trading conversations in electronic commerce have been identified. The way users can criticize the items proposed by the system has also been described.

Finally, related research work has been pointed out, and the major drawbacks of most of the commercial electronic catalogs using user profiles have been outlined.

[6]See Section 1.2.5 for a brief introduction on recommender systems.

Chapter 3

Modeling Electronic Catalogs as CSP

Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.

Eugene C. Freuder
Constraints International Journal, April 1997

3.1 Introduction

Constraint Satisfaction technology allows us to model a large set of problems in an elegant way and solve them using intelligent backtracking-based algorithms. Basically, implementing a constraint-based system consists of two main steps:

1. **Modeling** the problem into a Constraint Satisfaction Problem (CSP), and
2. **Solving** the CSP by using one of the well-known CSP algorithms[1].

Simple catalogs can be modeled just as a list of possible products from which the user has to select the preferred option. This is reasonable if the number of possible products is small and the user has few of preferences about the product. In such situations, simple filtering information methods suffice to support the user to find the best option. Consider, for example, a catalog modeling the television stock of a web-based store. If the store proposes a dozen television models, the catalog can be approached by just showing to the user the list of offers. Moreover, by means of simple filtering mechanisms, the system could rank the items according to some criteria (*e.g.*, his preferences). User criteria can be used as hard filtering constraints (not allowing optimization) or as soft constraints. Stolze's approach [231], called product scoring catalog (PSC), consists on considering user's preferences as soft constraints for scoring products accordingly.

[1] CSPs can also be solved with stochastic methods as *genetic algorithm* [99], *simulated annealing* [130, 29] or *tabu search* [97, 98] which are not CSP dedicated algorithms.

However, when the items of the catalog are configurable[2] and a large set of possibilities exists (complex catalogs), a more powerful encoding technique is needed. In this chapter, the problem of modeling complex catalogs by using constraint satisfaction methods is addressed.

- **Simple catalogs → information filtering methods**
- **Complex catalogs → constraint satisfaction methods**

Constraint satisfaction technology has been shown to be very efficient and suitable for modeling configuration problems. Nevertheless, the classic concept of constraint satisfaction is not powerful enough to express user's preferences nor optimization constraints. For this reason, our model is an extension of the classical concept of CSP.

In the following, we define the main notions of the constraint satisfaction technology and describe how to model electronic catalogs with configuration rules, user's preferences and optimization criteria into CSPs. Solving strategies of constraint satisfaction for electronic catalogs are described in Chapter 4.

3.2 Constraint satisfaction problems (CSPs)

Constraint Satisfaction Problems (CSPs) [246, 134, 140, 175, 160] are ubiquitous in applications like configuration [204, 173, 111], planning [229, 183, 26, 142, 94], resource allocation [209, 32], scheduling [71, 72, 6, 191], timetabling [165, 164] and many others. A CSP is specified by a set of variables and a set of constraints among them. A solution to a CSP is a set of value assignments to all variables such that all constraints are satisfied. There can be either many, one or no solutions to a given problem. Basically, the main advantages of constraint programming can be summarized as follows:

- it offers a general framework for stating many real world problems in a succinct, elegant and compact way,
- the nature of the representation allows formal description of the problem as well as a declarative description of search heuristics, and
- there are numerous methods for resolving conflicts, visualizing solutions, and characterizing the solution space.

3.2.1 Notational conventions

Throughout this work, some notational conventions are assumed. The following notations refer to concepts that will be defined during this chapter.

- Sets and vectors are noted by using calligraphic capital letters $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$. The set of variables of a problem is noted $\mathcal{V}$, a subset of variables is noted $\mathcal{W}$, a set of domains is noted $\mathcal{D}$ and a set of constraints is noted $\mathcal{C}$. The number of variables, $|\mathcal{V}|$, is written n.

[2] *I.e.*, choices on options restrict other options by means of configuration rules.

- Variables are noted with an upper case letter, usually V, and V_i indicates the i-th variable of $\mathcal{V}$.
- Values are noted with the lower case letter v, and v_i indicates that $v_i \in D_j$ and it is the i-th value in the domain D_j (domains are enumerated).
- An assignment of a value v to a variable V is written $\{V \leftarrow v\}$. Value tuples for a subset of variables $\mathcal{W} = \{V_{s_1}, \ldots, V_{s_i}\} \subseteq \mathcal{V}$ are written $\langle v_{t_1}, \ldots, v_{t_i} \rangle$ and express a collection of variable assignments $\{\{V_{s_1} \leftarrow v_{t_1}\}, \ldots, \{V_{s_i} \leftarrow v_{t_i}\}\}$.
- Domains are noted with the upper case letter D, and D_i indicates the i-th domain of $\mathcal{D}$ which is the domain of V_i.
- Constraints are noted with the upper case letter C, and C_i indicates the i-th constraint of $\mathcal{C}$.
- Soft constraints have an associated valuation function. A valuation function for a constraint C_i is written φ_{C_i}. An assignment (partial or total) is usually written A, and a solution (partial or total) is written S. Valuation functions are also applied to assignments and solutions, and they are written respectively, $\varphi(A)$ and $\varphi(S)$.

inconvinient

3.2.2 Classical CSPs definitions

In the following, the basic notions related to finite and discrete Constraint Satisfaction Problems are described. In this section, constraints are all *hard* constraints (no preferences nor optimization criteria) as opposed to the discussion in Section 3.1. The concepts are illustrated with an example of a toy problem (the crossword puzzle problem). This section introduce the basis of our constrained-based model for electronic catalogs which is explained in Section 3.5.

Definition 3.1 (Constraint Satisfaction Problem (CSP)) A *CSP* $\mathcal{P}$ is a tuple $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, where:

- $\mathcal{V} = \{V_1, \ldots, V_n\}$ is the set of variables involved in $\mathcal{P}$,
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ is the set of domains[3] associated to variables, V_i has domain D_i, and
- $\mathcal{C} = \{C_1, \ldots, C_m\}$ is the set of constraints which must be satisfied for any solution of $\mathcal{P}$. A constraint C_i involving a set of variables $\mathcal{W}_i = \{V_{i_1}, \ldots, V_{i_j}\} \subseteq \mathcal{V}$ is defined by a tuple $(scope, def)$:

[3]Domains are normally specified by a explicitly enumerated finite set of discrete values. However, in some applications, domains can be very large and difficult to enumerate them. In theses cases, a function `next-value` will be used to enumerate the domain.

- the scope of a constraint, $scope(C_i)$, is the set of variables $\mathcal{W}_i$ involved in the constraint.
- $def(C_i)$ is a set of value tuples[4] for the set of variables $\mathcal{W}_i = \{V_{i_1}, \ldots, V_{i_j}\}$, $def(C_i) \subseteq D_{i_1} \times \cdots \times D_{i_j}$. $def(C_i)$ is called the *definition* of the constraint and denotes the value tuples that satisfy C_i.

The arity of a constraint C is the size of its scope. Unary constraints only affect one variable, and the scope of binary constraints equals to 2. If the size of the scope of a constraint is higher than 2, the constraint is generally called a n-ary constraint.

If the constraints of the CSP are binary and unary, the CSP is called binary (non-binary or n-ary otherwise).

Definition 3.2 (Size of a CSP) The size of the CSP is the number of variables in the problem, and it is usually denoted by $n = |\mathcal{V}|$.

Indeed, modeling a problem into a CSP consists of identifying:

- the **variables** of the problem (problem choices) with their **domains**, and
- the **constraints** among variables. Constraints express what combinations of values are valid to compose a correct solution to the problem.

A solution to a CSP is an assignment to all variables in the problem, such that all the constraints are satisfied. In the following the definitions of partial and total assignment are given:

Definition 3.3 (Assignment or Compound Label) An *assignment* or *i-compound label* A on a set of variables $\mathcal{W}_A = \{V_{A_1}, \ldots, V_{A_i}\} \subseteq \mathcal{V}$ is expressed by the tuple (var, val), where $var(A)$ is the set of variables $\mathcal{W}_A$, and $val(A)$ is a value tuple $\langle v_{t_1}, \ldots, v_{t_i} \rangle$, $v_{t_k} \in D_{A_k}$. The value v_{t_k} corresponding to the variable V_{A_k} in A is denoted by $val(A,k)$. In a similar way, the variable V_{A_k}, to which the value v_{t_k} is assigned, is denoted by $var(A,k)$. A *partial assignment* of a CSP is an i-compound label where $i < n$ and a *total assignment* is a n-compound label that includes all the variables of the problem. Note that an assignment A over $\mathcal{W}$ can be expressed as a constraint C with $scope(C) = \mathcal{W}$ and $def(C) = \{val(A)\}$.

An assignment can be projected over a subset of its variables:

Definition 3.4 (Projection of an assignment over a set of variables) Given two sets of variables $\mathcal{W}_i = \{W_{i_1}, \ldots, W_{i_k}\}$, $\mathcal{W}_j = \{W_{j_1}, \ldots, W_{j_m}\}$, $\mathcal{W}_j \subseteq \mathcal{W}_i \subseteq \mathcal{V}$ and an assignment A over $\mathcal{W}_i$ with $val(A) = \langle v_{t_1}, \ldots, v_{t_k} \rangle$, a *projection of the assignment A over $\mathcal{W}_j$*, written $A \downarrow_{\mathcal{W}_j}^{\mathcal{W}_i}$, is an assignment A' with:

- $var(A') = \mathcal{W}_i \cap \mathcal{W}_j$, and
- $val(A') = \langle v_{A'_1}, \ldots, v_{A'_m} \rangle$ where $v_{A'_r} = v_{t_s}$ if $W_{j_r} = W_{i_s}$.

[4] A constraint C_i can also be stated in an implicit way, where a predicate $P(v_{i_1}, \ldots, v_{i_j})$, $v_{i_k} \in D_{i_k}$ will be true if the values are allowed, false otherwise.

Solution and partial solution (or consistent assignment) to a CSP are analog concepts to total and partial assignments but satisfying all the constraints of the problem. The set of all the solutions to the problem is called the solution space of the problem. Formally, these concepts are defined as follows:

Definition 3.5 (Solution, Partial Solution and Solution Space) A *solution* or *total consistent assignment* of a CSP is a total assignment (n-compound label) in such a way that all the constraints of the problem are satisfied. In an analytical form, a solution of a CSP is an assignment S with $var(S) = \mathcal{V}$ and $val(S) = \langle v_1, \ldots, v_n \rangle$ such that $\forall c \in \mathcal{C}, val(S \downarrow^{\mathcal{V}}_{scope(c)}) \in def(c)$. A *partial solution* or *partial consistent assignment* of a CSP is a partial assignment (k-compound label, where $k < n$) in such a way that all the constraints of the problem are satisfied. The *solution space* of a CSP is the set of all the solutions to the CSP.

A CSP is often graphically represented by a graph. This representation allows us to easily take a look at the problem topology and it is called the constraint graph associated to the problem.

Definition 3.6 (Constraint Graph) A *constraint graph* (also called *constraint network*) is a graphical representation associated to a CSP. Nodes of the graph stand for the variables of the problem, and constraints are represented by edges that link the implied variables. The labels of the edges define the constraints and the labels of the nodes represent the domain of the variables.

The search space of a CSP is all the possible n-compound labels in the CSP and the associated search tree is its graphical representation. Both concepts can be defined as follows:

Definition 3.7 (Search Space) The *search space* of a CSP $\mathcal{P}$ is the set of all the possible assignments to all variables, *i.e.*, the cartesian product of all the variable domains. Therefore, the *size of the search space* is the product of the size of each domain:

$$\text{search space} = \{D_1 \times \cdots \times D_n\}, \qquad |\text{search space}| = \prod_{1 \leq i \leq n} |D_i|$$

Definition 3.8 (Search Tree) The search space associated to a CSP can be represented as a *search tree*. A search tree for a CSP requires an order of the problem variables $V_{i_1}, \ldots, V_{i_n}$. The root node stands for the *empty assignment*. The level k of the tree involves the variable V_{i_k}. At the level k, the tree has one node per value in D_{i_k}. A path in the search tree from the root to a node at the k level can be seen as the (partial) assignment implying variables $V_{i_1}, \ldots, V_{i_k}$ taking values represented by the nodes of the path. Paths from the root to the leaves of the search tree are total assignments and represent the whole search space.

In the following, the problem of a crossword puzzle is proposed to illustrate the basic concepts related to classical CSPs.

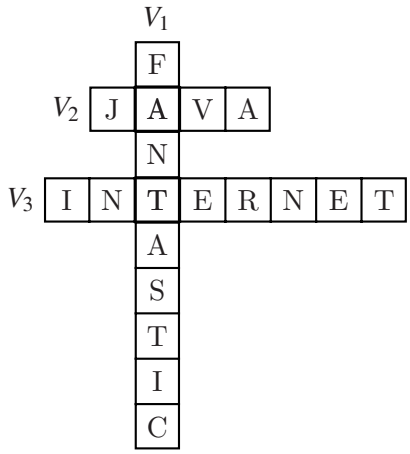


Figure 3.1: The crossword puzzle structure of Example 3.1 with a solution.

Example 3.1: A crossword puzzle.
The crossword puzzle problem is just a toy example to illustrate the formalization process of a problem into a CSP. The problem consists of placing words of a dictionary in a given puzzle satisfying certain constraints about the size of the words and the puzzle structure. The structure of the crossword puzzle is a set of columns and rows with overlapping positions. The rows and columns in the crossword puzzle are represented by the variables of the problem. The domains of the variables are the words of the dictionary. Unary constraints guarantee that words fit into the row or column. Binary constraints make sure that the letters of the overlapping positions are the same. Let us imagine that the dictionary is composed by $v_1 =$ `fantastic`, $v_2 =$ `java`, $v_3 =$ `internet`, $v_4 =$ `hello`, $v_5 =$ `constraint` and $v_6 =$ `insatisfiable`; and the crossword puzzle structure is defined by two rows and one column of size 4, 8, 9 respectively. Variable V_1 represents the column of size 9. Variables V_2 and V_3 represent the rows of size 4 and 8 respectively. Figure 3.1 shows the puzzle structure of the example. In order to express the constraints of the problem we need some notations and functions:

- $\mathcal{V} = \{V_1, V_2, V_3\}$, and $D_1 = D_2 = D_3 = \{v_1, v_2, v_3, v_4, v_5, v_6\}$.
- $v_{i,k}$ represents the value k in the domain for the variable V_i.
- $size(v_{i,k})$ gives the size of the word represented by the value $v_{i,k}$.
- $pos(v_{i,k}, p)$ indicates the letter in the position p of the word represented by the value $v_{i,k}$.

From the puzzle structure (Figure 3.1), the following constraints can be easily identified:

- Each variable has a unary constraint. In our example: $size(v_{1,k}) = 9$, $size(v_{2,l}) = 4$ and $size(v_{3,m}) = 8$.

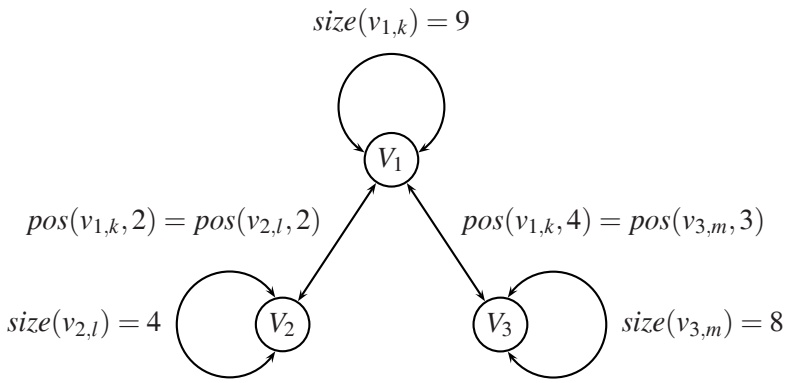


Figure 3.2: The CSP graph representing the crossword puzzle of Example 3.1.

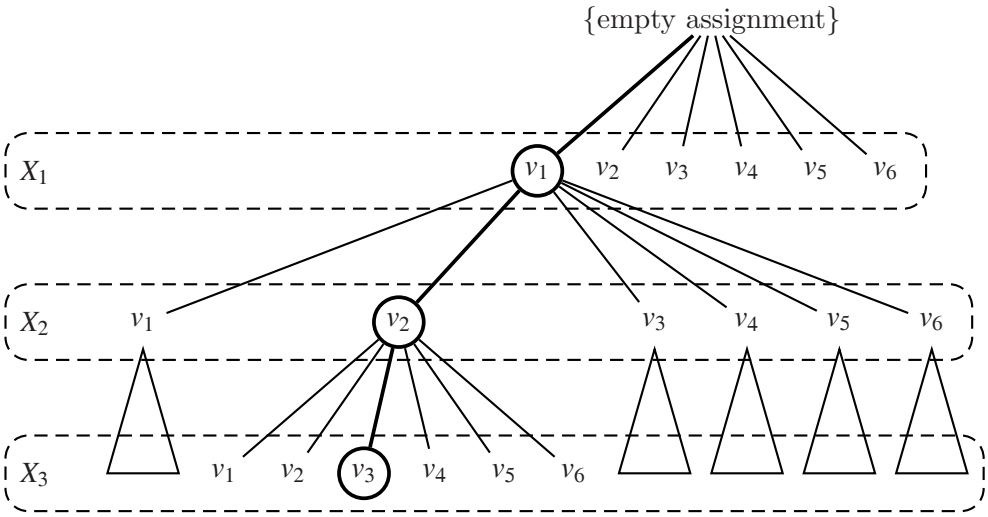


Figure 3.3: The search tree associated to the search space of the Example 3.1. For simplicity, only the branches containing a solution are expanded. The search tree of this example has 216 (6^3) leaves. Leaves represent all the possible total assignments. The nodes being part of the solution are displayed in bold.

- For each overlapping position in the crossword puzzle there is a binary constraint. In our example: $pos(v_{1,k},2) = pos(v_{2,l},2)$ and $pos(v_{1,k},4) = pos(v_{3,m},3)$.

The words placed in the puzzle shown in Figure 3.1 compose the only solution to the problem, *i.e.*, all the other combinations violate some of the constraints.

Figure 3.2 illustrates the concept of a CSP graph. Figure 3.3 shows part of search tree of the crossword puzzle example, where the leaves of the tree represent all the possible n-compound labels of the problem.

Note that CSPs are useful to model combinatorial problems. For instance, our toy example which is a very small problem has a search space of size 216. If we consider a more serious crossword puzzle with 15 columns and 15 rows and a dictionary of 200 words, the size of the search space will be $200^{15+15} \approx 10^{68}$! Several techniques have been developed

to efficiently solve combinatorial problems with a large search space. Appendix B reviews the most relevant solving techniques for classical CSPs.

3.2.3 Extending classical CSPs

Often one needs more powerful frameworks to model real-life scenarios. Classical CSP framework admits constraints which are *hard* and *crisp*. But, in many applications, constraints as defined in the classical CSP model are not enough. For example, consider constraints modeling some cost functions where the goal is to minimize such costs. This is the case when modeling user's preferences where the associated constraints must be satisfied as much as possible but they are not completely mandatory.

In the literature, different terminology is used to refer to different types of constraints. In the following, the terminology that will be used in the rest of the work is stated. Basically, one can classify the type of a constraint depending on two characteristics:

- Level of necessity:
  - **Hard** constraints which *must* be satisfied. These constraints are mandatory, they must be satisfied to form a solution. An assignment that violates a hard constraint is an inconsistent assignment.
  - **Soft** constraints which *might* be violated if necessary. When it is not possible to satisfy all the soft constraints to form a solution, soft constraint violations are admitted.
- Degree of satisfaction:
  - **Crisp** constraints which can be either totally satisfied or totally violated. They do not accept any degree of violation.
  - **Flexible**[5] constraints which can be satisfied (or violated) at a certain degree.

According to the above classification, three different types of constraints are identified[6]. An example of each type of constraint is given (for more examples of constraints see Section 3.9).

Hard and Crisp constraints are classical constraints, formally defined in Section 3.2.2. For example, modeling a catalog for a photo equipment with a camera body and a lens, requires the consideration of brand compatibility between both components. This requirement is an example of configuration constraint. If the brand of the camera body is not compatible with the brand of the lenses, the configuration constraint will be violated. If the brands of both components are compatible, the constraint will be satisfied.

[5]Flexible constraints are also called fuzzy constraints.

[6]It makes no sense to consider hard and flexible constraints since a violated hard constraint makes an assignment inconsistent, no matter the degree of its violation.

Soft and Crisp constraints can be violated and only accept one degree of violation. For instance, if the user prefers `Nikon` for the camera body, a `Canon` camera body completely violates the preference constraint. But, an assignment with the `Canon` camera body is still admitted as a potential solution, since even if the user preference is not respected at all, the assignment could make sense (*e.g.*, in the case that no `Nikon` camera body exist in the catalog, or the `Canon` camera body better satisfies other user preferences).

Soft and Flexible constraints can be violated and accept several degrees of violation. Usually, these constraints are modeled with a valuation function. Let us imagine that the user states a maximum price for a camera body of 2,000 CHF (preference constraint). A camera body that costs 2,100 CHF would be preferred than a camera body that costs 3,000 CHF, even if both violate the preference constraint. For instance, the first camera body would violate the constraint with a penalty of 100 and the second camera body would violate the constraint with a penalty of 1,000. Of course, a camera body that costs less than 2,000 CHF would completely satisfy the constraint, *i.e.*, there would not be any penalty.

Several extensions to the classical CSP model have been developed in order to model and solve problems that cannot be approached using standard CSPs:

Partial CSP [79] mainly deals with problems that are overconstrained, *i.e.*, problems that do not accept any solution. Another use of partial constraint satisfaction is to solve problems where finding a solution that respects all the constraints is too complex. In such cases, it is of interest to find solutions that satisfy the constraints as much as possible. MAX-CSP or *maximal* CSP is a general framework for modeling and solving partial CSPs. Larrosa describes algorithms for MAX-CSP and their evaluations in [148]. MAX-CSPs deal with CSPs with *soft* and *crisp* constraints.

Soft CSP is the generic term for CSPs that deal with constraints that are not *crisp* nor *hard*. Many different instances derive from soft CSPs:

- In Fuzzy CSPs [198, 55, 201] each tuple of values in the constraints has an associated preference level (also called valuation) from 0 to 1, where 1 is the best value and 0 the worst. A solution to a Fuzzy CSP is then a total assignment that maximizes the minimum constraint valuation in the assignment. Thus, a solution is a total assignment with the best (maximum) least (minimum) preferred constraint violated.
- Weighted CSPs associate a weight, cost or penalty to value tuples in constraints. A solution to a weighted CSP is then a total assignment that minimizes the sum of the costs of constraints. Weighted CSPs can be seen as a generalization of the MAX-CSP in the sense that a solution to a weighted CSPs minimizes not only the number of violated constraints but the sum of their costs.
- In Probabilistic CSPs [62] (Prob-CSPs), each constraint has an associated probability $p(c)$ to be part of the *real* problem. This formalism allows us to model

problems which are not completely known. A solution to a Prob-CSP can be defined in two different ways: a) a total assignment that maximizes the sum of the probabilities of the constraints, or b) a total assignment that maximizes the product of $(1-p(c))$ for each violated constraint.

- Possibilistic CSPs [212] associate to each constraint a possibilistic distribution among its value tuples. A solution to a possibilistic CSP is then a total assignment which minimizes the maximum valuation among the constraints. Possibilistic CSPs can be seen as the dual problem of fuzzy CSPs in the sense that the first uses a min-max schema and the second uses a max-min schema.
- Lexicographic CSPs [63, 56] deal with the problem of generating solutions in a too coarse way (drowning effect) in problems with min-max or max-min schemas. The drowning effect comes from the fact that only the constraint which is the most violated is taken into consideration. Thus, this approach can be used either in fuzzy CSPs or possibilistic CSPs. It consists of discriminating solutions with the same max/min constraint valuation by considering vectors of valuations ordered in a lexicographic manner. Therefore, two solutions to a fuzzy CSP (or possibilistic CSP) having the same minimum (or maximum) constraint valuation, can be differentiated by considering the second minimum (or maximum) constraint valuation. If the second most violated constraints are also violated to the same extent, the third will be then taken into consideration, and so on.

All these instances can be expressed by two general CSP frameworks, namely Valued CSPs [215] and Semiring-based [17] CSPs.

Constraints hierarchies [23, 74, 200] were introduced to solve over-constrained problems by associating degrees of importance or strengths to constraints in a hierarchical structure. The best solutions are those which satisfy all constraints up to a maximal level in the hierarchy.

Appendix C reviews in more details the most important extensions to the classical constraint satisfaction framework.

Our approach to model configurable catalogs is based on soft constraint satisfaction problems, and more concretely on weighted CSPs. This framework allows us to easily express in a uniform model all the constraints involved in configurable electronic catalogs with preferences and optimization constraints.

3.3 Encoding configurable electronic catalogs as CSPs

As stated before, encoding a problem as a constraint satisfaction problem implies to identify the variables and the constraints among them. In the following, we describe the variables and constraints for modeling any configurable catalog. At the end of the section, the interpretation of solution spaces of constraint satisfaction problems that model configurable catalogs are given.

3.3.1 Variables and values

Variables in a CSP that models a configurable catalog represent the components of the configurable product. Indeed, variables can also be called *choices*. For each choice there are several options which are the values of the variable.

The first step to model a configurable catalog as a CSP is to decide which are the variables. Then, the values of such variables can be identified easily. Values are the options the user has for each component of the product. Next step in the modeling process is to identify the constraints related to the catalog.

3.3.2 Constraints, preferences and optimization

By using classical CSPs, all the constraints involved in a product catalog can be modeled. However, as pointed out above, crisp and hard constraints are not always powerful enough to model real-life scenarios, and that is exactly the case when dealing with preferences and optimization criteria. Three different types of constraints in configurable catalogs are usually present: configuration constraints, user's preferences and optimization constraints.

Configuration constraints are related to the configuration task and define the compatibility among the different components that have to be composed to form a valid product of the catalog. These constraints are needed in the case that products can be only configured respecting certain compatibility rules. Configuration constraints cannot be violated, thus they are hard constraints. They are also crisp in the sense that configuration rules do not usually accept different degrees of satisfaction. Configuration constraints guarantee the feasibility and the correctness of a product (solution) to a catalog (problem).

User's preferences are used to take into consideration the user's specific needs about the product and are modeled by means of soft constraints. Tuples in these constraints have an associated penalty or cost indicating the degree of violation. Moreover, preferences can be stated by associating a fixed penalty or a function penalty. Preferences with a fixed penalty are crisp in the sense that they can be violated with the penalty or not violated at all. On the other hand, preferences with a function penalty are violated at some degree.

Often it is also necessary to have a context for a preference in order to express that the preference must only be applied in values where certain conditions applies. These preferences are called conditional preferences or contextual preferences.

Optimization constraints allow us to express criteria that can be assumed valid for every user regardless of his specific preferences. Optimization constraints determine the quality or optimality of a solution. These criteria are usually defined with a penalty function, the goal being to rank solutions that violate the user's preferences at the same level of satisfaction. Optimization constraints are clearly soft and can be either crisp or fuzzy.

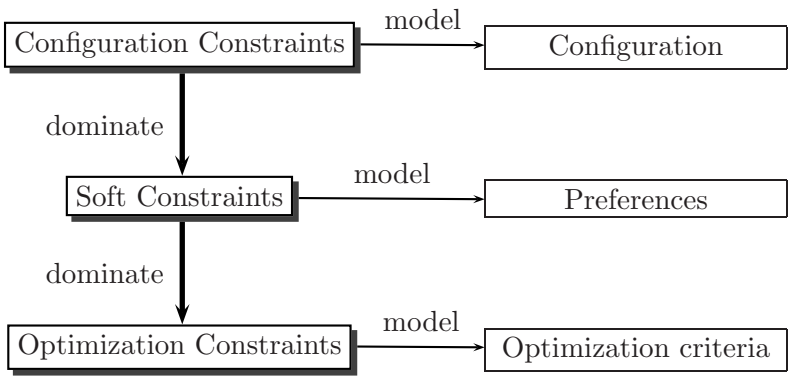


Figure 3.4: The constraint's hierarchy related to any electronic configurable catalog. Hard (configuration) constraints are more important than soft constraints which are more important than optimization constraints. A constraint type *dominates* another constraint type if constraints belonging to the first type have more important valuations than the second type.

Typically, user's preferences are always less important than constraints about the configuration task; and optimization constraints are less important than the preferences of the user (see Figure 3.4). Configuration constraints are hard constraints and preferences and optimization criteria are modeled with soft constraints.

It is worth to note that the distinction between preference or optimization constraints and configuration constraints is that the latter are mandatory. On the other hand, constraints for preferences or optimization criteria are basically expressed by the same type of constraints.

Example 3.2 is used to better explain the characteristics of the three types of constraints involved in a configurable catalog.

Example 3.2: A flight ticket from Geneva to Barcelona.
Let us imagine a simple catalog modeling the problem of finding flights for a round trip from Geneva (GVA) to Barcelona (BCN). Variable V_1 models the flights from GVA to BCN and variable V_2 models the flights from BCN to GVA. Therefore, domain D_1 associated to V_1 consists of all the possible flights from GVA to BCN and domain D_2 associated to V_2 consists of all the possible flights from BCN to GVA. The following constraints can be then stated:

- Configuration constraint: A binary constraint between V_1 and V_2 will make sure that the flight of V_1 arrives *before* the flight of V_2 departs. Without this constraint, there could be infeasible combinations of flights in a solution. This constraint guarantees that the combination of flights makes really sense.

- User's preference: For example, the user prefers flights departing before 10 a.m. Two possibilities exist for modeling such preference:

  – Fixed penalty for tuples in the constraint would be enough to discriminate

against flights which do not satisfy the preference. However, the degree of satisfaction of a flight departing at 10:10 a.m. would be the same as a flight departing at 4 p.m., *i.e.*, both flights completely violate the preference.

– Penalty functions for tuples allow us to have different degrees of satisfaction for flights in respect to the preference. For example, given a flight `f`, a function like:

$$\texttt{penalty(f)} \rightarrow \begin{cases} 0 & \texttt{if f.depTime} \leq \texttt{10 a.m.} \\ (\texttt{f.depTime} - \texttt{10 a.m.}) & \texttt{otherwise} \end{cases}$$

would be much more convenient for modeling the preference because it discriminates more accurately between the flights that violate the preference.

- Optimization criteria: For example, it can be assumed that, in general, people prefer short flights. In that sense, an optimization constraint prioritizing flights with a shorter flying time can be added to the CSP. Thus, there would be a unary optimization constraint for each variable with, for example, a penalty function `penalty(f) = f.flyingTime`.

Moreover, the preferences of a user are always ranked as being equally important. Actually, people tend to rank their preferences with respect to their degrees of importance. For example, consider a businessman planning to travel from Geneva to New York for a meeting at 4 p.m. He must arrive in New York before 4 p.m. and he prefers `Swiss` to any other airline. He could argue that his preference about the schedule is much more important than his airline preference because the first is a requirement to arrive on time for his meeting, and the second is more a personal preference. In a similar way, it is also convenient to be able to model optimization criteria using different degrees of priorities. In our model, degrees of importance for soft constraints, both user's preferences and optimization criteria, are defined by means of a constraint weight vector which is formally defined in Definition 3.13, Section 4.2.

Formal definitions for modeling constraints related to catalogs are given in Section 3.5.

3.3.3 Solution space

The solution space of an electronic configurable catalog represents the set of products satisfying the configuration constraints. In terms of constraint satisfaction, the solution space is the subset of the search space where the n-compound labels do not violate any hard constraint (*i.e.*, the set of feasible solutions). If there is no solution satisfying all the configuration constraints, it means that the catalog does not have a well-formed product (overconstrained situation). In this case, we say that the catalog is neither correct nor valid for the given problem.

3.3.4 An alternative CSP model

In the above encoding model, components of configurable products are modeled as variables of the CSP. Another CSP modeling approach would be to consider variables rep-

resenting key attributes of the components of the products. Key attributes refer to the those attributes that identify a concrete solution. Obviously, domains of variables would be the set of different options for each attribute.

This model require, in general, more hard constraints than the previous approach. In the above CSP model, configuration rules can be expressed by binary constraints among the different components. In this model, configuration rules between components must be encoded in regard of their attributes. On the other hand, preferences and optimization constriants can be easily expressed, because in general they depend on specific attributes of components.

The following example show how to model the problem illustrated in Example 3.2 using the alternative CSP model described in this subsection:

Example 3.3: A flight ticket from Geneva to Barcelona, using an alternative CSP model.
Let us imagine a simple catalog modeling the problem of finding flights for a round trip from Geneva (GVA) to Barcelona (BCN). The first step consists of identyfing the variables which are associated to key attributes of a solution, *i.e.*, the attributes that uniquely identify a product.

Locations do not need to be represented by variables because they are unique in this example. In the case that the trip would contain different possible departure or arrival locations, a variable representing departure or arrival location would be needed. In the following, the key attributes are listed:

- $\texttt{depTime}_1$ and $\texttt{arrTime}_1$: identify departure and arrival times for the flight from GVA to ZRH. Note that both times are needed since, the duration of different flights may not be the same. Another possiblity would be to consider $\texttt{depTime}_1$ and $\texttt{duration}_1$.

- $\texttt{depTime}_2$ and $\texttt{arrTime}_2$: identify the same as the above variables but for the return flight from BCN to GVA.

- $\texttt{depDate}_1$ and $\texttt{depDate}_2$: identify departure dates for the flight from GVA to ZRH and from ZHR to GVA respectively.

- $\texttt{airline}_1$ and $\texttt{airline}_2$: identify the operating airline of the flight from GVA to BCN and vice-versa.

- $\texttt{intermediate}_1$ and $\texttt{intermediate}_2$: identify the intermediate airports of flights from GVA to BCN and vice-versa.

Note that some other attributes can also be modeled as variables, even if they are not key attributes, for instance: $\texttt{aircraft}_1$ and $\texttt{aricraft}_2$ identifying the aircraft type of the flight from GVA to BCN and vice-versa, or `serviceClass` identiffying the calss of service (economy, business or first class). This variables may be usefull for modeling preference on such attributes.

Hard constraints among different variables must be then stated in order to ensure the feaseability of the solutions. On the other hand, user preferences (and optimization constraints) can be easily modeled as unary soft constraints on attributes.

The convinience of this model or the previous one strongly depends on the specific problem.

3.4 Extended CSP frameworks for electronic catalogs

Section 3.2.3 summarized the most important extensions to the classical CSP framework. Constraints which model user's preferences need to support different degrees of importance and at the same time different degrees of satisfaction. In the sequel, the extended CSP models are analyzed regarding the three types of constraints for electronic catalogs.

MAX-CSPs cannot be used, mainly because they deal only with crisp constraints. If configuration constraints and preference/optimization constraints are modeled in a crisp schema, the MAX-CSP could give solutions that violate hard constraints, *i.e.*, inconsistent solutions.

Constraint hierarchies have the ability to separate soft and hard constraints, thus they allow us to avoid the problem of having solutions that violate hard constraints. Nevertheless, the solutions are less intuitive to users. We would, for example, prefer to violate 10 times a constraint on the preferred airline rather than violate two different preferences on departure times. Even if preferences on departure time are more important than the preferred carrier, the result may not be intuitive at all to the user. Also, it is difficult to establish an ordering of constraints which reflects the user's preferences.

Instances derived from the general semiring-based CSP framework are the most promising ones, because they allow constraints to have different degrees of satisfaction:

- Fuzzy CSPs would find solutions with the best least preferred constraint. Basically, it takes into account only the worst unsatisfied criteria in each solution. The main disadvantage of this model is that solutions are not evaluated according to all constraints but only according to the less satisfied one.

- Lexicographic CSPs in the fuzzy framework are better suited than fuzzy CSPs for modeling preference/optimization constraints because they avoid a too coarse solution generation, however an even more general view of all constraints would be needed.

- Possibilistic and probabilistic CSP frameworks are not appropriate at all because preference/optimization constraints do not deal with probabilities nor possibilistic distributions.

- The Weighted CSP framework is the most well-suited framework for modeling configuration, preference and optimization constraints. In weighted CSPs, solutions are complete assignments that minimize the sum of all constraint violations. The inconvenience is that constraints are treated in the same way by summing up their

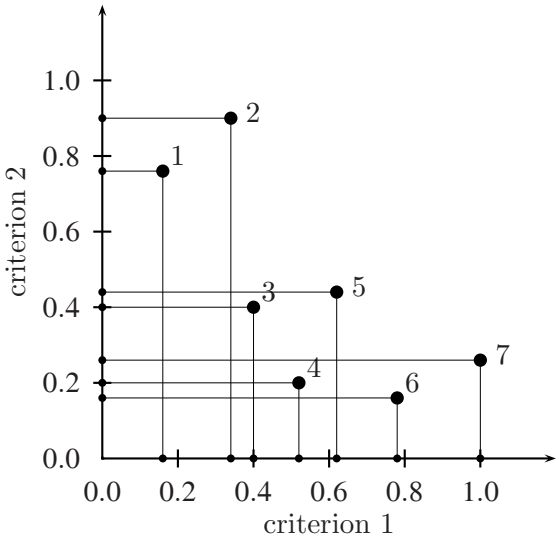


Figure 3.5: Example of solutions in a CSP with two soft constraints. The two coordinates show the values indicating the degrees to which criteria, horizontal and vertical, are violated. Solutions of the problem strongly depend on the CSP framework. Possible orderings of these solutions are presented in Table 3.1.

valuations without taking into consideration different degrees of importance among them.

In Figure 3.5, an example of solutions violating, to a certain degree, two different soft constraints is shown. Depending on the CSP framework, solving algorithms would give the results shown in Table 3.1. In Table 3.1, not only the most optimal solution is shown, but an ordered list of optimal solutions for each CSP framework.

CSP framework	Ordering of solutions
Constraint hierarchies:	
• if criterion 1 is more important than criterion 2	1, 2, 3, 4, 5, 6 and 7
• if criterion 2 is more important than criterion 1	6, 4, 7, 3, 5, 1 and 2
Fuzzy CSP	3, 4, 5, 1, 6, 2 and 7
Fuzzy CSP with lexicographic order	3, 4, 5, 1, 6, 2 and 7
Possibilistic CSP	no sense
Probabilistic CSP	no sense
Weighted CSP	4, 3, 1, 6, 5, 2 and 7

Table 3.1: Solving algorithms of different CSP frameworks may give different results for the same problem. The solutions to the problem are displayed in Figure 3.5.

3.5 Constraint satisfaction for configurable catalogs

Modeling electronic catalogs into CSPs implies to consider the three following types of constraints: configuration constraints, user's preferences and criteria optimization. In our model, configuration constraints are modeled with classical hard constraints. User's preferences and optimization constraints are modeled with soft constraints (crisp or flexible) that associate a cost or penalty to their tuples.

In the following definitions, our constraint satisfaction model is stated. Before defining the CSP model itself, the definitions of soft and hard constraints are given. Hard constraints are defined in a similar way as those in the case of classical CSPs but they are adapted in order to build a framework, where soft and hard constraints can be treated in the same way.

Definition 3.9 (Hard constraint) A *hard constraint* C_i with $scope(C_i) = \{V_{i_1}, \ldots, V_{i_j}\}$ is a constraint where its definition def_{C_i} is a valuation function (also called constraint valuation):

$$\varphi_{C_i} : D_{i_1} \times \cdots \times D_{i_j} \rightarrow \begin{cases} 0 & \text{if the tuple is allowed} \\ \infty & \text{if the tuple is disallowed} \end{cases}$$

∞ represents a value, called *hard violation*, that is used as a maximum violation for any assignment. Given a collection of values for constraint valuations $a_1, \ldots, a_k$, the following properties must hold:

1) $a_1 + \cdots + a_k = \infty \iff \exists i \in [1, \ldots, k], a_i = \infty$ and

2) $a_1 + \cdots + a_k < \infty \iff \forall i \in [1, \ldots, k], a_i < \infty$.

Note that the above properties are important in order to guarantee that a single violated hard constraint is enough to decide that a solution is infeasible (property 1) and that just a set of violated soft constraints never produces an infeasible solution (property 2).

Soft constraints are modeled as functions that map each solution into a numerical value that indicates to what extent the criterion is violated, *i.e.*, the lower the value, the better the solution.

Definition 3.10 (Soft constraint or Criterion) A *soft constraint* C_i with $scope(C_i) = \{V_{i_1}, \ldots, V_{i_j}\}$ is a constraint where its definition def_{C_i} is a valuation function $\varphi_{C_i} : D_{i_1} \times \cdots \times D_{i_j} \rightarrow \mathbb{R}^+ \backslash \{\infty\}$ that defines the degree to which value combinations violate the constraint; a value of 0 means that the constraint is completely satisfied.

The problem of solving a constraint satisfaction problem with hard and soft constraints is called *Multi-criteria Constraint Optimization Problem* (MCOP):

Definition 3.11 (Multi-criteria Constraint Optimization Problem (MCOP)) A *MCOP* $\mathcal{P}$ is a tuple $(\mathcal{V}, \mathcal{D}, \mathcal{HC}, \mathcal{SC})$, where:

- $\mathcal{V} = \{V_1, \ldots, V_n\}$ is the set of variables involved in $\mathcal{P}$,

- $\mathcal{D} = \{D_1, \ldots, D_n\}$ is the set of domains associated to variables,
- $\mathcal{HC} = \{HC_1, \ldots, HC_m\}$ is the set of hard constraints.
- $\mathcal{SC} = \{SC_1, \ldots, SC_l\}$ is the set of soft constraints.

Hard and soft constraints are separated to ease the explanations of the rest of the thesis, but they can be considered as the same type of constraints because both could be defined over functions of the type $\varphi_{C_i} : D_{i_1} \times \cdots \times D_{i_j} \rightarrow \mathbb{R}^+$.

Feasible solutions and partial feasible solutions for MCOPs are defined in the same way as solutions for classical CSPs, thus:

Definition 3.12 (Feasible Solution and Partial Feasible Solution) A *Feasible Solution*[7] S of a MCOP with n variables is a n-compound label in such a way that all the hard constraints are satisfied. In an analytical form, a solution of a MCOP is an assignment S with $var(S) = \mathcal{V}$ and $val(S) = \langle v_1, \ldots, v_n \rangle$ such that $\forall c \in \mathcal{HC}, \varphi_c(val(S \downarrow^{\mathcal{V}}_{scope(c)})) = 0$. A *partial feasible solution*[8] S of a MCOP is a k-compound label ($k < n$) where all the hard constraints are satisfied.

3.6 Optimal solutions to MCOPs

What the best solution to a MCOP is, depends strongly on the relative importance of different criteria. This may vary depending on the user, the time, or the precise values that the criteria can take. For example, in travel planning for some people price may be more important than the schedule, while for others it is the other way around. People find it very difficult to characterize the relative importance of their preferences by numerical weights. On the other hand, it is not always easy to define valuation functions for soft constraints in a way that they can be combined in a numerical way. Thus, two different situations are identified:

- **Quantitative approach**: this approach deals with problems where a numerical combination of the valuations for each constraint is feasible. This means that two conditions hold:

  1. the relative importance of the constraints (constraint weighting) is precisely known, and
  2. the valuation functions of the constraints are also known and comparable among them.

  In such a situation, the optimal solution to a MCOP is a total feasible assignment that minimizes its valuation. The valuation of an assignment is the sum of the violations of the solution for each constraint. This approach is described in detail in Section 3.7.

[7] Sometimes, it will be just called solution.
[8] Sometimes, it will be just called partial solution.

- **Qualitative constraint combination approach**: is an alternative approach when the above conditions do not hold. In this situation, it is not possible to identify a single best solution, but at least certain solutions can be classified as certainly *not* optimal (Section 3.8).

Actually, in both approaches, one might be interested in getting a set of good[9] solutions to the problem. This is especially important in the case of electronic catalogs because the user normally wants to evaluate a set of alternatives and choose the product that best fits his needs. Interestingly, most decision aids already return not the single optimal solution, but an ordered list of the top-ranked solutions. Thus, web search engines return hundreds of documents deemed to be the best matches to a query, and electronic catalogs return a list of possibilities that fit the criteria in decreasing degree. In general, these solutions have been calculated assuming a certain weight distribution among constraints. It appears that listing a multitude of nearly optimal solutions is intended to compensate for the fact that these weights, and thus the optimization criterion, are usually not accurate for the particular user.

We assume that the user is always interested in getting a set of *good* solutions and not only *the best*. Electronic catalogs are designed to support the user decision process but the user is likely to always make the final decision among several alternatives. People like to evaluate a set of choices and be able to select the best one. In that sense, an electronic catalog can be seen as a **decision aid system** for the purchase decision making process. For this reason, we are interested in getting the *k best* solutions to a MCOP, and this is valid for both approaches.

3.7 The quantitative approach

As explained above, it would be appropriate, therefore, to be able to express different degrees of importance of soft constraints. This applies both to user's preferences and to optimization criteria. In the quantitative approach, these degrees of relevance are numerically known and therefore can be defined by means of a vector called constraint weight vector.

Definition 3.13 (Constraint Weight Vector) Given a set of soft constraints $\mathcal{SC} = \{SC_1, \ldots, SC_l\}$, an associated *constraint weight vector* $\mathcal{CW} = \{w_1, \ldots, w_l\}, w_i \in \mathbb{R}$ defines different degrees of importance for the constraints in $\mathcal{SC}$. w_i is a multiplicative factor for the valuation function associated to SC_i.

Assignments in the quantitative approach can be viewed in terms of costs or penalties. In that sense, we define the valuation of an assignment by the sum of all the violations of the constraints implied in the assignment:

Definition 3.14 (Valuation of an Assignment) Given an assignment A for a MCOP

[9]Here, the term *good* (and *best*) has two different meanings depending on the approach it refers to.

with $var(A) = \mathcal{W} \subseteq \mathcal{V}$, $\varphi(A)$ denotes its valuation and it is computed as follows:

$$\varphi(A) = \sum_{\forall c \in \{HC \cup SC\}} \varphi_c(val(A \downarrow^{\mathcal{W}}_{scope(c)}))$$

An assignment A is a (partial) feasible solution if and only if $\varphi(A) < \infty$; otherwise A violates some hard constraints. If a constraint weight vector $\mathcal{CW} = (w_1, \ldots, w_l)$ is available, then the valuation of an assignment A is:

$$\varphi(A) = \sum_{\forall c \in HC} \varphi_c(val(A \downarrow^{\mathcal{W}}_{scope(c)})) + \sum_{\forall c_i \in SC} w_i \cdot \varphi_{c_i}(val(A \downarrow^{\mathcal{W}}_{scope(c_i)}))$$

In the quantitative approach, a MCOP problem can be mapped into an optimization problem with a single criterion, called Weighted Constraint Optimization Problem (WCOP):

Definition 3.15 (WCOP and Optimal Solution) A *WCOP* is a MCOP with an associated weight vector $\mathcal{CW} = \{w_1, \ldots, w_l\}$[10]. The *optimal solution* S to a WCOP is a feasible solution that minimizes its valuation function $\varphi(S)$. Note that there may be several optimal solutions to a WCOP.

When all the criteria are equally important, the associated constraint weight vector would be $(1/k, \ldots, 1/k)$ where k is the number of criteria. In such cases, the constraint weight vector is not needed and the WCOP is equivalent to a Weighted CSP (see Section 3.2.3) and therefore is also an instantiation of the semiring CSP framework. WCOP can be solved by branch-and-bound search algorithms. These algorithms can be easily adapted to return not only the best solution, but an ordered list of the k best solutions. These algorithms are described in Chapter 4.

3.7.1 Order of feasible solutions

The relations *better than* and *equally good as* applied to feasible solutions allow us to have a totally ordered set of the solution space.

Definition 3.16 ($\preceq$, *better than* relation over the feasible solutions to a WCOP) Given a WCOP with feasible solutions $\mathcal{S}$, two solutions $A, B \in \mathcal{S}$, and a valuation function as described in Definition 3.14, we say that solution A is *better than* solution B, written $A \prec B$, if and only if $\varphi(A) < \varphi(B)$. Solution A is *equally good as* solution B, written $A \doteq B$, if and only if $\varphi(A) = \varphi(B)$. Solution A is *better than or equally good as* solution B, written $A \preceq B$, if and only if $\varphi(A) \leq \varphi(B)$.

Property 3.1: $\preceq$ totally orders the set of feasible solutions to a WCOP. Given a WCOP with $\mathcal{S}$ feasible solutions, the following properties hold:

1. Reflexivity: $A \preceq A, \forall A \in \mathcal{S}$.

[10] In the case of having several different types of criteria, we could have one constraint weight vector for each type of criteria.

2. Weak antisymmetry: $A, B \in \mathcal{S}, A \preceq B$ and $B \preceq A$ implies $A \doteq B$.

3. Transitivity: $A, B, C \in \mathcal{S}, A \preceq B$ and $B \preceq C$ implies $A \preceq C$.

4. Comparability: For any $A, B \in \mathcal{S}$, either $A \preceq B$ or $B \preceq A$.

Proof. Each property is proved in the following:

1. Reflexivity:
   $A \preceq A \Rightarrow \varphi(A) \leq \varphi(A)$. Since $\varphi(A) = \varphi(A)$, $\varphi(A) \leq \varphi(A)$ always holds.

2. Weak antisymmetry:
   $A \preceq B$ and $B \preceq A \Rightarrow \varphi(A) \leq \varphi(B)$ and $\varphi(B) \leq \varphi(A)$. Thus, $\varphi(A) = \varphi(B)$ which implies that $A \doteq B$.

3. Transitivity:
   $A \preceq B$ and $B \preceq C \Rightarrow \varphi(A) \leq \varphi(B)$ and $\varphi(B) \leq \varphi(C)$. Thus, $\varphi(A) \leq \varphi(C)$ which implies that $A \preceq B$.

4. Comparability:
   For any $A, B \in (S)$, either $\varphi(A) \leq \varphi(B)$ or $\varphi(B) \leq \varphi(A)$ is true by the Trichotomy Law[11].

□

Since $\preceq$ totally orders the feasible solutions to a WCOP, the *best k* solutions to a WCOP can be defined as follows:

Definition 3.17 (k Best Solutions to a WCOP) Given a WCOP with m feasible solutions $\mathcal{S} = \{S_1, \ldots, S_m\}$, the *best k ($k \leq m$) solutions to a WCOP* is a set of feasible solutions $\mathcal{S}' = \{S_1, \ldots, S_k\} \subseteq \mathcal{S}$ such that:

$$S_i \preceq S_j, \forall i = 1, \ldots, k,\ j = k+1 \text{ and } S_k \preceq S_l, \forall l > k$$

As in the case of an optimal solution, the k best solutions to a WCOP may not be a unique set.

3.7.2 User preferences and optimization criteria

In the quantitative approach, constraint valuations are combined in a linear combination with a constraint weight vector. In our framework, electronic catalogs are modeled by two kind of soft constraints: user's preferences and optimization criteria. Thus, soft constraints are combined together, and therefore to avoid over (or under) dominance of some of the criteria, it is convenient that the following two properties hold:

[11] For arbitrary real numbers a and b, exactly one of the relations $a < b$, $a = b$, $a > b$ holds (see [5], page 20).

1) Soft constraints (both optimization criteria and user's preferences) should be comparable among them. This property is necessary, since a solution to a WCOP is a feasible assignment that minimizes the linear combination of the soft constraint violations with their weight factors.

2) Optimization criteria should only differentiate solutions that have the same valuation in regard to the user's preferences. In other words, optimization constraints only play a role among solutions that violate the user's preferences at the same degree (see Section 3.3.2).

The process of modifying the valuation functions of the soft constraints in order to satisfy the above conditions is called respectively *normalization* and *rescaling*:

1) **Normalization**: Valuation functions of soft constraints should be transformed to the same range. Originally, soft constraints are defined through valuation functions of the form $\varphi : D_1 \times \cdots \times D_n \rightarrow \mathbb{R}^+ \backslash \{\infty\}$. To make these valuations comparable among them, a transformation of valuation functions from $\varphi : D_1 \times \cdots \times D_n \rightarrow \mathbb{R}^+ \backslash \{\infty\}$ to $\varphi : D_1 \times \cdots \times D_n \rightarrow [0, \texttt{max-val}], \texttt{max-val} \in \mathbb{R}^+ \backslash \{\infty\}$ is needed. For the rest of the document, we assume that `max-val` is equal to 1. After this transformation, a valuation of `max-val` indicates that the soft constraint is totally violated, a valuation of `max-val`/2 means that the soft constraint is half violated, and a valuation of 0 indicates that the soft constraint is completely satisfied. How to make such a transformation depends strongly on the constraint itself (and its semantics), but in general we can consider two kinds of valuation functions for soft constraints:

   – Fixed degrees of violations. For instance, consider that a user prefers to fly with Swiss Air Lines (`LX`). This preference can be expressed by three degrees of violations:

$$\texttt{penalty(f)} \rightarrow \begin{cases} 0 & \texttt{if f.airline} = \texttt{LX} \\ 100 & \texttt{if f.airline} \in \texttt{alliance(LX)} \\ 500 & \texttt{otherwise} \end{cases}$$

   The normalized form to the range $[0,1]$ of this valuation function could then be:

$$\texttt{normalized-penalty(f)} \rightarrow \begin{cases} 0 & \texttt{if f.airline} = \texttt{LX} \\ 0.2 & \texttt{if f.airline} \in \texttt{alliance(LX)} \\ 1 & \texttt{otherwise} \end{cases}$$

   – Continuous degrees of violations. For example, the user prefers flights departing before 10 a.m. As seen in Example 3.2, p42, this preference can be expressed by a function like:

$$\texttt{penalty(f)} \rightarrow \begin{cases} 0 & \texttt{if f.depTime} \leq \texttt{10 a.m.} \\ (\texttt{f.depTime} - \texttt{10 a.m.}) & \texttt{otherwise} \end{cases}$$

In order to normalize such preference to the range $[0,1]$, the maximum `f.depTime` of all the possible flights must be known. If we call this maximum departure time `maxDepTime`, then the normalized function could be stated as:

$$\texttt{normalized-penalty(f)} \rightarrow \begin{cases} 0 & \texttt{if f.depTime} \leq \texttt{10 a.m.} \\ \frac{\texttt{f.depTime} - \texttt{10 a.m.}}{\texttt{maxDepTime} - \texttt{10 a.m.}} & \texttt{otherwise} \end{cases}$$

2) **Rescaling**: The second step consists of transforming the valuation functions after the step 1) in order to force a priority for user preferences over optimization criteria (see Section 3.3.2), *i.e.*, optimization criteria should only differentiate solutions that violate the preferences at the same degree. This step must guarantee that even if all the optimization constraints are violated at their maximum degree, a difference in any of the violations in the preferences should be more important than all the optimization criteria violations.

Let us assume that there are op optimization constraints with valuation functions to the range $[0,1]$. Then, the sum of the optimization valuations is always at the range $[0, op]$. Valuation functions for preferences can be discretized in k values, and their discrete form could then be:

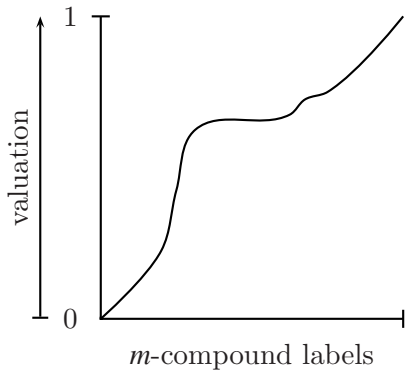
$$\varphi_{\texttt{discrete}} = \begin{cases} 0 & \iff 0 \leq \varphi < \frac{1}{2 \cdot k} \\ \frac{op}{2} + i \cdot op & \iff \frac{1}{2 \cdot k} + \frac{i}{k} \leq \varphi < \frac{1}{2 \cdot k} + \frac{i+1}{k}, i = 0, \ldots, k-1 \\ k \cdot op & \iff \frac{k-1}{k} \leq \varphi \end{cases}$$

The parameter k heavily depends on the application domain. In general, as k increases, the *precision* of the valuation functions for the preferences also increases. In this way, variances on the violations for preferences are always more relevant than any violations of the opitmization constraints.

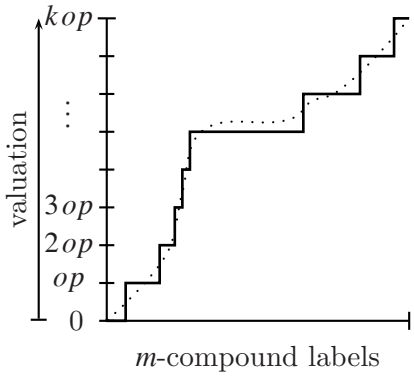
Another option could be just to transform φ into

$$\varphi_{\texttt{rescaled}} = \varphi \cdot k \cdot op$$

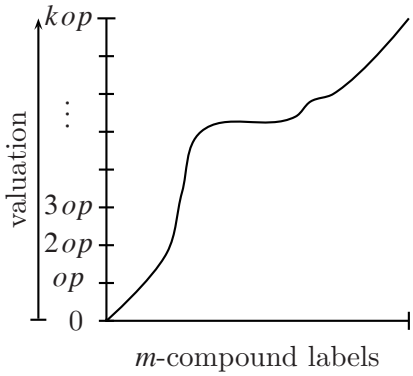
Figure 3.6 shows the process of transforming a normalized valuation function for a criteria (Subfigure a) in its discrete (Subfigure b) and rescaled (Subfigure c) forms. The advantage of using $\varphi_{\texttt{rescaled}}$ instead of $\varphi_{\texttt{discrete}}$ is that preferences do not loose precision in their valuation functions. The disadvantage of using $\varphi_{\texttt{rescaled}}$ instead of $\varphi_{\texttt{discrete}}$ is that in some cases, the optimization criteria can be more relevant than some slight variance in the satisfaction of preferences (which contradicts the property 2 above described). Again, the choice between $\varphi_{\texttt{rescaled}}$ and $\varphi_{\texttt{discrete}}$ depends on the application domain. In some applications it could even be convenient to have some preferences modeled with discretized functions and some others modeled in their rescaled form.



(a) A normalized valuation function for a preference involving m variables.



(b) A discretized valuation function for a preference involving m variables.



(c) A rescaled valuation function for a preference involving m variables.

Figure 3.6: Subfigure (a) is an example of a valuation function for a preference normalized in the range $[0,1]$. If we assume that the problem has op optimization constraints, then the valuation of all the optimization constraints lies within the range $[0,op]$. Subfigure (b) shows a possible way of transforming the preference valuation function in a discrete function with k values. Subfigure (c) plots the valuation function for the same preference rescaled to the range $[0,k \cdot op]$.

3.8 The qualitative constraint combination approach

When relative importance of criteria is unknown or the valuation functions for soft constraints can not be combined together, it is not possible to identify a single best solution. But at least certain solutions can be classified as certainly *not* optimal. This is the case when there is another solution which is as good as or better in all respects (*i.e.*, in all soft constraints). We say that a solution S_1 *dominates* another solution S_2 if for every soft constraint, the violation cost in S_1 is no greater than that in S_2, and if for at least one

constraint, S_1 has a lower cost than S_2. Solution dominance is defined formally as follows:

Definition 3.18 (Solution Dominance) Given a MCOP P with l soft constraints $\mathcal{SC} = \{SC_1, \ldots, SC_l\}$ and two feasible solutions S_1 and S_2 of P:

$$S_1 \text{ dominates } S_2 \iff \begin{cases} \forall c \in \mathcal{SC}, \varphi_c\left(val\left(S_1 \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) \leq \varphi_c\left(val\left(S_2 \downarrow^{\mathcal{V}}_{scope(c)}\right)\right), \text{ and} \\ \exists c \in \mathcal{SC}, \varphi_c\left(val\left(S_1 \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) < \varphi_c\left(val\left(S_2 \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) \end{cases}$$

The solution dominance defines a relation on the set of feasible solutions to a MCOP P. The expression S_1 dominates S_2, written $S_1 \prec S_2$, means that S_1 is *better than* S_2. When solution S_1 and solution S_2 violate the constraints at the same degree, we say that the solutions are *equally good*, written $S_1 \doteq S_2$:

$$S_1 \doteq S_2 \iff \forall c \in \mathcal{SC}, \varphi_c\left(val\left(S_1 \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) = \varphi_c\left(val\left(S_2 \downarrow^{\mathcal{V}}_{scope(c)}\right)\right)$$

Consequently, $S_1 \preceq S_2$ means that solution S_1 dominates S_2 or S_1 is equally good than solution S_2: $S_1 \preceq S_2 \iff S_1 \prec S_2$ or $S_1 \doteq S_2$. Furthermore,

$$S_1 \preceq S_2 \Rightarrow \forall c \in \mathcal{SC}, \varphi_c\left(val\left(S_1 \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) \leq \varphi_c\left(val\left(S_2 \downarrow^{\mathcal{V}}_{scope(c)}\right)\right)$$

Please, note that $\prec$, $\preceq$ and $\doteq$ have two different meanings, depending on the context (quantitative or qualitative), however, their meaning should always be clear within the context.

3.8.1 Order of feasible solutions

In the qualitative constraint combination approach, the feasible solutions to a MCOP can be partially ordered by the solution dominance relation (see Definition 3.18).

Property 3.2: $\preceq$ partially orders the set of feasible solutions to a MCOP. Given a MCOP with $\mathcal{S}$ feasible solutions, the following properties hold:

1. Reflexivity: $A \preceq A, \forall A \in \mathcal{S}$.
2. Weak antisymmetry: $A, B \in \mathcal{S}, A \preceq B$ and $B \preceq A$ implies $A \doteq B$.
3. Transitivity: $A, B, C \in \mathcal{S}, A \preceq B$ and $B \preceq C$ implies $A \preceq C$.

Proof. Each property is proved in the following:

1. Reflexivity: $A \preceq A \Rightarrow A \prec A$ or $A \doteq A$. It is clear that $A \doteq A$ (A is equally good than A).
2. Weak antisymmetry: $A \preceq B$ and $B \preceq A \Rightarrow (A \prec B$ or $A \doteq B)$ and $(B \prec A$ or $B \doteq A)$. By Definition 3.18, the following expressions are contradictory:
   - $A \prec B$ and $B \prec A$.
   - $A \prec B$ and $B \doteq A$.

Thus, we have that $A \doteq B$ is true.

3. Transitivity: $A \preceq B$ and $B \preceq C \Rightarrow$

$$\begin{aligned} \forall c \in \mathcal{SC}, \varphi_c\left(val\left(A \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) &\leq \varphi_c\left(val\left(B \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) \text{ and} \\ \forall c \in \mathcal{SC}, \varphi_c\left(val\left(B \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) &\leq \varphi_c\left(val\left(C \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) \end{aligned}$$

that implies $\forall c \in \mathcal{SC}, \varphi_c\left(val\left(A \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) \leq \varphi_c\left(val\left(C \downarrow^{\mathcal{V}}_{scope(c)}\right)\right)$. Therefore, by Definition 3.18, $A \preceq C$.

□

3.8.2 Pareto optimality

The idea of *Pareto*[12]*-optimality* [184] is to consider all solutions which are not dominated by another one as potentially optimal:

Definition 3.19 (Pareto-optimal Solution and Pareto-optimal Set) Given a MCOP P with feasible solutions $\mathcal{S}$, any feasible solution $S \in \mathcal{S}$ which is not dominated by another is called *Pareto-optimal Solution* of P:

$$S \text{ is Pareto-optimal} \iff \nexists S' \in \mathcal{S}, S' \prec S$$

The *Pareto-optimal Set*[13] $\mathcal{PO}$ of a MCOP P with feasible solutions $\mathcal{S}$ is the set of solutions which are not dominated by any other one: $\mathcal{PO} = \{S \in \mathcal{S} \mid \nexists S' \in \mathcal{S}, S' \prec S\}$.

In Figure 3.7, the Pareto-optimal set is $\{1,3,4,6\}$, as solution 8 is dominated by solution 1, solution 7 is dominated by 4 and 6, 5 is dominated by 3 and 4, and 2 is dominated by 1.

Example 3.4: User profiles and optimality in the travel domain.
This example illustrates how different user profiles (constraint weight vectors) influence the optimality of a solution for each individual user. Consider users willing to plan the same itinerary. The users have exactly the same criteria:

1. leave after 17:00,
2. mileage with `United Airlines`,
3. no change in London Heathrow, and
4. as cheap as possible.

A first naive analisys could yield to the interpretation that all the users would prefer the same solution, since they all have the same criteria for the same itinerary. Nevertheless, criteria have different importance for individual users.

[12] Vilfredo Pareto, 1848-1923, Italian economist.
[13] also called the efficient frontier of P.

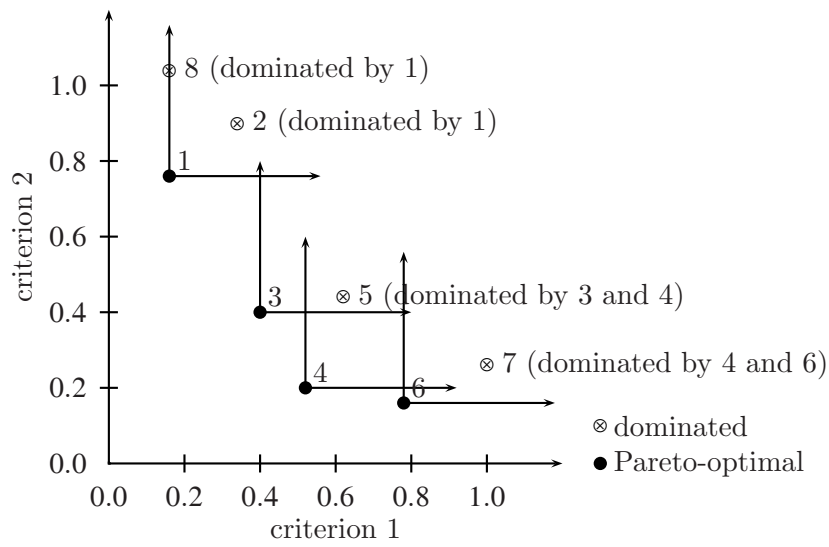


Figure 3.7: Example of Pareto-optimal solutions in a MCOP with two preference criteria. The two coordinates show the values indicating the degrees to which criteria, horizontal and vertical, are violated. Each solution dominates solutions in a rectangle. The solutions which are not dominated (1, 3, 4 and 6) are Pareto-optimal.

In Table 3.2, different solutions are shown. Each solution satisfies some of the criteria of the users. A *conscientious* user, for example, would prefer to satisfy the preferences about time and cost (criteria 1 and 4). In regard to *selfish* travelers, the airline and intermediate airports preferences (criteria 2 and 3) have a higher degree of importance, whilst the cost (criterion 4) or the departure time (criterion 1) are less important. Finally, for a *bureaucrat* traveler, time and airline preferences (criteria 1 and 2) are the most important criteria.

On the other hand, the last solution is *dominated* by solutions 1, 3, and 4. Note that, the only satisfied criteria for the dominated solution is the airline preference, all the other criteria are unsatified. Solutions 1, 3, and 4 satisfy the airline preference and others at the same time, thus they dominate solution 5. In a closer look at Table 3.2, one could think solution 4 is also dominated by solution 1, since solution 1 satisfies the same criteria than solution 4 plus criterion 4. However, the first criterion can accept different degrees of satisfaction, and solution 4 better satisfies it than solution 1 (solution 4 has a later departure time than solution 1).

The following two basic lemmas will be used in the rest of the chapter.

Lemma 3.1: At least there is one Pareto-optimal solution to a MCOP.
A MCOP P with $\mathcal{S} = \{S_1, \ldots, S_n\}$, $n \geq 1$, solutions has at least one Pareto-optimal solution.

Proof. Let us consider the sets of solutions

$$\mathcal{S}_1 = \{S_1\}, \mathcal{S}_2 = \{S_1, S_2\}, \mathcal{S}_3 = \{S_1, S_2, S_3\}, \ldots, \mathcal{S}_n = \mathcal{S} = \{S_1, S_2, \ldots, S_n\}$$

and their corresponding sets of Pareto-optimal solutions $\mathcal{PO}_1, \ldots, \mathcal{PO}_n$.

Proving that $|\mathcal{PO}_n| \geq 1$ by induction, implies to demonstrate that:

		criterion			
user profile	solution	1	2	3	4
selfish	1. `United Airlines`, 14:55 → 13:50, CHF 900	✗	✓	✓	✓
conscientious	2. `British Airways`, 18:50 → 18:15, CHF 789	✓	✗	✗	✓
bureaucrat	3. `United Airlines`, 17:30 → 18:15, CHF 2,300	✓	✓	✗	✗
selfish	4. `United Airlines`, 15:30 → 13:10, CHF 1,175	✗	✓	✓	✗
dominated	5. `United Airlines`, 12:35 → 12:25, CHF 1,500	✗	✓	✗	✗

Table 3.2: Individual users with the same criteria would choose different flights because they give different weights to their criteria. The last solution is dominated, thus it would never be rationally preferred for any user profile.

1. $|\mathcal{PO}_1| \geq 1$, and
2. $|\mathcal{PO}_k| \geq 1 \Rightarrow |\mathcal{PO}_{k+1}| \geq 1, \forall k, 1 \leq k \leq n-1$

Hence,

1. $|\mathcal{PO}_1| \geq 1$ is obviously true. S_1 is not dominated by any other solution in $\mathcal{S}_1$. Therefore S_1 is Pareto-optimal for $\mathcal{S}_1$.
2. There are the two following cases:
   - $\exists S \in \mathcal{PO}_k \,|\, S \prec S_{k+1}$. Consequently, $\mathcal{PO}_k = \mathcal{PO}_{k+1}$. Therefore, $|\mathcal{PO}_{k+1}| \geq 1$.
   - $\nexists S \in \mathcal{PO}_k \,|\, S \prec S_{k+1}$. That means that $\forall S \in \mathcal{PO}_k \,,\, S_{k+1} \prec S$ or $S_{k+1} \doteq S$. In both cases, $S_{k+1} \in \mathcal{PO}_{k+1}$. In consequence, $|\mathcal{PO}_{k+1}| \geq 1$.

Thus, by the induction principle, $|\mathcal{PO}_n| \geq 1$ is true. □

As shown in Figure 3.7, solutions to a MCOP with k criteria can be mapped into points in a k-dimensional space. Formally, these concepts are defined as follows:

Definition 3.20 (Criteria Space and Mapping Solutions to it) Given a MCOP P with $\mathcal{SC} = \{SC_1, \ldots, SC_k\}$ soft constraints, its *criteria space* is a k-space $\mathbb{R}^k$, where its dimension i represents criterion SC_i. A solution S to P can be then mapped to a point p in the criteria space as follows:

$$point(S) = p = \left(\varphi_{C_1}\left(val\left(S \downarrow^{\mathcal{V}}_{scope(C_1)}\right)\right), \ldots, \varphi_{C_k}\left(val\left(S \downarrow^{\mathcal{V}}_{scope(C_k)}\right)\right)\right)$$

The solutions that are mapped to a point p are denoted $sol(p)$, and the point to where a solution is mapped is called $point(S)$. It is also useful to differentiate the mapping of a solution S to different criteria spaces. For that, the mapping of a solution S to a k criteria space will be noted $point_k(S)$.

It is convenient to differentiate between Pareto-optimal solutions and points where these Pareto-optimal solutions are mapped:

Definition 3.21 (Set of Pareto-optimal Points of a MCOP) Given a MCOP P with $\mathcal{SC}$ soft constraints and k Pareto-optimal solutions $\mathcal{PO}$. The *set of Pareto-optimal points* of P, denoted $\mathcal{POP}$, is the set of points where Pareto-optimal solutions are mapped in the criteria space of P.

Note that a Pareto-optimal point can represent several Pareto-optimal solutions, thus in general, $|\mathcal{POP}| \leq |\mathcal{PO}|$.

Lemma 3.2: Adding one criterion to a MCOP can only add more Pareto-optimal points.
Given a MCOP P with $\mathcal{SC} = \{SC_1, \ldots, SC_k\}$ soft constraints, $\mathcal{S}$ solutions, $\mathcal{PO}$ Pareto-optimal solutions and $\mathcal{POP}$ Pareto-optimal points. Consider a MCOP P' with $\mathcal{SC}'$ soft constraints, equivalent to P but with one more criterion: $\mathcal{SC}' = \mathcal{SC} \cup \{SC\}$. Let $\mathcal{PO}'$ be the set of Pareto-optimal solutions of P' and let $\mathcal{POP}'$ be the set of Pareto-optimal points of P'. Then, $\mathcal{POP} \subseteq \mathcal{POP}'$.

Proof. $\mathcal{POP} \subseteq \mathcal{POP}' \Leftrightarrow \forall p \in \mathcal{POP}, \exists p\prime \in \mathcal{POP}'$ such that $\exists S, S \in sol(p) \wedge S \in sol(p\prime)$. Let p be any Pareto-optimal point of $\mathcal{POP}$. Let S be any solution in $sol(p)$. Since $point(S) \in \mathcal{POP}$, S is Pareto-optimal, by Definition 3.21. By Definition 3.18 and Definition 3.19:
$\nexists Y \in \mathcal{S}$, such that :

$$\begin{aligned} \forall c \in \mathcal{SC}, \varphi_c\left(val\left(Y \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) &\leq \varphi_c\left(val\left(S \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) \text{ and} \\ \exists c \in \mathcal{SC}, \varphi_c\left(val\left(Y \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) &< \varphi_c\left(val\left(S \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) \end{aligned}$$

which is equivalent to:
$\forall Y \in \mathcal{S}$,

$$\exists c \in \mathcal{SC}, \varphi_c\left(val\left(Y \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) > \varphi_c\left(val\left(S \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) \text{ or} \tag{3.1}$$

$$\forall c \in \mathcal{SC}, \varphi_c\left(val\left(Y \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) = \varphi_c\left(val\left(S \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) \tag{3.2}$$

Therefore, any solution $Y \in \mathcal{S}, Y \neq S$ falls in one of the following cases:

(3.1) in this case it is clear that adding one more criterion to the problem would not imply in any case that solution Y dominates solution S. In consequence, $point_{k+1}(S) \in \mathcal{POP}'$.

(3.2) this second case implies that $Y \doteq S$. Adding one more criterion, SC_{k+1}, to the problem implies the following cases:

1. $\varphi_{SC_{k+1}}\left(val\left(Y \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) = \varphi_{SC_{k+1}}\left(val\left(S \downarrow^{\mathcal{V}}_{scope(c)}\right)\right)$, or
2. $\varphi_{SC_{k+1}}\left(val\left(Y \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) < \varphi_{SC_{k+1}}\left(val\left(S \downarrow^{\mathcal{V}}_{scope(c)}\right)\right)$, or
3. $\varphi_{SC_{k+1}}\left(val\left(Y \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) > \varphi_{SC_{k+1}}\left(val\left(S \downarrow^{\mathcal{V}}_{scope(c)}\right)\right)$.

However, in all cases there will be always at least one Pareto-solution: S and Y in the first case, Y in the second case and S in the third case. In consequence, $point_{k+1}(S) \in \mathcal{POP}'$.

Thereby, $\mathcal{POP} \subseteq \mathcal{POP}'$. □

3.8.3 User's preferences and optimization criteria

In the quantitative approach, user's preferences and optimization criteria were treated in the same way. Using Pareto-optimality over user's preferences and optimization criteria would lead to unexpected solutions. That is because we could get solutions that are optimal in respect to optimization criteria but they are not optimal in respect to user's preferences. Hence, two different strategies can be considered:

- User's preferences as the main set of criteria and optimization criteria as a second set of criteria.

  Pareto-optimality is only applied to user's preferences. Optimization criteria can be then used to rank Pareto-optimal solutions. In this way, solutions are firstly selected by their Pareto-optimality in respect to user's preferences, and secondly by their optimality in respect to optimization criteria. A special case is when there are no user's preferences at all. In this situation, Pareto-optimality can be used over optimization criteria.

- All optimization criteria as one more criterion.

  By using this second strategy, the combination of all optimization criteria is one single criterion for Pareto-optimality to be considered together with the user's preferences. The combination of the optimization criteria can be done in the same way as in the quantitative approach, *i.e.*, by summing up their valuations.

3.8.4 Pareto-optimal solutions on the convex hull

In Section 3.8, it has been explained that Pareto-optimal solutions dominate the rest of the solutions, therefore Pareto-optimal solutions are always better for the user than the dominated ones.

However, not all the Pareto-optimal solutions are equally optimal regarding different user profiles. A user profile is here represented by a constraint weight vector as defined in Definition 3.13. In other words, for the same user's preferences two different users might prefer different solutions because they have different constraint weights. In Figure 3.8, the projections of each Pareto-optimal solution to the different user profile vectors indicate to what extent the solution satisfies the criteria for the associated user. Therefore, in Figure 3.8, different users would prefer solutions in different orders: `user 1` $\rightarrow$ 6, 4, 3 and 1; `user 2` $\rightarrow$ 4, 6, 3 and 1; `user 3` $\rightarrow$ 4, 6, 1 and 3; `user 4` $\rightarrow$ 1, 4, 3 and 6.

On the other hand, we pointed out that Pareto-optimal approach is used when the constraint weight vector is unknown. It is worth to remember that if the relative importance of the constraints (*i.e.*, the constraint weight vector) is precisely known, the quantitative approach gives the optimal solutions and they can be totally ordered. However, even if the constraint weight vector is unknown, some Pareto-optimal solutions can be removed because they would never be optimal for any user profile. These Pareto-optimal solutions are the ones that are not on the convex-hull (Definition 3.22) of the set of feasible solutions.

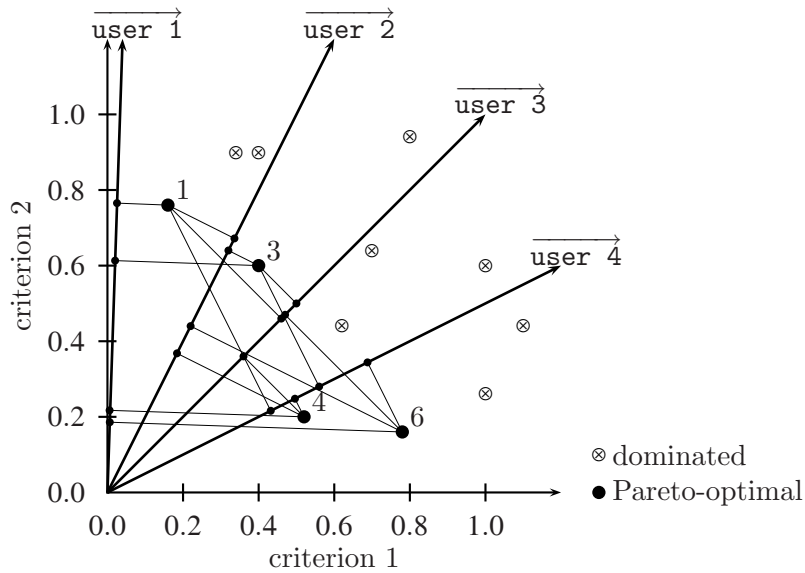


Figure 3.8: The two coordinates show the values indicating the degrees to which criteria, horizontal and vertical, are violated. Pareto-optimal solutions (1, 3, 4 and 6) are shown. The constraint weight vector associated to 4 different users are depicted as vectors (`user 1`, `user 2`, `user 3` and `user 4`). The projections of each Pareto-optimal solution to the different user profile vectors indicate to what extent the solution satisfies the criteria for the associated user.

Definition 3.22 (Convex-hull of a set of solutions to a MCOP) Given a MCOP P with $\mathcal{SC} = \{C_1, \ldots, C_m\}$ soft constraints (criteria) and with $\mathcal{S} = \{S_1, \ldots, S_s\}$ solutions. Consider each solution $S_i \in \mathcal{S}$ as a point p_i in the m-Space $\mathbb{R}^m$:

$$p_i = \left(\varphi_{C_1}\left(val\left(S_i \downarrow^{\mathcal{V}}_{scope(C_1)}\right)\right), \ldots, \varphi_{C_m}\left(val\left(S_i \downarrow^{\mathcal{V}}_{scope(C_m)}\right)\right)\right)$$

The *convex-hull* of the set of solutions to a MCOP is the smallest convex set that contains all the points p_i representing a solution $S_i \in \mathcal{S}$, and it is noted $conv(\mathcal{S}) \subseteq \mathcal{S}$.

For example, consider the Figure 3.9. Solution 3 is Pareto-optimal but it is not on the convex-hull of the set of feasible solutions. It can be observed that for user profiles with $\alpha > 90^o$ Pareto-optimal solution 4 is always more preferred than Pareto-optimal solution 3; for user profiles with $\alpha < 90^o$ Pareto-optimal solution 1 is always more preferred than Pareto-optimal solution 3; and for user profiles with $\alpha = 90^o$ both Pareto-optimal solutions 1 and 4 are always more preferred than Pareto-optimal solution 3. Pareto-optimal solution 3 is never the most preferred one for any profile, because it is not on the convex-hull.

It is worth noting that the above explanation is valid under the assumption that user profiles are defined linear with respect to criteria.

3.8.5 What are the k best solutions?

In the quantitative approach it was possible to define the k best solutions to a problem because the feasible solutions can be totally ordered (see Property 3.1). But, when using the Pareto-optimal concept, the solutions to a MCOP are only partially ordered

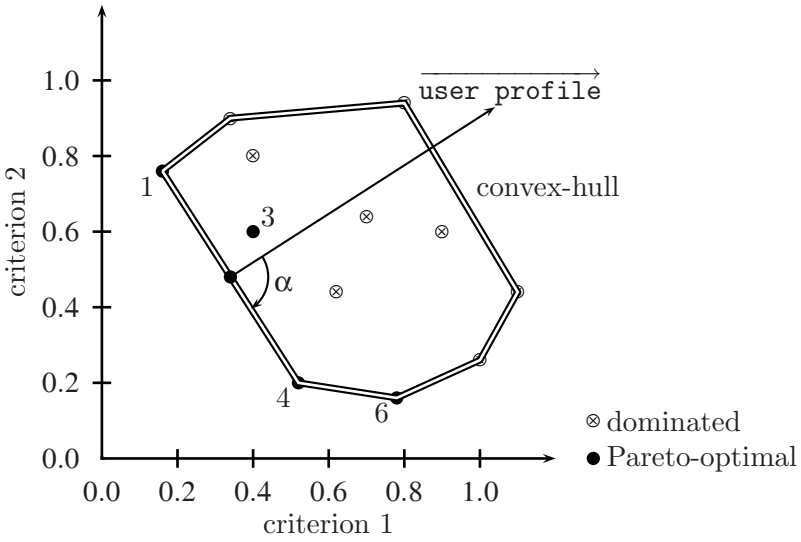


Figure 3.9: The two coordinates show the values indicating the degrees to which criteria, horizontal and vertical, are violated. Pareto-optimal solutions (1, 3, 4 and 6) are shown. The solutions on the convex-hull are linked by a double line. Pareto-optimal solution 3 is not optimal for any user profile because it is not on the convex-hull.

(see Property 3.2), *i.e.*, the feasible solutions can be classified in Pareto-optimal solutions and dominated solutions.

As stated in Section 3.6, we are concerned in getting the k best solutions to a MCOP. A partial order on the solutions is not enough to decide what are the k best solutions to the problem. One characteristic of the Pareto-optimal set is that it usually contains many solutions; in fact, all solutions could be Pareto-optimal. Also, it could happen that a problem has very few Pareto-optimal solutions, however there is always at least one Pareto-optimal solution (see Lemma 3.1).

Ideally, the best solutions to the user for a MCOP are the Pareto-optimal solutions on the convex hull (see Section 3.8.4), however, due to the complexity of computing the solutions on the convex hull when having more than two criteria ($O(n^{\lfloor n/2+1 \rfloor})$, see [226]), several alternatives are suggested.

Let us assume that we are looking for the k best solutions to a MCOP with p Pareto-optimal solutions. Pareto-optimal solutions are, by definition, optimal in respect to any other dominated solution. Therefore, Pareto-optimal solutions should always be preferred over dominated ones.

When dealing with the problem of obtaining the k best solutions to a MCOP with p Pareto-optimal solutions, two different situations are possible[14], either there are $p < k$ Pareto-optimal solutions (too few Pareto-optimal solutions) or $p > k$ Pareto-optimal solutions (too many Pareto-optimal solutions). In the sequel, both situations are described, and different strategies are proposed.

[14]The case where $p = k$ is negligible because then the k best solutions are the solutions which are Pareto-optimal.

3.8.5.1 Too few Pareto-optimal solutions

A problem with few Pareto-optimal solutions is a problem where many solutions are dominated by few ones. Thus, electronic catalogs with this situation are offering few optimal choices to the user. In consequence, it would apparently be enough to support the user in the buying process by only showing these few Pareto-optimal solutions. Nevertheless, in many cases, it can be convenient to give more options to the user, even if some of these options are dominated by others. This is mainly due to the fact that some preferences involved in the user's decision process can not always be expressed by formal constraints. In the buying decision making process, there are usually emotional factors and feelings about products that cannot be easily stated in a formal way. On the other hand, people like to have several options to examine and then choose the one that best suits their needs.

In the case that there are less Pareto-optimal solutions than solutions to be shown to the user, three alternatives are proposed:

1. The Pareto-optimal set is enlarged with dominated solutions that minimize the valuation function (as defined in Definition 3.14) to form the set of solutions to be shown to the user. This alternative could be considered as a mix between the quantitative approach and the Pareto-optimal approach. The quantitative approach is used here to totally order the dominated solutions and take the best $p-k$ ones, in order to complete the set of solutions to be shown to the user. If a constraint weight vector is not available, the preferences of the user are considered equally relevant.

2. Let us assume that the problem has a set of feasible solutions $\mathcal{S}$ with a Pareto-optimal set $\mathcal{PO}$. Then, we could consider the Pareto-optimal solutions in respect to the set $\mathcal{S}-\mathcal{PO}$ to get a second set of Pareto-optimal solutions $\mathcal{PO}'$. The interpretation of this alternative is that as it considers the Pareto-optimal solutions at several levels, it could be seen as a hierarchy of Pareto-optimal sets. Imagine, for example, that one solution dominates all the other ones. In such a case it would be convenient to consider the Pareto-optimal set without taking into consideration this optimal solution.

   When removing a Pareto-optimal set $\mathcal{PO}$, the next $\mathcal{PO}'$ may be too large, leading to the case where there are too many solutions. In this case, one of the strategies proposed in Section 3.8.5.2 can be considered.

3. Finally, another alternative is to ask the user to add more preferences to the system. In Chapter 4, we show experimentally how by increasing the number of criteria, the number of Pareto-optimal solutions also increases. Thus, if there are too few Pareto-optimal solutions and the user adds some more preferences, it is likely that the number of Pareto-optimal solutions will increase. Actually, when adding a new criteria to a MCOP with p Pareto-optimal points, there will be at least p Pareto-optimal points to the new problem (see Lemma 3.2).

   Similarly, when adding more criteria, the problem can then have too many Pareto-optimal solutions. In such a case, one of the strategies proposed in the following

section can be adopted.

3.8.5.2 Too many Pareto-optimal solutions

When a catalog has too many Pareto-optimal solutions to be shown to the user, one could say that the products have very balanced trade-offs in respect to the criteria. It means that the user has a lot of optimal choices regarding his preferences. Ideally, it would be convenient to show all the Pareto-optimal solutions and let the user choose. However, most of the current graphical user interfaces cannot show, in an appropriate manner, more than a certain number of products. Therefore, some strategies to select a subset of the Pareto-optimal set are proposed:

1. One strategy is to filter out the k best solutions in the Pareto-optimal set in respect to the valuation function defined in Definition 3.14. This alternative leads to a mixed approach between the quantitative and the Pareto-optimal approach. This procedure is similar to the one for enlarging the optimal solutions set in the case of too few Pareto-optimal solutions.

2. The other alternative, is to compute a subset of the Pareto-optimal set that is representative of the whole set. This alternative seems to be complex, especially if the number of criteria is high.

3. Ideally, the system should only give the Pareto-optimal solutions that are on the convex-hull (see Section 3.8.4). The drawback with this alternative is its computational complexity (see Chapter 4, Section 4.3.4).

3.8.5.3 Totally ordering for Pareto-optimal solutions

The ideal way of presenting Pareto-optimal solutions to the user is by showing them in a graphical display, where the user can identify to what extent each criteria is satisfied. However, this can be very difficult when the number of criteria is large. A special case is when there are only 2 criteria, where a bidimensional plot (similar to the one shown in Figure 3.7) is the most comprehensible display for people, because they can easily see what solution is good for their specific criteria weighting.

When showing solutions to the user, very often it is convenient to rank and present them in an ordered list. For that, a way of ranking the Pareto-optimal solutions is needed. By using the valuation function of each solution, the Pareto-optimal solutions can be totally ordered, as in the case of the quantitative approach.

3.9 An example: photo equipments

As an example to illustrate the modeling process described in this chapter we propose to analyze a catalog for SLR (Single Lens Reflex) photo equipment. This example is complex enough to argue that it cannot be solved by just presenting a list of all the possible combinations to the user, therefore our model will fit well for this problem.

A complete SLR equipment is composed of a set of camera bodies, a set of lenses and a set of flashes. Simplifying, each component of a SLR system can be specified by the following attributes (with their possible values):

camera body :

`brand` → {`Nikon,Canon,Minolta,Sigma,Pentax`}
`model` → `reference`
`max-shutter-speed` → `num value`
`manual-automatic` → {`manual,auto`}
`autofocus` → {`yes,no`}
`price` → `num value`

lens :

`brand` → {`Nikon,Canon,Minolta,Sigma,Tamron,Tokina,Pentax`}
`mount-type` → {`Nikon,Canon,Minolta,Sigma,Pentax`}
`max-aperture` → `num value`
`min-aperture` → `num value`
`min-zoom` → `num value`
`max-zoom` → `num value`
`price` → `num value`

flash :

`brand` → {`Nikon,Canon,Minolta,Sigma,Metz`}
`brand-compatibility` → {`Nikon,Canon,Minolta,Sigma,Pentax`}
`guide-number` → `num value`
`price` → `num value`

Each component of the equipment is a choice for the user, *i.e.*, a variable in the constraint satisfaction problem. Therefore, if the user is looking for an equipment with i camera bodies, j lenses and k flashes, the problem is modeled with $i+j+k$ variables: i `body-var` variables, j `lens-var` variables, and k `flash-var` variables.

The values for each variable are the available options for each choice. The domain sizes for each type of variable are shown in Table 3.3[15].

The user may be interested in a complete equipment with several lenses, flashes, and bodies[16]. Thus, the photo equipment catalog accepts different instances of the problem depending on how many components the user is looking for. Table 3.4 shows the size of search spaces corresponding to different instances of the catalog. Such instances are derived by varying the number of components to be selected (the choices or variables). Note that the search space for such problems quickly becomes very large, therefore simple filtering methods are not enough to support users to find the right products in an efficient manner.

Let us illustrate the concepts described in this chapter with a complete catalog for an equipment with one camera body, two lenses and one flash. The associated variables for

[15]Real data taken from each brand's web site on June 2002.
[16]Professional photographs are normally interested in several bodies with a set of lenses and flashes.

Domain sizes

Brand	`body-var`	`lens-var`	`flash-var`
Nikon	9	54	9
Canon	10	58	8
Minolta	8	53	8
Pentax	7	28	9
Sigma	2	51	4
Tamron	-	31	-
Tokina	-	63	-
Metz	-	-	22
Total	36	338	60

Table 3.3: Domain sizes for each type of variable in a CSP for modeling a photo equipment.

num. of b	num. of l	num. of f	search space
1	1	1	$36 \cdot 338 \cdot 60 = 7.3 \cdot 10^5$
1	2	1	$36 \cdot 338^2 \cdot 60 = 1.4 \cdot 10^6$
1	3	1	$36 \cdot 338^3 \cdot 60 = 4.9 \cdot 10^8$
1	3	2	$36 \cdot 338^3 \cdot 60^2 = 2.9 \cdot 10^{10}$
2	2	1	$36^2 \cdot 338^2 \cdot 60 = 5.2 \cdot 10^7$
2	3	1	$36^2 \cdot 338^3 \cdot 60 = 1.7 \cdot 10^{10}$

Table 3.4: num. of b, num. of l and num. of f indicate the number of bodies, lenses and flashes respectively in the problem. The search space for different instances of the CSP modeling photo equipments. Different instances correspond to different sets of components to be selected in the catalog.

each component are called: `body-var`, `lens-var-1`, `lens-var-2`, and `flash-var`. Note that more than one million of the possible combinations (see Table 3.4) of the components exist without taking the hard constraints, *i.e.*, configuration rules, into consideration.

The following configuration rules define the hard constraints involved in the example:

- A lens is compatible with a camera body if the `brand` of the camera body and the `mount-type` of the lens are the same. This compatibility rule implies the following binary hard constraints between each `body-var` variable and each `lens-var` variable:

$$\varphi(A) = \begin{cases} 0 & \texttt{if body-var.brand} = \texttt{lens-var.mount-type} \\ \infty & \texttt{otherwise} \end{cases}$$

- In a similar way, a flash is compatible with a camera body if the `brand` of the camera

body and the `brand-compatibility` of the flash are the same, thus:

$$\varphi(A) = \begin{cases} 0 & \texttt{if body-var.brand} = \texttt{flash-var.brand-compatibility} \\ \infty & \texttt{otherwise} \end{cases}$$

There is no direct compatibility rule between a flash and a lens. Since flashes and lenses are always plugged into a camera body, if they are compatible with the body, then they are also compatible between themselves.

In general, the following optimization constraints can be assumed:

- It is supposed that equipments with all the components of the same brand are better than equipments combining several different brands (even if the components are compatible amongst themselves). This criterion implies an optimization constraint involving all variables (n-ary constraint). This constraint can be defined over a valuation function $\varphi(A) \rightarrow [0,1]$ (A is a total assignment):

  $$\varphi(A) = \frac{\texttt{num-brands}(A) - 1}{n-1}$$

  where n is the number of components and $\texttt{num-brands}(A)$ gives the number of brands present in the assignment A. A total assignment that only combines one brand, would have a valuation of 0, *i.e.*, totally satisfies the optimization criteria. In our example, with 4 components, a total assignment that combines 2 different brands would have a valuation of $1 - 2/3 = 1/3$. The worst case (valuation of 1) in regard to this constraint is a total assignment that combines 4 different brands.

- The price of the whole product should always be minimized as much as possible[17]. This criteria can be split in n unary-constraints, one for each component with a `price` attribute. This criterion can be defined over a valuation function $\varphi(A) \rightarrow [0,1]$ (A is an assignment for one variable):

  $$\varphi(A) = \frac{\texttt{val}(A)\texttt{.price}}{\texttt{max-var-price}}$$

  where `max-var-price` gives the maximum price for any value in the domain of the variable.

- One of the factors indicating the quality of a lens is the `max-aperture`: the lower value, the better lens.

Typically, for this example, the above described constraints would remain for different users. Hard constraints ensure the correctness of the products of the catalog whereas optimization constraints rank solutions according to criteria that can be assumed valid for any user.

User's preference about the components are defined through binary constraints. Some examples just to illustrate how to model user's preferences are:

[17] Of course, this constraint is completely in the user's interest. The seller could be interested in a constraint for maximizing the total price, but this depends on the selling strategy and is out of the scope of this thesis.

- Preference about the zoom range of a lens. Normally, a user wants to specify a range zoom [`pref.min`, `pref.max`] to be covered by the lens. Thus, the valuation function, $\varphi(A) \rightarrow [0,1]$ (A is an assignment for a variable `lens-var`), is based on a function `f` that indicates which part of the range is covered by any value in the `lens-var` variable:

$$\mathtt{f}(A) = \frac{min(\mathtt{val}(A).\mathtt{max\text{-}zoom}, \mathtt{pref.max}) - max(\mathtt{val}(A).\mathtt{min\text{-}zoom}, \mathtt{pref.min})}{\mathtt{pref.max} - \mathtt{pref.min}}$$

  and therefore, its valuation is:

$$\varphi(A) = 1 - \mathtt{f}(A)$$

- Valuation functions for preferences about brands can be simply stated as follows:

$$\varphi(A) = \begin{cases} 0 & \mathtt{if\ val}(A).\mathtt{brand} = \mathtt{preferred\text{-}brand} \\ 1 & \mathtt{otherwise} \end{cases}$$

  where the `preferred-brand` is the preferred brand.

- Another example are preferences where the user specifies a maximum price `pref.price` for a component with a `price` attribute:

$$\varphi(A) = max\left(0, \frac{\mathtt{val}(A).\mathtt{price} - \mathtt{pref.price}}{\mathtt{max\text{-}var\text{-}price} - \mathtt{pref.price}}\right)$$

  where `max-var-price` gives the maximum price for any value in the domain of the variable.

3.10 Summary

Firstly, the main notions related to classical constraint satisfaction problems are presented. Modeling electronic catalogs requires to express three types of constraints: configuration constraints which are classical (hard and crisp), and user's preferences and optimization constraints which are soft and flexible. It has been argued that classical constraint satisfaction models are not enough to model the constraints involved in electronic catalogs.

Multi-Criteria Optimization Problems for modeling electronic catalogs by using constraint satisfaction techniques are proposed. Two main different approaches have been analyzed:

The quantitative approach assumes that the preferences can be combined numerically and that the relative importance among them is precisely known. In such cases, the feasible solutions to a catalog can be totally ordered, thus the k best solutions to the problem can be easily identified.

The qualitative constraint combination approach deals with Pareto-optimality for electronic catalogs. The feasible solutions to a catalog are then partially ordered in Pareto-optimal solutions and dominated solutions. Several strategies to choose the k best solutions to a catalog using this approach are proposed.

Finally, an example of an electronic catalog for photo equipment has been used to illustrate the main ideas presented in this chapter.

The solving strategies for the model described in this chapter, are presented in the following chapter.

Chapter 4

Searching Methods for Electronic Catalogs as CSP

4.1 Introduction

In this chapter we propose two different approaches for solving electronic catalogs modeled with the framework provided in Chapter 3.

The first approach (quantitative approach) is based on the well-known branch and bound algorithm. It finds feasible solutions (satisfying the configuration constraints) that minimize the sum of the valuations for all the soft constraints implied in an electronic catalog: user's preferences and optimization criteria. This approach assumes that soft constraints in the problem can be combined together. It implies that valuation functions associated to soft constraints are normalized (and rescaled to differentiate preferences from optimization constraints) and the constraint weight vector is numerically known. The main algorithm together with its main variants are described. Basically, Section 4.2 reviews the existing algorithms to solve partial constraint satisfaction problems and describes the necessary modifications to adapt them in the framework described in Chapter 3, Section 3.7.

The second approach (qualitative constraint combination approach) is based on the Pareto-optimality concept. Solving algorithms to find approximations of the Pareto-optimal set are proposed. Unfortunately, exact algorithms to compute the Pareto-optimal set of a multi optimization problem are too costly, in general, to be applied to interactive electronic catalogs. It has been shown (Chapter 3, Section 3.8.4) that Pareto-optimal solutions in the convex hull are always optimal for a specific constraint weight vector (user profile). Again, the complexity of the existing algorithms to compute the convex hull of a set of feasible solutions with more than 2 criteria, does not allow to apply these algorithms for interactive applications such as electronic catalogs.

4.2 The quantitative approach

Depth First Branch and Bound is a well-known algorithm for solving NP-hard combinatorial optimization problems. It can be seen as the analogous algorithm of the simple

backtracking algorithm [101, 22] but applied to optimization problems. Freuder and Wallace discuss in [79] the needed variations of the classical CSP methods (*simple backtracking*, *retrospective strategies*, and *prospective strategies*) in order to apply them in the context of partial constraint satisfaction. Partial constraint satisfaction is a classical CSP where the goal is to find solutions that minimize the number of violated constraints. In [167], the authors review the approaches for solving over-constrained problems. Both papers, [79] and [167], together with Larrosa's thesis [143] are guides to the algorithms described in this section. Some variations and considerations are given to completely adapt the methods for partial constraint satisfaction problems to our framework described in Chapter 3, Section 3.7.

The goal of the quantitative approach is to find feasible solutions to a WCOP (Definition 3.15, page 50) that minimize[1] their valuation function (Definition 3.14, page 49). The valuation function for an assignment A implying variables $\mathcal{W}$ is recalled in the following expression:

$$\varphi(A) = \sum_{\forall c \in HC} \varphi_c \left(val \left(A \downarrow^{\mathcal{W}}_{scope(c)} \right) \right) + \sum_{\forall c_i \in SC} w_i \cdot \varphi_{c_i} \left(val \left(A \downarrow^{\mathcal{W}}_{scope(c_i)} \right) \right) = \varphi_{HC}(A) + \varphi_{SC}(A)$$

Note that the first term of $\varphi(A)$, $\varphi_{HC}(A)$, indicates if the assignment A is feasible, *i.e.*, if all hard constraints are satisfied. On the other hand, the second term of $\varphi(A)$, $\varphi_{SC}(A)$, indicates *how good* is the assignment. Therefore, this solving approach must find total assignments A with $\varphi_{HC}(A) = 0$ (feasible assignments) that minimize their valuation function with respect to the soft constraints $\varphi_{SC}(A)$.

It has been shown in Property 3.1 that feasible solutions to a WCOP can be totally ordered by a *better than* relation. In the following, a complete algorithm to obtain the *best* solution to a WCOP is described. Also, adaptations to this algorithm to find the k *best* solutions are given. And finally, some improvements to the basic branch and bound based algorithm are proposed.

Unary, binary and n-ary constraints The algorithms presented in the following sections assume that WCOPs only have binary constraints. The reason for this assumption is that algorithms are easier to understand for binary CSPs, and that any non-binary CSP can be transformed into a binary CSP in polynomial time. Section 4.2.7 explains how to deal with non-binary constraints. On the other hand, in electronic catalogs, user's preferences are usually modeled with unary soft constraints (Section 3.9). How to deal with unary constraints is described in Section 4.2.6.

4.2.1 Depth First Branch and Bound (DFBB) algorithm

Depth First Branch and Bound [149, 196, 182, 8] (noted DFBB) explores the search tree associated to the problem using a first-depth search schema. Each internal node represents a partial assignment whereas a leaf represents a total assignment (a solution). The algorithm solves the WCOP by incrementally extending a partial assignment to a total

[1] The maximization case is symmetric and therefore can be easily adapted from the minimization case.

assignment, starting from the empty assignment. At the level k of the associated search tree, variable k is instantiated with its next value. At each node, if the current assignment satisfies all the hard constraints ($\varphi_{HC}(\mathsf{currentAssig}) = 0$), its valuation function with respect to the soft constraints ($\varphi_{SC}(\mathsf{currentAssig})$) is computed. This valuation is called the *Lower Bound cost function* (LB) in branch-and-bound terminology. LB is an underestimation of the valuation function for all the nodes below the current node. It is relevant to note that the lower bound cost function must be non-decreasing along a path of the search tree. This is true in our case since the costs associated to soft constraints are always positive. The *Upper Bound cost function* (UB) is the valuation of the best solution found so far (initially set to ∞[2]). When the current assignment is not feasible or $\mathsf{UB} \leq \mathsf{LB}$, the algorithm is in a *dead-end*. A *dead-end* situation indicates that the current branch cannot lead to a feasible solution or to a better solution than the best solution found so far. Therefore, in a *dead-end* node, all the successors of that node are pruned and the algorithm backtracks to the previous node in order to explore a new branch. When the algorithm reaches a leaf of the search tree and $\mathsf{UB} > \mathsf{LB}$, the current assignment is the best solution found so far. Then, the UB is updated with the valuation function of the solution found, and the algorithm backtracks.

The basic DFBB algorithm for our framework is described in detail in Algorithm 1. Assignments (and therefore also solutions) are expressed as sets of variable instantiations which are noted $\{\mathsf{var} \leftarrow \mathsf{val}\}$. `PreviousVariable` returns `nil` if there are no more previous variables. When `PreviousVariable` returns `nil` and there are no more values to try for the current variable, the whole search tree has been explored and the search terminates.

The conditional **if**s before and after the line 3 of Algorithm 1 could be merged in one single line. However, in some cases, it could make sense to check the feasibility of the assignment currentAssig before computing its φ_{SC} (the lower bound). In this way, if currentAssig is not a feasible assignment, the algorithm can already prune the branch below currentAssig. In other cases, it can be more efficient to test $\mathsf{LB} < \mathsf{UB}$ before testing the feasibility of the assignment. Whether to compute one test before the other one or vice-versa depends strongly on the topology of the problem. The goal for improving the efficiency of branch and bound algorithms is always to detect *dead-end* nodes as soon as possible. Therefore, if the problem is highly constrained with respect to the hard constraints, it is preferable to make the feasibility test before. In the opposite case, the order of the tests will be performed the other way around. In general, one could think about a mixing order of the tests. At the beginning of the search, the UB is likely to be very high, and as long as the search finds new best solutions, the UB becomes lower. In consequence, it could be convenient to start the search applying the hard constraint checking before the soft constraint checking, and swapping this order at some more advanced state of the search process. The effectiveness, however, of such a technique depends strongly on the concrete problem topology.

The computation of $\varphi_{HC}(\mathsf{currentAssig})$ is done by only checking the constraints that

[2] In Section 4.2.5, other initial values for the UB cost function are suggested to improve the efficiency of DFBB algorithms.

imply the last assigned variable of the currentAssig with any of the previously instantiated variables. Constraints implying previous instantiated variables but not the last instantiated variable have already been checked, therefore it is not necessary to check them again. If one of the hard constraints fails, then the computation returns ∞ without regarding any further constraints. If all the hard constraints are satisfied it returns 0. Formally, given a feasible assignment A_k with $var(A_k) = \{V_1, \ldots, V_k\}$, the computation of $\varphi_{HC}(A_{k+1})$ for an assignment A_{k+1} with $var(A_{k+1}) = \{V_1, \ldots, V_k, V_{k+1}\}$ can be expressed as:

$$\varphi_{HC}(A_{k+1}) \rightarrow \begin{cases} 0 & \texttt{if } \varphi_c\left(val\left(A_{k+1}\downarrow^{var(A_{k+1})}_{scope(c)}\right)\right) = 0, \forall c \in \mathcal{HC} \text{ such that} \\ & scope(c) \subseteq var(A_{k+1}) \text{ and } V_{k+1} \in scope(c) \\ \infty & \texttt{otherwise} \end{cases}$$

Similarly, the computation of $\varphi_{SC}(\mathsf{currentAssig})$ is also done retrospectively. Only the soft constraints involving the last instantiated variable and some of the previously instantiated variables are evaluated. These new valuations are then added to the last LB, and the result is the current LB. Formally, given an assignment A_k with $var(A_k) = \{V_1, \ldots, V_k\}$ and $\mathsf{LB}(A_k)$, the updated LB for an assignment A_{k+1} with $var(A_{k+1}) = \{V_1, \ldots, V_k, V_{k+1}\}$ is computed as:

$$\mathsf{LB}(A_{k+1}) = \mathsf{LB}(A_k) \quad + \quad \sum w_i \cdot \varphi_{c_i}\left(val\left(A_{k+1}\downarrow^{var(A_{k+1})}_{scope(c_i)}\right)\right), \forall c_i \in SC \text{ such that} \\ scope(c_i) \subseteq var(A_{k+1}) \text{ and } V_{k+1} \in scope(c_i)$$

Actually, the test $\mathsf{LB} < \mathsf{UB}$ can be done during the computation of LB. In this way, if at any moment of the LB computation, LB is already higher than UB, no more constraints need to be checked, and the test returns false directly[3].

Algorithm 1 can be easily adapted to return the k best[4] solutions instead of only the best one. For that, the first k feasible solutions are kept without computing LB nor UB. When the first k feasible solutions are found, UB is set to the worst valuation (the maximum) of all the k feasible solutions found so far. From that point on, LB is always computed as in the original algorithm. When a new solution is found, the solution of the k best solutions found with a worst valuation is replaced with the new solution and UB is updated with the new worst valuation of the k best solutions. The modifications of Algorithm 1 for computing the k best solutions are shown in Algorithm 2.

[3]Since the costs associated to soft constraints are always positive, the valuation function of an assignment is non-decreasing along a path in the search tree.

[4]*best* in the sense of the relation *better than* defined in Chapter 3, Definition 3.16

Algorithm 1: Depth First Branch and Bound (DFBB) algorithm to find the best solution to a WCOP.

```
function DFBB
    Input: WCOP P = (X, D, HC, SC, W)
1   Output: bestSol: the best solution to P
2   bestSol ← ∅
    currentAssig ← ∅
    LB ← 0
    UB ← ∞
    var ← FirstVariable ()
    val ← FirstValue (var)
    end ← false
    while ¬ end do
        currentAssig ← currentAssig ∪ {var ← val}
        if φ_HC(currentAssig) = 0 then
3           LB ← φ_SC(currentAssig)
            if LB < UB then
                if LastVariable?() then
4                   bestSol ← currentAssig
5                   UB ← LB
                    NewBranch (var, val)
                else
                    var ← NextVariable ()
                    val ← NextValue (var)
            else
                NewBranch (var, val)
        else
            NewBranch (var, val)
6   return bestSol

procedure NewBranch (var,val)
    while LastValue?(val) and ¬ end do
        var ← PreviousVariable ()
        if var ≠ nil then
            val ← CurrentValue (var)
        else
            ; the whole search space has been explored!
            end ← true
    if ¬ end then
        val ← NextValue (var)
```

Algorithm 2: Depth First Branch and Bound (DFBB) algorithm to find the k best solutions to a WCOP.

```
   function DFBB for the k best solutions
      Input: WCOP P = (X, D, HC, SC, W)
             k : the number of best solutions to compute
1     Output: bestSolutions: the k best solutions to P
2     bestSolutions ← ∅
      worstSol ← ∅ ; the worst solution of the bestSolutions
      ⋮
      if |bestSolutions| ≥ k then
         ; LB is only computed if there are k solutions in bestSolutions
3        LB ← φ_SC(currentAssig)
      ⋮
      if |bestSolutions| ≥ k then
         worstSol ← WorstSolution (bestSolutions)
4        bestSolutions ← ReplaceSolution (worstSol, currentAssig)
         worstSol ← WorstSolution (bestSolutions)
5        UB ← φ_SC(worstSol)
      else
         ; the first k feasible solutions are always kept
         bestSolutions ← bestSolutions + currentAssig
      ⋮
6     return bestSolutions
```

It is worth noting that there could be more feasible solutions with the same valuation as the worst solution of the k best solutions found by Algorithm 2. Thus, if the goal is to also keep all the feasible solutions with the same valuation than the worst solution of the k best solutions found, some slight variations must be taken into consideration:

- the condition $\mathsf{LB} < \mathsf{UB}$ just after the line 3, should be replaced by $\mathsf{LB} \leq \mathsf{UB}$.
- the `ReplaceSolution` (worstSol, currentAssig) method in line 4 should be replaced by the following code:

```
if LB = UB then
   bestSolutions ← bestSolutions + currentAssig
else
   ; LB < UB
   bestSolutions ← ReplaceSolution (worstSol, currentAssig)
   if currentAssig is exactly the k best solution in bestSolutions then
      remove all solutions S from bestSolutions such that φ_SC(S) > LB
```

DFBB algorithm has an exponential worst case complexity because in the worst case the algorithm visits all the nodes in the search tree and checks all the constraints involved in the problem. Therefore, strategies to avoid achieving this bound have been proposed in the literature. The following sections review the main techniques to improve the basic DFBB and describes the needed adaptations to apply them to our framework.

4.2.2 Prospective strategies

Prospective strategies (also called *local consistency*, *constraint propagation*, or *look-ahead*) enforce any assignment to be extensible to other variables. Freuder in [76] defines *consistency*, in a general sense, as: *i-consistency* ensures that any consistent assignment involving $i-1$ variables is extensible to include any additional variable resulting in a new consistent[5] assignment involving i variables. Different levels of consistency exist, for example *arc-consistency* [158] or 2-*consistency*, *path-consistency* or 3-*consistency* and so on. Moreover, *i-strong consistency* [77] ensures that a problem is *k-consistent* for all $k \leq i$.

Prospective strategies can be applied as a *preprocessing* technique prior to search or can be used in search algorithms (*hybrid algorithms*). The goal of these strategies is always to prune values that do not meet local consistency. When prospective strategies are applied prior to search, some values are likely to be removed from their domains leading to a simpler but equivalent problem. On the other hand, prospective strategies embedded within search algorithms allow to filter inconsistent values from the domains of future variables and therefore backtrack in a more efficient way.

4.2.2.1 Partial Forward Checking

Forward Checking (FC) for classical CSPs [163, 109] performs constraint checks between the current instantiated variable and future (unassigned) variables. When values from future variables are inconsistent with the current assignment, they are removed because they cannot lead to a solution along the current branch. When a node has no successor, it means that FC cannot continue the search because there are no further consistent values for the next variable, so it backtracks. The added cost of FC compared to simple BT is that when FC backtracks it must restore the values that were previously removed. Actually, FC can be viewed as the complement of simple backtracking: BT checks backward the consistency of the current assignment, whereas FC checks consistency forward to ensure that any value in future variables is consistent with the current assignment.

In [79], Freuder and Wallace adapt classical FC algorithm to the framework of partial constraint satisfaction. The resulting algorithm is called *Partial Forward Checking* (PFC). Mainly, the adaptation arises from the fact that the condition to detect a *dead-end* node in partial CSPs is not the same as in the case of classical CSPs. PFC is very similar to FC in the sense that: when a new node is visited, it checks all the constraints involving the current variable with any unassigned variable. Moreover, in PFC, each value dynamically keeps

[5] A consistent assignment was defined in Chapter 3 as feasible assignment, *i.e.*, an assignment that satisfies all the hard constraints.

track of the number of its inconsistencies that have been detected, called *inconsistency count* (ic). The inconsistency count of values for future variables is used to:

- **Prune values**. A value b for a variable j with a ic_{jb} is removed if $\mathsf{LB} + ic_{jb} \geq \mathsf{UB}$ because it cannot lead to a solution improving the current UB. Larrosa comments in his thesis [143] that the above expression given in [79] can be improved by simply adding the sum of minimum inconsistency counts for each unassigned variable, thus:

$$\mathsf{LB} + ic_{jb} + \sum_{k \in F-j} min_v(ic_{kv}) \geq \mathsf{UB}$$

  where F is the set of unassigned variables.

- **Improve the lower bound**. The computation of the lower bound cost function can be improved by taking into consideration the ic of values for unassigned variables. The improved lower bound, written LB', is computed as follows:

$$\mathsf{LB}' = \mathsf{LB} + \sum_{j \in F} min_v(ic_{jv}) \tag{4.1}$$

  In other words, LB' is an underestimation of the cost that any total assignment that extends the current assignment would have.

The efficiency of branch and bound algorithms strongly depends on the quality of its bounds. Improvements on how to increase the lower bound cost function and how to decrease the upper bound cost function will make the search more efficient by enabling the prune condition $\mathsf{LB} \geq \mathsf{UB}$ early. Several improvements to the basic lower bound of the PFC have been described in the literature. For example, Wallace suggests in [255] to compute *directed-arc consistency* in a process prior to search in order to improve the lower bound cost function. This preprocessing is used to compute *directed arc-inconsistency count* (DAC) for each value of each variable. DAC for a value b of a variable j, dac_{jb}, is the number of variables which are arc-inconsistent with the value b of variable j and appear after variable j in the variable ordering. This preprocessing requires a fixed order of the variables that must be used in the search. Wallace proposes then to improve LB by the following expression:

$$\mathsf{LB} + \sum_{j \in F} min(ic_{jb} + dac_{jb}) \tag{4.2}$$

where F is the set of unassigned variables. PFC using this dac counter is called PFC-DAC and requires a static order of the variables.

In [252], another improvement is proposed, deriving in an algorithm called *Russian Doll Search* (RDS). Other more complex improvements of the lower bound cost function are out of the scope of this thesis, the reader is referred, for example, to Larrosa's thesis [143]. A more recent review on improvements of lower and upper bounds for partial constraint satisfaction is [167]. Other relevant papers about more sophisticated ways to improve lower bounds for partial constraint satisfaction are [2], [12], [95], [145], [147], [146], [148] and [213].

Application to our framework Our model deals with hard and crisp constraints and with soft and flexible constraints. Note that in partial constraint satisfaction (or MAX-CSP) the goal is to find solutions that minimize the number of violated constraints. In our framework, hard constraints cannot be violated (in order to form a feasible solution) and soft constraints have a valuation function as a cost or penalty. The goal is to find feasible solutions that minimize their valuation function with respect to the soft constraints. Algorithm 3 shows the PFC applied to our framework.

The main difference between PFC for partial constraint satisfaction and PFC for our framework is the computation of inconsistency counts. In partial constraint satisfaction, ic_{jb} is the number of inconsistencies for the value b of the variable j with respect to a current assignment. In our framework, ic_{jb} is computed as the *extra* cost of having the value b of the variable j into the current assignment A regarding the set of soft constraints in the problem. In a formal way:

$$\begin{aligned} ic'_{jb} &= \sum_{c_i} w_i \cdot \varphi_{c_i}\left(val\left(\{A+\{j \leftarrow b\}\} \downarrow^{var(A)+j}_{scope(c_i)}\right)\right), \forall c_i \in SC \text{ such that} \\ & scope(c_i) \subseteq \{var(A)+j\} \text{ and } j \in scope(c_i) \end{aligned} \tag{4.3}$$

The computation of ic'_{jb} is done in a cumulative manner. At each propagation, the ic' associated to values for future variables is updated. For a value b for a future variable j, ic'_{jb} is updated with the valuations of soft constraints involving the last instantiated variable and j. Constraints involving previous variables and variable j have already been considered. This guarantees that constraints are only checked once for any value tuple.

`Propagation` procedure in our framework takes into consideration hard and soft constraints, thus it can be split in two tasks:

- Soft propagation: updates ic'_{jb} for any value b in all unassigned variables. A value b for a future variable j is removed if

$$\mathsf{LB} + ic'_{jb} + \sum_{k \in F-j} min_v(ic'_{kv}) \geq \mathsf{UB}$$

  where F is the set of unassigned variables.

- Hard propagation: works as in the case of classical CSPs. The values for unassigned variables which violate any hard constraint with respect to the last instantiation are removed because they would make the assignment infeasible or inconsistent.

Another measure to improve the lower bound (and value prune conditions) for partial constraint satisfaction is *dac* (Equation 4.2). To adapt *dac* to WCOPs, noted dac', the same considerations taken for adapting *ic* must be taken. In the case of WCOPs, unsupported domains cannot be considered but the minimum cost carried by a variable can be computed. As in the case of partial constraint satisfaction, these counts are computed beforehand, prior to search, and it requires a static variable ordering that must be used in the search process. Let us assume a fixed ordering of variables $\mathcal{W} = \{V_1, \ldots, V_n\}$. Then,

dac' for a value b of a variable V_k, dac'_{kb}, can be computed as follows:

$$dac'_{kb} = \sum_{\forall l>k} min_v \left(\varphi_{SC} \left(val \left(\{\{V_k \leftarrow b\}, \{V_l \leftarrow v\}\} \downarrow^{\{V_k,V_l\}}_{scope(c)} \right) \right) \right), \forall v \in dom(V_l) \tag{4.4}$$

The function `UpdateLowerBound` (in Algorithm 3) can be computed by using one of the expressions proposed in the literature (for example, Equation 4.1 or 4.2), but replacing ic_{jb} by ic'_{jb} as given in Equation 4.3, and replacing dac_{jb} by dac'_{jb} as given in Equation 4.4.

`NewBranch` is basically the same procedure as described in Algorithm 1: it backtracks to the previous visited node and selects the next node to visit. `NewBranch` updates the value `end` to finish the search when there are no more values to try for the current variable and `PreviousVariable` returns `nil`, *i.e.*, when the whole search space has been explored. However, each time that `PreviousVariable` is called, the procedure needs to restore the effects of the last propagation (`UnPropagate`). `UnPropagate` procedure consists in:

- restoring the values that were removed after the last propagation (coming either from the soft propagation or from the hard propagation), and

- update `LB` by subtracting the constraint valuations produced by the constraints involving the last assigned variable and all values of any future variables. The *ic* of values for future variables must be restored accordingly, taking into consideration the constraints that involved the last instantiated variable and the future ones.

A new best solution is found when the current assignment has been extended to a total assignment. It has not to verify if the total assignment is feasible or if it is improving the current best solution, because `Propagation` makes sure that any current assignment is feasible and has a better lower bound than the best solution found so far.

The adaptation of Algorithm 3 to compute the k best solutions to a WCOP is very similar than the needed variations of Algorithm 1 resulting in Algorithm 2. Basically, the algorithm must update the bounds according to a set of solutions instead of one single solution. That means:

- when a new solution is found, it must replace the worst solution among the k best solutions found so far, and afterwards

- `UB` is updated with the valuation of the worst solution among the best k solutions.

Algorithm 3: Partial Forward Checking (PFC) algorithm to find the best solution to a WCOP.

```
function PFC
    Input: WCOP P = (X,D,HC,SC,W)
    Output: bestSol: the best solution to P
    bestSol ← ∅ ; currentAssig ← ∅
    LB ← 0; UB ← ∞
    var ← FirstVariable (); val ← FirstValue (var)
    end ← false
    while ¬ end do
        currentAssig ← currentAssig ∪ {var ← val}
        if LastVariable?() then
            bestSol ← currentAssig
            UB ← LB
            NewBranch (var, val)
        else
            Propagation (UB)
            LB ← UpdateLowerBound (LB)
            var ← NextVariable () ; val ← NextValue (var)
            if val = nil then
                NewBranch (var, val)
    return bestSol

procedure NewBranch (var,val)
    ⋮
    UnPropagation ()
    var ← PreviousVariable ()
    ⋮
procedure Propagation (UB)
    updates ic′ for any value in all unassigned variables
    removes future values which ic′ ≥ UB
    removes future values which violates any hard constraint
procedure UnPropagation ()
    restores the values that were removed during the last Propagation
```

4.2.2.2 Preprocessing look-ahead techniques

Preprocessing look-ahead techniques in constraint satisfaction are used prior to search in order to produce an equivalent problem which is easier to solve. In classical CSPs, these algorithms enforce some level of local consistency, typically arc-consistency. However, in some cases, achieving local consistency may degrade the performance of the search process [187, 203]. There is a trade-off between enforcing local consistency and the search

itself [52, 109, 189].

Moreover, the adaptation of local consistency methods prior to search for partial constraint satisfaction problems is not easy. Values can not be removed from the problem because before the search, there is no way to compute an upper bound cost function. However, Freuder and Wallace propose in [79] to keep, for each value, the number of domains that would not be supporting the value, called *arc consistency count*. Then, during the search the arc consistency counts can be used to improve the lower bound cost function. The arc consistency method must be computed once before the search, and it has a time complexity of $O\left(cd^2\right)$, where c is the number of constraints and d the maximum domain size of the variables.

Application to our framework Regarding our framework, look-ahead techniques, as processes prior to search, can be applied with respect to hard constraints. In this way, some values may be removed and therefore the search space can be reduced.

On the other hand, it is not straightforward how to apply preprocessing look-ahead techniques regarding soft constraints. It is not possible to compute the arc consistency counts as in the case of partial constraint satisfaction. All values are supported by the domains. Therefore, the concept of arc consistency counts must be adapted. For a value b of variable j, the equivalent counter of ac, ac'_{jb}, would be the sum, for each variable $k \neq j$, of the minimum cost produced by constraints among value b of variable j and any value of variable k. Formally:

$$ac'_{jb} = \sum_{\forall k \neq j} min_v \left(w_i \cdot \varphi_{c_i} \left(val \left(\{\{j \leftarrow b\}, \{k \leftarrow u\}\} \downarrow^{\{j,k\}}_{scope(c_i)} \right) \right) \right), \forall u \in dom(k), \tag{4.5}$$

4.2.3 Search ordering heuristics

DFBB, as any branch and bound or backtracking algorithm, visits nodes in the order specified by the heuristics to select variables and values. In the given description of the algorithms, these methods are `First/NextVariable` and `First/NextValue`. Different heuristics for such ordering methods may have an important effect in the efficiency of the algorithms, as shown in [52].

4.2.3.1 Static and dynamic ordering

Ordering heuristics can be classified in two main groups:

- Static ordering: establishes an ordering for variables and values before the search. Thus, the search tree structure is fixed prior to search and maintained along the search process.

- Dynamic ordering: makes selection of variables and values during the search. Each time a new node has to be visited, the heuristic chooses a variable and a value. By using dynamic ordering, the search tree structure is not known before the search and it changes during the search.

Static and dynamic orderings can be combined to form more complex heuristics. Typically, when using dynamic ordering, static ordering heuristics are used to break ties.

4.2.3.2 Variable ordering

The goal of variable ordering heuristics for partial constraint satisfaction is to increase the lower bound as quick as possible along a path in the search tree. In this manner, *dead-end* situations are detected early.

Static variable ordering Several static variable ordering heuristics have been reviewed in [167] for partial constraint satisfaction problems:

- *Decreasing backward degree* (BD) [257] establishes an ordering with respect to the number of constraints involving a variable with past variables. First selected variables will then be more constrained regarding past variables, therefore they will have a high inconsistency count, thus a higher lower bound. The drawback for this heuristic is the lack of information at the highest levels of the search tree.
- *Decreasing forward degree* (FD) [145] establishes the complementary ordering of BD: instead of considering past variables it considers future variables. This heuristic tries to propagate inconsistency counts to future variables. It has the opposite drawback of BD: the lack of information at the deepest levels of the search tree.
- *Decreasing degree* (DG) [257] is a combination of BD and FD: at the highest levels of the tree FD is used whereas BD is used at the deepest levels of the tree. Consequently, it intends to avoid the disadvantages of BD and FD.
- *Decreasing AC mean* (AC) [257] heuristic considers first variables with a high *ac*. These variables will likely have a high *dac* as well. Then, when using *dac* in the lower bound cost function, it tends to increase the lower bound function quickly.

These heuristics and their combinations have been tested in [145] which concludes that the best heuristic combination is FD as a first criterion and BD for breaking ties.

Dynamic variable ordering The most used heuristic for dynamically selecting variables is the *minimum domain* (DOM) which selects first variables with small domain sizes. This heuristic is only used when the size of domains are different.

Application to our framework

- **Static variable ordering**: BD, FD and DG consider the number of constraints implied in a variable (with past or future variables) as a measure for the ordering heuristic. In our framework, two kind of constraints exist: hard and soft. Therefore, BD, FD and DG can only consider hard constraints, soft constraints or both of them. When considering soft constraints, it can be convenient to differentiate user's preferences and optimization criteria because they do not usually have the same

valuation ranges. User's preferences should count more than optimization criteria. The same idea can be applied between hard and soft constraints. The efficiency of these different measures depends on the specific topology of the problem.

AC heuristic can also be used in our framework by considering ac' (Equation 4.5).

- **Dynamic variable ordering**: DOM heuristic for dynamic variable ordering can be directly used in our framework since it does not depend on the constraints but on the size of the domains.

4.2.3.3 Value ordering

Few literature exist about value ordering heuristics for partial constraint satisfaction. The goal of value ordering heuristics for branch and bound based algorithms is to select first values that can decrease the upper bound cost function as quick as possible. For that, the most promising values must be selected first. Thus, values are ordered by its increasing ic, $ic+dac$ or $ic+ac$, depending on which counts are kept for values.

Application to our framework When applying value ordering heuristics to our framework, only soft constraints must be taken into consideration. As in the case of partial constraint satisfaction, the goal of value ordering heuristics is to decrease the upper bound as quick as possible. Hard constraints have no effect on the lower bound since upper bound is always computed for feasible solutions that do not violate any hard constraint. Therefore, the order of the values must be independent of hard constraints.

The existing value ordering heuristics for partial constraint satisfaction can be used in our framework. Clearly, these heuristics must use the counters adapted to our framework: ic' (Equation 4.3), dac' (Equation 4.4) and ac' (Equation 4.5).

4.2.4 Retrospective strategies

In [79], Freuder and Wallace propose two retrospective algorithms which are adaptations of the counterparts for classical constraint satisfaction problems: *backjumping* and *backmarking*. The first allows us to backtrack in a more intelligent manner than chronological backtracking whereas the second can avoid some redundant constraint checks.

4.2.4.1 Backjumping

In classical CSPs, *backjumping* (BJ) [87] keeps track of previous failures in the search to behave more efficiently when backtracking occurs. *Backjumping*, as well as any other backtracking based algorithm, backtracks when all the tried values for a variable were inconsistent. The difference with respect to simple backtracking is that BJ *jumps* to the deepest level of the tree that was causing an inconsistency with any of the tried values. The analog BJ for DFBB must take into consideration that the condition to be in a *dead-end* is different. In classical constraint satisfaction, backtracking occurs when all the values for a variable violate some constraints. In partial constraint satisfaction, BJ backtracks

to the deepest level, l, that causes any violation with the current assignment only if the nodes below l do not cause any violation [79]. This is due to the fact that there could be some assignments below the level l that add some constraint violations to the lower bound cost function.

Application to our framework WCOPs have both classical hard constraints and soft constraints. Therefore, DFBB algorithm applied to WCOPs backtracks when the current assignment is too far from the best solution (or the k best solutions) found so far or when the current assignment violates some hard classical constraints. *Backjumping* applied to WCOPs means to consider both cases. In the previous described DFBB algorithms (Algorithm 1 and Algorithm 2) there are two calls to the backtracking method `NewBranch` (var,val). The BJ version would have two different `NewBranch` methods: one for the hard constraint check ($\varphi_{HC}(\mathsf{currentAssig}) = 0$) and the other one for the soft constraint check ($\mathsf{LB}<\mathsf{UB}$).

4.2.4.2 Backmarking

Backmarking (BM) applied to classical CSPs [89], allows the backtracking algorithm to avoid some redundant successful consistency checks, as well some redundant discoveries of inconsistencies. For that, it uses a marking schema for avoiding consistency checks that have been done. [79] describes this technique applied to partial constraint satisfaction problems.

Application to our framework The needed modifications of this technique for solving WCOPs is similar to the one needed for adapting *backjumping*. The algorithm must consider both type of constraints, hard and soft. Therefore two different marking schemas would be needed: one for the classical constraint checks and another one for the computation of the lower bound cost function.

4.2.5 Improving initial upper bound

Initial upper bound can also improve the efficiency of branch and bound algorithm. In [256], Wallace presents some experiments using local search techniques to compute initial upper bound for partial constraint satisfaction problems. The methods were used as anytime algorithms, *i.e.*, after some amount execution time the algorithms are stopped. The cost function of the best solution found by the local search method is taken as initial value for the upper bound in the branch and bound search algorithm. The three tested methods in [256] are:

- *min-conflicts* [171] consists in improving an initial complete assignment by searching through the space of possible repairs. The search is guided by a value ordering heuristic, the min-conflicts heuristic, that attempts to minimize, at each step, the number of constraint violations.

- *break-out procedure* [177] change the weight of constraints in order to escape from local minima.
- *weak commitment search* [264] was proposed by Yokoo as a method that combines the advantages of iterative methods and backtracking methods.

Another way of improving initial upper bound by using *mean field annealing* has been pointed out by Cabon *et al.* in [27].

Application to our framework In our framework, local repair search methods can be used to find a feasible solution and use its valuation as the initial upper bound value. However, if the goal is to find the k best solutions to a WCOP, the local search methods must find k solutions, and the upper bound would be then the worst valuation of these k solutions found. The heuristics mentioned above must be adapted to consider the valuation functions of the soft constraints and not only the number of violated constraints. On the other hand, hard constraints must be taken into consideration in repair methods, since only feasible assignments must be considered.

4.2.6 A note on unary constraints

Often, in electronic catalogs, user's preferences can be naturally expressed using unary constraints. These constraints model preferences of the user about a component of the catalog. Several examples of this kind of preferences for a catalog of photo equipments are described in Section 3.9, Chapter 3. Unary constraints, in electronic catalogs, are usually soft. Actually, a hard unary constraint would just imply to remove the values that do not satisfy it. Unary soft constraints can be checked prior to search. They can be used to initialize the counts to be kept for each value (Equation 4.3, Equation 4.4, or Equation 4.5) that are used to compute the lower bound of the current assignment.

4.2.7 A note on non-binary constraints

Bessière [14] briefly summarizes the state of the art in non-binary classical CSPs and provides recent work in the domain. Larrosa and Dechter [144] present the theoretical equivalence between binary and non-binary semiring-based CSPs. However, transforming non-binary problems into binary problems is not always convenient, and nowadays the interest of solving non-binary constraints directly is recognized [168].

A first approach to handle non-binary constraints of WCOPs in basic branch and bound (Algorithm 1 and Algorithm 2) is to consider that a constraint can only be checked if the current assignment contains all the variables implied in the constraint. For PFC (Algorithm 3), one can consider the propagation of a k-arity constraint only when the current assignment implies $k-1$ of the variables of the constraint. Then, the propagation procedure updates the *count* [6] for each value of the future variable k which is implied in the constraint. Obviously, the drawback of this approach is that consistency checks can only

[6] It can be any of the counts for PFC described in the previous sections.

take place in assignments of k or $k-1$ variables for DFBB or PFC algorithms respectively. This means, that the effect of non-binary constraints is only considered in deep nodes of the search tree, and therefore *dead-end* situations are not likely to be detected early. This approach is the simplest way of handling non-binary constraints because the same algorithms for the binary case can be applied in the non-binary case, but its efficiency can be improved.

In [168], Meseguer *et al.* present a deep study on how to handle non-binary constraints for PFC algorithm in constraint optimization problems. They propose to propagate a k-arity constraint at each node that implies just one variable of the constraint. But, the propagation can not take place for every future variable implied in the constraint, because then its effect would be considered more than one time in the lower bound computation. Thus, the algorithm selects one of the future variables of the constraint in order to compute its new count produced by the constraint [166].

4.2.8 DFBB as an anytime algorithm

An *anytime* algorithm is an algorithm that can provide a solution at any time, and can provide a better solution with more computation [49]. It is easy to see that DFBB produces suboptimal solutions along its execution, and therefore DFBB can give a suboptimal solution at any time of its execution [120]. In [268], Zhang compares DFBB as anytime algorithm (called *truncated* DFBB) with other local search methods applied to the *asymmetric Traveling Salesman Problem* (ATSP) [127]. Zhang concludes that truncated DFBB can outperform local search methods for finding suboptimal solutions. In [268], it is also claimed that this is not a particular and isolated observation only for ATSP. Similar observations have been made for other problem domains, for instance, number partitioning [136], or random coding networks [128].

DFBB-based algorithms (simple DFBB and PFC) can be seen as anytime algorithms. Electronic catalogs, designed as interactive applications, must compute solutions quickly. In catalogs where the complete execution of branch and bound methods is not possible due to the size of the associated search tree, the anytime versions of DFBB or PFC can be considered. For that, a maximum execution time t has to be preset beforehand. Then, when the running time of the search algorithm exceeds t, it stops and gives the suboptimal solution found so far as a result.

4.2.9 Soundness, completeness and complexity

Soundness and completeness Clearly, DFBB and PFC are sound algorithms since regions of the search tree that do not lead to optimal solutions are not considered. Note that the anytime versions of these algorithms are not sound, *i.e.*, they can find sub-optimal solutions, but this is inherent to the nature of any anytime algorithm.

Branch and bound based algorithms are *systematic* because of the following properties [185]:

- do not leave any stone unturned, and
- do not turn any stone more than once.

The first point is equivalent to the concept of completeness, *i.e.*, the algorithm finds either a solution[7] or it concludes that the problem is unsolvable[8]. The described algorithms (DFBB and PFC) are systematic because they explore the whole search space that is potentially containing a solution to the problem. Some parts of the search tree can be pruned if and only if they cannot contain any optimal solution. On the other hand, it is clear that the algorithms do not visit the same node more than once.

Again, the anytime versions of DFBB and PFC are not complete for the same reasons that they are not sound.

Time and space complexity Worst-case time complexity bound for DFBB algorithm is exponential with respect to the number of variables (as simple backtracking algorithm) as, in the worst case, the algorithm will try all the nodes of the search tree and will check all the constraints involved in the problem. The worst-case time complexity for DFBB is $O(d^n)$, where n is the number of variables in the problem and d is the maximum domain size of the variables.

Space complexity for DFBB algorithm and its variants is linear with respect to the number of variables: $O(n)$. In other words, the space complexity is bounded by the length from the root node of the search tree to any leaf. It has the same space complexity as any other search algorithm based on a depth-first search schema.

4.3 The qualitative constraint combination approach

In the qualitative constraint combination approach (Section 3.8.2, Chapter 3), the solutions to a MCOP are the feasible solutions which are Pareto-optimal with respect to the soft constraints.

Pareto-optimal solutions are hard to compute because unless preference criteria involve only a few of the variables, the dominance relation can not be evaluated on partial solutions. Research on better algorithms for Pareto-optimality is still ongoing (see, for example, Gavanelli [91]), but since it cannot escape this fundamental limitation, generating all Pareto-optimal solutions is likely to always remain computationally very hard. Therefore, Pareto-optimality has so far found little use in practice, despite the fact that it characterizes optimality in a more realistic way. This is especially true when the Pareto-optimal set must be computed very quickly, for example in interactive configuration applications (*e.g.*, electronic catalogs). Due to the complexity of computing the whole set of Pareto-optimal solutions [199, 230], methods for approximating the Pareto-optimal set are suggested in the following.

[7] Here, a solution is the set of the k best solutions to a WCOP.

[8] In our framework, a WCOP is unsolvable if there is not any complete assignment satisfying all the hard constraints.

4.3.1 Methods for approximating pareto optimal solutions

To approximate the set of Pareto-optimal solutions, the simplest solution is to simply map the MCOP into an optimization problem with a single criterion obtained by a fixed weighting of the different criteria. In other words, the suggested approach consists in building a WCOP, from the given MCOP, and solving it with any of the methods described in the previous section. Actually, a WCOP is a MCOP with a constraint weight vector (Definition 3.15).

In practice, it turns out that among the k best solutions to a WCOP, many are also Pareto-optimal. Theorem 4.1 shows indeed that the optimal solution of a WCOP is always Pareto-optimal, and that furthermore among the k best solutions all those which are not dominated by another one are Pareto-optimal for the whole problem.

Theorem 4.1: Pareto-optimality of a MCOP from a WCOP.
Given a MCOP P with variables $\mathcal{V}$, and l soft constraints $\mathcal{SC}$. Let be P' the WCOP obtained from P with a constraint weight vector $\mathcal{W} = (w_1, \ldots, w_l), w_i > 0$. Let be $P'_k(\mathcal{W})$ the set of the k best solutions of P'. If $S \in P'_k(\mathcal{W})$ and S is not dominated by any $X \in P'_k(\mathcal{W})$, then S is Pareto-optimal of P.

Proof. Assume that S is not Pareto-optimal of P'. Then, there is a solution $Y \notin B_k(\mathcal{W})$ which dominates solution S, and by Definition 3.18:

$$\begin{aligned} \forall c \in \mathcal{SC}, \varphi_c\left(val\left(Y \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) &\leq \varphi_c\left(val\left(S \downarrow^{\mathcal{V}}_{scope(c)}\right)\right), \text{ and} \\ \exists c \in \mathcal{SC}, \varphi_c\left(val\left(Y \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) &< \varphi_c\left(val\left(S \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) \end{aligned}$$

As a consequence, we also have:

$$\sum_{i=1}^{l} w_i \cdot \varphi_{c_i}\left(val\left(Y \downarrow^{\mathcal{V}}_{scope(c_i)}\right)\right) < \sum_{i=1}^{l} w_i \cdot \varphi_{c_i}\left(val\left(S \downarrow^{\mathcal{V}}_{scope(c_i)}\right)\right)$$

therefore Y must be better than S according to the weighted optimization function (*i.e.*, according to the WCOP P'). But this contradicts the fact that $Y \notin P'_k(W)$ while $S \in P'$. □

From Theorem 4.1, it is easy to see that the optimal (the best) solution to a WCOP (with positive constraint weights) that was built from a MCOP, is a Pareto-optimal solution of the MCOP.

Corollary 4.1: The optimal solution to a WCOP is Pareto-optimal.
Given a MCOP P with variables $\mathcal{V}$, and l soft constraints $\mathcal{SC}$. Let be P' the WCOP obtained from P with a constraint weight vector $\mathcal{W} = (w_1, \ldots, w_l), w_i > 0$. Let be S the best solution of P'. S is Pareto-optimal of P.

Proof. The same proof of Theorem 4.1 can be applied by considering $k = 1$. □

Corollary 4.2: Pareto-optimality of the optimal solutions to a WCOP.
Given a MCOP P with variables $\mathcal{V}$, and l soft constraints $\mathcal{SC}$. Let be P' the WCOP obtained from P with a constraint weight vector $\mathcal{W} = (w_1, \ldots, w_l), w_i > 0$. Let be S the best solution of P'. Any other solution to the WCOP with the same valuation than S is also Pareto-optimal of P.

Proof. The proof is similar to the proof for Theorem 4.1. Assume that a solution S' with the same valuation of S is not Pareto-optimal of P. Then, there is a solution Y which dominates S', and by Definition 3.18:

$$\begin{aligned} \forall c \in \mathcal{SC}, \varphi_c\left(val\left(Y \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) &\leq \varphi_c\left(val\left(S' \downarrow^{\mathcal{V}}_{scope(c)}\right)\right), \text{ and} \\ \exists c \in \mathcal{SC}, \varphi_c\left(val\left(Y \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) &< \varphi_c\left(val\left(S' \downarrow^{\mathcal{V}}_{scope(c)}\right)\right) \end{aligned}$$

As a consequence, we also have:

$$\sum_{i=1}^{l} w_i \cdot \varphi_{c_i}\left(val\left(Y \downarrow^{\mathcal{V}}_{scope(c_i)}\right)\right) < \sum_{i=1}^{l} w_i \cdot \varphi_{c_i}\left(val\left(S' \downarrow^{\mathcal{V}}_{scope(c_i)}\right)\right)$$

therefore Y must be better than S according to the weighted optimization function (*i.e.*, according to the WCOP P'). But this contradicts the fact that S' has the same valuation than the best solution of P'. □

Theorem 4.1 and its corollaries justify the use of WCOPs to find not just one, but a larger set of Pareto-optimal solutions. In particular, by filtering the k best solutions returned by a WCOP algorithm (DFBB or PFC) to eliminate the ones which are dominated by another one in the set, only solutions which are Pareto-optimal for the entire problem are found. Thus, it is possible to bypass the costly step of providing non-dominance on the entire solution space of the problem.

In the following, two main methods for approximating Pareto-optimal sets are proposed: the simple method and the iterative method which is based on several runs of the simple method.

4.3.1.1 Simple method

The first suggested algorithm (Algorithm 4) to approximate the Pareto-optimal set of a MCOP consists in:

- modeling the MCOP with p criteria as a WCOP P with a constraint weight vector[9] $\mathcal{W} = (w_1, \ldots, w_p)$,
- generating the k best solutions of P, and
- filtering them to retain only those which are not dominated.

[9] Usually the unit vector $(1, \ldots, 1)$. However, if the user is able to state some kind of degrees of importance for each criteria, this information can be used in the constraint weight vector.

Algorithm 4: Method for approximating the Pareto-optimal set of a MCOP by using a single WCOP.

function `SimpleMethod`
- **Input**: P: the MCOP to solve.
  - $\mathcal{W} = (w_1, \ldots, w_p)$: the constraint weight vector.
  - k: the maximal number of solutions to compute.
- **Output**: $\mathcal{PO}$: an approximation of the Pareto-optimal set of P.
- $\mathcal{S} \leftarrow$ `SolveWCOP` (`BuildWCOP` (P, $\mathcal{W}$), k)
- $\mathcal{PO} \leftarrow$ `FilterDominated` ($\mathcal{S}$, $\emptyset$)
- **return** $\mathcal{PO}$

function `BuildWCOP` (P, $\mathcal{W}$)
- transforms the MCOP P into a WCOP by considering the constraint weight vector $\mathcal{W}$

function `SolveWCOP` (P', k)
- finds the k best solutions to the WCOP P' by any of the branch and bound methods (DFBB or PFC)

The function `BuildWCOP` is very simple. Given a MCOP and a constraint weight vector $\mathcal{W} = (w_1, \ldots, w_p)$, for each criteria c_i, it transforms its valuation function φ_{c_i} into a new valuation function $\varphi'_{c_i} = w_i \cdot \varphi_{c_i}$.

The function `SolveWCOP` uses one of the branch and bound based algorithms to find the k best solutions to a WCOP.

The `FilterDominated` (described in detail in Algorithm 5) builds a Pareto-optimal set taking into account two sets: $\mathcal{S}$ which is the set of potential new Pareto-optimal solutions, and $\mathcal{PO}$ which is the set of Pareto-optimal solutions found so far. For the simple method, `FilterDominated` is only called with $\mathcal{S}$[10]. Initially, the Pareto-optimal set is composed by the union of all the solutions in $\mathcal{S}$ that have the best valuations (line 1), and the Pareto-optimal set $\mathcal{PO}$. By Corollary 4.2, these solutions are Pareto-optimal for the entire problem. After this initialization, the algorithm removes the Pareto-optimal solutions from $\mathcal{S}$ (line 2), which is the set of solutions to be filtered. At that moment, the solutions in $\mathcal{S}$ which are dominated by any of the solutions of $\mathcal{PO}$ (line 3) can also be eliminated from $\mathcal{S}$ because they are not potential Pareto-optimal solutions. Afterwards, for each solution $s \in \mathcal{S}$, it checks if s dominates some other solutions in $\mathcal{S}$ (line 4). The solutions which are dominated by s can be directly removed because they will not be Pareto-optimal. In the case that, another solution in $\mathcal{S}$ dominates solution s (line 5), solution s is no more considered as a potential Pareto-optimal solution, and the algorithm continues with the next solution in $\mathcal{S}$. When a solution $s \in \mathcal{S}$ is not dominated by any other solution in $\mathcal{S}$ (line 6), solution s is Pareto-optimal.

`FilterDomainated`($\mathcal{S}$) uses a predicate called `dominates`. This predicate just checks all the criteria valuation functions for two given feasible solutions. Given two solutions,

[10] In the iterative method, the function `FilterDominated` is called with the two parameters.

X and Y, X `dominates` Y is true if and only if for all criteria X is better or equally good than Y and for at least one criterion X is strictly better than Y.

Algorithm 5: Method for filtering a set of solutions to get a Pareto-optimal set.

```
   function FilterDominated (S, PO)
1    PO ← PO ∪ {s | s ∈ S and ∀y ∈ S, φ_SC(s) ≤ φ_SC(y)}
2    S ← S \ PO
     foreach po ∈ PO do
       foreach s ∈ S do
3        if po dominates s then
           S ← S \ {s}
     foreach s ∈ S do
       pareto-ok ← true
       foreach y ≠ s ∈ S do
4        if s dominates y then
           S ← S \ {y}
5        else if y dominates s then
           pareto-ok ← false
           break
6      if pareto-ok then
         PO ← PO ∪ {s}
         S ← S \ {s}
     return PO
```

4.3.1.2 Iterative method

The above method (Algorithm 4) has the weakness of generating solutions that are optimal with respect to a certain constraint weight vector and thus likely to be very similar to one another. The iterated *weighted-sums* approach, as described for example by Steuer in [230], attempts to overcome this weakness by calling a WCOP method several times with different constraint weight vectors. Each WCOP will give us a different subset of Pareto-optimal solutions, and a good distribution of constraint weight vectors should give us a good approximation of the Pareto-optimal set.

Basically, the proposed iterative method (Algorithm 6) consists in performing several runs over WCOPs with different constraint weight vectors and one run over the unit constraint weight vector $(1,\ldots,1)$. The method has two parameters:

1. k, which is the maximal number of solutions to be found, and
2. $\mathcal{W} = \{W_1,\ldots,W_p\}$, which is the collection of constraint weight vectors.

Algorithm 6 performs $p+1$ iterations, one for each different WCOP with constraint weight vector W_i, and one for a WCOP with the unit constraint weight vector $(1,\ldots,1)$.

At each iteration, it computes the best $k/(p+1)$ solutions to the corresponding WCOP. At the end of each iteration, dominated solutions are filtered out, so by Theorem 4.1, the resulting set of solutions are Pareto-optimal.

Algorithm 6: Weighted-sums method for approximating the Pareto-optimal set of a MCOP. The collection of p weight vectors $\mathcal{W}$ are generated to give an adequate distribution of solutions.

```
function IterativeMethod
    Input: P : MCOP.
           k : the maximal number of solutions to compute.
           W = {W_1, ..., W_p}: W_i is a collection of weight vectors.
    Output: PO: an approximation of the Pareto-optimal set.
    S ← SolveWCOP (BuildWCOP (P, (1,...,1)), k/(p+1))
    PO ← FilterDominated (S, ∅)
    foreach W_i do
        S ← SolveWCOP (BuildWCOP (P, W_i), k/(p+1))
        PO ← FilterDominated (S \ PO, PO)
    return PO
```

Different distributions of constraint weights A key issue regarding the iterative method for approximating the Pareto-optimal set of a MCOP is how to generate the adequate distribution of constraint weights. In the following, some distributions are suggested:

- **Random**: $m+1$ iterations using m randomly generated weight vectors and the unit vector $(1,\ldots,1)$.
- **One criterion left out**: $l+1$ iterations, one for each criteria and one for the unit vector. The iteration for the constraint C_i, is performed with the constraint weight vector $(w_1,\ldots,w_l)$, where $w_{k=i}=1$ and $w_{k\neq i}=100$. The idea behind this variant is to reduce the effect of one of the criteria at each iteration.
- **Two criteria left out**: $\frac{l(l-1)}{2}+1$ iterations, one for each couple of criteria and one for the unit vector. The iteration for C_i and C_j, is performed with the weight vector $(w_1,\ldots,w_l)$, where $w_{k=i}=1$, $w_{k=j}=1$, and $w_{k\neq i,k\neq j}=100$. The idea behind this variation is very similar to the *one criteria left out* but considering each couple of criteria.
- **One criterion and two criteria left out**: $\frac{l(l-1)}{2}+l+1$ iterations, one for each criterion, one for each couple of criteria, and one for the unit vector. This is the mixed approach of the *one criterion left out* variant and *two criteria left out* variant.

4.3.2 Soundness, completeness and complexity

Soundness and completeness By Theorem 4.1, the simple and iterative algorithms described in this section are obviously sound in the sense that solutions found are really

Pareto-optimal solutions for the problem.

The simple method and the iterative method only approximate the set of Pareto-optimal solutions of a MCOP, therefore both algorithms are not complete.

4.3.2.1 Complexity

The computational complexity of the `dominates` predicate (used in Algorithm 5) is linear with respect to the number of criteria. Given two solutions S and Y to a MCOP P with $\mathcal{SC} = \{SC_1, \ldots, SC_m\}$ criteria, the Y `dominates` S predicate must check (Definition 3.18) if for all criteria Y is better or equally better than S and that for at least one criterion Y is strictly better than S. In consequence, the time complexity in the worst case of the `dominate` predicate is $O(m)$.

Simple method Basically, the simple method performs the two following steps:

1. `SolveWCOP` amounts to solve a WCOP using one of the methods described in Section 4.2. As it was argued in Section 4.2.9, the worst-case time complexity of this step is $O(d^n)$, where n is the number of variables in the problem and d is the maximum domain size of the variables.

2. `FilterDominated` executes the `dominated` predicate

$$|\mathcal{PO}| \cdot |\mathcal{S}| + 2 \cdot \sum_{i=0}^{|\mathcal{S}|-2} (|\mathcal{S}| - i) \cdot (|\mathcal{S}| - i - 1)$$

   times in the worst case. Thus, the worst-case time complexity of `FilterDominated` is[11] $O\left(|\mathcal{S}|^2\right)$.

In consequence, the worst-case time complexity of the simple method is $O\left(d^n + m \cdot |\mathcal{S}|^2\right)$, where n is the number of variables in the problem, d is the maximum domain size of the variables, m is the number of criteria in the problem and $|\mathcal{S}| = k$ is the number of the best computed solutions. In general, $n \gg 2$, thus the worst-case time complexity can be considered exponential with respect to the number of variables in the problem $O(d^n)$.

Iterative method The iterative method performs $p+1$ times the steps `SolveWCOP` and `FilterDominated`. This indicates that the worst-case time complexity for the iterative method is, in general, $O((p+1) \cdot d^n)$, where n is the number of variables in the problem, d is the maximum domain size of the variables, and p is the number of constraint weight vectors.

[11] Observe that $|\mathcal{PO}| \leq |\mathcal{S}|$, thus $|\mathcal{PO}| \cdot |\mathcal{S}| \leq |\mathcal{S}|^2$

4.3.3 Evaluation

Different instances and variations of the described methods for approximating Pareto-optimal sets of MCOPs have been tested for randomly generated MCOPs. Next section, describes how the random problems were generated. Results of the tests are presented afterwards.

4.3.3.1 Random MCOP generation

The topology of a random binary[12] MCOP is defined by: $\langle n, m, hc, ht, sc, st, maxV \rangle$, where:

- n is the number of variables in the problem,
- m is the size of variable domains,
- hc is the graph density in percentage for unary and binary hard constraints,
- ht is the tightness in percentage for disallowed tuples in unary and binary hard constraints,
- sc is the graph density in percentage for unary and binary soft constraints,
- st is the tightness in percentage for unary and binary soft constraints,
- $maxV$ indicates the maximum valuation for soft constraints. Therefore, soft constraint valuations can take values from 0 to $maxV$.

For simplicity, hard and soft constraints are separated and mixed constraints are not considered, therefore $hc + sc \leq 100$. For building random MCOP instances, the variables for each constraint are chosen following a uniform probabilistic distribution. In the same way, we choose the tuples in constraints. Valuations for soft tuples are randomly generated between 0 and $maxV$ and valuations for hard tuples are represented by a maximum valuation (∞).

The algorithms have been tested with different set of problems of soft CSPs with 5 and 10 variables and 10 values for each variable. Hard unary/binary constraint density hc has been varied from 20% to 80% in steps of 20, and the tightness for hard constraints ht varies also from 20% to 80% in steps of 20. Soft unary/binary constraint density sc has been varied from 20% to 80% in steps of 10, with tightness fixed at $st = 100$. In the case of 5 variables, in total there could be $5 + (5 * 4)/2 = 15$ soft constraints (5 unary constraints and 10 binary constraints). In the case of 10 variables, in total there could be $10 + (10 * 9)/2 = 55$ soft constraints (10 unary constraints and 45 binary soft constraints). Observe that hard and soft constraint graph densities are varied according to the expression $hc + sc \leq 100$. Thus, for instance, when $hc = 40\%$, sc varies from 20% to 60%.

For every different class of problems, 50 different instances were generated, and each instance has been tested with the simple method (with the unit vector as a constraint weight vector) and the iterative method. The iterative method has been evaluated with

[12] A binary MCOP is a MCOP with unary and binary constraints.

Hard tightness	Number of feasible solutions
20%	51,121
40%	21,792
60%	6,444
80%	778

Table 4.1: The number of of solutions in average for generated problems with 5 variables, 10 values per variable and 20% of hard unary/binary constraint density.

the different proposed distributions of the constraint weight vector (see Section 4.3.1.2). The methods have been tested varying the number of total solutions to be computed from 30 to 530 in steps of 50, from 530 to 2,030 in steps of 100, from 2,030 to 10,030 in steps of 1,000 and from 10,030 to 20,030 in steps of 10,000.

4.3.3.2 Results on random problems

Tests were performed on a PC Pentium III 600Mhz with 256 MB RAM, and the algorithms were implemented in Java. The implemented search algorithm to find the best k solutions to a WCOP is PFC. For the different problem topologies, the average of the results for each problem topology are evaluated in the following section.

Firstly, it is of interest to know how many Pareto-optimal solutions there are in a problem depending on the number of criteria (soft constraints). In Figure 4.1, it is shown that the number of Pareto-optimal solutions clearly increases when the number of criteria increases. The same phenomena applies for instances with 5 and 10 variables. Actually, this result is in accordance with Lemma 3.2 provided in Section 3.8.2.

On the other hand, we have observed that even if the number of Pareto-optimal solutions decreases when the problem gets more hard-constrained (less feasible solutions) the percentage with respect to the total number of solutions increases (Figure 4.2). Thus, the proportion of the Pareto-optimal solutions is clearly more important when the problem gets more constrained.

Figure 4.3 shows the proportion in average of Pareto-optimal solutions found by the different methods for problems with 6 soft constrains, 10 variables, 10 values per variable, 40% of hard constraint density and 40% of hard constraint tightness. The different evaluated methods are:

- the simple method with the unit constraint weight vector,
- the iterative method, with the following variations:
  - with $p = 2, 4, 6, 10, 14, 18$ randomly generated constraint weight vectors.
  - one criteria left out (see Section 4.3.1.2),
  - two criteria left out (see Section 4.3.1.2),
  - one and two criteria left out (see Section 4.3.1.2),

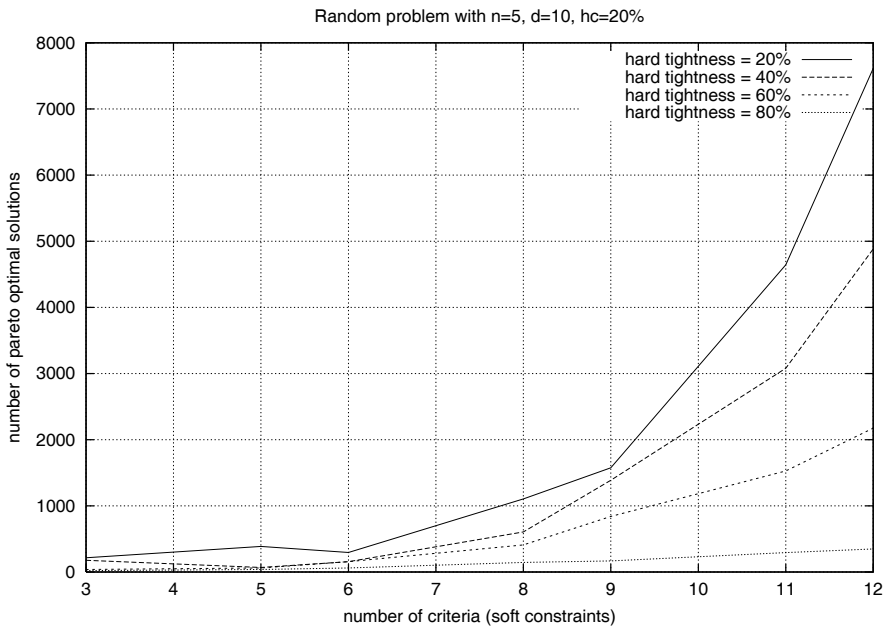


Figure 4.1: Number of Pareto-optimal solutions depending on the number of soft constraints for random generated problems with 5 variables, 10 values per domain and 20% of hard unary/binary constraint density. The results, in average, are shown for problems with hard tightness of 20%, 40%, 60% and 80%. The total number of solutions in average for these problem topologies are presented in Table 4.1.

- the lexicographic fuzzy CSP model. The interest of evaluating lexicographic fuzzy CSP model is that numerous properties and algorithms exist for such a CSP model [17]. Unfortunately, for approximating Pareto-optimal solutions, this model showed very poor results compared to the other methods.

The evaluation of the methods are shown up to 530 solutions because in real applications it could not be feasible to compute a larger set of solutions. When computing up to 20,030 solutions, the behavior of the different methods does not change significantly, see Figure 4.4. The 50 randomly generated problems used for Figure 4.3 and Figure 4.4 had in average 134,661 feasible solutions (satisfying hard constraints) and 991 Pareto-optimal solutions. The iterative methods perform better than the simple search algorithm with respect to the total number of solutions computed. It is worth to note that the iterative methods find more Pareto-optimal solutions when the number of iterations increase.

Lexicographic Fuzzy method results in finding a very low percentage of Pareto-optimal solutions (less than 2%). Note that, for the lexicographic fuzzy CSP, Theorem 4.1 does not apply, thus the percentage shown of Pareto-optimal solutions is computed *a posteriori* by filtering out the Pareto-optimal solutions that were not really Pareto-optimal for the entire problem.

Another way of evaluating the different methods is to compare the number of Pareto-optimal solutions found with respect to the computing time. Figure 4.5 shows the results

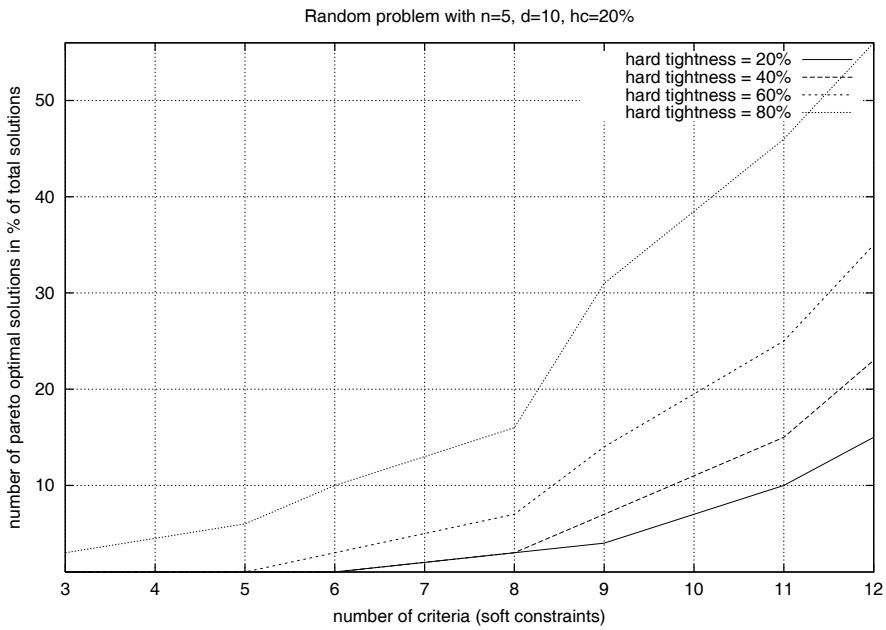


Figure 4.2: Number of Pareto-optimal solutions in percentage of the total number of solutions depending on the number of soft constraints for random generated problems with 5 variables, 10 values per domain and 20% of hard unary/binary constraint density. The results, in average, are shown for problems with hard tightness of 20%, 40%, 60% and 80%.

up to 5 seconds. Using this comparison, the simple method performs the best. The performance of the variants of the iterative method decreases when the number of iterations increases. This is due to the fact that the iterative methods must perform p search process over the whole search tree.

In general, it can be observed that when the number of iterations of the methods increases, the performance regarding the total number of computed solutions also increases but the performance regarding the computing time decreases. This behavior is similar for the other topologies of randomly generated MCOPs. This is due to the fact that the computing time of finding the k best solutions with a branch and bound algorithm is not linear with respect of finding the k best solutions with m iterations (k/m solutions per iteration). For example, computing 1,030 solutions with one iteration took in average for our problems 0.3114 seconds and computing 1,030 solutions with 7 iterations (of 147 solutions) took 1.049 seconds.

Even if the tests based on the iterative method takes more time than the simple method for getting the same percentage of Pareto-optimal solutions, they are likely to produce a more representative set of the Pareto-optimal set.

Using a brute force algorithm that computes all the feasible solutions and filter out those which are dominated, took on average 12.23 seconds for the same problems as in the above figures. This demonstrates the interest of using approximative methods for computing Pareto-optimal solutions, especially for interactive configuration applications

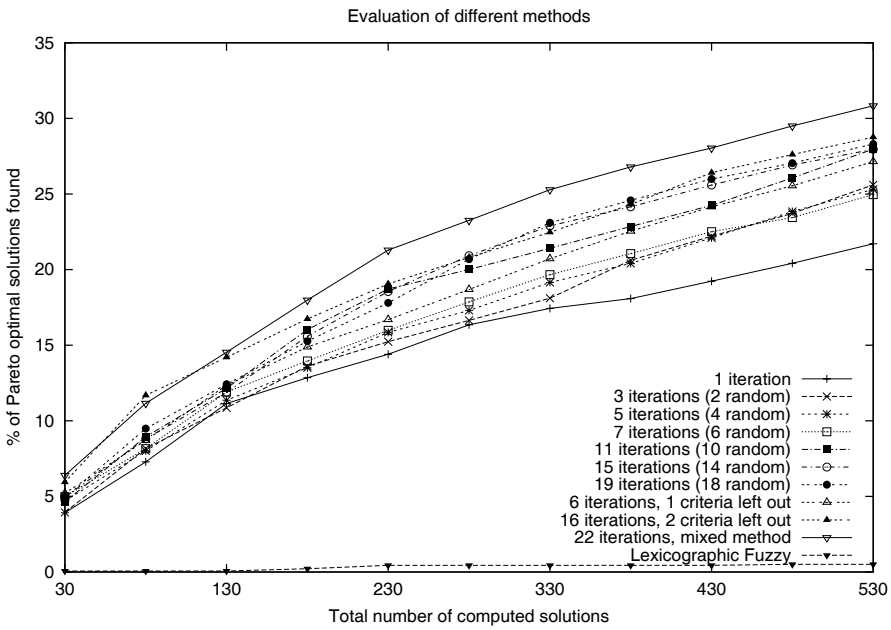


Figure 4.3: Pareto-optimal solutions found by the different proposed methods (in %). Methods are applied to 50 randomly generated problems with 10 variables, 10 values per domain, 40% of density of hard unary/binary constraints with 40% of hard tightness and 6 criteria (soft constraints). The number of total computed solutions for each method varies from 30 to 530 in steps of 100.

(*e.g.*, electronic catalogs).

4.3.4 Solutions on the convex hull

Ideally, the best solutions to a MCOP for the user are the Pareto-optimal solutions on the convex hull (see Section 3.8.4). Unfortunately, due to the complexity of computing the solutions on the convex hull when having more than two criteria ($O(n^{\lfloor n/2+1 \rfloor})$, see [226]), computing the Pareto-optimal solutions which are on the convex hull is not feasible for interactive applications. However, in some cases it could be feasible to compute the Pareto-optimal solutions on the convex-hull, for example, in systems where the user states the problem, expresses his preferences and checks the results after some time. This can be done in non-interactive systems. In Section 4.5.2.5, the main existing algorithms to find the convex-hull of a set of points are reviewed.

4.4 Converging to satisfactory solutions

As discussed in Chapter 2, Section 2.2.2, the goal of our mixed-initiative system is to converge to a reduced set of satisfactory solutions as long as the user expresses his own preferences. In Section 2.2.2, two different mechanisms for achieving that goal were proposed, namely: by ranking and by reducing the set set of solutions to be shown to the

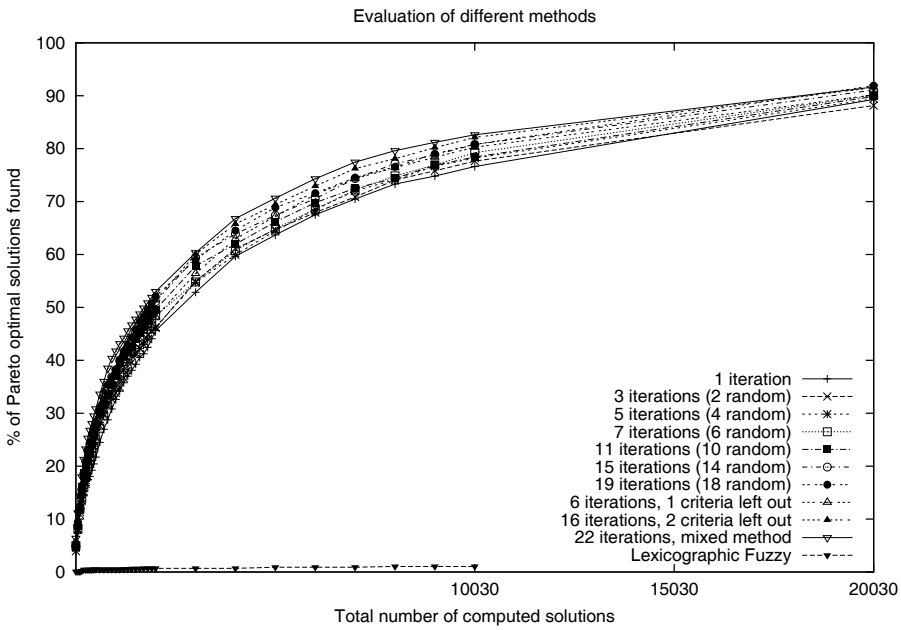


Figure 4.4: Pareto-optimal solutions found by the different proposed methods (in %). Methods are applied to 50 randomly generated problems with 10 variables, 10 values per domain, 40% of density of hard unary/binary constraints with 40% of hard tightness and 6 criteria (soft constraints). The number of total computed solutions, for each method, varies from 30 to 1,030 in steps of 1,000.

user. In the following the behavior of the quantitative and the Pareto-optimal approaches is analyzed in order to see if when adding criteria, *i.e.*, soft constraints, to a problem the satisfactory set of solutions is reduced.

4.4.1 The quantitative approach

Experiments on randomly generated problems have been performed in order to find out the distribution of the best solutions in terms of their optimality. The experiments were done in problems with 10 variables, 10 values per variable and 40% of density of hard unary/binary constraints with 40% of tightness. The number of criteria have been stated to 3, 5, 7, 9, and 12. For each different topology, 20 randomly problems were solved by branch and bound to find the best 1,000 solutions. The valuations of the 1,000 best solutions were normalized on the range [0..1], in order to be able to compare the results. The valuation of a solution s, $\varphi_{SC}(s)$, was normalized by the following expression:

$$\varphi'_{SC}(s) = \frac{\varphi_{SC}(s) - \varphi_{SC}(bestSol)}{\varphi_{SC}(worstSol) - \varphi_{SC}(bestSol)}$$

where *bestSol* is the solution with the lowest valuation among the 1,000 best solutions, and *worstSol* is the solution with the highest valuation. This normalization was necessary to compute the average of the results obtained for each instance.

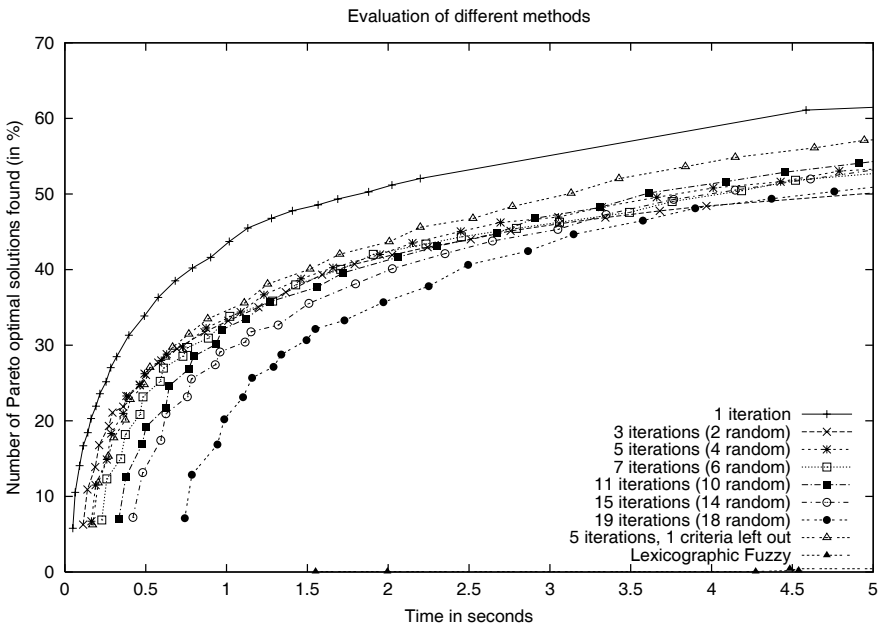


Figure 4.5: Number of Pareto-optimal solutions found by the different proposed methods with respect to the computing time. For this plot, the problems have 10 variables, 10 values per domain with 40% of hard unary/binary constraints with 40% of hard tightness and 6 criteria (soft constraints).

num. of criteria	num. of acceptable solutions
3	255
5	150
7	105
9	87
12	60

Table 4.2: The number of of solutions, in average, under the 0.5 of the quality range of the 1,000 best solutions for problems with different number of criteria.

Figure 4.6 clearly shows that solutions' optimality degrades faster for problems with more criteria. Figure 4.7 shows the same plot as in Figure 4.6 but for the 200 best solutions to ease the analysis of the results.

In other words, when the number of criteria increases, the set of the most optimal solutions is reduced. For example, let us fix a bound of acceptable solution quality at the half of the quality of the 1,000 best solutions (0.5). With this quality bound, Table 4.2 gives the number of acceptable solutions, in average, for problems with different number of criteria. Clearly, when the number of criteria increases, the number of acceptable solutions decreases. This means that as long as the user expresses his criteria, satisfactory solutions are reduced, thus the solutions converge to a small set of satisfactory solutions.

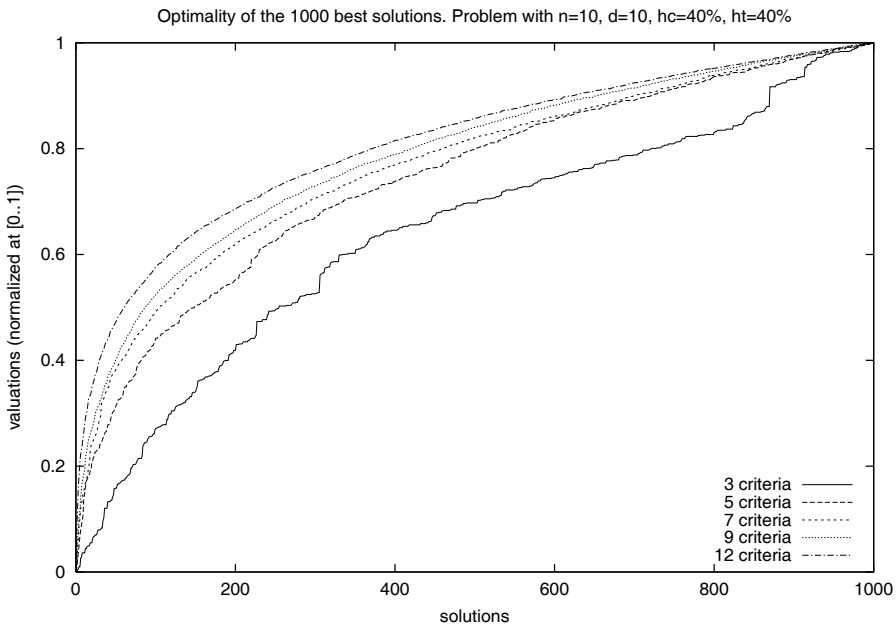


Figure 4.6: The optimality (valuation normalized on the range [0..1]) of the best 1,000 solutions. The valuations are computed on average for 20 randomly generated problems with 10 variables, 10 values per variable, 40% of density of hard unary/binary constraints with 40% of hard tightness. The number of criteria have been stated to 3, 5, 7, 9, and 12.

4.4.2 The qualitative constraint combination approach

In regard to the qualitative constraint combination approach, the results are completely different to those obtained for the quantitative approach. This behavior can be observed in Figure 4.1: the number of Pareto-optimal solutions increases with the number of criteria in the problem.

This behavior of the Pareto-optimality is a drawback for our general approach. It would not be acceptable to show more and more solutions to the user as long as he expresses his preferences. This conduct would definitely be very contraindicative with the user's intuition.

In the next subsection a mixed approach is suggested to overcome the aforementioned drawback of the Pareto-optimality approach and, at the same time, benefit from its main advantage.

4.4.3 A mixed approach

As it has been shown above, the main advantage of the quantitative approach is that best solutions are decreasing when the number of criteria increases. This was a requirement for the suggested mixed-initiative approach described in Chapter 2, Section 2.2.2. On the other hand, the main advantage of the Pareto-optimality approach is that dominated solutions are filtered out because they should never be rationally chosen by any user.

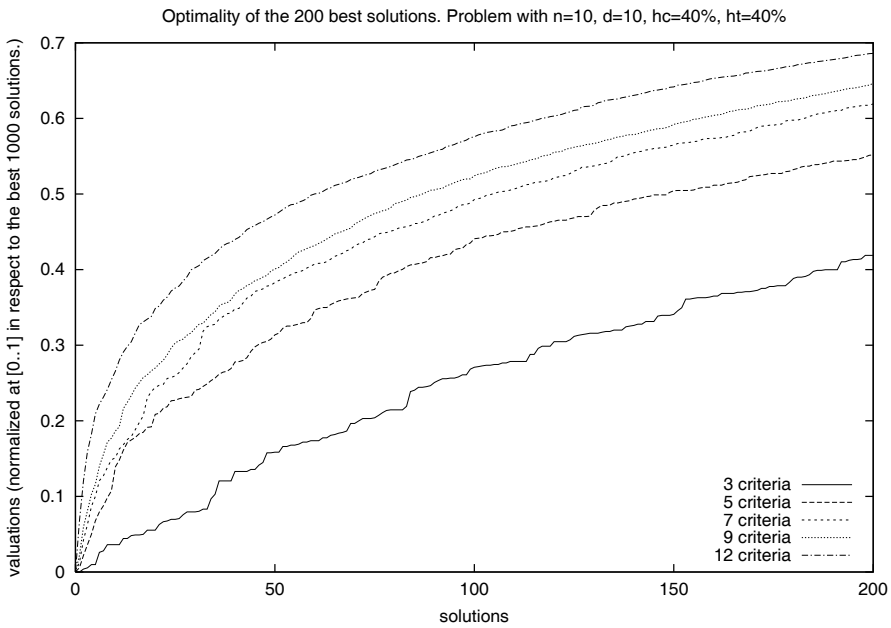


Figure 4.7: The optimality (valuation normalized on the range [0..1]) of the best 1,000 solutions. The valuations are computed on average for 20 randomly generated problems with 10 variables, 10 values per variable, 10% of density of hard unary/binary constraints with 40% of hard tightness. The number of criteria have been stated to 3, 5, 7, 9, and 12.

Combining both approaches is straightforward: solutions are computed using the quantitative approach, and dominated solutions are filtered out. It is worth noting that in Chapter 3, Section 3.8.5.2, this method was already mentioned.

4.5 Related work

4.5.1 The quantitative approach

The quantitative approach deals with finding the best k solutions to a WCOP. This approach is also known as a single objective optimization problem. The optimization function is usually stated as a scalar function that combines all the criteria (soft constraints). Basically, the existing algorithms can be classified in:

4.5.1.1 Systematic search

The main systematic search techniques to solve a single objective optimization problem (namely DFBB and PFC) and their variants have already been described in Section 4.2.

4.5.1.2 Local search

Greedy local search methods are incomplete, *i.e.*, they find suboptimal solutions without knowing how far the solution found is from the optimal one. However, they have been

successfully used due to their efficiency. Basically, local search algorithms start from a initial solution and try to improve it by searching neighborhood solutions. Meseguer *et al.* in [167] review the main local search methods as follows:

Min-Conflicts algorithms were initially proposed by Minton *et al.* in [170, 171]. Several improvements of the initial algorithm have been proposed incorporating more efficient search techniques [258, 256]. Min-conflicts algorithm explores the neighborhood of the current assignment by choosing a promising value for any of the variables which is in conflict with a constraint. The efficiency of local search algorithms strongly depends on the size of the neighbors to be considered. To avoid visiting some of the neighbors, Fast Local Search [247] has been proposed by Tsang *et al.* . It basically tries to avoid neighbors which are unlikely to be promising by guiding the search with heuristics. GENET is a connectionist approach [48] for solving CSPs, based on the Min-conflicts algorithm. By means of a constraint weighting system, it escapes from local minima. The weights of the constraints which are violated in a local minima are increased allowing the algorithm to escape from the local minima.

Genetic algorithms (GA) [99], based on the ideas from evolution [115]. The idea behind GA algorithms is to evolve a population of solution candidates. Depending on a metric (the objective function in optimization problems), the candidates have a degree of chance to continue in the population. GA algorithms have been applied to the context of constraint satisfaction (see [58, 202, 33]). In the case of constrained optimization problems, the key issue is how to incorporate the information from the constraints into the genetic metric function, in order to guide the search in an appropriate way. In [38], the authors propose the usage of penalty functions.

Simulated Annealing (SA) [130] has also been applied to solve constrained optimization problems. At the beginning of the search the moves to visit neighbors are all accepted regardless of whether they are uphill or downhill. As long as the search advances, the moves are more accepted uphill than downhill. In [152], Li proposes a variant of the SA algorithm in which the search space is divided into disjoint subspaces. These subspaces are then approached by localized SA algorithms. Different strategies have been proposed. For example, Wha and Wang in [254] present a framework where the SA algorithm is combined together with the Lagrange multipliers theory. This last approach has been applied to both discrete and continuous problems.

Approximating methods In Section 4.2.8, the anytime interpretation of branch and bound algorithms was presented. Cabon *et al.* , present in [12] three different adaptations of the systematic search algorithms to produce anytime lower bounds algorithms, namely:

- *Problem simplification*: a lower bound is computed by solving a simplified problem systematically [95].

- *Objective simplification*: a lower bound is computed by considering a simpler objective.
- *Search simplification*: a lower bound is computed by performing a limited search tree and exploiting subproblem lower bounds (see [12] for search simplification algorithms).

4.5.2 The qualitative constraint combination approach

Our qualitative constraint combination approach can be seen as a more general and well-studied topic called *Multi-criteria combinatorial optimization problems*[13] (MOCO). See [250, 156, 57] for complete reviews on this topic. A MOCO is defined in [57] as a discrete optimization problem, with n variables x_i, $i = 1, \ldots, n$, Q objectives[14] z_j, $j = 1, \ldots, n$ and a specific constraint structure defining the feasible set of solutions to be considered. The complexity of MOCO problems has been shown to be NP, see [59, 219]. In our framework, the constraint structure is defined by means of hard constraints (configuration constraints), and the objectives are defined through user's preferences and optimization criteria. In the following, the main existing methods for solving MOCO problems are reviewed. Basically, such algorithms can be classified depending on their solutions:

- **The whole Pareto-optimal set**. These methods find the *whole* Pareto-optimal set, *i.e.*, they are **systematic algorithms**.
- ***Supported* Pareto-optimal solutions**. These algorithms are able to find all the *supported* Pareto-optimal solutions. A *supported* Pareto-optimal solution is a solution that is optimal for some weighted sums of the objectives[15]. These methods are exact in the sense that they find solutions that are Pareto-optimal, but they are incomplete in the sense that they do not find the *whole* Pareto-optimal set.
- **An approximation of the Pareto-optimal set**. These algorithms (which are based on Metaheuristics[16]) find solutions that are likely to *represent* the Pareto-optimal set but they are not necessarily Pareto-optimal. An interesting review on such techniques is the dissertation of Hansen [108]. Basically, these algorithms are based on **local search algorithms** and **genetic algorithms**. Local search methods use one single Pareto-optimal solution candidate at anytime, while population based methods derive a set of candidate solutions directly to an approximation of the Pareto-optimal set.

As seen in Section 3.8.4, Pareto-optimal solutions on the convex-hull of the Pareto-optimal set are very interesting because they are optimal for some user profiles (*i.e.*, some

[13] Special solving methods exist for bicriteria problems, but they are not discussed in this section since our interest is focused on problems with more than two criteria.

[14] Objectives are usually considered as linear functions.

[15] These solutions are those which are in a concave part of the efficient frontier in the case of minimization (convex in the maximization case) [230].

[16] *i.e.*, heuristic principles that are not problem specific but are applicable to a large range of problems.

concrete criteria weighting). A review on the state of the art of methods for finding the convex-hull of a set of points in the n-space is presented in Section 4.5.2.5.

4.5.2.1 Systematic search algorithms

The following methods find *all* Pareto-optimal solutions. Surprisingly, not much work has been done to solve a MOCO using systematic search algorithms. To our knowledge, Gavanelli has presented the first branch and bound based method for solving multi-criteria combinatorial optimization problems.

Gavanelli's algorithm Gavanelli addresses in [90, 92, 91] the problem of multi-criteria optimization in constraint problems directly. His method is based in a branch and bound schema where the Pareto dominance is checked against a set of previously found solutions using *Point Quad-Trees* (see [66] for a description of Point Quad-Trees and [105] for an explanation of how to use such data structure applied to the domination concept). Point Quad-Trees are useful for efficiently bounding the search. However, the algorithm can be very costly if the number of criteria or if the number of Pareto-optimal solutions are high. Gavanelli's method significantly improves the approach of Wassenhove-Geders [259]. The Wassenhove-Geder's method basically consists of performing several search processes, one for each criteria. Each iteration takes the previous solution and tries to improve it by optimizing another criteria. Using this method, each search produces one Pareto-optimal solution, so many search processes must be done in order to approximate the Pareto-optimal set.

The Normal-Boundary Intersection Method Das and Dennis [46] have identified two main problems with parametric scalarizing methods (see Section 4.5.2.2), namely:

- they only produce Pareto-optimal solutions in convex[17] Pareto-optimal frontiers, and
- there is no guarantee that the generated Pareto-optimal solutions are spread along the Pareto-optimal frontier, even if the parameters are varied in a well distributed manner.

Following the above two observations, Das and Dennis propose a new method called *Normal-Boundary Intersection* (NBI) [47, 44, 43]. The authors argue that the NBI method is able to produce an evenly distributed set of points in the Pareto set given an evenly distributed set of parameters.

The NBI method has been successfully applied to large assembly systems [42] and to error digital communications problems [45].

4.5.2.2 Methods for finding *supported* Pareto-optimal solutions

The following methods find *supported* Pareto-optimal solutions. Actually, these methods can be solved several times by changing some parameters. At each execution, a new

[17] Convex for minimization functions, concave for maximization functions.

Pareto-optimal solution is found.

Parametric scalarization methods The most commonly used approach for solving a Multi-criteria Optimization Problem is to convert the problem into a single-criterion optimization problem by means of a scalarizing function. In other words, solving a MCOP, using a parametric scalarization method, amounts to transform the problem into a COPs[18] which can be solved using standard single-criteria optimization techniques. Varying the parameters of the scalarizing function, several COPs can be obtained from the original MCOP. The optimal solution to each COP is then a Pareto-optimal solution of the MCOP. Many researches have proposed different scalarizing functions and parameters. Steuer's book [230] gives a deep study on different ways to map a multi-objective optimization problem into a single objective optimization problem. The drawback of these methods is that some Pareto-optimal solutions cannot be found if the efficient frontier is not concave[19].

Given any point $z = (z_1, \ldots, z_p) \in \mathbb{R}^p$, an ideal (or reference) point $z' = (z'_1, \ldots, z'_p) \in \mathbb{R}^p$ (which is optional) and a vector of weights[20] $w = (w_1, \ldots, w_p)$, several parametric scalarizing functions can be defined [167], namely:

- the weighted sum is the most popular scalarizing function:

$$g(z,w) = \sum_{i=1}^{P} w_i(z_i - z'_i)$$

- the weighted Chebychev scalarizing function:

$$g(z,w) = max_i\{w_i(z_i - z'_i)\}$$

  or the augmented weighted Chebychev scalarizing function:

$$g(z,w) = max_i\{w_i(z_i - z'_i)\} + \varepsilon \sum_{i=1}^{P} w_i(z_i - z'_i)$$

  where ε is a small number greater than zero.

- the l_p-norm function: given $p \geq 1$,

$$g(z,w) = \left(\sum_{i=1}^{P} w_i \left(z_i - z'_i\right)^p \right)^{\frac{1}{p}}$$

  Note that when $p = 1$, this function is equivalent to the weighted sum scalarizing function. When $p \notin \{1, \infty\}$, the resulting objectives become nonlinear.

[18] A Constraint Optimization Problem (COP) is a WCOP where the constraint weight vector is encoded in the optimization functions.

[19] In the case that the optimization function is a minimization function, convex if the optimization function is a maximization function.

[20] Usually, the vector of weights meet the following conditions: $\forall i w_i \geq 0, \sum_{i=1}^{P} = 1$ and is then called normalized weight vector.

- other scalarizing functions:

$$g(z,w) = (1-\alpha)\, g_c(z,w) + \alpha/k\, g_w(z,w)$$

where g_c is the Chebychev function, and g_w the weighted sum.

The methods that have been suggested in this chapter for the qualitative constraint combination approach are basically based on this approach, more precisely the weighted sum approach.

The Hierarchical method and Trade-off method consists in ranking the optimization criteria in order of importance [230]. Hence, a single objective optimization problem is carried out for the most important objective subject to additional constraints on the other objectives. These additional constraints are stated as allowable values for each other objective. Then, by changing the most important objective and the allowable values for the other objectives, one can get several Pareto-optimal solutions. This method is also called the ε-*constraint method*. The method is further explained in [180, 155, 28, 119].

The Global Criterion method tries to solve a MCOP as a COP where the criterion to be optimized is a distance function to an *ideal* solution z'. The ideal solution is precomputed by optimizing each criteria independently. Then, a single objective optimization problem can be solved by minimizing [180]:

$$g(z) = \sum_{i=1}^{P} \left(\frac{z'_i - z_i}{z'_i} \right)^p$$

where p defines the type of distance. For instance, Boychuk and Ovchinnikov [24] have suggested $p = 1$, and Salukvadze [206] has suggested $p = 2$.

Other distance functions can be used for this method which is further detailed in [266, 267].

Goal Programming approach amounts to defining the objective functions as goals[21] with priorities or weights [230, 122]. Then, measure functions give information about how much a goal has been achieved. The overall goal is to minimize the deviation from the specified goals. For example, minimizing objective z_i is transformed into two constraints: $z_i - d_i = t_i$, $d_i \geq 0$, where t_i is the goal to be achieved. In this way, the problem is transformed in a problem in which all the d_i have to be minimized, and therefore it can be solved by any of the previous methods. For more details on goal programming applied to MOCO problems the reader is referred to [30, 151, 150, 121].

Interactive methods were first described in Steuer's book [230]. The system interacts with the user in order to find a satisfactory solution. Interactive methods are based on

[21] Usually, the decision maker specifies the goals.

well-known interactive methods for the continuous case. Shin reviews the main interactive methods for the continuous case in [223]. Such methods include objective articulation throughout the solving process. The objectives are not stated at the beginning but defined by the decision maker along the solving method. They are often called *progressive* methods. In [125], Jaszkiewicz lists the following interactive methods: Zionts method [269], Korhonen, Wallenius and Zionts method [138], Köksalan, Karwan and Zionts method [133], Korhonen method [137], Malakooti method [162], Taner and Köksalan method [236], AIM method [154], Light Beam Search-Discrete method [126] and Interquad method [235].

4.5.2.3 Local Search methods

The following methods find *approximate* Pareto-optimal solutions. The goal of local search methods aims to find solutions which are *optimal enough* to solve the problem efficiently. Such methods yield a good tradeoff between the quality of an approximation of the Pareto-optimal set and computing requirements.

Simulated Annealing (SA) algorithms deal with probabilities for accepting a neighbor solution. The main idea of SA algorithms applied to MOCO problems is to always accept a neighbor solution if it dominates the current solution, and accept a neighbor solution with a given probability if it is dominated by the current solution. Obviously, the key point is how to deal with neighbor solutions which do not dominate the current one neither are dominated by the current one. Sarafini [220, 221] suggests transition probabilities, considering two rules: a) only dominating neighbors should be accepted with probability one (strong rule), and b) only dominated neighbors should be accepted with probability less than one (weak rule). Tests showed that the best approach is however to use a composite of the two rules.

Ulungu in [248, 251, 249] proposed a SA based algorithm for MOCO problems called MOSA (Multiple Objective Simulated Annealing). In this approach, the same probability transitions as in the Sarafini's approach is used, but the search is repeated with different weights, in order to generate an approximation of the whole Pareto-optimal set. Experiments for the knapsack and assignment problems only concerning two criteria have been reported.

Another approach that uses SA, proposed by Czyzak and Jaszkiewicz in [39, 40], is called PSA (Pareto Simulated Annealing). PSA consider a sample of solutions which are moved to the Pareto-optimal frontier instead of just considering one single solution. There is also a dispersion mechanism in which the solutions are moved away from the other solutions by means of calculating different weights for each solution. This mechanism will likely produce an approximation uniformally spread along the Pareto-optimal frontier.

Tabu Search (TS) [97, 98] is a local search method which repeatedly moves from a current solution to the most promising neighbor solutions. TS escapes from local minima by keeping a tabu-list of forbidden neighbors.

The first approaches for solving MOCO problems with tabu search [41, 114] consist in solving several single optimization problems and therefore, getting one *supported* solution at each execution.

Gandibleux *et al.* state in [85] the basic principles of TS algorithms to be applied in the context of approximating the Pareto-optimal set. This first adaptation of TS for solving MOCO problems is called MOTS (Multiple Objective Tabu Search) [86]. In MOTS, neighbor solutions are chosen among the one with the highest value of a weighted scalarizing function. The weight vector is changed in such a way that the objectives which are greatly improved are degraded. The tabu-list is used to prevent to visit already checked neighbors and to change the weights of the scalarizing function. In [107], a version of the MOTS algorithm considering a set of solutions is presented.

Hybrid local search algorithms Recently, simulated annealing and tabu search have been combined by Alves and Climaco in [4], resulting in a new algorithm.

4.5.2.4 Genetic algorithms

Genetic algorithms (GA) are based on the theory of evolution. The pioneer GA based method for multicriteria optimization was the VEGA (Vector Evaluated Genetic Algorithm) approach presented in [211] by Schaffer. Mainly, GA algorithms, applied to the MOCO problem, consists in maintaining a population of solutions via self adaptation and cooperation. Self adaptation means that the individuals evolve independently while cooperation implies an exchange of information among the candidate solutions. The idea is simply to group those solutions in the population which are not dominated by any other, into the rank 1. From the solutions which are not in rank 1, those which are not dominated by any other are grouped into rank 2. This grouping process is done until all the individuals (solutions) are assigned in one rank. A function based on the ranking (the fitness function) can then be used in selection for reproduction.

Niching principles have been widely used in genetic algorithms to achieve solution diversity [100]. The interpretation of the niching principle for MOCO problems is to reproduce a set of well distributed solutions along the Pareto-optimal frontier instead of just covering a small area of it. For a deep study on niching methods, please refer to the PhD thesis of Mahfoud [161].

Many different variants of the GA algorithm for solving MOCO problems have been proposed[22], namely: Multiple Objective GA (MOGA) [67], Non-dominated sorting GA (NSGA) [228], Niched Pareto GA (NPGA) [117], and GA based on a min-max strategy [35, 36].

In MOGA, the ranking concept is modified. Each solution is assigned to the rank 1 plus the number of solutions in the population that dominate it. For more details, see [11, 178, 69, 70]. NSGA follows the suggestion of Goldberg and uses the niching principle. NPGA is based on the binary tournament selection. It uses randomly generated

[22] At `http://www.lania.mx/~ccoelo/EMOO`, Coello lists more than 320 papers on genetic algorithms for multiobjective optimization.

comparison sets. The winner of the tournament is the individual that is not dominated by any of the individuals in the comparison set. When there is no dominance in a tournament, the niched principle is used as a tiebreak by choosing solutions which are far away in the objective space.

For further information about GA algorithms applied to MOCO problems, please refer to the surveys of Fonseca and Fleming [68] and Horn [116].

Hybrid local search and genetic algorithms Tabu search algorithm and genetic algorithm have been combined in [1], resulting in a new algorithm for multicriteria optimization problems. The algorithm has been evaluated for knapsack problems. The thesis of Knowles deals with local-search and hybrid evolutionary algorithms for Pareto optimization [131].

4.5.2.5 Pareto-optimal solutions on the convex-hull

This problem, can be seen as a geometric problem in which the goal is to find the smallest convex set covering the whole set of points in a p-space. Unfortunately, due to the complexity of computing the solutions on the convex hull when having more than two criteria ($O(n^{\lfloor n/2+1 \rfloor})$, see [226]), computing the Pareto-optimal solutions which are on the convex hull is not feasible for interactive applications.

The problem of finding Pareto-optimal solutions in the convex-hull for a given MCOP can be restated as finding the convex-hull of a set of points in the m-Space, where m is the number of criteria of the MCOP. Consider each solution $S_i \in \mathcal{S}$ as a point p_i in the m-Space $\mathbb{R}^m$ (Definition 3.22):

$$p_i = \left(\varphi_{C_1}\left(val\left(S_i \downarrow^{\mathcal{V}}_{scope(C_1)}\right)\right), \ldots, \varphi_{C_m}\left(val\left(S_i \downarrow^{\mathcal{V}}_{scope(C_m)}\right)\right)\right)$$

The *convex-hull* of the set of solutions to a MCOP is the smallest convex set that contains all the points p_i representing a solution $S_i \in \mathcal{S}$, and it is noted $conv(\mathcal{S}) \subseteq \mathcal{S}$.

This problem is an open problem in the computational geometry field and has been studied for many years. Among all the existing methods[23], the *Quickhull algorithm* is the most popular method (see for example [9]).

Incremental algorithms for the convex-hull problem have also been widely used. They tread repeatedly a point in the set together with the already processed points. Of particular interest is the *Beneath-Beyond algorithm* [103]. Another type of incremental methods is called *randomized incremental algorithms*, firstly proposed by Clarkson and Shor in [34].

In [217], Seidel reviews the main existing solving methods for computing the convex-hull of a set of n-dimensional points. Several implementations of the main methods[24] for for solving the convex-hull problem in two and three dimensions can be found on the web, for example see `http://www.cse.unsw.edu.au/~lambert/java/3d/hull.html`.

[23] Specialized methods for the bidimensional case have been developed, but are not applicable, in general, to our framework.

[24] Incremental, gift wrap, divide and conquer, and Quickhull algorithms

4.6 Summary

This chapter explores solving techniques for the two approaches presented in Chapter 3:

- **Solving techniques for the quantitative approach** assumes that all the soft constraints involved in the problem can be numerically combined together in a single scalar function. Relative importance of the different criteria to be optimized are stated by means of a constraint weight vector. The main techniques for finding the k best solutions to a WCOP have been described. Such techniques are based on the well-known branch and bound algorithm, namely: the DFBB and the PFC. Basically, the existing techniques have been reviewed and adapted to suit our model.
- **Solving techniques for the qualitative constraint combination approach** deals with the concept of Pareto-optimality. Since computing directly the Pareto-optimal set to a MCOP can be too costly for interactive applications (*e.g.*, electronic catalogs), methods for approximating the Pareto-optimal set to a MCOP are suggested. Some evaluation has been done to validate the methods.

Finally, related work is presented.

Chapter 5

The *SmartClient* Architecture

5.1 Motivation

An important issue which has to be faced in information systems is *scalability*: the ability to support large numbers of simultaneous users. Client-server computing allows such scalability by distributing the computational load to the client computers. The concept of *thin* clients has extended client-server computing to much larger systems, in particular the Internet. Internet is the most widely used platform for electronic commerce, however more local computer networks can be considered such as LANs[1] and WANs[2]. Most of the current electronic catalogs are server-centric, using customer's computers for just rendering results given by servers. In such standard models, the persistence layer, business logic layer and most part of the presentation layer are implemented on the server side.

Since most problem-solving methods are either compute- or knowledge-intensive, providing such functionality poses severe scalability problems. In information systems that have to serve thousands of users with small response times, the time that can be allotted to each individual user is very small, however, the evolution of the distributed computing technologies has been enlarging the potential use of client's computers by:

- speeding up data (and programs) transmission between servers and clients, and
- enabling the execution of programs on the client side which are sent through the web, *e.g.*, via applets or plug-in programs for browsers. It is also worth noting that personal computers and other personal devices (such as mobile phones and PDAs[3]) are becoming quickly more and more powerful[4] in respect of computing process and memory capacities.

[1] LAN stands for Local Area Network.

[2] WAN stands for Wide Area Network.

[3] PDA stands for Personal Digital Assistant.

[4] `http://www.intel.com/labs/eml`: Expanding Moore's Law. Gordon Moore predicted in 1965 that transistors on a chip would double every year. In 1970 the Moore's Law was revised, from 1 year to 2 years for doubling the transistor's density. Lately, the Moore's Law has been updated, and it is believed that each 18 months the transistor's density is doubled. Anyhow, the power of processors is increasing exponentially while their production costs are decreasing.

Traditional wisdom would say that in order to make a client intelligent, it will have to include a lot of complex code and data, *i.e.*, be a very *fat* client. The key point exploited in *SmartClient*[5] architecture is that by using constraint satisfaction techniques (as described in Chapter 3 and Chapter 4), one is able to develop smart (intelligent) clients for electronic catalogs that are also thin.

5.2 Software architectures

This section gives basic definitions related to software architectures, and identifies the architectural topics which are of interest in this work.

5.2.1 Definitions

There is little agreement among researchers about what exactly should be included in the definition of software architecture. In this section, the definitions given by Fielding in his PhD dissertation [65] are recalled[6]. This terminology and definitions will be assumed throughout the rest of the chapter.

Definition 5.1 (Software Architecture) A *software architecture* is defined by a *configuration* of architectural elements (*components*, *connectors*, and *data*) constrained in their relationships in order to achieve a desired set of *architectural properties*.

Definition 5.2 (Component) A *component* is an abstract unit of software instructions and internal state that provides a transformation of data via its interface.

Definition 5.3 (Connector) A *connector* is an abstract mechanism that mediates communication, coordination, or cooperation among components.

Definition 5.4 (Data) A *datum* is an element of information that is transferred from a component, or received by a component, via a connector.

Definition 5.5 (Configuration) A *configuration* is the structure of architectural relationships among components, connectors, and data during a period of system run-time.

Definition 5.6 (Architectural Properties) The properties of a software architecture can be functional or non-functional:

- **Functional properties** are given by the system requirements. In other words, functional properties of a software architecture indicate if the whole system achieves the functionalities it was designed for.

- **Non-functional properties** are quality attributes of the architecture such as ease

[5]SmartClient architecture is US patent pending.
[6]See [65] for insight details, justifications and related research about these definitions.

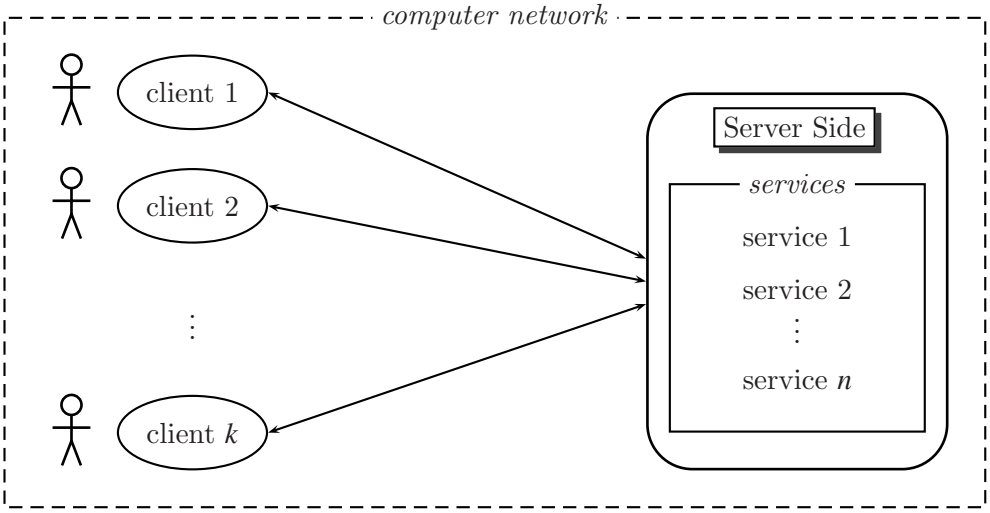


Figure 5.1: Basic client-server architecture. Clients simultaneously access services offered by a server through a computer network.

of evolution, reusability of components, efficiency, portability, reliability, and scalability.

5.2.2 Client-server architectures

Client-server architectures are commonly used in network-based applications. Users access [7] functionalities (a set of services) at the same time, from remotely located computers or other devices, such as as mobile phones and PDAs, through a computer network, as shown in Figure 5.1.

A client-server architecture can be briefly described as in [65]:

- A **server** component, offering a set of services, listens for requests upon those services. A server is usually a non-terminating process and often provides service to more than one client simultaneously.
- A **client** component, desiring a service to be performed, sends a request to the server via a connector. The server either rejects or performs the request and sends the response back to the client component.

5.2.3 Stateful vs stateless servers

In client-server architectures, the state, also called session state, of a server can be defined as the information that it maintains about the status of ongoing interactions with every specific client. Two different approaches must be considered: client-stateless-server and client-stateful-server.

[7] Note that the term *server* will be referred throughout this chapter as an abstract component, but it can be composed of a battery of physically different server computers.

Client-stateless-server architectures do not keep any information about the session state on the server side. Therefore, each request from the client must contain all the needed information to process the request. Session state is entirely kept on the client side. The main disadvantages of such stateless-servers are:

- requests must be processed from scratch without considering the context of previous client requests. This implies that services offered by the server must be context-independent.
- network performance may be decreased, because of the repetitive data to be sent within requests from clients. This is due to the fact that repetitive data along a series of requests cannot be stored on the server side.

On the other hand, client-stateless-server architectures benefit from the following advantages:

- facilitate the **monitoring** of the system without requiring to analyze beyond a request datum, since requests contain all the needed information to be understood.
- improve the **reliability** of the system, since requests that fail can be relaunched without any other considerations such as context or previous requests.
- improve the **scalability** of the system, since no information is kept on the server side, less memory requirements are needed for each request.

Client-stateful-server architectures keep the information about the session state on the server side. Client-stateful-server architectures loose the advantages of having a stateless server, *i.e.*, these architectures are more difficult to monitor, less reliable and less scalable. On the other hand, requests to the server component are usually smaller, since they do not need to transfer the state information which is kept on the server. Another advantage is that business logic for processing requests can, in general, be more sophisticated taking into account the information kept on the server session.

5.2.4 Layered architectures

A layered architecture is defined by Garlan and Shaw in [102] as an architecture which is organized hierarchically, each layer providing service to the layer above it and serving as a client to the layer below. In some layered systems inner layers are hidden from all except the adjacent outer layer. In consequence of such architectural organization, several desirable properties are reached, namely:

- to support design based on increasing **levels of abstraction**. This allows developers to implement different parts of complex systems independently. Therefore, layers can be implemented by different developer teams that only look at the interfaces of the above and below layers, without considering the implementation details of the rest of the system.

- to ease **enhancements** of functional properties of the system. Since layers are only directly connected to their adjacent layers, enhancing the functionality of one layer, only affects at most the layers immediately below and above.
- to support **reuse**. Layer implementations can be interchanged without any major consideration, but just ensuring that they provide the required functionality defined by the interface of the layer with its adjacent layers.

Layered architectures cannot always be applied to all kinds of systems, because some systems cannot be logically structured in different layers. The efficiency of layered architectures directly depends on the number of its layers, *i.e.*, on the different levels of abstractions. Each couple of layers defines an interface and a communication mechanism, thus having many layers can lead to inefficient architectures. As often happens in computer science, when designing a software system, there is a trade-off between efficiency and the desirable aforementioned properties. Note that the layered structure of the system is not directly related to the distribution of the layers to client and server sides. One could have, for example, a part of a layer in the client and another part of that layer in the server. In information systems literature, two types of standard layered architectures exist: two-tier and three-tier.

5.2.4.1 Two-tier architectures

Two-tier architectures emerged around 1980s to improve file server software architectures, where files were stored in servers and could be accessed directly by remote clients. The main improvement of two-tier architectures over file server architectures was to provide a user-friendly interface on the client side to access the file information stored on the server side, instead of directly accessing the files. The layer on the client side is called presentation layer[8], and the layer on the server side is called persistence layer[9]:

$$\boxed{\text{presentation} \Longleftrightarrow \text{persistence}}$$

5.2.4.2 Three-tier architectures

Three-tier architectures, also known as three layer architectures or multi-tier architectures, emerged in the 1990s to overcome the limitations of two-tier architectures. In three-tier architectures, a new layer between persistence and the presentation layers was introduced. This intermediate layer is called business logic layer[10] .

As any layered architecture, layers communicate with above and below layers. Thus, in the case of a three-tier architecture, layer communication is as follows:

$$\boxed{\text{presentation} \Longleftrightarrow \text{business logic} \Longleftrightarrow \text{persistence}}$$

[8]Different terminology is used for the presentation layer, for instance, user interface layer.

[9]Different terminology is also used for the persistence layer, for instance, data management layer, backend layer or data access layer.

[10]Different terminology exist for the business logic layer, such as middle layer, management services layer and application server layer.

5.2.5 Topics of interest

Software architecture, on its own, is a field of research in computer science. Nevertheless, this thesis only attacks some of the issues related to software architecture applied to electronic catalogs. Basically, this chapter deals with the issue of how to design software architectures that enable to:

1. support the user by **solving problems** and not just providing information,
2. build **mixed-initiative** systems, such as the one described in Chapter 2, and
3. build **scalable** systems.

Specifically, the topics of interest about designing an architecture for electronic catalogs include: server services, layer organization and client-server distribution.

In regard to the services offered by the server for electronic catalogs, only the service of supporting the user in finding the right product will be faced. Services providing functionality such as shopping baskets, purchase statistics or payment methods are out of the scope of this thesis. Communication mechanisms between the different layers are also out of the scope of this work, and thus they will not be discussed. In addition to these issues, technology insights about implementation topics are also out of the scope of this chapter.

5.3 Standard architectures for electronic catalogs

Most of the current implementations of electronic catalogs are based on standard layered client-server architectures, with persistence, business logic and presentation layers, as shown in Figure 5.2:

- the **server side** implements the following layers:
  - **persistence layer** which access the internal or external databases or other information systems, such as web sites and legacy systems, taking into account the request of the user.
  - **business logic layer** which generates solutions from the data received in the persistence layer.
  - **presentation layer** which formats solutions in order to be rendered by the client's computer, *e.g.*, in HTML[11] pages. The component of this layer communicates with its counterpart in the client side. Formated solutions are sent to the client side.
- the **client side** is only used to get requests from users and display the solutions sent by the server. This component on the client side is considered as part of the **presentation layer** of the whole system.

[11] HTML stands for HyperText Markup Language.

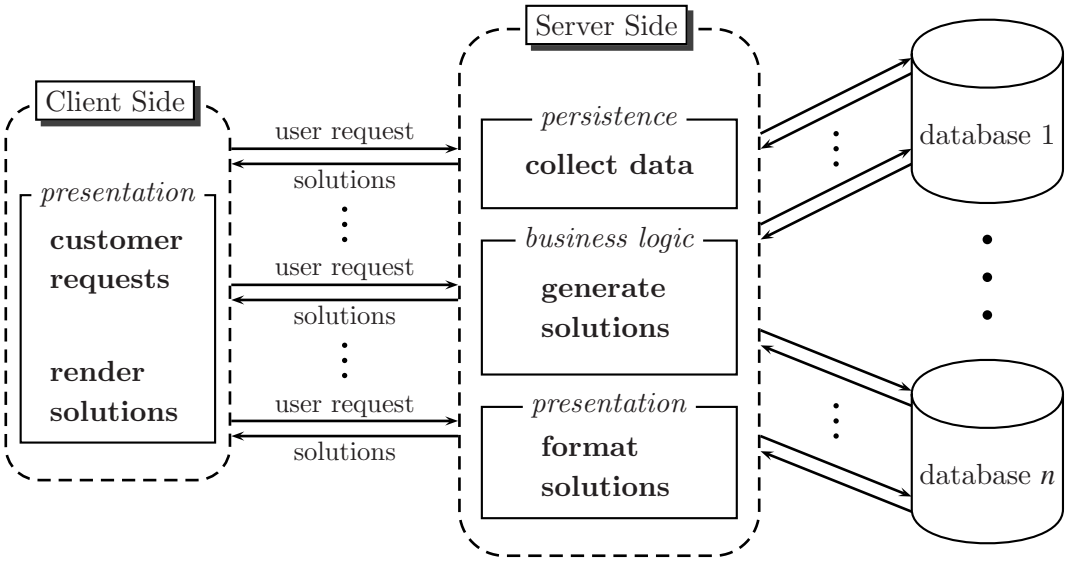


Figure 5.2: The standard client-server architecture for electronic catalogs. Lines between the client side and the server side show the interactions, *i.e.*, data transfers through the computer network. Almost all the processing is done on the server side, whilst the client side is only used for rendering results.

5.3.1 Purchase decision making process

The support for the user in his purchase decision making process in standard architectures for electronic catalogs is very low.

On one hand, in standard catalogs with **non-configurable products**, the system basically shows a small set of products according to some initial set of user criteria. In the case the user does not like any solution, a function like *next solutions* allows him to receive another set of solutions. If the user wants to go back to the previously shown set of solutions, he can use the *previous solutions* functionality.

On the other hand, in respect to catalogs with **configurable products** there are mainly two ways of supporting the user to find the right product configurations:

- **one-step** configuration: the system proposes complete configured products according to the user request and his preferences. This methodology can be used for simple configurable products, *i.e.*, products with few components or few options per component. Non-configurable products can be seen as simple configurable products with one single component.

- **sequential** configuration: the system supports the user in configuring his best product, by selecting one choice for each component in a sequential order. Complex configurable product catalogs are usually faced with sequential configuration processes. See Section 1.3.1 for examples of such complex configurable products.

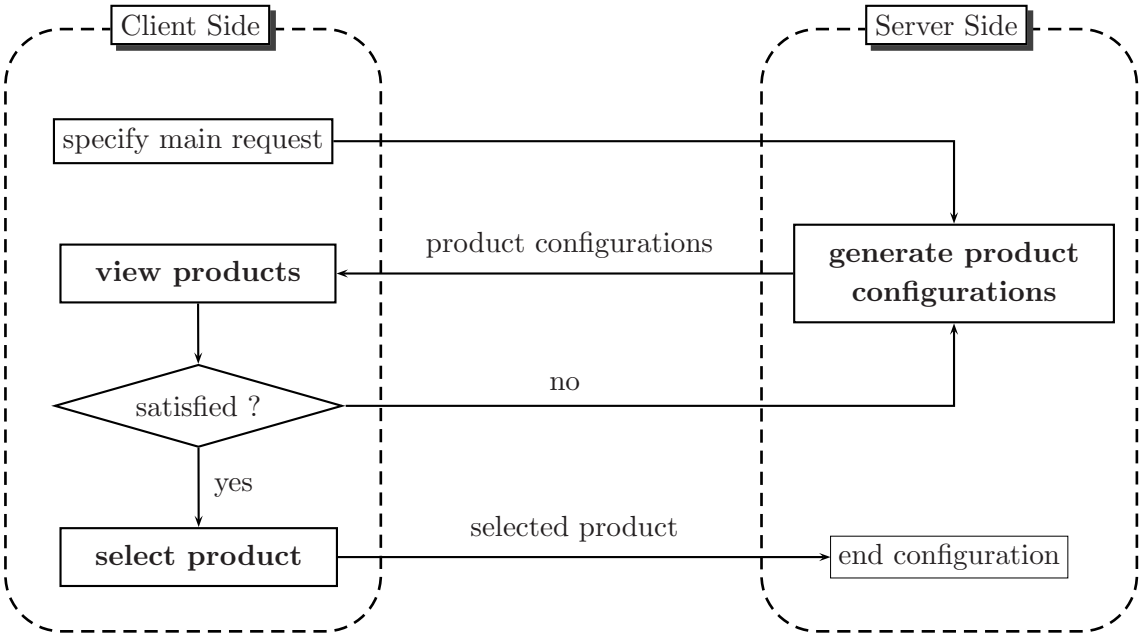


Figure 5.3: One-step decision making process for configurable products in standard electronic catalog architectures. A loop implying the server side and the client side allow the user to browse through product configurations until he finds the right product.

5.3.1.1 On-step configuration

Electronic catalogs with one-step configurators can be applied to products which are simple to configure, *i.e.*, without a huge combinatorial explosion. As shown in Figure 5.3, the system consists of a loop where the user receives a set of complete product configurations. If the user does not find the right product, more configurations are proposed, usually through a *next products* mechanism. In the same way, the user can also go back to the previous set of solutions. Clearly, this methodology can be applied to catalogs with atomic products as well. Preferences can also be taken into account. This functionality is normally embedded into a *refine* mechanism that proposes other solutions according to the user's criteria. Both functionalities, *next products* and *refine*, are depicted in Figure 5.3 as the *generate product configurations* component on the server side.

5.3.1.2 Sequential configuration

In sequential configuration systems, the system suggests to the user to select each component of the product sequentially. Each choice of the user restricts the choices for next components. These restrictions are made according to the configuration rules in order to produce valid product configurations[12]. Due to the sequentiality of the selection process of the choices for the components of the product, the user realizes his mistakes once the system proposes the choices for the next component. Therefore, when the user receives

[12]See Section 1.2.3 for more details on the configuration task.

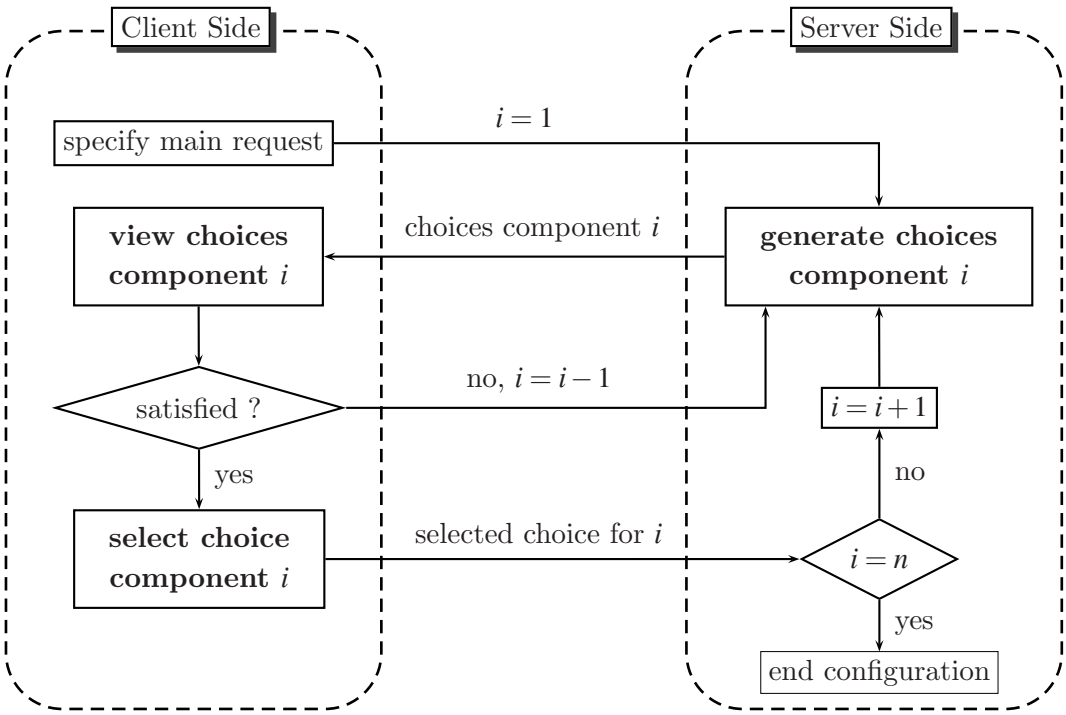


Figure 5.4: Sequential decision making process for configurable products with n components, in standard electronic catalog architectures. A loop implying the server side and the client side allows the user to configure the product step by step, *i.e.*, at each step a component of the product is selected. When the user realizes he made a mistake in selecting a choice for component i, the server proposes again the choices for that component.

the choices for the component i and he realizes (actually he guesses) that a mistake on the selection of another component j was done, the process must be restarted from the wrong choice of component j. Usually, these configurators offer mechanisms to go back to previous component ($j = i - 1$) selection, therefore several back-steps may be needed. At the end, the user hopefully gets a complete configuration of the product. Electronic catalogs for configurable products using such a decision making process are, for instance, Travelocity[13] in the travel domain, and Audi CarConfigurator[14] in the car industry.

As indicated before, the client side is only used for presenting the results to the user without any relevant business logic process, which is located at the server side. In Figure 5.4, the aforementioned processes are depicted for configurable products, *i.e.*, for complex catalogs. Such decision making processes are sequential, at each step a choice for a component of the product is selected. Since choices for different components are interrelated, by configuration rules, users are forced to go back and forth to different components. In consequence, at each step of the configuration process, the *generate solutions* component on the server side is requested.

[13] `www.travelocitiy.com`.
[14] `www.audi.ch`, in French, German or Italian.

5.3.2 Stateless server vs stateful server

In Section 5.2.3, basic concepts about client-stateless-server and client-stateful-server architectures were given. This section discusses the advantages and disadvantages of architectures with stateless or stateful servers for electronic catalogs. As explained before, there are mainly two kind of electronic catalogs in regard to the configuration process: one-step and sequential processes. In the case of one step configuration, the data returned by the server is a set of product configurations. In the case of sequential configuration, the data returned is a set of choices for the requested component. Nevertheless, for the following discussion, both type of processes are considered similar because they share the same advantages and disadvantages.

5.3.2.1 Client-stateless-server

Client-stateless-server architectures for electronic catalogs imply the access to external or internal databases, or other information systems, at each time the user is requesting more product alternatives. In addition to that, the requests to the server side must contain the information to know what solutions have already been shown (an index can be enough). In the case that the user expresses preferences, information about such criteria must also be included in each request to the server side. Clearly, the main disadvantage in such stateless architectures is that, at each client request, the data sources must be accessed. In the case that data is locally stored in the server's computers, the response time for each user interaction may be reasonable. But, if the data needed to process client requests is located in remote systems (legacy systems, other information systems, or external databases), stateless architectures often make the server response time unacceptable for interactive electronic catalogs.

5.3.2.2 Client-stateful-server

The main advantage of stateful servers compared to stateless servers, is that the number of accesses to information source systems can be reduced. All the product configurations that fit the user's request can be stored at the session in the server. In this manner, only the main request of the user produces an access to the information data sources, while next requests only access the information stored in the session state. This allows to build electronic catalogs, with quick response times enabling the design of interactive catalogs. The main drawback of such systems concerns scalability. Servers can easily become overloaded by the necessity of keeping a session state for each individual user accessing the catalog simultaneously. This is especially true if the sessions contain many product configurations, or choices for each component, and hundreds or thousands of users access the catalog simultaneously.

5.3.2.3 Trade-off between stateless-servers and stateful-servers

In Table 5.1, the main pros and cons for electronic catalogs in architectures with stateless and stateful servers are summarized. Actually, the choice between stateful and stateless

servers is a trade-off between scalability and efficiency, depending on the following factors:

- the number of users that may be accessing the catalog simultaneously.
- the complexity of the catalog in terms of number of choices for each component and number of constraints relating components. There is a direct relation between the complexity of the catalog and the complexity of the business logic to provide product configurations satisfying the criteria of the users.
- response time of accessing data from information sources. The response time of information sources strongly depends on where the information sources are located, and the way they are connected to the system. Information sources located on the server side can be accessed quicker than external information sources such as legacy systems, information systems or external databases.

	scalability	efficiency
stateless server	✓	✗
stateful server	✗	✓

Table 5.1: Trade-off between efficiency and scalability regarding client-stateless-server and client-stateful-server architectures for electronic catalogs.

5.4 The *SmartClient* architecture

SmartClient [238][15] architecture allows systems to have smart and thin clients, by marrying the two following characteristics of constraint satisfaction techniques:

- Constraint satisfaction provides search algorithms which are both very simple and compact to implement, and at the same time, implement complex behaviors with reasonable efficiency, and
- Constraints allow representing complex information in a compact form.

As a result, basic *SmartClient* architecture configuration is based on two main novelties [245]:

1. Constraint-based solving algorithms can easily be executed on the client side because they implement complex behaviors while being compact.
2. Instead of transferring solutions (or choices) to the client side, *SmartClient* architecture sends entire problems to the client side. This implies to execute solving algorithms on the client side.

[15] Republished in [239, 240].

Several architecture configurations can derive from the *SmartClient* main concepts. In Section 5.4.3, different architecture configurations following the *SmartClient* paradigm are discussed. The basic *SmartClient* architecture configuration is explained in the following subsections.

5.4.1 *SmartClient* main concepts

The basic *SmartClient* architecture configuration is shown in Figure 5.5. The client sends a request containing the main user query to the server. The server accesses information sources in order to generate the corresponding CSP taking into consideration the user request. The CSP is then transfered to the client side. Before sending the CSP to the client side, some preprocessing techniques for simplifying the CSP can be applied. Such techniques are basically based on hard arc- and path-consistency algorithms for classical CSPs (see Appendix B for details on preprocessing consistency algorithms for classical CSPs). On the client side, compact solving constraint satisfaction algorithms are able to solve the CSP and support the user in the purchase decision making process. In this way, the user can browse through the different solutions by interacting with the client side locally. The *SmartClient* architecture, in its basic configuration, can be summarized as follows:

- **Compiling CSPs on the server side**: On the server side, the system compiles all relevant data from information sources according to the user request into the corresponding CSP. The way the server component compiles all the information into a CSP, depends on how the catalog is modeled. Preprocessing constraint satisfaction techniques can be applied in order to transform the CSP in a smaller and simpler but equivalent problem. The CSP is then transferred to the client side.

- **Solving CSPs on the client side**: On the client side, the user is able to browse through the solutions of the CSP, *i.e.*, the products of the catalog. By means of incremental constraint posting, as explained in Chapter 2, the user narrows down the set of solutions according to his preferences.

5.4.2 Representing solution spaces

Compacting solution spaces is a key point for *SmartClient* technology where solutions spaces are sent to the user's computer. The issue of how CSPs can help compacting solutions spaces is discussed in the following.

Combinatorial problems can have enormous number of solutions, arising though the combinations of variable values. For a problem with n variables of uniform domain of size d, there can be up to d^n different solutions. A complete enumeration of all solutions would require

$$n \cdot d^n \tag{5.1}$$

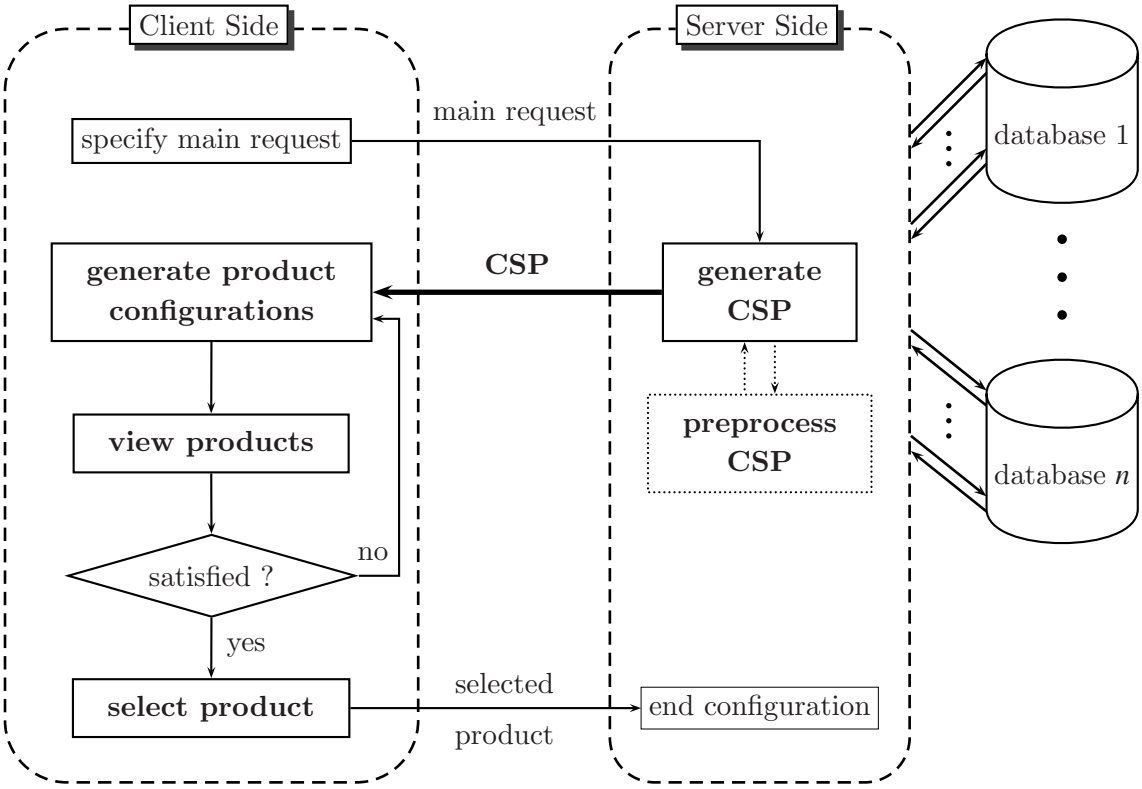


Figure 5.5: Decision making process for configurable products in *SmartClient* architectures. Most of the business logic is carried out on the client side, allowing the user to benefit from having a mixed-interactive system as explained in Chapter 2.

units of storage. If variables are completely independent, the space can be represented as a cross product of all their values. This representation would require only $n \cdot d$ units of storage.

In most cases, however, the admissible combinations are restricted by constraints[16]. In the worst case, a constraint can be stored as a list of all the admissible tuples. In such a case, a k-ary constraint can require up to d^k units of storage. In a CSP network with n variables, the arity of its constraints are in the range $[1,\ldots,n]$. In other words, a CSP with n variables can have constraints of arity $1,2,\ldots,n$. There can be at most C_k^n k-arity constraints, where C_k^n is the number of combinations without repetition of k elements out of n. Thereby, there can be at most

$$C_k^n = \binom{n}{k} = \frac{n!}{(n-k)! \cdot k!} = \frac{n \cdot (n-1) \cdots (n-k+1)}{k!}$$

k-arity constraints in a problem with n variables. In consequence, storing the k-arity

[16]In our model, these constraints are the hard constraints, since soft constraints do not restrict the admissible or feasible set of solutions. However, soft constraints must also be considered in regard to storing requirements of a problem.

constraints for a problem with n variables would require in the worst case

$$d^k \cdot \frac{n \cdot (n-1) \cdots (n-k+1)}{k!}$$

units of memory. Since a problem with n variables can have constraints with arity $1, 2, \ldots, n$, in the worst case storing a CSP with n variables would require:

$$\sum_{k=1}^{n} d^k \cdot \frac{n \cdot (n-1) \cdots (n-k+1)}{k!} \tag{5.2}$$

units of storage. Clearly, Equation 5.2 can be bounded by the following expression:

$$\sum_{k=1}^{n} d^k \cdot \frac{n \cdot (n-1) \cdots (n-k+1)}{k!} \leq \sum_{k=1}^{n} d^k \cdot \frac{n^k}{k!} \leq n \cdot d^k \cdot \frac{n^k}{k!} = \frac{d^k \cdot n^{k+1}}{k!}$$

If k is relatively small with respect to n, comparing Equation 5.2 and Equation 5.1 would imply:

$$\frac{d^k \cdot n^{k+1}}{k!} \ll n \cdot d^n$$

Actually, if k is relatively small with respect to n, storing a CSP with n variables (Equation 5.2) is *exponentially better* than storing all admissible combinations (Equation 5.1).

Note that, on one hand, many problems can be formulated as binary CSPs, and on the other hand, any non-binary CSPs can be theoretically transformed into binary CSPs in polynomial time. In the case of a binary CSP ($k = 2$), with n variables (usually $n \gg k$), storing the constraint graph instead of an enumeration of all solutions is dramatically better:

$$d \cdot n + \frac{d^2(n^2 - n)}{2} \ll n \cdot d^n$$

5.4.3 *SmartClient* architecture configurations

Up to this point, the basic *SmartClient* architecture has been described, however, different architecture configurations can be considered. *SmartClient* architecture configurations depend on the type of the client present in the system:

- **Stand-alone application clients**: are implemented as stand-alone applications. Stand-alone application clients receive problem-related data sent by the server through a network. Examples of this type of client are any program that can be executed in the client's machine.

- **Remote application clients**: capable of executing transferred code. Remote application client components are able to execute code which is sent by the server through a network. The executable code is not stored locally in the client's machine, but sent through a network. For example, clients with java run-time environments, where applets[17] contain the code to be executed. Note that applets can be executed

[17] `http://java.sun.com/applets`: An applet is a program written in the Java programming language that can be included in an HTML page. When using a Java technology-enabled browser to view a page that contains an applet, the applet's code is transferred to your system and executed by the browser's Java Virtual Machine (JVM).

in several devices such as personal computers, java phones and PDAs. Other technologies that enable these kind of clients are browser plug-ins, or specific scripting languages for browsers, such as ActiveX[18], JavaScript[19], or Windows Script[20].

- **Front-end clients**: capable of displaying information. Front-end client components are only capable of displaying information sent by the server through the computer network. Example of such clients are typically WWW[21] browsers in personal computers, or WAP[22] browsers in mobile devices such as mobile phones and PDAs.

Table 5.2 shows different configurations of the *SmartClient*. Basically, these configurations derive from considering a specific type of client. Note that the configuration with an application client is the configuration called *basic* through this chapter.

client type	server transmission	server type	num., size requests
stand-alone	problem data (CSP)	stateless	1, medium
remote	problem data (CSP) + code	stateless	1, medium-large
front-end	subset of solutions	stateful/stateless	n, small

Table 5.2: *SmartClient* architecture configurations derived from considering different types of clients.

In the following subsections the aforementioned *SmartClient* architecture configurations are analyzed by identifying their main usages, advantages and disadvantages.

5.4.3.1 Stand-alone application clients

The *SmartClient* architecture configuration with application clients is considered the basic *SmartClient* configuration.

Usages: Users willing to access electronic catalogs implemented in *SmartClient* architectures with application clients, must install the application in their device, usually a personal computer. Therefore, the usage of this *SmartClient* configuration is targeted to users that need to access the catalog often. Sporadic users would not be interested in such *SmartClient* architecture configuration.

Advantages: This configuration is the most scalable among the other possible configurations because:

[18] `http://msdn.microsoft.com`: ActiveX is a set of technologies that enable software components to interact with one another in a networked environment.

[19] `http://wp.netscape.com`: JavaScript is Netscape's cross-platform, object-based scripting language for client and server applications.

[20] `http://msdn.microsoft.com`: Windows Script provides two script engines, Visual Basic Scripting Edition and Microsoft JScript, which can be embedded into Windows Applications.

[21] WWW stands for the World Wide Web.

[22] WAP stands for Wireless Application Protocol.

- mostly all the business logic of the system is carried out by the client side, and
- the transmission between the server side and the client side is minimized.

In regard to the user, the main advantage is that the application client would only access the server side only once for getting the CSP. Once the CSP is transferred onto the client side, the user can interact with the application in an interactive process, without any delay. If, however, the main request of the user is very vague, the response time of the server can be long since it must collect all the information related to the user's query.

Disadvantages: The main disadvantage of this approach is that users must install an application. Therefore, sporadic users may dislike this architecture configuration.

Another inconvenience of this *SmartClient* architecture configuration is that sending the CSP can be time and bandwidth consuming, depending on the CSP size. However, this is the key for only accessing the server side once, as explained above.

5.4.3.2 Remote application clients

This *SmartClient* architecture configuration differs from the basic one (stand-alone application clients) in the transmission between the server side and the client side. In this configuration, the server side sends the CSP and the code to solve it. Therefore, this configuration can be seen as a slightly adapted version of the basic *SmartClient* architecture configuration.

Usages: *SmartClient* architectures with remote application clients are targeted to users that require intelligent tools for supporting the purchase decision making process without installing any software. Thus, it is a convenient *SmartClient* configuration for sporadic users that still require to be supported in the selection decision making process.

Advantages: This configuration also benefits from the fact that mostly all the business logic is processed on the client side. Nonetheless, the transmission between the server side and the client side is more important than in stand-alone application clients, since the code is also transmitted apart from the data of the problem. Therefore, this configuration is also highly scalable, but not at the same level as the basic configuration (with application clients).

Disadvantages: Within this configuration, users must have a device capable of executing the code sent by the server. Moreover, bearing in mind that,

1. Internet is a very general-purpose network, and
2. nearly all electronic catalogs are designed for being used on the Internet,

the requirement of having specific device capabilities can be a drawback for many users.

Obviously, a drawback of this configuration is that the code to be executed on the client side is sent by the server side every time a user wants to access the electronic catalog.

5.4.3.3 Front-end clients

SmartClient architecture with front-end clients can be considered as a compromise between standard catalog architectures and the benefits of using constraint satisfaction techniques. Within this configuration, all the business logic is located on the server side, while the client side is only used to display the information provided by the server component.

Usages: *SmartClient* architectures with front-end clients are convenient for providing intelligent tools without the requirements of the above configurations, namely:

- installing a stand-alone application (basic *SmartClient* configuration), and
- the client device must be able to execute the code sent by the server component.

This configuration is the most convenient for catalogs that must be accessed by many users with minimal requirements, *e.g.*, a standard web browser. This *SmartClient* variant is also indicated for devices with low computing and memory capacities, such as mobile phones or PDAs.

Advantages: The main advantage of such configuration with front-end clients is the *universality* of the whole system, since it is similar to the standard existing architectures for electronic catalogs. Thus, it does not demand any special requirements for users.

Disadvantages: As discussed in Section 5.3, executing complex business logic on the server side directly affects the scalability of the whole system. In this configuration the whole business logic is executed on the server side. Thus, when supporting hundreds or thousands of users accessing the system simultaneously, the server component could become overloaded decreasing its efficiency. However, since constraint-based solving is only executed when users enter or modify a preference, the processing only happens in small fractions of time. Most of the time, users are just evaluating the product proposed by the system.

Stateless-servers vs stateful-servers: This configuration accepts stateless-servers and stateful-servers. Basically, the advantages and disadvantages when analyzing both type of servers for *SmartClient* architectures are the same as those explained in Section 5.3.2. Nevertheless, some specific considerations must be made.

Stateless-servers imply to access the information sources and build the CSP each time the user criticizes the solutions in order to narrow down the proposed set of solutions according to his preferences. The required time to access the information, build the corresponding CSP, and solve it, may produce a system very tedious to use.

On the other hand, stateful-servers can keep the corresponding CSP for each user in the session state. In that way, the interaction between the client side and the server side can be dramatically improved. However, session states containing the whole CSP for each user may produce a low scalable system.

An important consideration about this *SmartClient* configuration in terms of scalability must be done. Usually, scalability problems are due to memory usage on the server side, and are not related to processing power limitations. Thus, it is worth to note that the memory required by constraint satisfaction techniques is not very significant, and during the CSP solving, the memroy requirement is bounded by the number of variables and the number of solutions to be computed. Memory requirement for constraint-based solving is only linear with respect to the number of variables in the problem.

5.5 The scalability of the *SmartClient* concept

The scalability of any client-server architecture strongly depends on two factors: a) the number of transfers between the client and the server, and b) the size of such data transfers. Let us consider an example in the travel domain to illustrate the scalability of the *SmartClient* concept. As we stated several times in this document, our work basically approaches complex problems such as travel planning. The travel problem to be considred is the following:

> I live in Bern, Switzerland, and would like to visit colleagues in Princeton (New Jersey), and London. I would like to spend at least two days in each place, and will need to travel in the first two weeks of February.

Since I live in `Bern`, I can leave from any of three Swiss airports (`Zurich`, `Basel`, `Geneva`, abbreviated as `ZRH`, `BSL`, `GVA`). Also, for `Princeton` I can fly to two `New York` airports (JFK, EWR) or to `Philadelphia` (`PHL`), and there are three airports in `London` to consider (`LGW`, `LHR`, `LCY`). Finding the best plan for my trip involves checking all combinations of flights between these airports on the dates which are specified in the main query. Thus, a system considering all the possibilities from the example described above will access the flight information for the initial query as follows:

1. 1st leg from Bern to Princeton: flights from `ZRH/BSL/GVA` to `JFK/EWR/PHL` on the dates from 1st to 10th February,

2. 2nd leg from Princeton to London: flights from `JFK/EWR/PHL` to `LGW/LHR/LCY` on the dates from 4th to 12th February, and

3. 3rd leg from London to Bern: flights from `LGW/LHR/LCY` to `ZRH/BSL/GVA` on the dates from 6st to 14th February.

Considering only direct flights, there are in fact more than 4 million solutions for this problem arising from all the possible combinations of the flights for each leg.

Let us analyze these two questions in traditional flight information systems and in a *SmartClient* system. In Table 5.3 we show the approximated answers to the example described above using conventional approach versus the basic *SmartClient* configuration architecture.

	Conventional approach	*SmartClient* approach
Server access	447,039	1
Size of 1 transfer	60Kb	500 Kb
Size of total transfers	26.82 Gb	500 Kb

Table 5.3: Conventional approach vs. *SmartClient* approach. The data of the table is calculated considering that the user is interested in all possible combinations of flights that match the initial query of the example mentioned before.

5.5.1 Conventional approach

Let us consider planning such a trip using the systems currently available on the WWW. One type of system, most commonly offered by airlines themselves, simply allows the user to inspect flights or connections for one particular leg at a time. On a multi-leg trip such as this one, this would require the customer to carefully note down all solutions for the different legs and finally put together a solution by hand - not a very satisfactory way of planning such a trip.

Fortunately, tools such as Travelocity[23] allow us to configure multi-leg trips. Complete itineraries are constructed on the server and returned to the customer for selection. In an example as the one given above, we could in principle browse through all the 4 million possible solutions, evaluating each manually as to whether it satisfies our constraints. Considering that solutions are displayed in web pages with about 10 solutions at a time, this would involve an enormous number of transfers. Each web page sent back has about 60 Kbytes, so in total we need to transfer (and look at) about 24 Gbytes ($4 * 10^6/10 * 60$ Kbytes) of information to see the complete solution space.

However, a smart user can save some time by exploiting regularities of the domain, such as the fact that most flights operate daily and usually have space available. But this means precisely that the tool is not very intelligent: it still requires the customer to do most of the work.

5.5.2 Basic *SmartClient* configuration

SmartClient concept offers the possibility to support the customer's decision-making process with an intelligent scratchpad. In contrast to conventional tools, it can keep track of all options and choices and avoid having to reload information which had already been requested earlier.

First of all, the customer specifies initial constraints about the trip: the departure and arrival airports, departure dates, and some general preferences. The *SmartClient* server then collects information about *all* flights which could be part of a solution in one single server access. Since the information is encoded as a CSP, it only needs to record the *sum* of the information for each possible flight, not all their combinations. In this example, there are 795 possible flights, each of which is encoded in a text line containing no more than 80

[23] `http://www.travelocity.com`

bytes. Thus, the entire CSP takes up no more than 63 Kbytes ($795 * 80$ bytes). Considering that the size of an average response page from a conventional server such as Travelocity is already 60 Kbytes, this is not a particularly large agent. It can be transmitted through the Internet in less than 1 minute through a standard modem. However, the price to pay is to solve the CSP on the client side. The CSP in the *SmartClient* implicitly contains all 4,470,396 possible solutions. Because it runs locally on the customer's computer, the best combinations can be searched using the techniques described in Chapter 4.

5.6 Summary

This chapter has briefly reviewed the main notions related to the design of software architectures.

Standard currently used architectures for implementing electronic catalogs have been described. Two different methodologies for configuring products have been reviewed and compared, namely: one-step and sequential configuration. From the architectural point of view, client-stateless-server and client-stateful-server architectures have been described, both for one-step and sequential configuration processes.

The basic *SmartClient* architecture configuration has been presented. It is primarily based on two concepts:

- dealing with search spaces (problems) instead of individual solutions, and
- executing constraint satisfaction problems on the client side.

Finally, different *SmartClient* architecture configurations have been considered and analyzed by identifying their typical usages, advantages and disadvantages.

Chapter 6

The Travel Planning Problem

Ab uno disce omnes. *From one example, learn about all.*

Virgil, Aeneid 2. 1-200, 70-19 B.C.

6.1 Motivation

Nowadays, travel planning is one of the most complex task that one can face by using Internet-based electronic catalogs. In that sense, Bill Gates reportedly once said that Microsoft started Expedia[1] because no other industry was as complex as travel, with so much constantly changing electronic information and consumers who wanted to become personally involved in the reservation booking process.

In general, electronic catalogs for planning travels imply to plan a set of services together such as flights, hotels and cars. Designing electronic catalogs for supporting the user to find his best travel configuration implies to deal with:

- information provided by external information sources,
- configuration constraints, and
- complex user's preferences and optimization constraints.

Thereby, catalogs for supporting users to plan travels, perfectly suit as an example for illustrating this thesis work.

6.2 The traditional travel industry

According to the Travel Industry Association of America[2], tourism generated over USD 580 billion in total expenditures in 2001. Every second in the United Sates, USD 18,500 is spent by resident and international tourists on travel and tourism. These figures rank the travel industry as the third-largest retail sales industry in the United States. Consequently, travel industry has an enormous business potential in the area of electronic commerce.

[1] `http://www.expedia.com`.

[2] Travel Industry Association of America (TIA): `http://www.tia.org`.

It is widely accepted by the travel industry itself, that a key factor that produced an enormous growing of the travel business was the deregulation of the airline companies. In the United States, deregulation of the airline business was pushed by Jimmy Carter in 1978. In Europe, deregulation of the airline business was put into effect in 1993 by the European Commission. Airline business deregulation has produced the liberalization impact on the travel industry, producing a larger, more accessible and more affordable industry.

6.2.1 Travel supply chain

The travel supply chain can be seen as a sequence where stakeholders take part in travel product sales, namely:

Providers: companies offering travel products and services such as airlines, hotels, and transportation companies in general.

Distributors: technology companies that consolidated supplier information, inventory and pricing data, and provided a way to electronically search, book and issue tickets and documents. These companies are called GDSs (Global Distribution Systems).

Travel agents: companies that intermediate between providers and customers, usually through distributors. They support the user by providing shopping guidance and personalized advice through the whole travel purchase process. They can offer their services both to business and leisure consumers.

Credit card companies: make purchase activities more convenient and secure for customers.

Travelers: are the end users of the whole travel supply chain, *i.e.*, the travel customers. They can be leisure travelers or business travelers, with different specific requirements.

Global Distribution Systems are a key factor when considering to build electronic catalogs for planning travels, since the product information comes from such legacy systems. In the following section, the history and main functionality of GDSs is briefly summarized (see [110] for more details and significant references).

6.2.2 Global Distribution Systems (GDS)

The origin of the Global Distribution Systems (GDS) is located in 1964, when American Airlines launched its Airline Reservation System (ARS), called Sabre[3], with the ability of keeping inventory data in real time, accessible to agents around the world. In 1964, Sabre ran on two IBM 7010, and was able to process 84,000 telephone calls per day. For the first time, agent travelers were able to call Sabre offices and get quick responses on availability

[3]Sabre: Semi-automated Business Research Environment, `http://www.sabre.com`.

and prices. Prior to this, manual systems required centralized reservation offices, groups of human beings in a room with physical cards in rotating trays with information about seats on airplanes.

After launching Sabre, other airlines followed with their own proprietary ARS. Quickly, a network concept involving several ARSs and travel agents emerged. The idea was to place the reservation technology for all airlines to travel agents' desktops. This is the origin of the Centralized Reservation Systems (CRS). In 1976, United Airlines began installing its own CRS, called Apollo[4], in travel agencies, and American Airlines soon followed this idea. Obviously, a first consequence for airlines following this strategy was to save important costs, by reducing the needed resources in the direct sales call-centers. The success of the ARS and CRS was evident. Within the next 10 years, European and Asiatic airlines began developing their own CRSs. By the mid-to-late 1990s, the major CRSs essentially became GDSs that travel agents used to check real-time flight schedules, seat availability and pricing information, make bookings and issue tickets. GDSs were incrementally incorporating other service providers such as cruise operators, hotels, railway companies and car rental companies. On the other hand, GDSs functionalities have became more reach, allowing for example to take into consideration special meal requests, managing seat allocation and performing back-office accounting for travel agents. By the mid-1990s, there were about a dozen major GDSs worldwide (the current ones). The airline reservation technology evolution, from ARSs to GDSs, can be summarized as follows:

$$\boxed{\text{ARS, 1964} \Longrightarrow \text{CRS, 1976} \Longrightarrow \text{GDS, 1995}}$$

Table 6.1 shows the GDSs in late 1990s, by regions, with the corresponding implied airline companies and their market share. Currently, GDSs mainly provide four functional components, namely:

1. **Inventory management and display** components collect inventory (flight seats, hotel rooms, cars, and so forth) of providers, and display the requested information to travel agents in specialized computers. Such special travel agents computers and software are able to show ordered lists of products. Thus, the GDS algorithms that provide these ordered lists have a great impact on the purchase decision of users. It has been shown that 90% of flight bookings are made using flights that appear on the first screen of flights. In consequence, GDSs display information algorithms have a relevant impact on the travel business.

2. **Pricing and fare search** engines basically compute a fare upon an itinerary request taking into account sophisticated pricing rules. Such rules determine prices depending on constraints on routings, stop-overs, advanced purchases, length of stay, season periods, stays over a weekend and so on.

3. **Ticketing generators** components allow travel agencies to produce physical or electronic tickets.

[4]Later, it became Galileo `http://www.galileo.com`.

Region	GDS	Market Share	Owner Airlines
North America	Sabre	22%	American
	Worldspan	10%	Delta, Northwest, TWA
	SystemOne	N/A	Continental
	Gemini	N/A	AirCanada, Canadian
North America Europe	Galileo International	22%	United, USAir, British Airways, SwissAir, KLM, Alitalia, Olympic, Air Canada, Aer Lingus, Austrian, Air Portugal
Europe	Amadeus	27%	AirFrance, Lufthansa, SAS, Iberia
Asia-Pacific	Abacus	9%	Cathay Pacific, Singapore, Malasya, Philipine, Royal Brunei, China Airlines
	Infiny		All Nipon
	Axes		Japan Airlines
	Topas		Korean Airlines
	Southern Cross		Australian, Ansett Airlines
	Fantasia		Quantas Airlines

Table 6.1: GDS markets in late 1990s (from [110]). Market share numbers do not equal 100% because not all systems were evaluated.

4. **Database reporting** engines serve both travel agencies and airline companies in several ways: reporting statistics on finances and accounting, facilitating trend analysis, or passenger searches.

In Figure 6.1, the travel industry supply chain is shown, from the distribution systems to the end user. The user can interact with physical booking engines or with online booking engines. Services offered by physical booking engines are either direct, *e.g.*, airline ticket office and phone booking, or assisted, *e.g.*, travel agent and travel managers. On the other hand, online booking services are either offered directly, *e.g.*, provider's website, or assisted, *e.g.*, travel agent electronic catalog.

6.3 The internet travel industry

Travel industry was one of the earliest to go online, mainly because its enormous business potential and its maturity in technology systems for providing travel information to the travel agents through electronic means. Electronic commerce for travel, also called e-travel or Internet travel, emerged in 1996. From then, thousands of sites offering travel products and services have arose. These travel web sites can be classified in two main types:

- **Service providers**: such as airlines, hotels and rental cars companies which develop

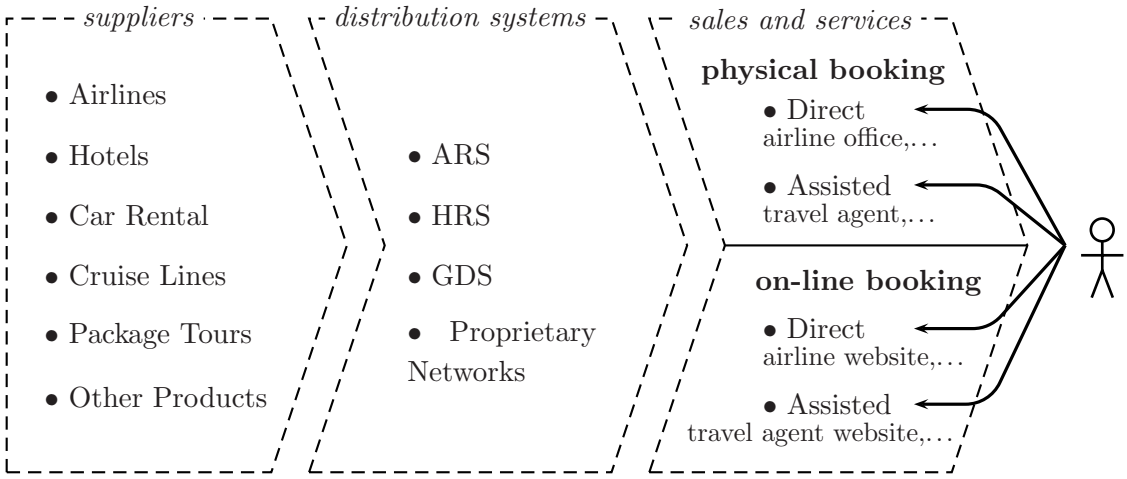


Figure 6.1: The travel industry supply chain (from [110]). Users purchase travel products offered by suppliers through the services managed by physical or online booking processes. Distribution systems provide the required travel data to physical or online booking services.

and operate websites that sell directly their products to the end users, *i.e.*, potential customers. Almost every airline is currently offering such online services.

- **Internet travel agencies**: offer at least the same kind of services as traditional travel agencies, *i.e.*, travel booking assistance. On top of that, web-based travel agencies are able to offer some additional services, such as real-time weather information, online currency exchange, and so forth. Nevertheless, standard Internet travel agencies often lack the quality of support provided by traditional travel agencies. Some examples of the most successful and advanced Internet travel agencies are: Expedia[5], Travelocity[6], and Orbitz[7].

The goal of electronic catalogs for the travel industry is to provide similar services as those offered by traditional travel agents, under electronic means. In other words, electronic catalogs for travel would replace traditional travel agencies by offering similar services under possibly better conditions. These better conditions are basically those that one can find in any electronic commerce activity:

- 7/7, 24/24 available services,
- accessing the services from home or work,
- advantageous prices (less costs than traditional industries),

Internet travel agencies use the GDSs to acquire the travel product data, while service providers usually access directly their own proprietary CRSs.

[5] `http://www.expedia.com`.
[6] `http://www.travelocity.com`.
[7] `http://www.orbitz.com`.

6.3.1 Main GDS/CRS functionalities

Every GDS or CRS offer a set of requests with their own specificities, however, three main requests are commonly implemented. In this section, brief descriptions of these main GDS/CRS functionalities[8], which are required for travel planning, are given together with some illustrating examples. The examples are given in a very simplified form and all the used codes are the official IATA[9] codes. In the rest of the chapter, the three requests will be considered uniform across the different GDS or CRS. Examples are retrieved from the Functional Description of the services provided by Galileo International [83, 84].

6.3.1.1 `AirAvailability` request

Given a pair of locations A and B, and a date d, `AirAvailability` request returns a list of sets of flights to go from A to B on date d. Note that a set of flights for going from A to B can contain only one (direct) flight or several flights (including intermediate airports). In Table 6.2, an `AirAvailability` request is shown from going from `LON` (London) to `RIO` (Rio de Janeiro). In London, several airports exist (`LHR`, `LGW`, `LCY`), thus the GDS may contain in its response any of these three airports in London. In this example IATA codes returned by Galileo GDS are used[10].

Dep	Arr	DepDate
`LON`	`RIO`	19/10/2002

Table 6.2: GDS `AirAvailability` request example: from London to Rio de Janeiro on the 19/10/2002.

In Table 6.3, only three sets of flights are shown. The first is a direct flight, while the two others are combination of flights with intermediate airports in Amsterdam and Frankfurt.

Note that in the example of Table 6.3, all the flights have 0 stops. An example of a flight with one stop, would be a flight from Geneva to Sydney stopping at Melbourne. Such a flight is a direct flight, however, it includes an (technical) stop to Melbourne. These stops are not intermediate locations, because there is no change of aircraft, it is just a technical stop.

Several attributes returned by `AirAvailability` request are omitted in Table 6.3 to ease the explanation. Missing attributes are for instance, the flight number, operating carrier (which can be different to the carrier of the flight), available booking classes for

[8]Other functionalities such as booking, statistic reporting or payment are not considered because they are out of the scope of this thesis. Functionalities for supporting car rental and hotel bookings are also offered by GDSs and CRSs.

[9]IATA (International Air Transportation Association) is an international association which monitors and regulates many aspects of the airline industry including standards and industry cooperation.

[10]`LHR` → London Heathrow International Airport, `LCY` → London City Airport, `LGW` → London Gatwick Airport, `GIG` → Antonio Carlos Jobim International Airport, `AMS` → Amsterdam Schiphol Airport, `FRA` → Frankfurt Airport, `RG` → Varig Brasil Airline, `KL` → Royal Dutch Airlines.

Dep	Arr	DepDate	DepTime	ArrTime	Carrier	Aircraft	Stops	Availability
LHR	GIG	19/10/2002	22:00	06:30	RG	M11	0	F,C,Y
			...					
LHR	AMS	19/10/2002	06:50	09:05	KL	737	0	C,Y
AMS	GIG	19/10/2002	10:25	17:05	KL	74M	0	F,C,Y
			...					
LGW	FRA	19/10/2002	19:00	21:30	LH	310	0	C,Y
FRA	GIG	19/10/2002	22:20	06:15	LH	340	0	F,C,Y

Table 6.3: GDS `AirAvailability` response example: from London to Rio de Janeiro on the 19/10/2002.

each class of services (Y: economy, C: business , F: first), animal transportation allowed flag, passenger nationality restrictions, special meals, smoking allowed flag, and so forth.

6.3.1.2 `PriceItinerary` request

The `PriceItinerary` request is specified by a sequence of flights and a booking class, and returns an applicable fare accordingly. The response includes many information about different prices for types of travelers (infants, military, adult, senior citizen), last date to purchase the ticket, airport taxes, and so on.

6.3.1.3 `BestBuy` request

The `BestBuy` request is defined by a sequence of locations with departure dates, and returns priced sequence of flights. Unfortunately, this request is very limited, *i.e.*, it only works for one-way and round-trips and economy class or special offers.

GDS requests may be really complex with many details. Since GDSs are legacy systems primarily used by human travel agents, many of the electronic functionalities are not really adapted for electronic commerce. Nonetheless, these last years, GDSs are making considerable efforts to provide interfaces to their legacy systems which are becoming more and more suitable for e-commerce.

6.3.2 Standard Internet travel planning systems

In general, one could say that standard electronic catalogs for planning travels are simply implemented as interfaces to GDSs or CRSs. In regard to the mechanisms that GDSs and CRSs offer (see Section 6.3.1) to plan a trip, in general, two different processes are implemented:

1. **Standard** planning process is carried out with two basic GDS requests, namely: `AirAvailability` and `PriceItinerary`.

2. ***BestBuy***[11] planning process is carried out with the `BestBuy` GDS request.

It is worth noting that the aforementioned processes to configure a trip fit well into the classification made in Chapter 5, Section 5.3.1, in the following manner:

Standard travel planning	$\Longrightarrow$	sequential configuration
BestBuy travel planning	$\Longrightarrow$	one-step configuration

Standard and *BestBuy* planning strategies share all the advantages and disadvantages of sequential and one-step configuration process for electronic catalogs (see Section 5.3.1 for more details about these configuration process variants). In the following section, one-step and sequential configurators for planning travel are briefly commented.

6.3.2.1 One-step configuration

In one-step configuration process, the user directly gets configured products. In travel planning, this means that the traveler evaluates complete itinerary solutions. In standard travel planners, this is achieved by using the `BestBuy` request. Unfortunately, `BestBuy` request only gives few very simple itineraries with economy class and special offers. This approach can only be taken for one-leg trips and round-trips.

6.3.2.2 Sequential configuration

Sequential configurators are also available in standard travel planning systems. This model is more flexible allowing complex itineraries. The user has to select one option for each part of the itinerary, *i.e.*, for each leg. Options for a leg are given by means of `AirAvailability` requests. Each time that the user selects an option, the system restricts the options for the next leg according to configuration rules and optimization criteria which are implemented by GDSs. At the end, once all the legs are selected with an option, the system is able to price the itinerary with a `PriceItinerary` request.

Actually, using our approach it is possible to use `AirAvailability` and `BestBuy` requests into a one-step configuration process. The model allowing to use both requests together is discussed in Section 6.5.6.

As it has been shown in previous sections, standard processes to plan (configure) trips on the web are not fully supporting the user in his purchase decision-making process. The user has still to do almost all the work for analyzing the proposed flights, and selecting the preferred options according to his personal criteria. Next sections illustrate how to model the problem of planning travels within the framework described in Chapter 3. Such problem modeling enables to apply the all the concepts described in previous chapters.

[11] *BestBuy* is the accepted terminology in the travel industry for this functionality.

6.4 The problem of planning travels

The problem of configuring a trip can be roughly stated as finding a combination of transport means to go to a collection of locations on certain dates. In the following, only the air travel is considered as a mean of transport, however, other means of transports such as rental cars, trains and buses could be considered in the same model without major modifications. Hotel booking and other travel services may also be considered.

At a first glance, the problem of planning air travel for a given itinerary des not seem to be a very complex problem. Nevertheless, the problem of finding flight combinations for traveling to a collection of n locations has a combinatorial explosion, as roughly shown in Figure 6.2. This combinatorial explosion comes from the possible flight combinations to travel to the n locations. Suppose a user residing in Geneva wants to go to London, New York, and then back. The sequence:

Geneva	→	London,	on December 10th 2002
London	→	New York,	on December 15th 2002
New York	→	Zürich,	on December 19th 2002

is called an *itinerary*. Assuming that 20 different flights exist for each pair of locations, the whole search space of the itinerary is $20^3 = 8,000$. Actually, an `AirAvailability` GDS request usually returns about 60 different flights for a pair of locations on a given date, thus the whole search space of the above example would be then $60^3 = 216,000$ potential solutions. The combinatorial explosion of an flight planning problem becomes much more important when considering several locations and several dates for each segment of the trip, for example, the following itinerary

Geneva or Zürich	→	London,	on December 10/11th 2002
London	→	New York,	on December 15/16/17th 2002
New York	→	Geneva or Zürich,	on December 19/20th 2002

defines a search space of $(60*4)*(60*3)*(60*4) = 10,368,000$ possible flight combinations.

6.4.1 Statement and definitions

An *itinerary* defines a trip between a *start location* and an *end location* where some *intermediate locations* are visited as well. A set of dates are associated to each pair of consecutive locations, defining the dates where the user wants to travel. A location is either an airport or a city. A city can include several airports, *e.g.*, New York City includes three airports: `JFK`[12], `LGA`[13], and `EWR`[14].

An itinerary is composed of an ordered collection of *legs*, *e.g.*, Geneva → London on December 10th 2002. A leg is specified by a *departure* and *arrival* locations, and a set of

[12]IATA code for the John F. Kennedy International Airport.
[13]IATA code for the La Guardia Airport.
[14]IATA code for the Newark Liberty International Airport.

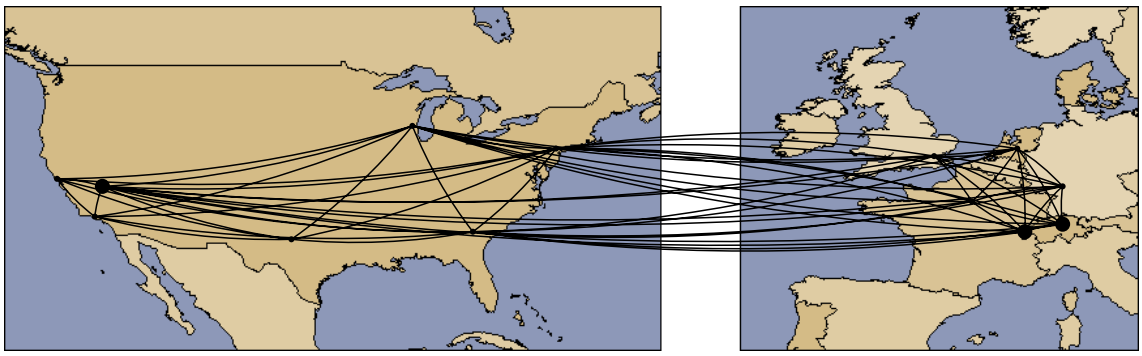


Figure 6.2: Combinatorial explosion in travel planning.

dates. The departure location corresponds to the start or an intermediate location of the itinerary, whereas the arrival location is an intermediate or end location. The set of dates associated to a leg defines the possible dates where the leg can take place. Tow legs are called *consecutive* if they directly follow each other, *i.e.*, there is no other leg in between. Note that two consecutive legs may not share a same intermediate location, that is, the arrival location of a leg may not be the same as the departure location of the following leg.

A leg is composed of one or several *flights*. If a leg is composed of several flights, the traveler is forced to change planes at *transit locations* (airports). The number of transits of a leg is the number of times the traveler must change flights on the leg, and corresponds to the number of flights minus one.

A flight may include several *stops* between the departure and arrival location. During a stop, the traveler does not need to change planes and generally stays in the plane (technical stops).

These concepts can be formally defined as follows. Note that, locations (airports or cities), carriers and aircraft types are specified by the standardized codes proposed by IATA.

Definition 6.1 (Flight) A *flight* is defined by:

- `depLoc` is the departing location of the flight,
- `arrLoc` is the arriving location of the flight,
- `depDate` is the departure date of the flight,
- `depTime` is the departure time in `depLoc` local time of the flight,
- `arrTime` is the arrival time in `arrLoc` local time of the flight,
- `numStops` is the number of stops of the flight,

- `carrier` is the carrier that operates the flight,
- `flightNum` is the flight number of the flight, and
- `availability` specifies what classes of services are available for the flight. There are three classes of services: economy, business and first (not available for European flights). Within each class of service, different booking classes exist depending on fare conditions such as the possibility of rebooking, special offers, weekend rules and so on. Actually, `availability` gives the available seats on each booking class.

Given a flight `f`, their attributes will be denoted as `f.attribute`. For example, `f.arrTime` denotes the arrival time of the flight `f`.

Definition 6.2 (Flight Sequence) A *flight sequence* is an ordered collection of m flights $\{\mathtt{f}_1,\ldots,\mathtt{f}_m\}$ from a departure location `departureLoc` to an arrival location `arrivalLoc` on a certain departure date `departureDate`, where:

- $\mathtt{f}_1\mathtt{.depDate} = \mathtt{departureDate}$,
- $\mathtt{f}_1\mathtt{.depLoc} = \mathtt{departureLocation}$,
- $\mathtt{f}_m\mathtt{.arrLoc} = \mathtt{arrivalLocation}$,
- $\mathtt{f}_i\mathtt{.arrLoc} = \mathtt{f}_{i+1}\mathtt{.depLoc}$, and
- $(\mathtt{f}_i\mathtt{.arrDate} = \mathtt{f}_{i+1}\mathtt{.depDate}$ **and** $\mathtt{f}_i\mathtt{.arrTime} <_b \mathtt{f}_{i+1}\mathtt{.depTime})$ **or**

  $(\mathtt{f}_i\mathtt{.arrDate} <_b \mathtt{f}_{i+1}\mathtt{.depDate})$.

The operator $<_b$ denotes the precedence of times and dates, *i.e.*, $t_1 <_b t_2$ means that t_1 is before t_2. The notation for the attributes of a flight sequence is similar to the one for flights, for instance `fs.departureLoc` denotes the departure location of `fs`.

In other words, a flight sequence is a collection of consecutive flights going from a location to another where the first flight departs on a specified date. Two consecutive flights are linked by the same airport, *i.e.*, a connection or transit airport.

Definition 6.3 (Itinerary Problem) An *itinerary problem* is specified by an ordered collection of n legs $\{leg_1, leg_2, \ldots, leg_n\}$. A leg_i is composed by:

- a set of departure locations $DL_i = \{dl_1,\ldots,dl_k\}$,
- a set of arrival locations $AL_i = \{al_1,\ldots,al_l\}$,
- a set of departure dates $DD_i = \{d_1,\ldots,d_m\}$, and
- a set of flight sequences $\mathtt{flightSeqs}_i = \{\mathtt{fs}_{i,1},\ldots,\mathtt{fs}_{i,p}\}$, where $\forall\, j, 1 \leq j \leq p$:
  - $\mathtt{fs}_{i,j}\mathtt{.departureLoc} \in DL_i$,
  - $\mathtt{fs}_{i,j}\mathtt{.arrivalLoc} \in AL_i$, and

– $\texttt{fs}_{i,j}\texttt{.departureDate} \in DD_i$.

Note that since the legs of an itinerary are ordered, the following property must hold:

$$\forall 1 \leq i < n, \forall d_k \in DD_i,\ \exists d_l \in DD_{i+1} \text{ such that } d_k \leq_b d_l$$

Two consecutive legs can depart on the same date, but the predecessor leg can never leave after the departure date of the following leg.

Definition 6.4 (Itinerary Specification) An *itinerary specification* is an itinerary problem without the flight sequences, and it corresponds to the user request.

Definition 6.5 (Itinerary Solution) A *solution* to a given itinerary with n legs is a collection of n flight sequences $\texttt{solFlightSeq}\ \{\texttt{sol-fs}_1,\ldots,\texttt{sol-fs}_n\}$, where $\texttt{sol-fs}_i \in \texttt{flightSeqs}_i$ and

$$\forall 1 \leq i < n,\ \texttt{lastFlight(sol-fs}_i\texttt{)} <_b \texttt{firstFlight(sol-fs}_{i+1}\texttt{)}$$

where $\texttt{f}_i <_b \texttt{f}_j$ means that
$(\texttt{f}_i\texttt{.arrDate} = \texttt{f}_j\texttt{.depDate}$ **and** $\texttt{f}_i\texttt{.arrTime} <_b \texttt{f}_j\texttt{.depTime})$ **or**
$(\texttt{f}_i\texttt{.arrDate} <_b \texttt{f}_j\texttt{.depDate})$

Example 6.1 illustrates the above definitions with a concrete scenario.

Example 6.1: Itinerary and itinerary solution.
Considering the following itinerary specification:

Geneva or Zürich	→	London,	on December 10/11th 2002
London	→	New York,	on December 15/16/17th 2002
New York	→	Geneva or Zürich,	on December 16/17th 2002

its itinerary problem definition includes 3 legs:

- *leg*1
  - $DL_i = \{\texttt{GVA},\texttt{ZRH}\}$,
  - $AL_i = \{\texttt{LON}\}$,
  - $DD_i = \{12/10/2002, 12/11/2002\}$
  - $\texttt{flightSeqs}_1 = \{\texttt{fs}_{1,1},\texttt{fs}_{1,2},\texttt{fs}_{1,3},\texttt{fs}_{1,4},\ldots\}$, where
    - $\texttt{fs}_{1,1} = \{\texttt{[GVA,LGW,12/10/2002,07:10,07:55,BA,A320,0,\{C,Y\}]}\}$
    - $\texttt{fs}_{1,2} = \{\texttt{[GVA,LCY,12/11/2002,11:05,11:45,LX,A320,0,\{C,Y\}]}\}$
    - $\texttt{fs}_{1,3} = \{\texttt{[GVA,CDG,12/11/2002,09:00,09:45,AF,A320,0,\{Y\}]},$ $\texttt{[CDG,LHR,12/11/2002,10:00,12:15,AF,A320,0,\{C,Y\}]}\}$
    - $\texttt{fs}_{1,4} = \{\texttt{[ZRH,BSL,12/10/2002,06:00,06:25,LX,A320,0,\{Y,C\}]},$ $\texttt{[BSL,LCY,12/10/2002,06:50,08:10,LX,A320,0,\{C\}]}\}$

    ⋮

- *leg*2
  - $DL_i = \{\texttt{LON}\}$,
  - $AL_i = \{\texttt{NYC}\}$,
  - $DD_i = \{12/15/2002, 12/16/2002, 12/17/2002\}$,
  - $\texttt{flightSeqs}_2 = \{\texttt{fs}_{2,1}, \texttt{fs}_{2,2}, \texttt{fs}_{2,3}, \ldots\}$, where
    - $\texttt{fs}_{2,1} = \{$[LHR,EWR,12/16/2002,12:50,04:15,AA,A340,0,{F,C,Y}]$\}$
    - $\texttt{fs}_{2,2} = \{$[LHR,JFK,12/15/2002,13:20,04:10,UA,A340,0,{F,C}]$\}$
    - $\texttt{fs}_{2,3} = \{$[LHR,AMS,12/17/2002,06:25,07:25,DL,A320,0,{Y,C}], [AMS,JFK,12/17/2002,07:50,13:15,DL,A320,0,{F,C}]$\}$

      $\vdots$
- *leg*3
  - $DL_i = \{\texttt{NYC}\}$,
  - $AL_i = \{\texttt{GVA}, \texttt{ZRH}\}$,
  - $DD_i = \{12/16/2002, 12/17/2002\}$,
  - $\texttt{flightSeqs}_3 = \{\texttt{fs}_{3,1}, \texttt{fs}_{3,2}, \texttt{fs}_{3,3}, \ldots\}$, where
    - $\texttt{fs}_{3,1} = \{$[EWR,GVA,12/16/2002,21:25,14:10+1 day,BA,A340,0,{F,C,Y}]$\}$
    - $\texttt{fs}_{3,2} = \{$[JFK,ZRH,12/17/2002,18:10,07:50+1 day,AA,A340,0,{C,Y}]$\}$
    - $\texttt{fs}_{3,3} = \{$[JFK,LHR,12/17/2002,21:10,13:00+1 day,DL,A320,0,{Y,C}], [LHR,ZRH,12/18/2002,13:30,14:10,DL,A320,0,{F,C}]$\}$

      $\vdots$

A solution to this itinerary problem could be, for example $\{\texttt{fs}_{1,2}, \texttt{fs}_{2,1}, \texttt{fs}_{3,1}\}$, but the set $\{\texttt{fs}_{1,2}, \texttt{fs}_{2,3}, \texttt{fs}_{3,1}\}$ is not a solution to the problem since the flight sequences $\{\texttt{fs}_{2,3}, \texttt{fs}_{3,1}\}$ are not compatible: the last flight of $\texttt{fs}_{2,3}$ arrives to NYC before the first flight of $\texttt{fs}_{3,1}$ departs from NYC.

6.5 Modeling the travel problem as a CSP

As indicated in Section 3.3, modeling an electronic catalog for a concrete user request implies to identify the variables with their domains, and the constraints among them. In this section, the process of modeling the travel planning problem with the framework proposed in Chapter 3 is detailed.

6.5.1 Variables and values

An itinerary specification (user request) determines the variables of the problem. Each leg of the itinerary specification corresponds to one variable. The variable associated to leg_i will be referred as to $\texttt{leg-variable}_i$. The domain for a variable $\texttt{leg-variable}_i$ is the set of flight sequences $\texttt{flightSeqs}_i$ of the itinerary problem. Thus, a value for a `leg-variable` is a sequence of flights. The justification of this problem modeling approach is strongly related to the way in which travel information can be accessed from GDSs.

In Chapter 7, the process of building the CSP for travel planning from a user request is explained. Basically, the user request (itinerary specification) defines the variables of the problem, and GDSs give the domains for such variables according to the specification of the itinerary.

Available class of services for a given flight are given by the `AirAvailability` request. Thus, for each class of service a value for a leg variable is created. For example, if the flights on a flight sequence are available in economy class and business class, two different values are created, one for the economy class and the other for the business class.

6.5.2 Configuration rules

Configuration rules expressed as hard constraints guarantee the feasibility of the solutions, *i.e.*, that the solutions are valid. In travel planning, when using `AirAvailability` request, hard constraints must guarantee that values for variables representing consecutive legs are compatible (Definition 6.5). This constraint is binary implying each pair of consecutive legs. The penalty associated to a tuple of values (values are flight sequences, a and b) of two consecutive legs, $\texttt{fs}_{i,a}$ and $\texttt{fs}_{i+1,b}$ is computed as follows:

$$\texttt{penalty}(\texttt{fs}_{i,a},\texttt{fs}_{i+1,b}) \rightarrow \begin{cases} 0 & \texttt{if isBefore}(\texttt{fs}_{i,a},\texttt{fs}_{i+1,b}) \\ \texttt{hard-violation} & \texttt{otherwise} \end{cases}$$

The predicate $\texttt{isBefore}(\texttt{fs}_{i,a},\texttt{fs}_{i+1,b})$ is true if the last flight of $\texttt{fs}_{i,k}$ arrives before the first flight of $\texttt{fs}_{i+1,l}$ departs. Actually, one could consider a minimum slot of time between two legs, for example 1 or 2 hours. Remember that legs are defined by the user's request. Thus, each location between legs is a destination for the user, and it makes sense to assume that the minimum time slot for a destination is at least 1 or 2 hours.

The feasibility of flights within a flight sequence `fs` in terms of sequentiality, is guaranteed by the `AirAvailability` request of GDSs, *i.e.*, the sequence of flights returned by GDSs are always valid combinations.

6.5.3 User preferences

Soft constraints are used to model user preferences. Preferences about any aspect of an itinerary are easily and naturally modeled using the constraint-based approach discussed in Chapter 3. In this subsection, only some user preferences are illustrated as examples, without giving an exhaustive list of all possible user preferences for the travel planning.

User preferences are modeled as crisp or flexible soft constraints (see Section 3.2.3 for descriptions on these types of constraints), depending on the type of the preference.

6.5.3.1 Soft and crisp constraints

Crisp constraints are either totally satisfied or totally violated, without accept any degree of violation. Examples of crisp preferences are:

- Preference to specify a preferred carrier alliance. This preference implies unary constraints over leg variables. The preference could be applied to the whole itinerary (a unary constraint for all leg variables) or to a specific leg (a unary constraint for the variable associated to the leg). The penalty function of this constraint for a leg_i over a flight sequence $\texttt{fs}_{i,a}$ is defined as:

$$\texttt{penalty(fs}_{i,a}\texttt{)} \rightarrow \begin{cases} 0 & \texttt{if isOperated(fs}_{i,a}\texttt{,prefAlliance)} \\ \texttt{soft-violation} & \texttt{otherwise} \end{cases}$$

  Note that this preference only considers if the whole flight sequence is operated by `prefAlliance`. However, a better model for this preference can be implemented within a flexible preference which counts the number of flights in the flight sequence which are not operated by `prefAlliance`.

- Preference to specify a preferred aircraft type for a leg i. The penalty function of such constraint for a leg_i over a flight sequence $\texttt{fs}_{i,a}$ is defined as:

$$\texttt{penalty(fs}_{i,a}\texttt{)} \rightarrow \begin{cases} 0 & \texttt{if aircrafts(fs}_{i,a}\texttt{)} = \texttt{prefAircraft} \\ \texttt{soft-violation} & \texttt{otherwise} \end{cases}$$

  The same note as in the previous example applies for this preference. A better model can consider the number of flights of the flight sequence with a different aircraft type than the preferred one.

- Preference to specify a preferred intermediate location for a given leg. The penalty function of this constraint for a leg_i over a flight sequence $\texttt{fs}_{i,a}$ is defined as:

$$\texttt{penalty(fs}_{i,a}\texttt{)} \rightarrow \begin{cases} 0 & \texttt{if hasLocation(fs}_{i,a}\texttt{,prefLocation)} \\ \texttt{soft-violation} & \texttt{otherwise} \end{cases}$$

From the above examples, it is easy to extend them to incorporate negative preferences, *i.e.*, instead of preferred features, they model disliked features.

6.5.3.2 Soft and flexible constraints

Flexible constraints are satisfied (or violated) at a certain degree, thus different degrees of violation are considered. Examples of flexible preferences are:

- Preferences about preferred departure/arrival times. These preferences are naturally expressed as flexible constraints because time is clearly a continuous parameter. For example a preference for stating to depart from a location of a leg_i before 10 a.m. are expressed by the following penalty function over a flight sequence $\texttt{fs}_{i,a}$:

$$\texttt{penalty(fs}_{i,a}\texttt{)} \rightarrow \begin{cases} 0 & \texttt{if f.depTime} \leq \texttt{10 a.m.} \\ (\texttt{f.depTime} - \texttt{10 a.m.}) & \texttt{otherwise} \end{cases}$$

  where `f` is $\texttt{firstFlight(fs}_{i,a}\texttt{)}$.

- Preferences about preferred carriers take into consideration the carriers present in the flight sequence which are not the same as the preferred one, and also the carriers which are not preferred but are in the same alliance. Carriers which are not preferred but are in the same alliance are better for the user than others.

As in the case of soft and crisp constraints, negative preferences can be similarly modeled, *i.e.*, instead of preferred features, they model disliked features.

6.5.3.3 Soft constraint weights

The user could also be interested in expressing weights for his preferences. In such a case, the WCOP associated to the user problem includes a constraint weight. Even users find it very difficult to numerically weight their preferences, some weight scales can be expressed. In this way, for example, the user could state his preferences as *low*, *medium* or *high* regarding their strength. From such classification, a constraint weight vector makes sense, assigning, for example, a weight of 0.25, 0.5, and 1 for *low*, *medium* and *high* preferences respectively.

6.5.3.4 Contextual soft constraints

This mechanism allows to naturally express the way of having preferences by humans. Contextual constraints are of the form: `condition` → `preference`, where `preference` is any constraint preference (as explained above). The `condition` of a contextual constraint is a boolean predicate over some feature of the value/s of the variable/s implied in the constraint. Examples of contextual preferences are:

- Departure/arrival time preferences with a context on the departure/arrival location. These constraints express time preferences for different locations for the same leg. For example, a user who lives in Bern, and can depart from Geneva after 10 a.m., but if he departs from Zürich then he can only depart after 11 a.m. because the train to go to Zürich takes more time than going to Geneva.

- Preferred carrier depending on the destination. These constraints express carrier preferences only for some destinations.

The function penalties of the above user preferences examples are not normalized nor rescaled with respect to the optimization criteria, please refer to Section 3.7.2 to see why and how penalty functions must be normalized and rescaled.

6.5.4 Optimization criteria

Optimization constraints are less strong than preferences, differentiating among solutions which violate preferences to the same extent. They can be seen as a secondary criterion to order solutions when solutions have the same penalty when only considering user preferences. Optimization constraints model general optimization criteria that are valid in all situations and by all users, and therefore they are built under *common sense* assumptions. Some examples of optimization criteria are:

- Minimize the number of transit locations within flight sequences. People do not like to stop at transit locations and change planes.
- Minimize the flying time of the whole itinerary, users normally prefer to spend as less as possible time traveling.
- Minimize the number of different carriers participating in an itinerary solution. It is known that combining different carriers strongly increases the price of the itinerary. Also, it is convenient to minimize the number of carriers participating in a solution which belong to different carrier alliances.
- Push itinerary solutions where the arrival airport for leg_i and departure airport for leg_{i+1} are the same. Remember that one could have an itinerary solution where the user arrives at `LGA` and departs from `JFK`. Both airports are in New York city, but the user may find it more comfortable to arrive and depart from the same airport, for example, if he would like to book an hotel near the airport.
- Minimize the number of different classes of service within the same itinerary solution. Travelers do not usually want to change class of service during the same itinerary. An exception must be taken into consideration for the first class, since it does not exist in European flights.

Note that these criteria are assumptions that are done under the *common sense*, however, in some cases they could be contrary to the wishes of the traveler. Let us imagine a person that would like to stop at a specific transit airport to do some shopping. But in these cases, the user can always express such preference by a constraint which would cancel the optimization criterion counterpart, because of their different strength of valuation functions.

6.5.5 Note on using fares

Up to this point, fares on itineraries were not discussed. Manifestly, the price of a whole itinerary solution is of a high interest for the user. The easiest approach to price an itinerary is to make use of the `PriceItinerary` request offered by GDSs. Nonetheless, this is not convenient for the user, since he can not take into consideration the price in his purchase-decision making process, *i.e.*, he will only be aware of the price when he selects a solution and is priced using the GDS access.

Another approach consists on accessing the fare descriptions of GDSs for a given itinerary. This can be done by the `FareQuote` request which is available for most of the GDSs. The `FareQuote` request just needs an itinerary specification. The difference between such request and the `PriceItinerary` request, is that the latter prices a collection of concrete flights, and the former gives fares applicable to an itinerary specification. A fare for a given itinerary specification mainly consists of a price (amount and currency), and a list of fare rules. The rules of a fare are conditions that must be satisfied in order to price the itinerary with such fare. These rules must be applied to flights. Examples

of fare rules are, stay over weekend, Saturday night flag, routings (this conditions may be sophisticated), and so forth. The main inconvenience of these rules is that they are entered by human operators in pseudo language, *i.e.*, natural language with some widely accepted codes. Therefore, in order to use `FareQuote` request in electronic travel planners, complex parsers must be available. Noteworthy, `FareQuote` request is mainly designed to be read by human travel agents. They can easily interpret fare rules and price itineraries with.

From a CSP point of view, fares for an itinerary would be a domain of a `fare-variable`. Such variable would be then implied in binary constraints with leg variables in order to guarantee the applicability of the fares rules. Some of the fare rules may imply several legs, and thus k-ary constraints may be needed as well. Clearly, these constraints must be hard, since if the fare rules do not apply to flights, the fare can not be applied.

6.5.6 Note on using `BestBuy` request

`BestBuy` request gives priced itinerary solutions for a given itinerary specification. Unfortunately, it only gives very few simple itineraries solutions within economy class and some special offers. Nonetheless, this request could also be used and integrated in the CSP model described up to here. For that, a new variable is introduced, as a `fare-variable`. The domain for that variable is composed of all the fares given by the `BestBuy` request. A special value for the `fare-variable`, called *dummy-fare*, is also considered. On one hand, all the flight sequences (values for `leg-variables`) which were collected from `AirAvailability` requests are linked with the *dummy-fare* of the `fare-variable` by means of binary hard constraints. On the other hand, the flight sequences given by the `BestBuy` request are added to the leg variables. Such values are then linked to the values of the `fare-variable` by means of binary constraints. Since fares given by `BestBuy` requests only match one combination of flight sequences, the correctness of final solutions is guaranteed.

This model allows the user to evaluate within the same purchase decision-making process priced solutions and unpriced solutions. Itinerary solutions which are not priced are detected because they will have the *dummy-fare* assigned to their `fare-variable`. The interface must take priced and unpriced solutions into consideration.

Using `BestBuy` as explained, allows the user to realize that there are some simple solutions which are priced, and more complex solutions which are not priced. Leisure travelers or low-budget travelers would be more interested in priced solutions, whilst business travelers would prefer heterogeneous solutions (fitting their complex preferences) even if they are not firstly priced. Before purchasing an itinerary solution which is not priced, the system must price it with the standard `PriceItinerary` request.

6.6 An alternative CSP modeling approach

An alternative CSP modeling can be considered, similarly to the modeling approach described in Section 3.3.4. In this case, variables are not associated to legs, but to features

of the whole itinerary. An itinerary problem can be modeled with the following variables:

- For each leg of the itinerary:
  - A `depDate` variable with all the possible departing dates.
  - A `depTime` variable with all the possible departure times.
  - An `arrTime` variable with all the possible arriving times.
  - A `depLoc` variable with all the possible departure locations.
  - An `arrLoc` variable with all the possible arrival locations.
  - An `intermediateLoc` variable with all the possible intermediate locations.
- A `aircraftType` variable with all the possible aircraft types.
- A `carrier` variable with all the possible carriers.

  ⋮

The domains of variables come from the requests to GDSs according to the user request (the itinerary specification). Within this approach, many more hard constraints are need to guarantee the correctness of itinerary solutions. Actually each flight sequence given by the `AirAvailability` request must be translated in a set of hard constraints. In order to avoid having k-ary constraints ($k > 2$), a variable with the flight numbers (unique identifier for a flight) may be needed. The advantage of such model is that preferences (unary constraints) may produce much more propagation during the solving. Another advantage is that preferences (unary constraints) can be modeled more explicitly, they directly affect values of variables instead of attributes of values. However, the main drawback is that this model undoes the flight sequences given by GDSs. In other words, GDSs give correct flight sequences for each leg (correct sub-configurations), and this approach undoes this sub-configurations and produces a configuration task with more components to be configured by the solving algorithm.

This model was evaluated, in terms of solving efficiency, and performed worse than the model where variables are associated to the legs of the itinerary, and therefore such model was abandoned.

6.7 Related work

In [225], the authors propose a model-based travel planning framework. They offer a formally-based models which are well-defined in precise mathematical terms. The goal of this research is two-fold: on one hand to formalize the travel-related concepts mathematically, and on the other hand provide the basis for an engineering approach. Solving the travel planning problem is faced with an incrementally travel plan strategy. At each step, the travel plan is more and more concrete and complete, also closer to the user requirements. However, it is not clear how to build an electronic catalog for planning travels

using currently available technology (GDSs), and they do not propose any user interaction model nor an Internet-based architectural model.

Linden *et al.* in [153] propose a similar approach to the one described in this chapter, also using constraint satisfaction technology. Nevertheless, they do not deal with the travel information gathering aspect of the system. They consider the data available in local databases. Also, they do not explicitly describe the constraint model and their solving strategies.

Thompson *et al.* propose in [237] a conversational model where the solutions are shown to the user only when the solution space is reduced. Therefore, in their model, the user interacts with the systems without looking at solutions. Such a model would be difficult to be applied to planning travel because attributes of the products (travel items) are too complex to be asked without a criticizing process on proposed solutions.

Morris and Maes propose in [176] a framework for auctions and interactive negotiations between the seller and the buyer. SARDINE system is a system to submit ticket bids to airlines and for airlines to respond to each bid. However, they do not tackle the configuration problem, thus they deal with the travel planning problem as a multi-attribute atomic product.

In [31] the authors present a model for acquiring user prferences for product customization. For that, they use Multi-attribute Utility Therory (MAUT). This model can be considered close to ours but without considering hard constraints. The weakness of this approach is that is hard to deal with the combinatorial explosion of the potential solutions for a multi-leg trip.

6.8 Summary

This chapter firstly reviews the travel industry, and specially to the topics related to electronic catalogs for planning travels.

Secondly, the problem of planning travels has been stated formally, giving the related definitions. An example of modeling the travel planning problem according to the framework provided in Chapter 3 was explained. Basically, the modeling process applied to the travel domain has been shown.

Finally, related work on modeling the travel planning problem has been reviewed.

Chapter 7

A Commercial Application for Travel Planning: *reality*

In theory, there is no difference between theory and practice.
But, in practice, there is.

Jan L.A. van de Snepscheut, 1953-1994.

7.1 Context

reality is an application commercialized by i:FAO AG[1] for business travel planning. However, the origin of *reality* is located at the Swiss Federal Institute of Technology in Lausanne, in a joint research project with Swissair[2] started in September 1997. From 1997 to April 2000, a prototype for planning travels on the web, called *SmartTravel*, was developed by Sebastian Gerlach and Marc Torrens, under the direction of Boi Faltings and Pearl Pu. After realizing an increasing commercial interest by SwissAir about the prototype, Iconomic Systems SA[3] took off with the idea of commercializing *SmartTravel*. In Iconomic Systems SA, *SmartTravel* became *IsyTravel*. Finally, Iconomic Systems SA was acquired by i:FAO AG in May 2001. Since then, *IsyTravel* became *reality*. Many people[4] have contributed to the success of *reality*, and therefore this chapter only intends to illustrate some of the key implementation points, and validates the contributions of this dissertation.

From this point on, the application will be called *reality* even if some interesting points were only present in previous versions. In other words, some of the explained features are no longer present in *reality* but were developed in previous versions.

[1] i:FAO AG, `http://www.ifao.net`.
[2] In 2002, Swissair became Swiss: `http://www.swiss.com`.
[3] A startup company founded by Boi Faltings, Pearl Pu, Christian Frei and Marc Torrens.
[4] My deepest gratitude to all people that contributed and still contributes to the development of *reality*. See the acknowledgments at the beginning of this document for details about *who* contributed on *what*.

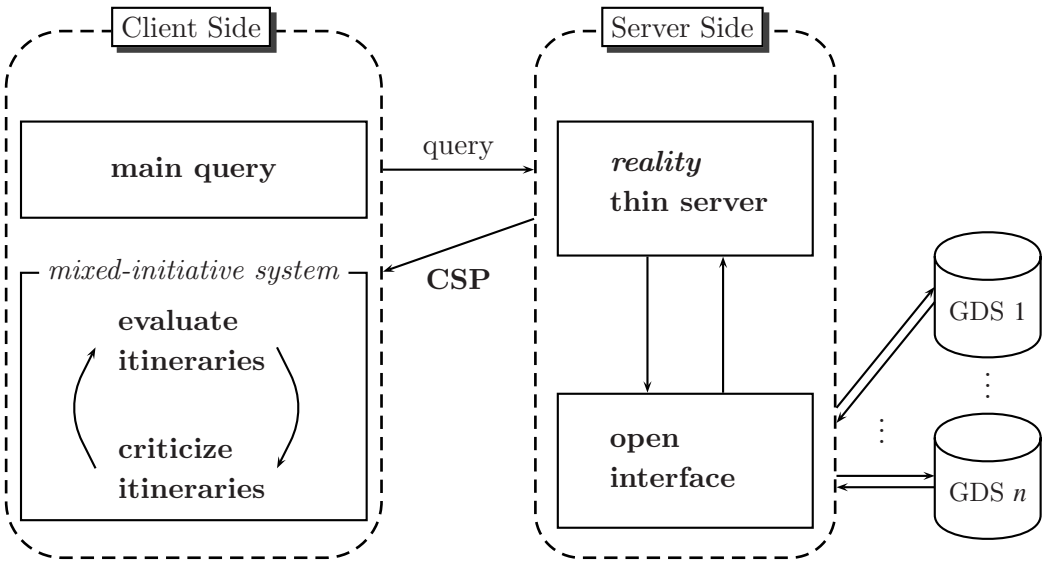


Figure 7.1: *reality* architecture: a *SmartClient* architecture for travel planning.

7.2 Architecture

The architecture of *reality* is shown in Figure 7.1. The system works basically as follows. The user enters his main request (it corresponds to an itinerary specification, see Definition 6.4). The request is then sent to the *reality* thin server where the gathering information process is done. After that, the user is able to use the mixed-initiative system on the client side, without accessing anymore the server side (except for booking a selected itinerary solution). The communication between the client side and the server side is done through HTTP[5] protocol, and the CSP is transferred using a proprietary compact form.

The following subsections explain the process of gathering information and planning travels in more detail.

7.2.1 Gathering information on the server side

The gathering travel information process is completely done on the server side. The *reality* thin server receives user main requests. These itinerary specifications are transformed into a set of queries to be sent to the *open interface*. The *open interface* component can be seen as a server that takes queries in a proprietary language and transforms them in GDS requests[6]. It is the bridge between *reality* thin server and GDSs. The reason of having a separate server to access the GDS information is to bring the whole *reality* system more independent from travel legacy systems. Actually, the *open interface* could be used by other applications, since it is developed independently of the specific *reality* requirements. The *open interface* processes the set of GDS requests in parallel.

Once the *open interface* returns the results back into proprietary data structures (simpler than those used by GDSs), the *reality* thin server builds a CSP corresponding to the

[5]HTTP: HyperText Transfer Protocol.

[6]The mechanism of the *open interface* access the GDSs is out of the scope of this thesis.

user requests with the GDS returned information. When the CSP is built, it is sent to the *reality* application, *i.e.*, to the client side.

Note that both server components, *reality* thin server and *open interface* process requests in parallel. The client for the *reality* thin server is the *reality* application, and the client of the *open interface* is the *reality* thin server.

7.2.2 Travel planning on the client side

One could consider that the travel planning starts once the *reality* application receives the CSP from the *reality* thin server. At that moment, the CSP is solved for the first time, and initial itinerary solutions are shown to the user. It is the first step of the mixed-initiative model based on a conversation. The user then has the opportunity to criticize any attribute of any of the proposed itinerary solutions. Such critiques are translated into soft constraints as explained in Chapter 6.

Partial Forward Checking is used to solve the CSP corresponding to the itinerary problem with user preferences and optimization criteria. See 4.2.2.1 more details on the PFC algorithm. The quantitative approach for combining constraints is adopted, and at each solving, the 30 best solutions are computed.

Value ordering is done according to the order of the flight sequences given by GDSs. Actually, GDSs order the flight sequences according to predefined criteria such as flying time, number of stops, and so on. Thus, this is a convenient order at the beginning when the user has not expressed his preferences. For subsequent searches with user preferences, the values are ordered according to the valuation of the unary constraints that express the preferences of the user. In the case that a fare variable is present, its domain is ordered by increasing the price of the fare values. Better values are tried first, as discussed in Section 4.2.3.

Variable ordering is done in the case a `fare-variable` is present in the CSP. In such a case, the variable modeling fares is selected first, since it produces many propagation through the leg variables.

7.2.3 Some implementation facts

All the described components, both on the client side and on the server side, are implemented in JAVA, using technologies like web services (weather forecast and currency exchange), EJBs[7], and servlets[8].

[7]Enterprise Java Beans: `http://java.sun.com/products/ejb`:That's because the EJB component model simplifies the development of middleware applications by providing automatic support for services such as transactions, security, database connectivity, and more.

[8]`http://java.sun.com/products/servlet/`: Servlet technology provides Web developers with a simple, consistent mechanism for extending the functionality of a Web server and for accessing existing business systems. A servlet can almost be thought of as an applet that runs on the server side (without a GUI).

The whole CSP solver (with data structures) is a JAR file of only 36 KB, showing that CSP algorithms are really extreme in their compactness as argued as a key point for *SmartClient* architecture (specially in its applet configuration).

The size of the transferred CSP is also a key point of the *SmartClient* architecture. The stream data size for CSP of a round-trip (two legs) is never more than only 4 KB, and less than 10 KB for a 5 legs. Noteworthy, the transferred CSP is compacted by a proprietary compacter component that avoid repeated data, like airlines, airport codes, aircraft types and so on. Thus, in our CSP data stream, the IATA codes are not repeated. Moreover the CSP data stream is zipped to even reduce more the CSP data stream size.

From an efficient solving performance point of view, solving a round trip takes about 100 ms, a three-leg itinerary takes 200 ms, and a four-leg itinerary takes about 500 ms. These solving times are approximative, and they depend on the specific problem and its user preferences.

7.3 Mixed-initiative system

In this section, the mixed-initiative system in *reality* is briefly described by pointing out some relevant points. Graphical user interfaces for the proposed mixed-initiative approach pose some challenges regarding the usability of the system. This thesis does not focus on this aspect, however, this section intends to explain some concepts that have been developed in *reality*, the goal being to illustrate the conversation that takes place between the user and the system. In Appendix A, a complete scenario using *reality* is discussed by providing a complete series of screenshots of the application.

7.3.1 Posting, modifying and retracting preferences

Posting, modifying and retracting preferences can be done in several ways. One mechanism could be the one shown in Figure 7.2, called *parallel coordinates* (inspired from [123]). In such display, each line represents one itinerary solution. Along horizontal space of the display, the relevant attributes of the itinerary specification are represented. For example, departure location, departure date, departure time, arrival time, arrival date, and arrival airport. The user can select one of the attributes for any criteria, for instance, a preferred arrival airport (`LHR` in the example of Figure 7.2). For expressing preferences on times, the user can select preferred time ranges by means of sliders (similarly to the diplay proposed in [224]).

Another type of display, based on menus, to express preferences is shown in Figure 7.3. In the solution display panel, on the bottom of the *reality* window, the attributes of the current itinerary solution are displayed. The current itinerary solution can be changed by means of next and previous buttons in the form of arrows. On each attribute, the user can activate a menu which will shown the options for a preference on that attribute. In Figure 7.3, the menu corresponds to a preference on the carrier. The two first options of the menu can be used to set a context for another preference (resulting in a contextual preference). Under the solution display, another display contains the entered preferences

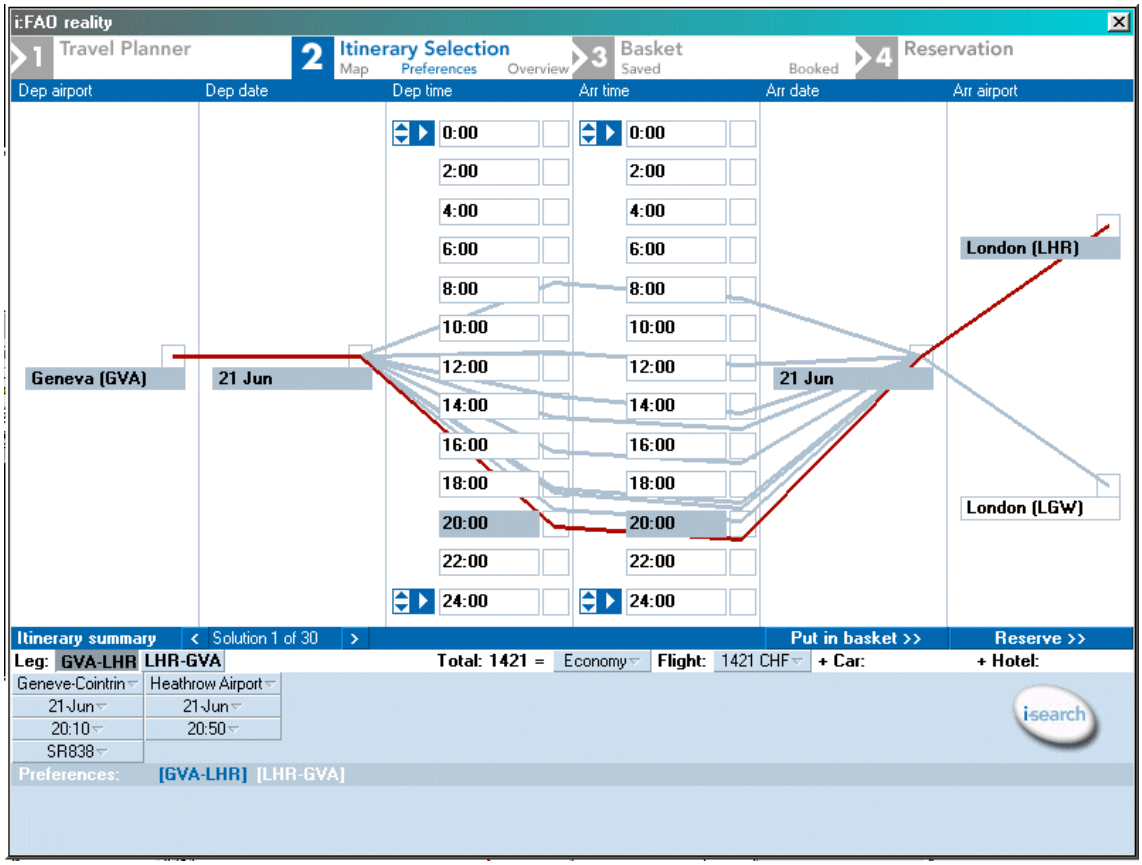


Figure 7.2: Parallel coordinates to express preferences.

in a textual mode. In this panel, the user can delete a preference, indicate to which leg the preference is activated, and also express weights to state different degrees of importance to preferences.

7.3.2 Showing violations

Showing the attributes of an itinerary solutions which are violated by some preference is a way of supporting the user in his purchase decision-making process. In Figure 7.4, a simple way of displaying attribute violations is shown. The attributes of the current solution which are violated are displayed in red. Different degrees of red, from light red to dark red, can be used to indicate different degrees of violation.

7.3.3 Solution displays

Apart from the standard way of displaying solutions, *i.e.*, by a textual description of their attributes, other sophisticated displays can be considered. For example, the parallel coordinates (shown in Figure 7.2) or the starfield plots (shown in Figure 7.3 and Figure 7.4). The starfield plot (similar to the one proposed in [3]) consists of a bidimensional plot where each axes represents a criteria. In that manner, the itinerary solutions can be graphically

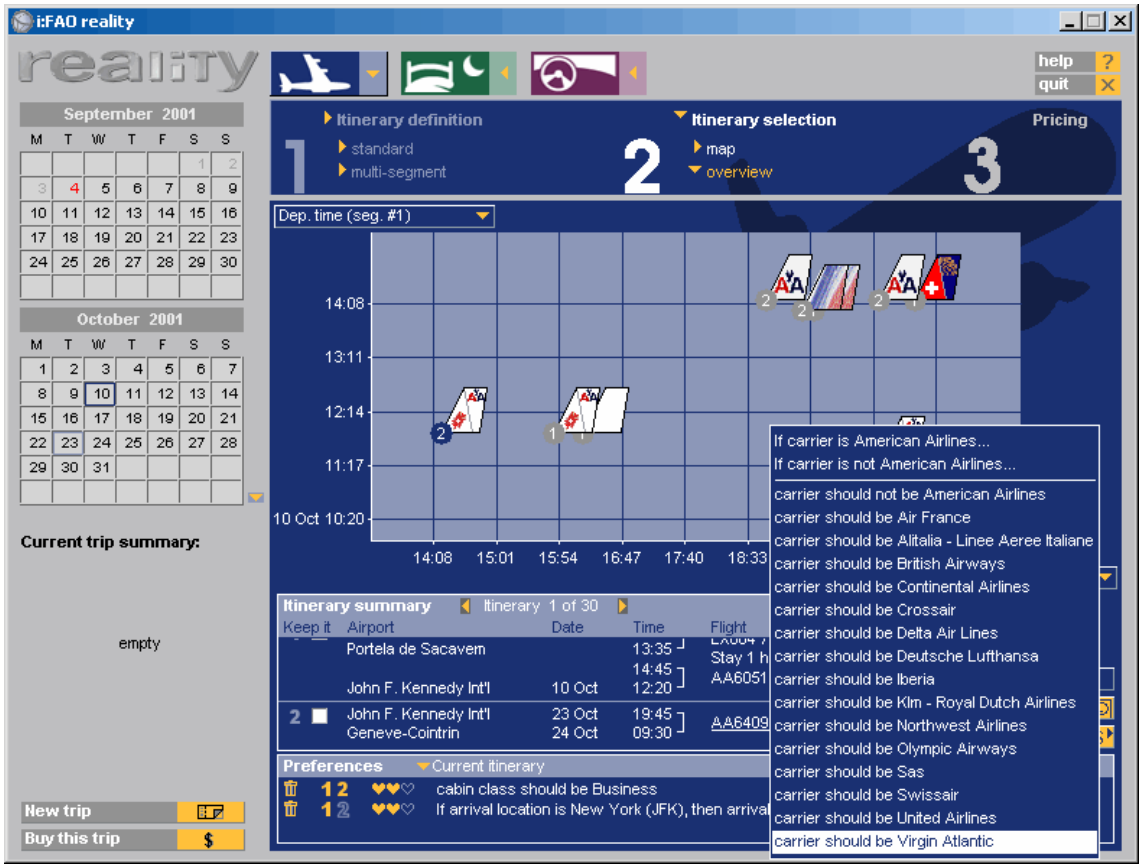


Figure 7.3: Menus and other graphical widgets to express preferences.

displayed accordingly. Each point in the plot where a solution is mapped is drawn with a flight tail with the airline logo of the solution (the empty tails represent an airline without a logo in the *reality* database). If the solution is operated by several airlines, several logos are displayed. The number on the bottom-left corner of the tail indicates the number of solutions which are mapped in the same point. The user can browse through the solutions on the same point by simply clicking on the flight tail.

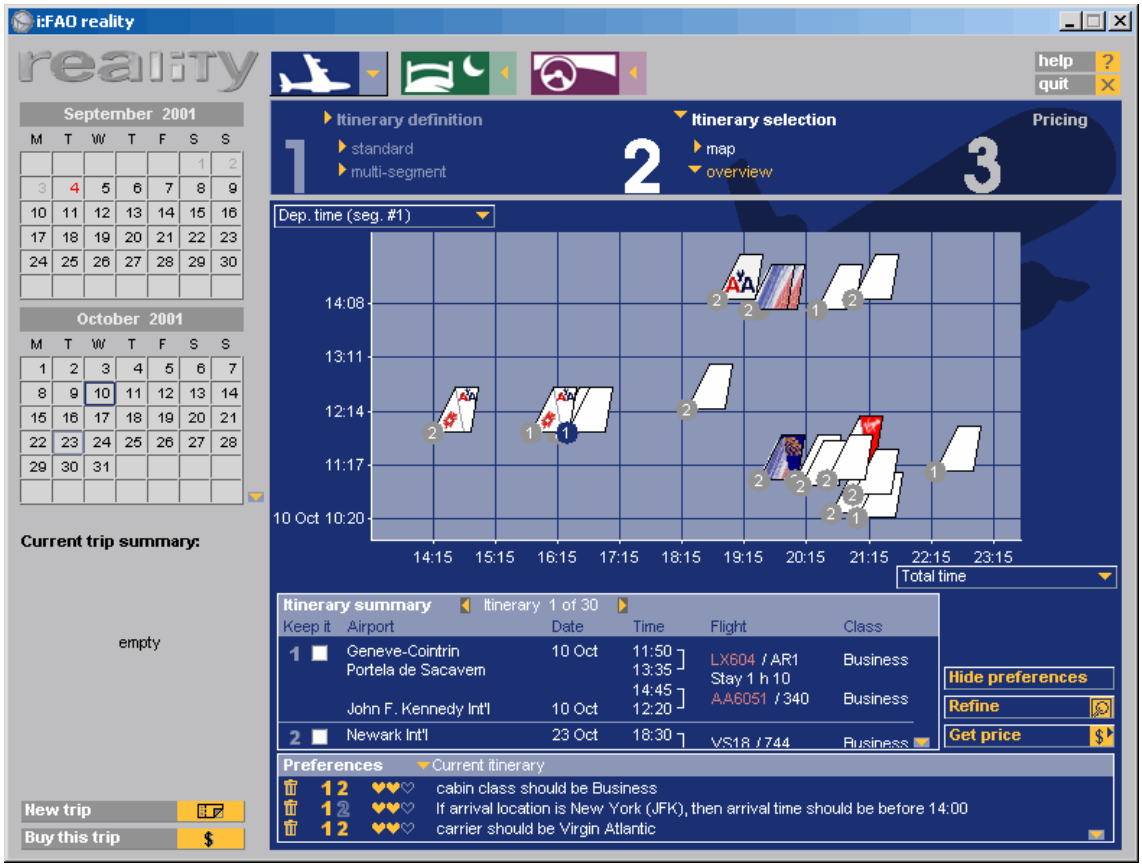


Figure 7.4: Showing violations on the attributes of an itinerary solution.

Chapter 8

Conclusions

Every problem becomes very childish once it is explained to you.

Sherlock Holmes, *The Adventure of the Dancing Men.*

8.1 Scope

Nowadays, Internet's world is full of information systems which make huge quantities of information available to people. This incredible amount of information is clearly overwhelming Internet end-users. As a consequence, intelligent tools to identify worthwhile information are needed, in order to fully assist people in finding the right information. Moreover, most information systems are ultimately used, not just to provide information, but also to solve problems.

Pushed by the growing popular success of the Internet and the enormous business potential of electronic commerce, e-catalogs have been consolidated as one of the most relevant types of information systems. Nearly all currently available electronic catalogs are offering tools for extracting product information based on key-attribute filtering methods. The most advanced electronic catalogs are implemented as recommender systems using collaborative filtering techniques.

Nevertheless, current electronic catalogs for complex domains are not directly facing the configuration problem with user's preferences. In complex domains, such as travel planning, interactive configuration tools are needed to fully support the user in the purchase decision making process. The main challenge of considering such advanced electronic catalogs is to incorporate sophisticated business logic while maintaining the scalability of the whole system. Traditional wisdom would state that, in order to build intelligent tools, complex code and data are unavoidable, consequently producing low scalable systems.

In this thesis, constraint satisfaction techniques applied to configurable electronic catalogs have been analyzed. Therefore, this thesis focuses on showing that constraint satisfaction is well-suited to cope with the difficulty of building intelligent and scalable electronic catalogs in complex domains. An electronic catalog for air travel planning has been developed to practically validate the techniques developed in this thesis work.

8.2 Contributions

This section outlines the main results and contributions of this thesis work.

8.2.1 Mixed-initiative system for electronic catalogs

The proposed mixed-initiative approach for electronic catalogs is presented in Chapter 2. A similar user interaction model was described by Linden *et al.* in [153] and a similar approach was used by a computer configurator in the Dell's web site. However, our approach relies on a more flexible constraint satisfaction model based on optimization techniques. Our conversational model is based on two observations of standard commercial activities. Firstly, personal criteria are not stated up front, and secondly, personal criteria are discovered by the customer by reacting to product examples during a dialog with the seller.

Following the above two observations, a mixed-initiative approach has been proposed for electronic catalogs. This mixed-initiative approach is based on a conversational model where the user criticizes product proposals provided by the catalog, the goal being to converge to a reduced set of satisfactory solutions. In networked environments, such as Internet, and in complex domains, such as travel planning, this interaction model clearly raises two challenges:

a) How can the system suggest the adequate products according to the critiques made by the customer to product proposals ?

b) How can the system be scalable while providing intelligent support to the user ?

Both questions are handled within this thesis, resulting in the following contributions.

8.2.2 Modeling and searching strategies for electronic catalogs

A uniform framework for modeling electronic catalogs using constraint satisfaction problems has been proposed. Three types of constraints have been identified, namely: configuration rules, user preferences and optimization criteria. In order to model these constraints, the classical CSP model is not flexible enough. It has been shown that weighted constrained optimization problems easily model all the constraints that a complex electronic catalog may involve. Two different approaches are considered:

Quantitative approach where all constraints are numerically combined. Such quantitative approach assumes that the user can precisely express the weights associated to the different criteria numerically. This assumption is particularly strong, since the user does not normally know the precise weights of his preferences, mainly because they dynamically change during the dialog with the system. The advantage of this approach is that it can be faced with very simple methods based on the well-known branch and bound algorithm.

Qualitative constraint combination approach based on the Pareto-optimality concept. This approach claims that only Pareto-optimal solutions are of interest to the user, thus dominated solutions can be filtered out. Since computing the whole Pareto-optimal set of solutions is computationally expensive, methods for approximating the Pareto-optimal set for an e-catalog have been proposed and evaluated. Surprisingly, the reported results show that branch and bound methods can be successfully used for finding approximations of the whole Pareto-optimal set.

A mixed approach. On one hand, the quantitative approach makes the assumption that constraint weights are numerically known, which could be considered too restrictive. On the other hand, the approach based on Pareto-optimality has the limitation of only providing a partial order. In addition to that, adding more criteria to a problem increases the number of Pareto-optimal solutions, without allowing the user to narrow down the set of solutions as desirable. A mixed approach based on the quantitative approach which filters out dominated solutions overcomes the main drawbacks of the Pareto-optimality based approach, while preventing to show dominated solutions.

8.2.3 *SmartClient* architecture

Scalability is a major problem in client-server architectures when handling complex business logic for many users simultaneously. *SmartClient* architecture is an architectural model that uses constraint satisfaction problems for representing solution spaces, instead of traditional models which represent solution spaces by collections of single solutions. This main idea is supported by the fact that constraint solvers are extreme in their compactness and simplicity, while providing sophisticated business logic. Different *SmartClient* architecture configurations are provided for different uses and architectural requirements.

8.2.4 Validation with a commercial application in the travel domain

The validation of the main contributions of this work has been done by applying them to a concrete application domain. The chosen domain is travel planning because of its complexity and relevance in the electronic commerce area. *reality* is a Internet-based application for travel planning that uses all the ideas presented in this thesis. In the travel domain, *reality* offers the purchase decision-making process that users need to solve problems, instead of just presenting product information.

From an engineering point of view, modeling electronic catalogs with constraint satisfaction techniques significantly simplify the maintenance of the business logic of the whole system. For example, considering a new kind of preference only implies the addition of a constraint in the model, the constraint solver remains exactly the same. The same advantage takes place when adding new variables, values or other constraints. In more standard programming paradigms, changing functional requirements would imply much more cumbersome adaptations from the implementation point of view.

8.3 Limitations

Limitations of the presented approach can be summarized in the following way:

Representative solution space generation: In the presented work, solution spaces are generated according to configuration rules, user preferences and optimization constraints. Therefore, the user gets solutions according to his personal criteria, however, it would be beneficial for the user, to get a representative set of the whole solution space as well. This is especially interesting at the beginning of the conversation with the system, when the user still does not know his preferences and would probably like to have an overview of the catalog. In addition to the solution space according to preferences and optimization constraints, it would be adequate to present extreme solutions, *i.e.*, solutions which are optimal for some set of predefined criteria. In this way, the user could evaluate a more heterogeneous set of solutions and discover more preferences.

Filtering out Pareto-optimal solutions: It has been argued in this thesis that dominated solutions should be filtered out because they would never be chosen rationally. Nonetheless, hiding dominated solutions may imply to miss good solutions, which are dominated merely because the user did not express all his criteria. Actually, such limitation is in relation to the previous paragraph.

Analysis of trade-offs: In order to better support the user, trade-off analysis of the proposed products would be useful. One could think about graphical displays helping users to evaluate what criteria are not satisfied and to what extent. Moreover, in many cases, satisfying one criteria implies the violation of another criteria, because they are directly interrelated. Such analysis of trade-offs would certainly help users to understand the available options of the catalog and select the best product. In *reality* a first approach for trade-off analysis is implemented as bidimensional starfield plots. Each axis of the plot represents one criteria which can be selected by the user. Points on the plot shows where the products are situated in respect to their criteria violations. Pu and Lalanne proposed in [194, 195, 141] methods for visualizing tradeoffs and to help the user to discover conflicts among his preferences. Such techniques could be of great support for the user in our framework.

Expressing complex preferences: Complex preferences naturally arise in complex domains, such as travel. People are used to formulate preferences with many conditions and relations among them. Constraint satisfaction can handle these kind of constraints, however, it is not clear how to express such preferences in standard graphical user interfaces.

Handling too many criteria: Using *reality*, it has been observed that users tend to express too many preferences. In these cases, the system becomes *overconstrained* and criteria do not behave intuitively. This is mainly due to the fact that in overconstrained

systems, criteria valuations tend to cancel each other out, producing unexpected results. Therefore, overconstrained situations should be prevented, in order to avoid the user being mislead.

Contradictory criteria: A mechanism to detect inconsistencies among personal criteria would improve the whole approach. Rationally, a user would never express contradictory criteria. Nevertheless, in cumbersome domains, users can easily be contradictory. Such phenomena is automatically corrected by a human seller, thus it would make sense to automatically detect these contradictions in electronic catalogs.

8.4 Further research

The above limitations indicate that more research can be undertaken to improve the approach presented in this thesis. Also, new research directions for electronic catalogs are pointed out. This section presents future research directions to improve and extend the presented work.

Applicability to other complex domains: This thesis has been applied and evaluated in the travel domain. Nonetheless, other complex domains must be deeply analyzed in order to evaluate the applicability of the main contributions of this work to other domains. Complex examples were enumerated in Section 1.3.1.

Multi-agent scheduling meetings: In this thesis, the air travel planning problem has been tackled. However, a more general problem in the travel domain is to arrange meetings for several participants taking into account constraints from personal agendas. Some research on multi-agent scheduling meetings using constraint satisfaction has been done [157, 73]. Since these research works already make use of CSP frameworks, it seems convenient to benefit from some of the results of this thesis in the problem of scheduling meetings using constraint satisfaction and multi-agent systems.

Collaborative filtering: Collaborative filtering reuse previous product selections to suggest adequate products to users. A mixed approach of collaborative filtering and the mixed-initiative model proposed in this thesis could be interesting. Especially at the beginning of the conversation between the user and the system, collaborative techniques could be used to propose initial sample of products. This would be beneficial at the beginning, when the user still does not precisely know his preferences. Instead of proposing products only based on optimization constraints, the system could propose products which have already been selected by other similar users. This could yield to a quicker convergence of the solution space.

Context-aware computing: A joint research project between i:FAO and the LIA at the EPFL has started about how to use contextual information in *reality* when planning

travels. Contextual information includes, for instance, weather information, statistics on delays in certain airports, or international news. The idea is to benefit from contextual information in order to suggest criteria to the user. For example, a user willing to travel to San Francisco in winter would be interested in knowing that Chicago airport tends to be a bad option because of terrible weather causing delays, so he could be alerted in order to avoid such an intermediate airport.

Open constraint satisfaction problems: Faltings *et al.* [61] describe a framework where OCSP addresses the issue of gathering information from different sources incrementally building up an OCSP while maintaining a correct solution set. The goal of an such approach is to avoid gathering information which could never be needed for a solution set. In this thesis, information gathering and constraint solving are decoupled, thus much accessed information could never be used in the solution set. OCSPs are especially interesting in *open-worlds* where choices and constraints are to be discovered from different servers in a network. Such approach could be analyzed together with our mixed-initiative model in the framework of complex electronic catalogs, where minimizing the amount of gathered information is an important issue for efficiency and scalability.

Constraint posting using natural language: Expressing complex preferences about product examples is a challenge from a graphical user interface point of view. Standard menus and graphical widgets are often not powerful enough. Natural language could be used to state complex preferences. An initial project in this direction was undertaken at the EPFL [80] with preliminary but successful results. However, deeper studies must be made in order to really evaluate the feasibility of expressing complex preferences by natural language in complex domains.

Abstractions and interchangeability: Abstractions and interchangeability in classical CSPs were deeply studied by Weigel in [260]. These concepts for soft CSPs have been recently tackled by Bistarelli *et al.* in [16]. Basically, these concepts can be used to simplify a CSP by identifying interchangeable values. In this way, a new solution can be produced by simply replacing the interchangeable values in an already known solution. These techniques could be used in our framework where values in domains present similarities and regularities. For example, in the travel planning problem, some airlines operate the same flight each day of the week.

Reuse of search tree: When the user adds or modifies a set of preferences, the search is performed from scratch without taking into account previous search processes. Nonetheless, the existing explored search tree could be reused. The penalties for all intermediate nodes and leaf nodes of the search tree need then to be updated. Solutions can then be recomputed using this updated tree. Note that some previously skipped subtrees may be explored in new searches. Reusing the work already done by previous searches could make the search process more efficient.

Representative solution generation: In our approach, solutions are generated according to user preferences and optimization constraints. This approach seems to be natural and logical, however, one inconvenience is that showing the solutions that best fit user's criteria may avoid to discover products that could also fit the user needs. In other words, some products could be never shown if the user do not express all his preferences. Since preferences are stated reacting to samples of products, it is of interest to show a representative set of products of the catalog, even if some of them do not fit the criteria expressed by the user. A representative solution generation would allow the user to discover criteria that would be otherwise hidden.

Learn user preferences Users tend to have the same preferences for different purchase situations. For example, a user that was interested once in terror movies, will be probably interested in this kind of movies again. In complex domains, such as travel, it is not so clear that preferences are similar for different itineraries and the same traveler, mainly because they strongly depend on the itinerary itself. However, some preferences can be repeated often by the same user, for example about certain airlines, or schedules. Techniques for learning those preferences depending on the historical preferences of a user could be used in complex electronic catalogs.

8.5 Conclusion

Currently, most of the electronic catalogs are designed merely to provide product information to the user. Nevertheless, in complex domains such as travel planning, electronic catalogs are not ultimately used just to provide product information but to solve problems, *i.e.*, to help the user to find his best product. This thesis has contributed to the challenge of designing electronic catalogs within complex domains that are able to really support end-users in their purchase decision-making process. The contribution ranges from a modeling framework to solving strategies, including software architectural aspects.

The presented concepts have been shown to be very effective in the framework of a commercial application for planning air travels. This application, called *reality*, clearly improves the functionalities provided by other available travel planning tools, while maintaining scalability in a client-server architecture.

Appendix A

Using *reality*: a scenario

The best way of showing *reality* in detail from a user's point of view, is by illustrating the usage of the system through a concrete scenario. The initial idea of the scenario and the associated screenshots have been provided by Patrick Hertzog[1]. The screenshots illustrating the scenario are placed at the end of the chapter for improve the readability of the text. More details about the implementation of *reality* can be found in [241].

A.1 The scenario using *reality*

Andy[2] is a busy businessman living in Bern. He has a meeting in Palo Alto (Silicon Valley) on January 13th and a conference about new emergent travel opportunities in New Orleans on January 15th. Since New Orleans is the greatest city in the world for jazz lovers, and Andy is a real jazz lover, he would like to take the opportunity to stay in New Orleans until January 16th.

Since Bern is between Geneva airport (`GVA`) and Zürich airport (`ZRH`), Andy may depart from either `GVA` or `ZRH`. He does not really know what is the nearest airport to Palo Alto, so he uses the *reality*'s world map to find out what are the right airports to go to (Figure A.1). Our user decides to only consider San Francisco airport (`SFO`) and San Jose airport (`SJC`) because he remembers now that in the Oakland's airport the shopping offer is not so good.

The first leg of Andy's itinerary, `GVA` or `ZRH` → `SFO` or `SJC` on January 12th, can be now defined as illustrated in Figure A.2. Dates can be entered by clicking on the graphical calendar or by typing them textually. In a similar way, Andy enters the data related to the second and third legs of his itinerary (Figure A.3 and Figure A.4). At each leg definition, a dotted line linking the airports of the current itinerary are displayed. Finally, the Andy's itinerary has been defined by a query with the following three legs:

1. `GVA` or `ZRH` → `SFO` or `SJC` on January 12th,

2. `SFO` or `SJC` → `MSY` on January 14th, and

[1] My deepest gratitude to him for his involvement in assisting me with the elaboration of this chapter.
[2] Any similarity between the contents of this chapter and real events are purely coincidental!

3. `MSY` → `GVA` or `ZRH` on January 17th.

After sending the main request through the `Search` button, Andy receives the first proposed itineraries (Figure A.5). At this initial stage, the main query defines a very huge product space (3,141,855 different alternatives !). Hence, Andy can start the conversation with the system by posting preferences, the goal being to reduce the solution space to a small set of satisfactory products.

Andy is a member of the frequent traveler program of the following airlines: Swiss (`LX`), American Airlines (`AA`), and United Airlines (`UA`). So, obviously, he prefers travel with those companies. Since he is a really busy businessman, he would like to be able to work in the plane, thus he prefers business class. In addition to that, he really does not want to flight in economy restricted class because he must be able to change his reservation if needed. All these preferences are entered by Andy, as shown in Figure A.6. Each time a new constraint (preference) is posted in the system, Andy receives new proposals according to the whole set of preferences.

The system now proposes an itinerary which is quite good for Andy, but since his meeting on January 13th starts early in the morning, he does not want to land at `SJC` so late. He thinks that it would be much better to arrive before 5 p.m. (Figure A.7). Now, the first leg of the proposed itinerary really satisfies him, and he decides to keep it (check box on the right, Figure A.7). By keeping a leg of an itinerary, the system will maintain it invariant and only the other legs will be changed according to the new preferences along the rest of the conversation with the system.

Now, Andy looks at the second proposed leg, and he realizes that he would fly with a `Boeing 737` and it is not his preferred aircraft. So, a constraint expressing that he does not like `Boeing 737` is entered (Figure A.8). The system now proposes to fly with an `Airbus 320` from `SJC` to Minneapolis `MSP` and with an `Airbus 319` from `MSP` to New Orleans `MSY`. Unfortunately, since the trip will take place in January, he does not want to connect via Minneapolis because in winter, the weather in that city is awful, and therefore the probability to be delayed is very high. Andy looks now at the possible connecting alternative airports (Figure A.9) and connecting to Houston (`IAH`) seems to be a much better idea. But now, the system suggests again to fly with a `Boeing 737`. At this stage, Andy thinks that choosing between to fly with the `Boeing 737` or to have a high probability to be late at the meeting, the choice is quickly done: he prefers to be on time. He understood that there was a trade-off to be done among these two inconvenient situations, and often the perfect solution does not exist. Thus, he decides to keep this second leg as illustrated in Figure A.9.

Looking at the third leg of the proposed itinerary, Andy realizes that the proposed solution includes a flight operated by `AirFrance` (`AF`). He has already flown with `AF` and had very bad meals, so he does not want to use it again. Such a preference posting is illustrated in Figure A.10. The proposed third leg of the itinerary arrives at Geneva

airport. Since Andy will leave from Zürich (first leg) and will go to the airport by car, he needs to come back at Zürich. Andy expresses this requirement as shown in Figure A.11.

Finally, the last proposed solution (Figure A.11) satisfies him, and Andy continue with the booking process. After selecting the right solution, Andy can now continue with the `Get Price` button. The system shows now the fare and fare rules of the selected itinerary, as shown in Figure A.12. Through the `Book Selected` button Andy can book the itinerary. He can also put it in the `My Choices` panel, and search for more potential better alternatives.

A.2 The scenario using a standard travel planning tool

In this section, the same scenario will be illustrated using *cytric v7*. *cytric v7* is a travel planning tool which is commercialized by i:FAO. The goal of showing the same scenario with *cytric v7* is to point out the advantages of using the concepts developed this thesis. It is worth to note, that both tools, *reality* and *cytric* use the same technology base, *i.e.*, the same access to GDSs. Actually, both applications use *arctic* which is an interface to several GDSs developed by i:FAO.

In *cytric v7*, the only way Andy has to enter his preferences about airlines is by using a user profile. Thus, in the following it is assumed that Andy has his preferred airlines (`LX`, `AA` and `UA`) stated in his profile.

In *cytric v7* the user is not able to enter multiple airports and dates. Thus, Andy has to choose only one departure airport. Let us say that he decides to take off from Geneva. If he wants to see if better solutions exist from Zürich, he has to start the whole planning process from scratch. For the destination in Palo Alto, he only enters San Francisco Figure A.13.

Once Andy is satisfied with the definition of his itinerary, the search is launched. The results are displayed as shown in Figure A.14. The different flights for each leg are displayed in a list format. In this scenario, 8 flights are proposed for the first segment, 8 for the second and 5 for the third. Therefore, Andy can evaluate and chose among $8*8*5 = 192$ solutions in total. Of course, all the available flights are not showed in such a first display: these are only the first ones ordered by their departure time and the user can get the follosing ones by clicking on the appropriate button. Each time Andy requests for the following flight combinations, a new request to the server is performed with the consequent response time until the new results are displayed. By using *reality*, the user can directly choose among all the possiblities, that means for that scenario, 3,141,855 different flight combinations.

By browsing the flights proposed for the first leg, he selects the one that best fits his needs, *i.e.*, leaving not too early, arriving not too late and in business class. Afterwards, Andy continues scrolling down to the proposed flights for the second segment, as shown in Figure A.15. He does not want to fly with Frontier Airlines because only the economy class is available and moreover hed dislikes the `Boeing 737`. Andy sees that there is a

direct flight operated by United Airlines but there are only sears in the first class (anyway, it is not possible to flight with business class for this leg). Andy is happy with this flight and selects is. Andy continues scrolling down the list, and gets the flights for the last leg as shwon in Figure A.16. for the last leg, he selects the Delta Air Lines flight and clicks on the *continue* button to review his selections and get the price Figure A.17.

A.3 Comparison between *reality* and *cytric v7*

Clearly, *reality* outperforms *cytric v7* (and other standard travel planning tools) in terms of functionality. Table A.1 summarizes the functionalities provided by *reality* which are missing in standard travel planning tools.

Missing functionality	Reason
Multiple airports	Combinatorial explosion: considering several airports per leg exponentially increases the number of possible flight combinations. Since flight combinations are displayed in lists, the number of flight combinations to be considered is reduced.
Multiple dates	Combinatorial explosion: idem as above.
Preference elicitation	Considering preferences in combinatorial problems is not feasible withouth using a poweful business logic. Standard web-based systems are not able to implement easily powerful business logic.
Conversational model	In standard web-based systems, each user interaction implies to send a request to the server side. The results of such requests may take long time. Consequently, mixed-initiative systems cannot be easily implemented using standard web-based techniques, where all the business logic is located on the server side.

Table A.1: Comparison of the functionality provided by *reality* which are missing in standard travel planning tools.

Figure A.1: By using the graphical world map in *reality*, the user can decide what airports are the most appropriate for his itinerary. On top of that, the user is also able to take a look at the current weather in the airports.

Figure A.2: Defining the main query. The user enters the dates and locations of the first leg of his trip.. Note that the *My Choices* panel contains previous booked itineraries.

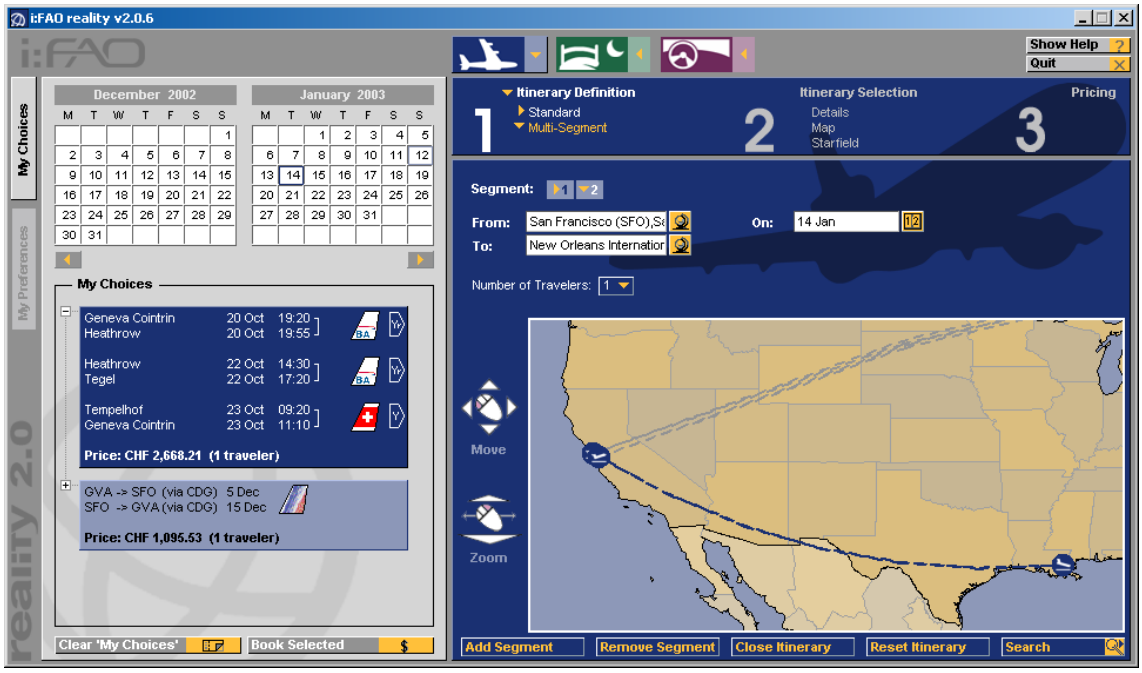


Figure A.3: Defining the main query. The user enters the dates and locations of the second leg of his trip.

Figure A.4: Defining the main query. The user enters the dates and locations of the third leg of his trip.

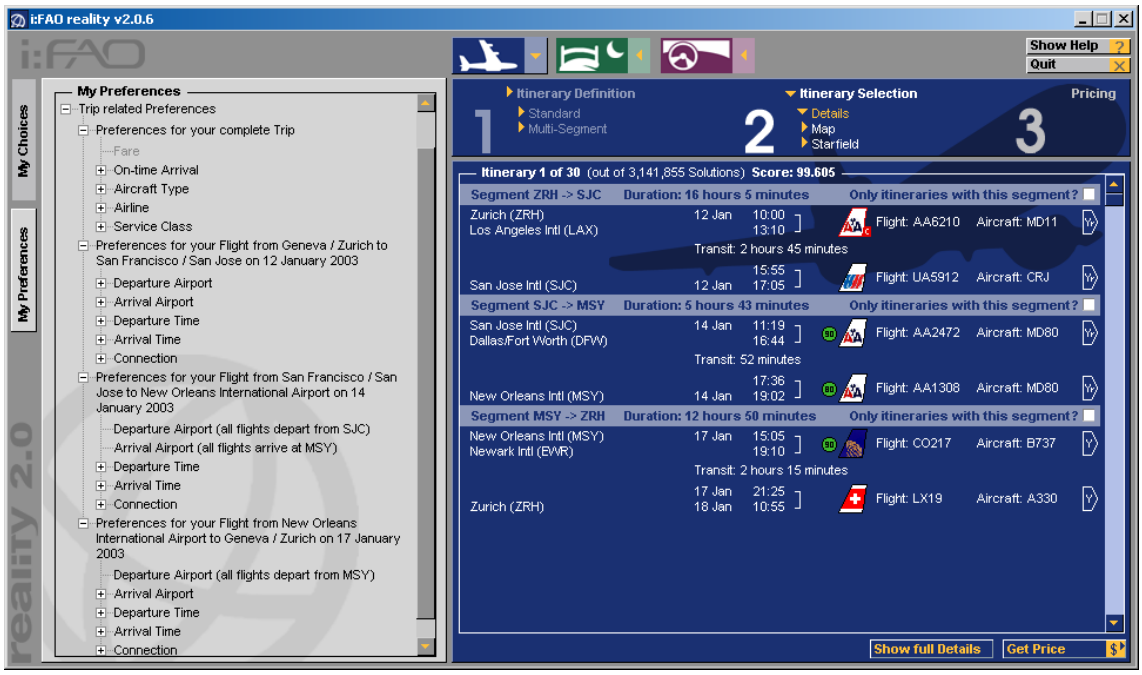


Figure A.5: The user receives the first solutions proposed by the system according to the main query. Since the initial query was very vague, it defines more than 3 million different alternatives.

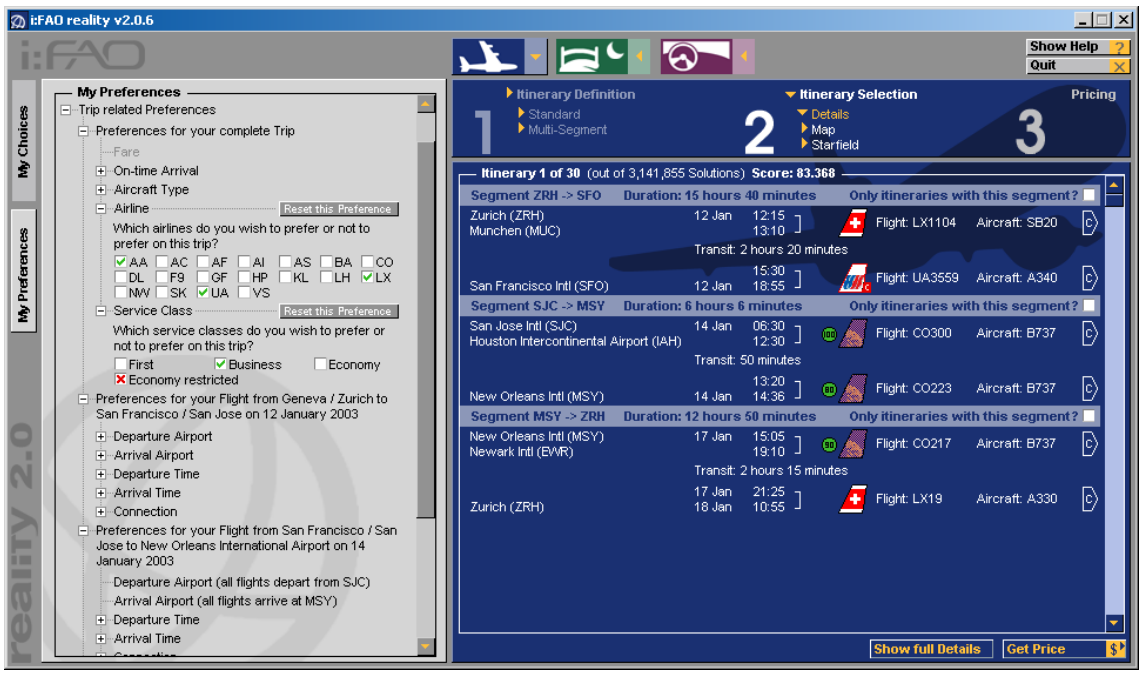


Figure A.6: After looking at the first solution, the user criticizes it by posting preferences. Preferred airlines: `AA`, `UA`, and `LX`. Preferred class of service: business, and disliked class of service: economy restricted.

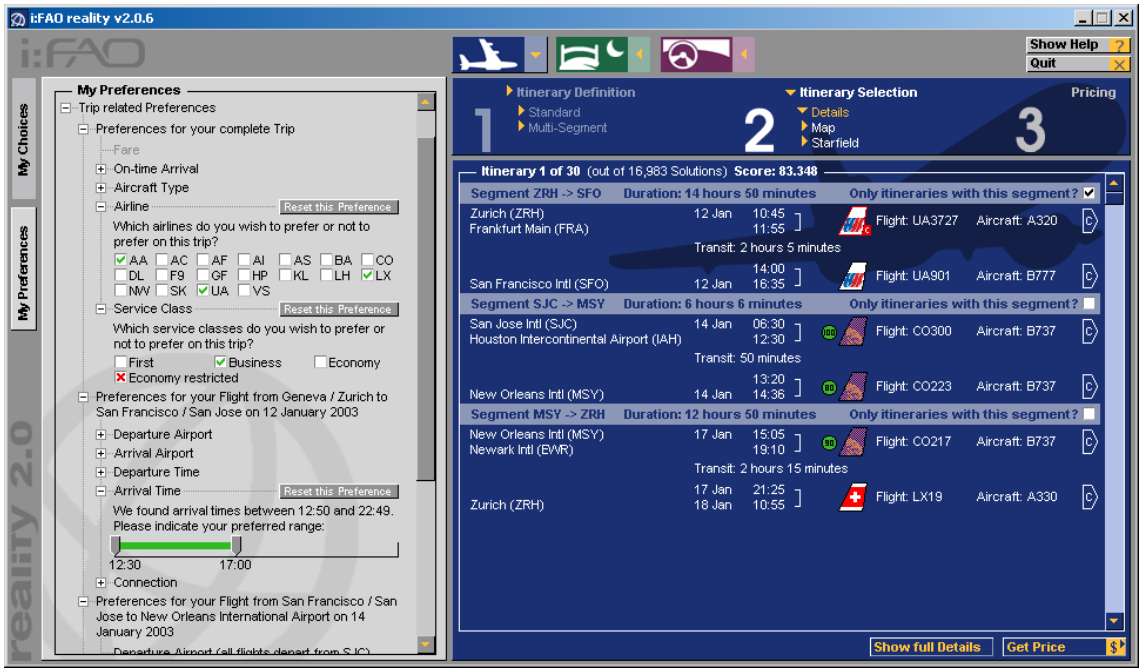


Figure A.7: The user prefers to arrive at the first destination before 5 p.m. Then, the first leg of the itinerary proposed by the system satisfies him, and decides to keep it by clicking on the check box on the right.

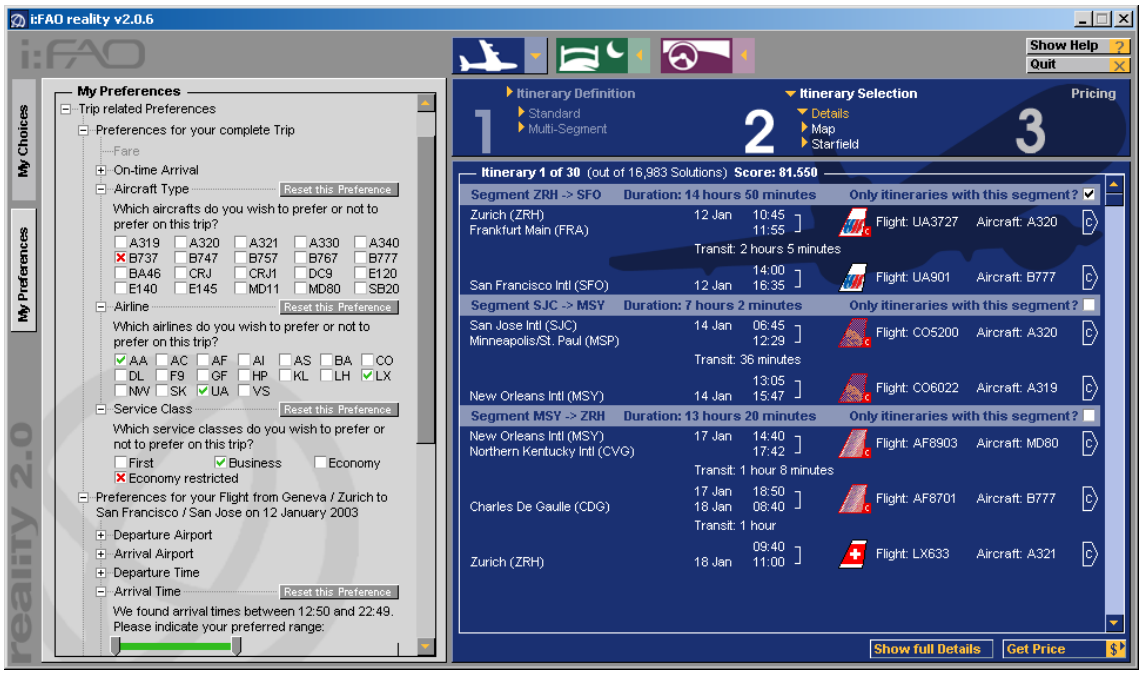


Figure A.8: The user does not like the aircraft of the second leg proposed by the system (a `Boeing 737`), so he indicates this preference. Then, the user receives a proposal for the second leg with an `Airbus A320`.

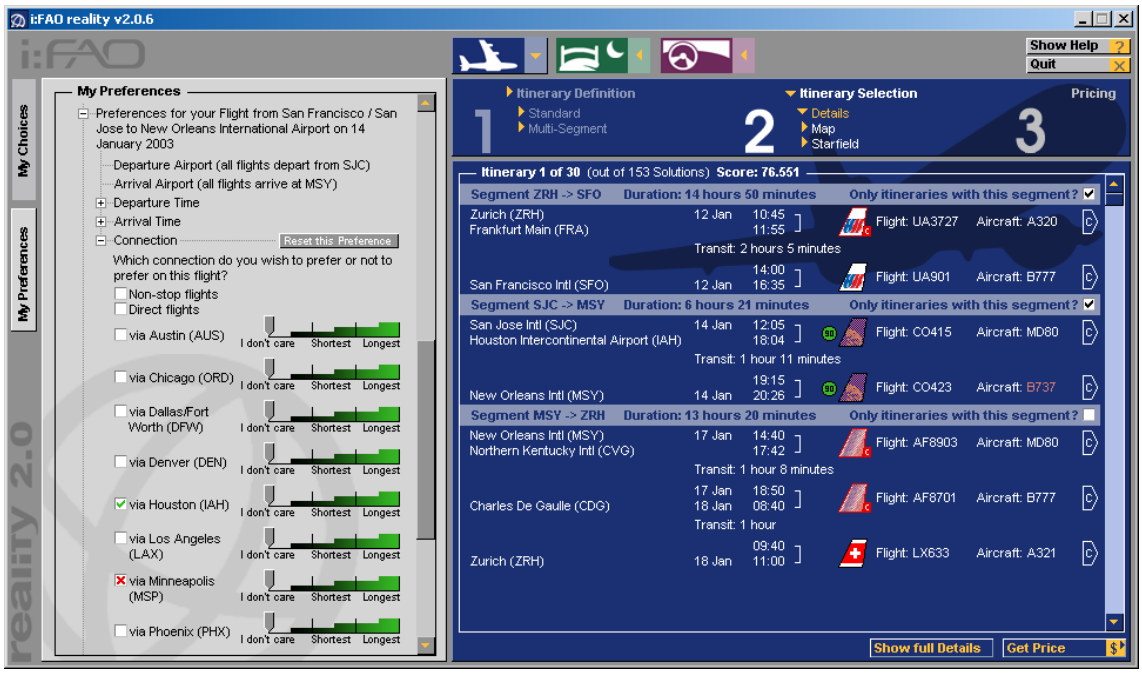


Figure A.9: In the received itinerary, the second leg connects via Minneapolis. Since in winter, Minneapolis is very cold and delays are frequent, the user decides to force the system to propose flights connecting via Houston. The new proposed second leg connects now via Houston, but contains flights with `Boeing 737`. However, the user thinks that it is a good trade-off and decides to keep it as is.

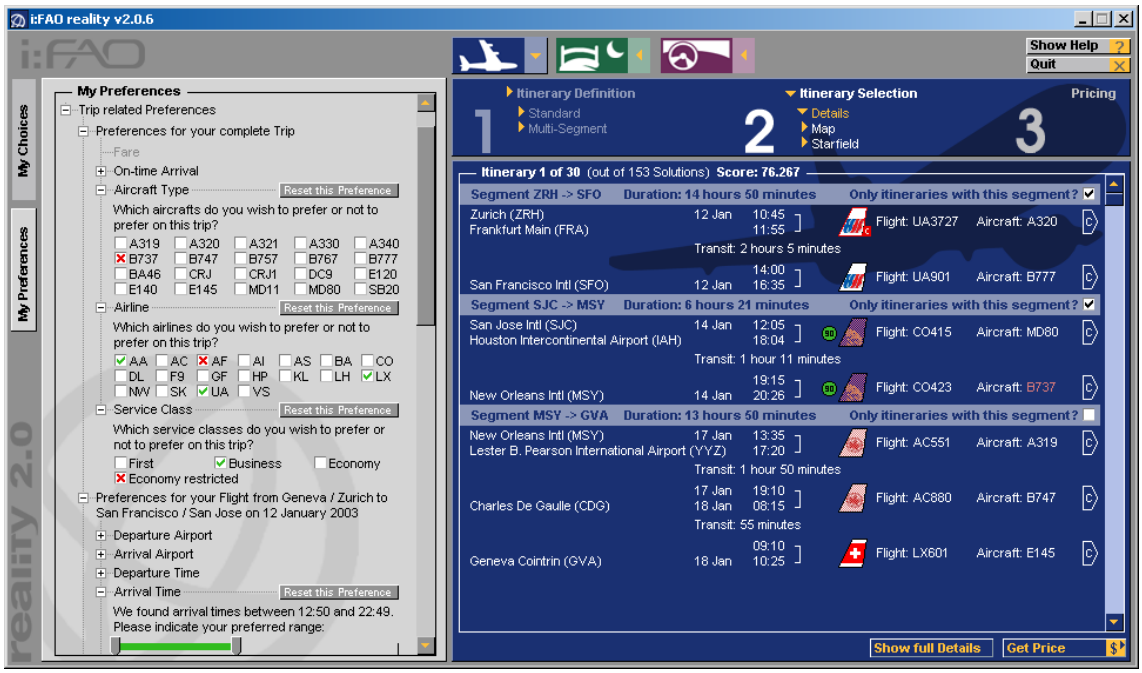


Figure A.10: The user does not like to fly with `AirFrance`, so he posts a preference for avoiding `AirFrance`.

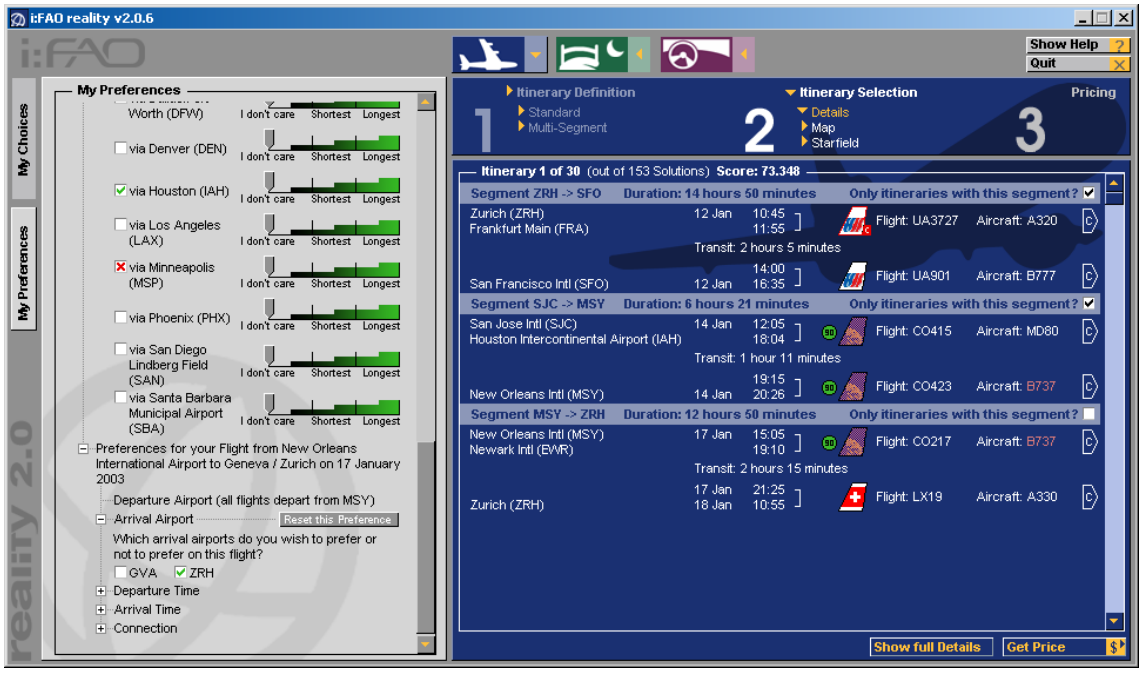


Figure A.11: Finally, the solution satisfies the user and decides to continue the process.

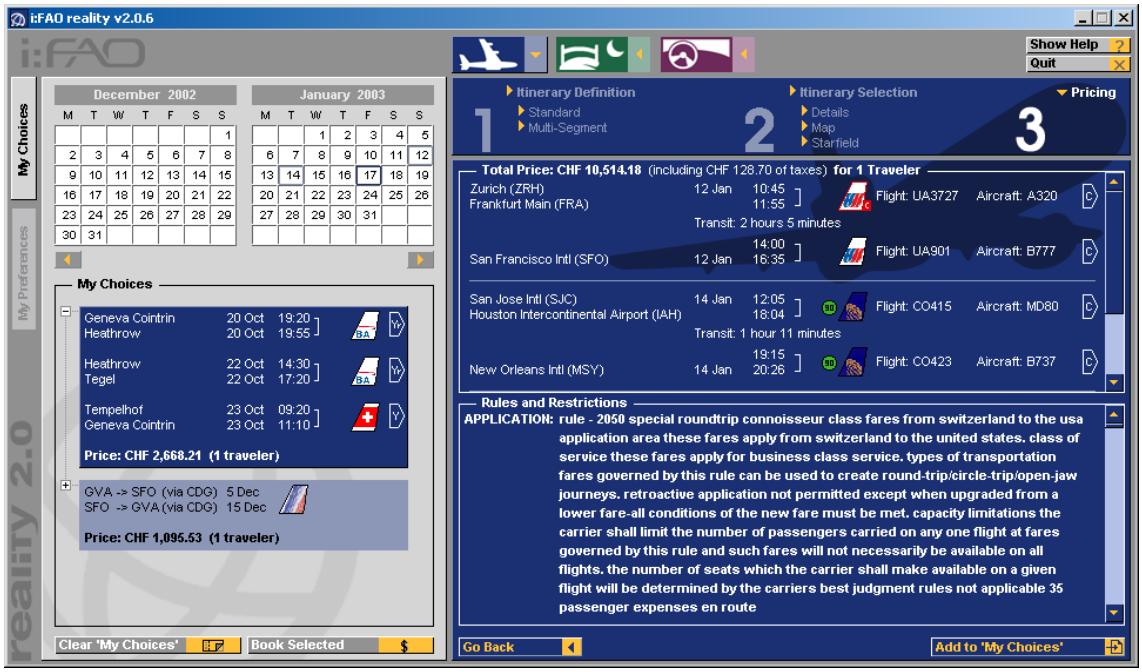


Figure A.12: The selected itinerary has been priced and fare rules are shown to the user.

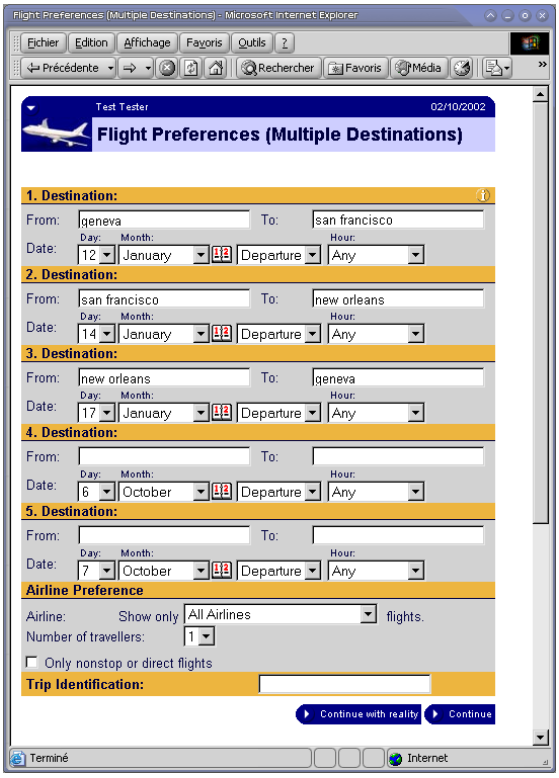


Figure A.13: Defining the itinerary of the scenario with *cytric v7*.

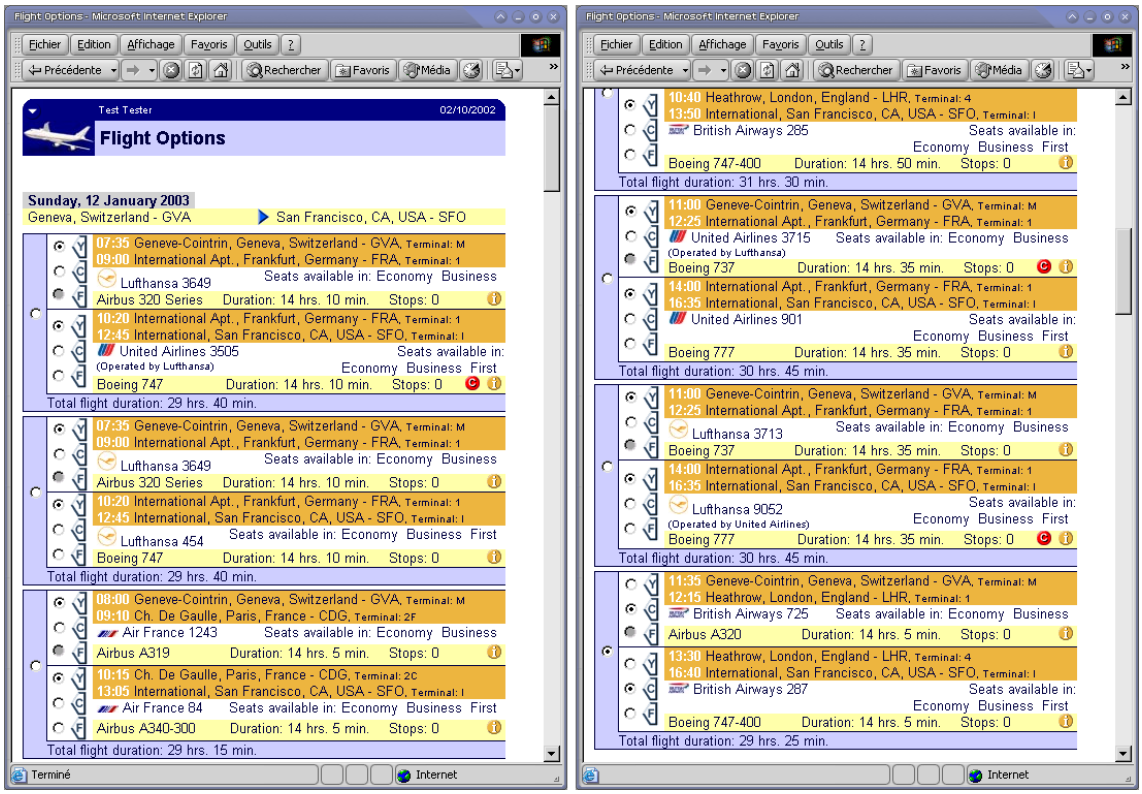


Figure A.14: Results for the flights of the first leg of the itinerary with *cytric v7*.

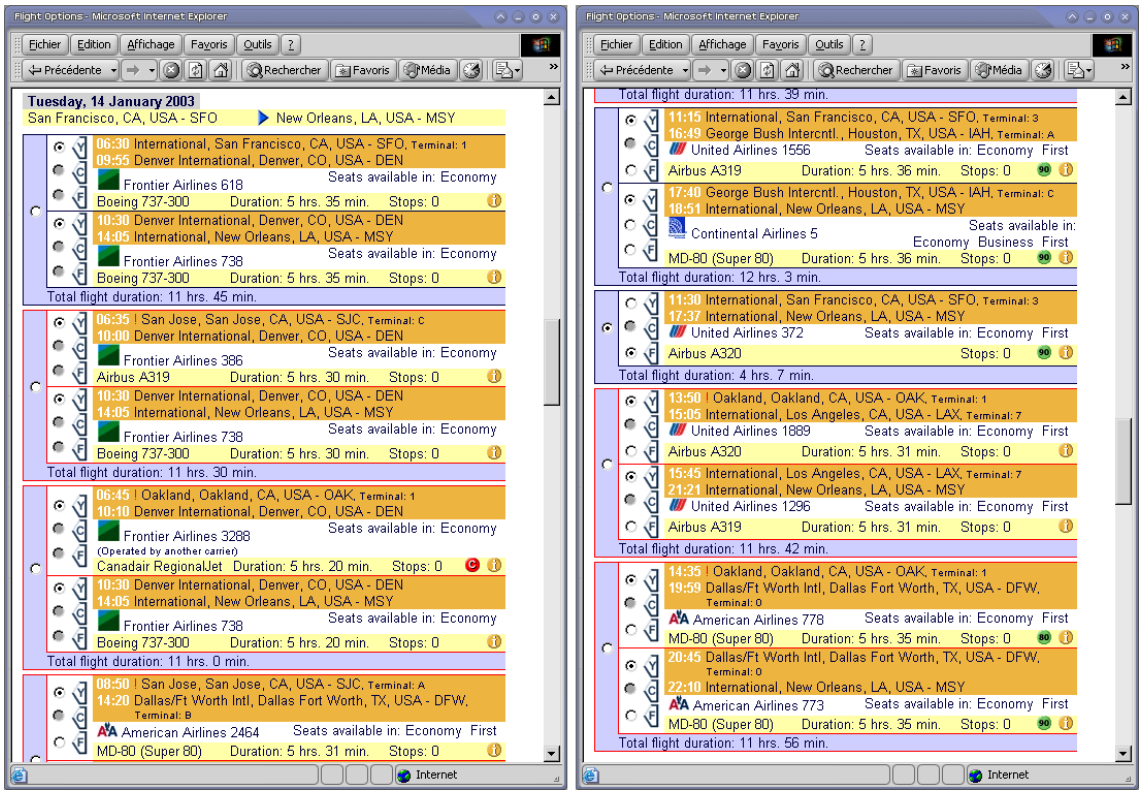


Figure A.15: Results for the flights of the second leg of the itinerary with *cytric v7*.

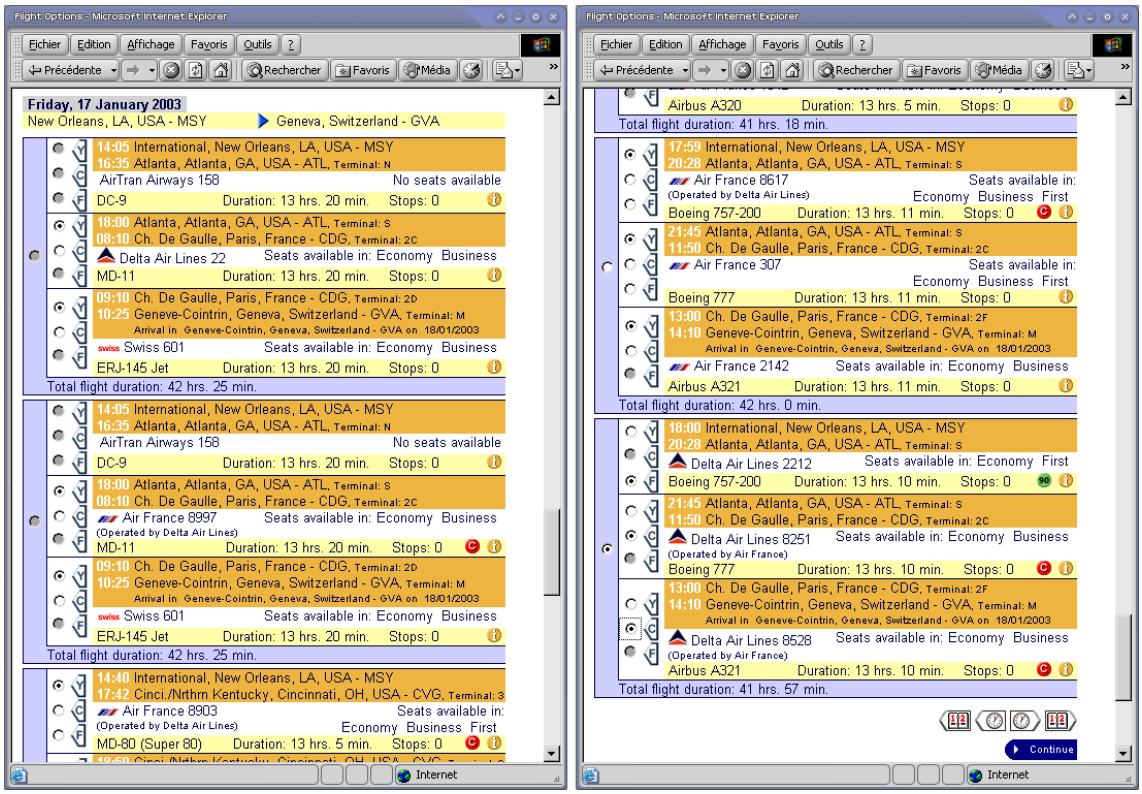


Figure A.16: Results for the flights of the third leg of the itinerary with *cytric v7*.

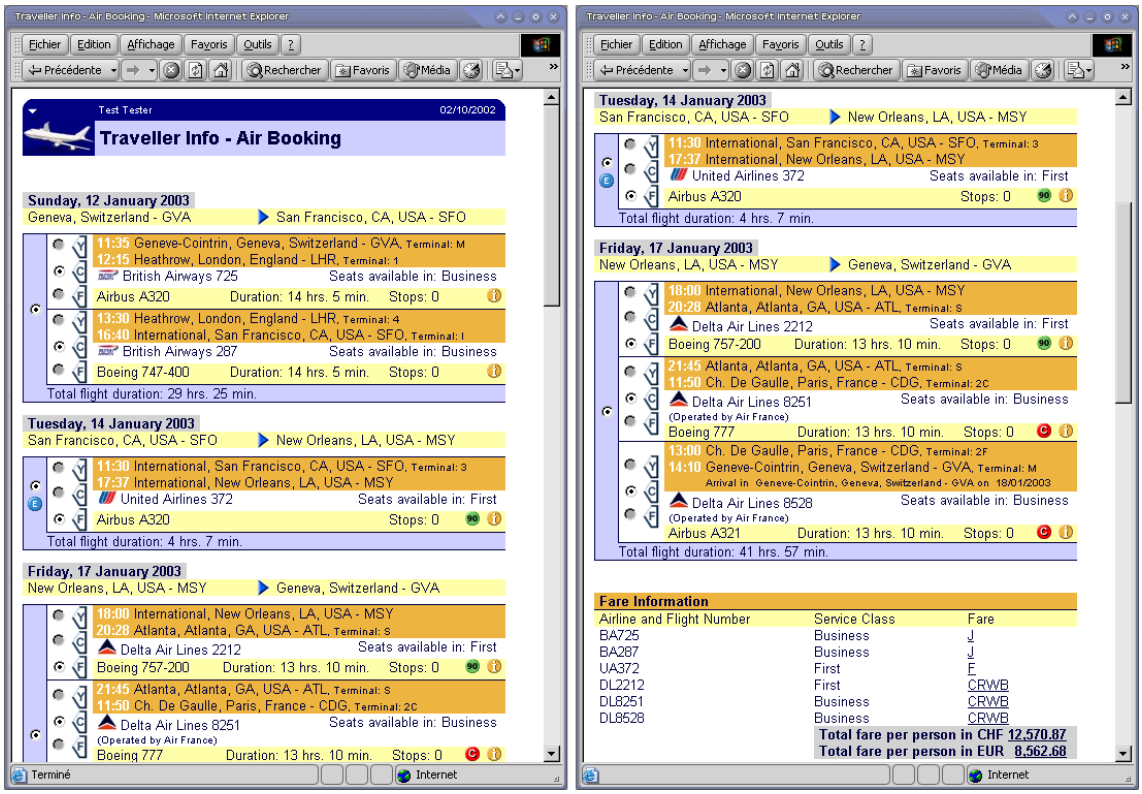


Figure A.17: The selected itinerary has been priced and details are shown with *cytric v7*.

Appendix B

Solving classical Constraint Satisfaction Problems

Constraint Satisfaction Problems (CSP) have been a subject of research in Artificial Intelligence for many years. Solving algorithms for classical CSPs deal with finding one solution or all the solutions to the problem. In the following, a brief description of the main solving algorithms for classical CSPs is given. For more detailed reviews about the main solving strategies, the reader is referred to [246, 140, 134, 81, 51, 10]. In the PhD dissertations of Christian Frei [75] and Javier Larrosa [143], recent complete reviews on solving algorithms for classical CSPs can also be found. This appendix is mainly based on the aforementioned reviews in Frei's and Larrosa's thesis [75, 143]. The International Conference on Principles and Practice of Constraint Programming and the journal *Constraints* give further specialized research papers on solving techniques for classical CSPs.

B.1 Systematic search algorithms

Systematic search methods completely explore the search space of the CSP. The basic systematic search algorithm that systematically explores the whole search space is called *Generate and Test* algorithm. Although it is a very trivial and inefficient algorithm, it makes the foundation of more advanced and efficient algorithms.

B.1.1 Generate and Test

The most trivial method is the *Generate and Test*(GT) algorithm. It systematically generates a complete assignment and check if it satisfies all the constraints. If the assignment does not satisfy all the constraints, it generates another assignment. The algorithm ends when it finds one solution. If the goal is to find all the solutions, all the complete assignments are generated. The algorithm is sound (solutions found are really solutions to the CSP) and complete (it can find all the solutions).

Systematic search algorithms for CSPs can be naturally described as search algorithms over their search tree (Definition 3.8). Basically, there are two different ways of systemat-

ically exploring a tree without visiting a node twice: depth-first and breath-first search. For solving CSPs depth-first strategy is more adequate because its memory requirement is always bounded by one complete assignment.

B.1.2 Chronological Backtracking

Chronological Backtracking (BT) [101, 22] is the simplest backtracking algorithm for solving a CSP. It explores the search tree using a first-depth search schema. Each node represents a partial assignment. The algorithm solves the CSP by incrementally extending a partial assignment to a complete assignment, starting from the empty assignment to a solution of the problem. At the node k, variable k is instantiated with its next value. The algorithm is in a *dead-end* when all the nodes below the current node produce some inconsistency with the current assignment. When a *dead-end* is found, the algorithm backtracks to the previous node and the next value is tried. The algorithm ends when a solution has been encountered or when all the search tree has been explored in the case one wants to find all the solutions. The major advantage of BT is that it eliminates parts of the search tree when a *dead-end* is found. Consequently, BT is more efficient than GT and it is also sound and complete. BT algorithm has mainly the three following drawbacks:

1. **Trashing**: the same constraint inconsistencies are rediscovered repeatedly [158].
2. **Redundant work**: constraint inconsistencies are not remembered, thus redundant work is done.
3. **Late detection**: the detection of a *dead-end* situation is done when it occurs, but it can be done before really occurs.

The first two drawbacks can be solved by *look-back propagation* algorithms (Section B.3) and the third drawback can be solved by *look-ahead propagation* algorithms (Section B.4).

Consistency techniques (Section B.2) can be used stand-alone, as preprocessing algorithms, to simplify a CSP before really solving it. Consistency techniques can be also used in combination with look-ahead propagation algorithms in order to improve search efficiency.

Variable and value ordering heuristics (Section B.5) also improve the efficiency of search algorithms.

B.2 Local consistency techniques

The most trivial consistency technique is referred to as *node-consistency* (NC). It simply removes values from variables which are not consistent with unary constraints on the respective variable.

In general, local consistency techniques (also called consistency enforcement, or constraint propagation) make partial consistent assignments extensible to other future variables [76, 158, 175].

The most widely used local consistency technique is called *arc-consistency* (AC). It removes values of variables which are not consistent with any of the values of other variables. *Path-consistency* (PC) ensures that a consistent assignment of two variables can be extended by a third variable consistently. Freuder in [76] defines *consistency*, in a more general sense, as: *i-consistency* ensures that any consistent assignment involving $i-1$ variables is extensible to include any additional variable resulting in a new consistent assignment. A problem is said *strongly k-consistent* [76] if it is j-consistent for all $j \leq k$. Following previous definition, NC is equivalent to strong 1-consistency, AC is equivalent to strong 2-consistency and PC is equivalent to strong 3-consistency. Note that a problem with n variables which is strongly n-consistent does not need any costly search to be solved, *i.e.*, it can be solved in a backtrack-free manner. A CSP with n variables which is strongly n-consistent is called *globally consistent*. Other variants of consistency techniques exist, for relevant literature on this variants see [77, 53, 54].

Consistency techniques can be applied as a preprocessing technique before the search and/or during the search process. The benefits of applying consistency techniques are the following [143]:

- **Problem simplification**: The transformed problem has additional explicit information. Typically, a search algorithm takes advantage of it, and improves its efficiency. However, in some cases achieving local consistency may degrade the performance of the search process [187, 203].

- **Unsolvability detection**: during their execution, local consistency algorithms may detect a problem is unsolvable.

The complexity of enforcing k-consistency is exponential in i [37]. Due to this high complexity, a trade-off between the effort spent in preprocessing local consistency algorithms and the efficient gain of solving algorithms must be considered. Theoretical and practical research on this trade-off have been presented in [52, 109, 189, 203]. In practice, only arc-consistency algorithms are used, either as preprocessing algorithm or in combination with a search method.

Different AC algorithms have been presented with an increasing level of sophistication: AC1, AC2, AC3 [158, 159]; AC4 [174]; AC5 [181]; AC6 [13] and AC7 [14].

B.3 Look-back strategies

Look-back strategies improve the BT algorithm by enhancing the backtracking phase of the algorithm. They basically use consistency checks among already instantiated variables (past variables). Actually, BT can be considered as the simplest look-back strategy. The following algorithms avoid the drawbacks of trashing and redundant work of BT.

B.3.1 Backjumping

Backjumping (BJ) [87, 88] avoids the trashing effect of BT. The only difference with respect to BT is on the backtracking point. When BT is in a *dead-end* situation it backtracks

chronologically (to the previous instantiated variable), while BJ backtracks directly to the conflicting variable. In a *dead-end* node, BJ computes for each value v_{ij} of the current variable V_i its culprit variable V_{cj}. A culprit variable V_{cj} for a value v_{ij} of the current variable V_i is computed as the highest variable, *i.e.*, the past variable which was allocated first, conflicting with the value v_{ij}. BJ then backjumps to the most recent instantiated variable among all culprit variables.

Conflict-Directed Backjumping and *Graph-Based Backjumping* are algorithms based on BJ with a more sophisticated backjumping behavior than simple BJ. The overhead costs of these algorithms are greater than BJ, since it is necessary to manage additional data structures. However, in some problems they can be more efficient than BJ.

B.3.2 Graph-based Backjumping

Graph-Based Backjumping (GBJ) [50], as CBJ, is an extension of the BJ algorithm, and it uses the knowledge about the CSP graph. This algorithm backjumps to the highest variable connected to the current one, *i.e.*, it jumps to the highest variable which is connected by a non trivial constraint[1]. This algorithm is useful only in the case that the associated constraint graph is sparse, if the graph is almost complete then the algorithm behaves as BT. The overhead of maintaining specific data structures is however small.

B.3.3 Conflict-directed Backjumping

Conflict-Directed Backjumping (CBJ) [189] is an extension of BJ and GBJ, and it uses the information about the conflicts between the current instantiation and future variables. Every variable has its own conflict set that contains the past variables which failed consistency checks with its current instantiation. And thus the CBJ backjumps to the highest variable in this conflict set. CBJ has the ability to perform *multiple backjumps*, that is, after the initial backjump from a *dead-end* it can continue backjumping across conflicts, which may potentially result in significant savings. This comes at the price of maintaining additional data structures, with higher overhead compared to BJ and GBJ.

B.3.4 Backmarking

Backmarking (BM) [89, 109] is able to reduce the number of consistency checks (redundant work) that BT algorithm inevitably performs.

The marking scheme is based on the following two observations:

1. If at the most recent node where a given instantiation was checked, the instantiation failed against some past instantiation that has not yet changed, then it will fail against it again. Therefore, all consistency checks involving this instantiation may be avoided.

[1] A *trivial constraint* is a constraint that does not restrict the combinations of values between the involved variables.

2. If, at the most recent node where a given instantiation was checked, the instantiation succeeded against all past instantiations that have not yet changed, then it will succeed against them again. Therefore we need to check the instantiation only against the more recent past instantiations which have changed.

BM was originally designed only for static variable ordering, however Prosser [188] and Bacchus and van Run [7] present variations of BM for dynamic variable ordering.

There are two algorithms that include the possibility of backjumping in the BM algorithm. They are called Backmarking and Backjumping (BM-BJ) and Backmarking and Conflict-Directed Backjumping (BM-CBJ). These algorithms are very similar to the BM algorithm, but they incorporate the backmarking information to decide which is the best variable to backtrack.

B.4 Look-ahead strategies

While look-back strategies use consistency checks among already instantiated variables (past variables), look-ahead strategies use consistency checks among uninstantiated variables (future variables).

Look-ahead strategies performs some level of local consistency at each instantiation of the search tree. Therefore, look-ahead strategies maintain an invariance during the search: for every future variable exists at least one value which is consistent with the past and current instantiations.

B.4.1 Forward Checking

Forward Checking (FC) [163, 109] differs from all algorithms described before because it performs the consistency checks forward. All back-look strategies do the consistency checks backward, *i.e.*, they do the consistency checks between the current variable and the past variables. In FC, at each level in the search tree, the domains of the future variables are filtered in such a way that all values inconsistent with the current instantiation are removed. In other words, at each variable instantiation a very limited form of arc-consistency is enforced. When a domain of a future variable becomes empty, the algorithm is in a *dead-end* situation, therefore the subtree below the empty variable can be skipped. In such situation, next value for the current variable is tried. When all the values for a variable failed, *i.e.*, some domain of a future variable becomes empty, the algorithm backtrack to the previous instantiation. When the instantiation that produced a value removal is undone, the removed values due to that instantiation must be restored accordingly. This produces an overhead with respect to simple BT. FC can be very efficient because of its ability to discover inconsistencies early, and therefore, reduce the size of the backtrack tree. However, FC can perform more consistency checks than backward algorithms, due to the local consistency enforcement.

Forward Checking hybrid algorithms combine backjumping with FC. There are mainly three FC hybrid algorithms, namely: Forward Checking and Backmarking (FC-BM), Forward Checking and Conflict-Directed Backjumping (FC-CBJ) and Forward Checking and

Graph-Based Backjumping (FC-GBJ). To determine the backtrack point these algorithms use the information about the variables that caused the current inconsistency. They differ in the way of computing the backtrack point.

B.4.2 Maintaining Arc Consistency

Maintaining Arc Consistency (MAC) [203] algorithm performs arc-consistency at each instantiation during the search. Experimental evaluations [203, 15] showed that MAC can outperform FC in hard problems, even the fact that MAC needs more processing for enforcing arc-consistency. Therefore, MAC algorithm is, in general, only more effective than FC for hard problems. MAC was combined with CBJ by Prosser in [190].

B.5 Variable and value ordering heuristics

In all the described algorithms, which are based on a first-depth search schema, the order in which variables and values are selected is not specified. Dechter and Meiri showed in [52] that variable and value orderings have an enormous impact on the efficient of solving algorithms. Ordering heuristics can be classified in:

- **Static** orderings: establish a selection order before the search process. Thus, the search tree is fixed beforehand, and it is maintained through the whole search process.

- **Dynamic** orderings: make selections during the search process, taking into consideration the current search state. Therefore, the search tree is built as long as the search process advances. During the search, at each level of the search three, there can be different selected variables, and for the same variable different value orderings can be established.

Dynamic and static ordering heuristics can be also combined together. Usually, static ordering heuristics are used to tie breaks of dynamic ordering heuristics.

B.5.1 Variable ordering

B.5.1.1 Static Variable Ordering

Static Variable Ordering (SVO) heuristics establish an order in which variables will be instantiated during the search. The following SVO heuristics have been suggested:

- **Maximum degree** heuristic orders the variables taking into account their constraint graph degree, *i.e.*, the number of the constraints involved in. The variables which are more constrained, *i.e.*, with a higher constraint graph degree, are selected first. The idea behind this heuristic is to first attempt to assign the variables which are likely to be the most difficult. As a consequence, inconsistencies are more likely to be discovered at high nodes of the search tree, and thus saving fruitless computations.

- **Minimal bandwidth** heuristic [265] gives a variable ordering under which the constraint graph bandwidth[2] is the lowest. The idea behind this heuristic is to order directly constrained variables closely in order to minimize the computational efforts when backtracking. However, finding such a variable ordering is computationally expensive.

- **Maximum cardinality** heuristic [52] selects an arbitrary variable, and in successive steps it selects variables which are connected to a maximal set of previously selected variables. When all the variables have been selected, a variable ordering can be established. The idea is similar to the minimal bandwidth heuristic, *i.e.*, directly constrained variables are closely ordered. However, it is less computationally expensive than the minimal bandwidth ordering heuristic.

- **Minimal width** heuristic [77] is similar to the maximum cardinality heuristic, but it establish the variable ordering from the last to the first. It starts selecting the last variable, and it successively selects variables which are less connected to already selected variables.

In the experiments reported in [52], no conclusion about the superiority of some of the heuristics could be done. Heuristic's benefits strongly depends on the topology of the problem.

B.5.1.2 Dynamic Variable Ordering

Dynamic Variable Ordering (DVO) heuristics are believed to be more effective than SVO heuristics. DVO heuristics are usually applied to look-ahead algorithms, since the information on future variables change along the search, thus such information can be used by the heuristics.

DVO heuristics are all based on the same principle proposed by Haralick and Elliot [109]: *To succeed, try first where you are most likely to fail.* The idea behind this principle is to try to discover fruitless search paths (and prune them) as early as possible. The principle can also be stated as[3]: *Deal with hard cases first: they can only get more difficult if you put them off.*

Minimum domain heuristic (MD) [109] orders the variables by their decreasing number of remaining values in their domains. In other words, variables with smaller domains are tried first.

Often, the minimum domain heuristic is combined with some information about the constraint graph topology in order to discriminate variables with same domain size. For example, *graph degree* [82], or *domain cardinality divided by degree* [15].

[2]The *bandwidth of a graph* is the maximum bandwidth among its nodes. The *bandwidth of a node* under an ordering is the maximum distance between the node and any other adjacent node.

[3]Extracted from On-line guide to Constraint Programming by Roman Barták: `http://kti.ms.mff.cuni.cz/~bartak/constraints/`

B.5.2 Value ordering

Once the decision to instantiate a variable is made, it may have several values available. Again, the order in which these values are considered can have substantial impact to find the first solution. However, if the goal is to find all solutions, or there are no solutions, then the value ordering is indifferent because all values will be tried once.

Value ordering heuristics follow the *succeed-first* principle proposed by Haralick and Elliot in [109]. This is the opposite principle applied for dynamic variable ordering. The goal of this principle is to find a solution as quick as possible, thus it seems adequate to try the most promising values first.

Dechter and Pearl [53] developed this idea proposing *advised backtracking* which estimates the goodness of a value based on a tree-relaxation of the problem. In the context of look-ahead strategies, another approach has been studied in [129, 93, 82]. In this approach, values are ordered by the pruning effect that they have on future domains.

B.6 Stochastic and heuristic algorithms

All the algorithms described above are complete, *i.e.*, they explore the whole solution space. These algorithms may be too costly in some applications, where finding a *good enough* solution may be enough. Stochastic and heuristic algorithms, also called greedy local search algorithms, use a *repair* or *hill-climbing* schema to move towards satisfactory solutions. The main problem of these algorithms is that they can be stuck at some local minimum. To escape from local minima, heuristics are proposed. Clearly, these algorithms loose the property of being complete provided by systematic search methods. Stochastic and heuristic algorithms can be seen as anytime algorithms, because they keep track of the best solutions found so far during the execution.

B.6.1 Basic local search schema: Hill-climbing

Hill-climbing is a basic form of local search. It starts from a randomly generated assignment of all variables, and iteratively, changes a value of some variable in such a way that the resulting complete assignment satisfies more constraints. Hill-climbing algorithm requires:

- an *objective function* that gives a numerical value for any given solution.
- a *neighborhood function* that maps every candidate solution s (called state) to a set of other candidate solutions (called neighbors of s).

In classical CSPs, the objective function for a solution s is a minimization function of the number of violated constraints by solution s. Hill-climbing starts from a candidate solution (initial state) which can be randomly or heuristically generated. The algorithm continues with a neighbor solution which is better according to the objective function. The search terminates when no better neighbor solution can be found. If the resulting solution

is not enough satisfactory, the whole search process can be restarted from different initial states.

Hill-climbing search is in a local minima when all the neighbors are worst than the current state which is not necessarily the best possible solution. To overcome this drawback different methods have been proposed (*Random Walk*, *Tabu Search* and *Simulated Annealing*). Another drawback of hill-climbing algorithms is that they have to evaluate all neighbors of the current state to decide which neighbor is the best to move to. *Min-conflicts* heuristic and *Fast Local Search* overcome this second drawback of hill-climbing basic schema. Methods build upon hill-climbing are often referred to as meta-heuristics.

B.6.2 Min-conflicts

Min-conflicts heuristic for hill-climbing was firstly proposed in [171]. This heuristic chooses randomly any conflicting variable, *i.e.*, variables that is involved in any unsatisfied constraint, and then picks a value which minimizes the number of violated constraints. Tie breaks are solved randomly. If no such value exists, it picks randomly one value that does not increase the number of violated constraints. This heuristics avoids to evaluate all the neighbors of the current state, however, it does not prevent from getting stuck in local minima. Next heuristics intend to escape from local minima.

B.6.3 Random walk

Random walk strategy [218] is combined with the min-conflicts heuristic to escape from local minima. For a given conflicting variable, the random walk strategy picks randomly a value with a probability p, and then apply the min-conflicts heuristic with probability $1-p$. The value of the parameter p has an important influence on the performance of the method (few experimental work reported that the feasible range for p is $0.02 \leq p \leq 0.1$).

B.6.4 Connectionist approach

GENET [48] is a connectionist approach to solve CSPs. Problems are represented as neural networks, where nodes in the network represent values which are linked following connections defined by constraints. Links of the network are associated to weights, which are subject to changes though reinforcement learning. The goal is that the network converges to states that represent solutions of the CSP. In order to escape from local minima, the weights of the constraints which are violated in a local minimum are augmented. This produces an increased objective function and allows the network to escape to other states. The algorithm starts with a random configuration of the network and recomputes the state of nodes (that can be activated or deactivated) repeatedly taking into account the neighbor states and the weights of connections to these nodes.

B.6.5 Guided Local Search and Fast Local Search

Guided Local Search (GDS) is a meta-heuristic algorithm for optimization problems in general. Considering the optimization function as the minimization function on the number

of unsatisfied constraints, GDS can be applied to classical CSPs. The idea is to escape from local minima by weighting the objective function with penalties.

Fast Local Search (FSL) [253, 247], which is combined with Guided Local Search (GDS), intends to overcome the cost of evaluating all the neighbors of the current state. This evaluation can be very costly because the number of the neighbors and the computational cost of the objective function. The intention of FLS is to ignore neighbors that are unlikely to lead to fruitful hill-climbs in order to improve the efficiency of the search process. Each neighbor move is associated with an activation bit (initially set to on). Only those neighbors whose bits are switched on will be considered. If a neighbor has been evaluated without leading to a better solution, its bit is switched off. The search is guided through sates where their number of active (switched on) sub-neighbors is maximized. To escape from local minima FLS uses the same idea than GDS. GDS and FSL approaches derive from the GENET model.

B.6.6 Tabu search

Tabu Search (TS) [97, 98] was firstly proposed by [96] and, independently, by [106]. TS builds a tabu list of states that have already been visited. This prevents the algorithm to visit nodes which have already been visited. *Aspiration criteria* are a set of conditions which if satisfied overrule the tabu restrictions. The most aspiration criterion is to accept a state which is in the tabu list if the state generates a solution better than any previously seen.

B.6.7 Simulated Annealing

Simulated Annealing (SA) [130], based on the ideas presented in [169], is based on a randomized schema which reduces the risk of getting stuck in local minima by allowing moves to inferior solutions. A move from a solution s to solution s' is only accepted if:

- s' is better than s (as minimum-conflict method), or
- s' is worse than s but $e^{\frac{-(g(s)-g(s'))}{T}} > R$

where $g(s)$ is the objective function, T is a control parameter called *temperature* and $R \in [0,1]$ is a uniform random number. The temperature parameter T is initially set to a high value, allowing many non-improving moves to be accepted and it is gradually reduced to a value where nearly all non-improving moves are rejected.

B.6.8 Genetic Algorithms

Genetic Algorithms (GA) [99] borrow their ideas from natural evolution theory [115]. The GA starts with an initial population of solutions (can be randomly generated) and evolves the population to a set of (sub) optimal solutions. The objective function (*e.g.*, the number of satisfied constraints in a maximization case) gives a fitness measurement for candidates to their chances to be selected. Genetic operators such as crossover and mutation are

applied to build a next generation of candidate solutions. Genetic algorithms have been applied to constraint satisfaction problems (see [58, 202, 33]).

Appendix C

Extended Constraint Satisfaction Problems

Classical constraints define the notion of allowed/disallowed combinations of values. Such concept is very convenient for problems that imply knowledge which is crisp or boolean. Unfortunately, this is not the case in many real-life problems. Very often, when facing real-life scenarios is necessary to extend the paradigm of classical discrete constraint satisfaction techniques. For example, extending the classical CSP paradigm is unavoidable for modeling user's preferences since people normally give different degrees of importance to their preferences. In Chapter 3, the main extensions to the classical CSP framework were briefly described. In this chapter, a review on the existing literature about the different extensions to the classical constraint satisfaction framework is given.

C.1 Constraint hierarchies

The theory of constraints hierarchies handles constraints with different priorities and was described in detail by Alan Borning *et al.* in [23]. In this theory, the authors define a hierarchy of constraints as a set of constraints classified by different strengths. In some problems like configuration with preferences, it is needed to define constraints that are *required*, *i.e.*, hard, and others that are *preferential*, *i.e.*, soft. The constraint hierarchy theory makes also possible to define several degrees of constraints in a hierarchy.

A solution to a constraint hierarchy is an assignment of values to all variables such that all the *required constraints* are satisfied and the *preferential constraints* are also satisfied as much as possible, according to their relative strengths. A solution has an evaluation with respect to a *comparator* that has to be *irreflexive* and *transitive*.

In [23] the authors describe a collection of algorithms for solving constraint hierarchies. Two of the most representative algorithms are outlined as follows:

- *Blue* and *DeltaBlue* algorithms for acyclic hierarchies based on local propagation, and
- *Orange* algorithms that solve sets of linear programming problems: each level of

constraints is reformulated as a separate problem.

Last algorithmic advances on solving constraint hierarchies are presented in the PhD dissertation of Rudovà [200].

C.2 Partial constraint satisfaction problems

Partial constraint satisfaction problems, introduced by Freuder and Wallace in [79], are of interest in one of the following situations:

1. when the CSP is overconstrained, *i.e.*, there is no solution satisfying all the constraints of the problem, or
2. when solving a CSP is too complex to solve it in a reasonable computing time, *e.g.*, in interactive and real-time applications.

In both situations, it is of interest to search for "good enough" solutions. This is referred to as partial constraint satisfaction. Maximal constraint satisfaction problems (MAX-CSP) can be seen as an instance of partial constraint satisfaction, where the goal is to maximize the number of satisfied constraints. MAX-CSPs have been deeply studied in Larrosa's thesis [143]. MAX-CSPs are specific instances of weighted CSPs (see Section C.3.4).

C.3 Soft constraint satisfaction problems

Soft constraint satisfaction problems arise from the requirement of adapting the classical concept of constraint to real problems. Thus, different soft constraint satisfaction problems can be defined through the following notions [200]:

- **Constraint**: defines some extension of the classical constraint concept.
- **Problem**: is defined according to the notion of constraint.
- **Satisfaction degree**: defines at what extent assignments satisfy the constraints in the problem. Satisfaction degree enables to compare different assignments.
- **Solution**: is defined through the optimality of assignments according to their satisfaction degree.

C.3.1 Fuzzy CSPs (FCSP)

Fuzzy CSPs [198, 55, 201] are able to model constraints which are not crisp. The tuples of values defining a constraint have an associated preference level. In this way, it is possible to model tuples that are more *preferred* to be satisfied than others. Usually, the associated preference level is between 0 and 1. Tuples which are associated with a preference level of 1 are indicating the best combination of values, while tuples with a level of 0 are indicating that the combination of values is not allowed (hard constraint).

Any assignment of a set of variables, has a degree of satisfaction that indicates how well the implied constraints are satisfied jointly. There are several ways of defining such degree of satisfaction based on: *conjunctive combination*, *productive combination*, *cumulative combination* and *average combination*. Among these different ways of computing degrees of satisfaction, the most commonly used is the *conjunctive combination*. Thus, the degree of satisfaction of a given assignment of variables is just the minimum preference level of all the implied tuples for all constraints. Therefore, a solution to a FCSP is an assignment to all variables such that its degree of satisfaction is maximized. The idea behind this min-max method relies on the concept of maximizing the satisfaction of the worst tuple in the solution.

In a more formal way, a FCSP is just a classical CSP with fuzzy constraints. A fuzzy constraint c is defined by a function level $l_c(a) \rightarrow [0,1]$, for all tuples a in the constraint. Thus, a solution to a FCSP is an assignment to all variables such that $min(l_c(a))$ is maximized for all the constraints in the problem and all tuples implied in the solution.

C.3.2 Probabilistic CSPs

Probabilistic CSPs [62] model problems where some part of the knowledge is uncertain. Constraints in probabilistic CSPs have associated a probability $p(c)$ which is independent from the other constraints. Such a probability does not refer to the constraint itself but to the probability that the constraint take place in the real problem. Sometimes, it could be useful to express a constraint that has some probability to occur in the real-life problem.

A solution to a probabilistic CSP has also a probability that it is really a solution in the real problem. For any assignment of a set of variables a, the probability that a is an assignment in the real problem can be defined by $P(a) = \prod(1 - p(c)) \mid c$ is violated by a. Moreover, a solution to a probabilistic CSP is an assignment a to all variables such that $P(a)$ is maximized.

C.3.3 Possibilistic CSPs

In [215], Thomas Schiex defines the concept of possibilistic CSPs. Similarly to the previous extensions to classical CSPs, a possibilistic CSP is a classical CSP with possibilistic constraints. Possibilistic constraints specify a possibilistic distribution among their tuples.

For any assignment of a set of variables a there is an associated valuation which corresponds to the maximum valuation among the violated constraints. Furthermore, a solution to a possibilistic CSP is a complete assignment a such that its valuation is minimized. Actually, possibilistic CSPs (min-max) can be seen as the dual problem of fuzzy CSPs (max-min).

One of the major problems with possibilistic CSPs is that when a tuple of a constraint is violated with valuation of α, all the tuples with valuations β, $\alpha > \beta$ are completely ignored, i.e. such tuple-constraints will no have any impact on the valuation of the solution. This effect is called the "drowning effect" and it is due to the idempotency of the max operator. In order to avoid such limitation, the notion of lexicographic CSP has been proposed in [63].

C.3.4 Weighted CSPs (WCSP)

Tuples in constraints of a Weighted CSP have an associated cost. Thus, weighted CSPs are very useful for modeling optimization problems where the goal is to minimize the total cost. The valuation (cost function) for any assignment a of a set of variables is the arithmetic sum of the constraints implied in the assignment. Moreover, a solution to a weighted CSP is an assignment a to all the variables such that its cost is minimized.

In a more formal way, a constraint c of a WCSP is defined by a cost function $C_c(a) \rightarrow [0,\infty]$, for all tuples a in the constraint. Thus, a solution to a WCSP is an assignment to all variables such that $\sum C_c(a)$ is minimized for all the constraints in the problem and all tuples implied in the solution.

C.3.5 Lexicographic CSPs

In order to avoid the "drowning effect" of the possibilistic CSPs and fuzzy CSPs, the notion of lexicographic CSPs was introduced by Fargier and Lang in [63]. Valuations in lexicographic CSPs do not only takes into account the levels of priorities but also the number of violated constraints at each level of priority.

C.4 Semiring-based constraint satisfaction problems (SCSP)

This section is a synopsis of a part of [17], taking explanations from [200]. Semiring-based CSP [19, 21, 17, 20] is meta-framework for soft CSPs where all their instances can be cast. Therefore, the properties of semiring-based CSPs can be inferred to other more specific soft CSP models.

Semiring-based CSPs are based on c-semirings which are a type of semirings:

Definition C.1 (semiring) A **semiring** is a tuple $(E,+,\times,\mathbf{0},\mathbf{1})$ such that:

- E is a set and $\mathbf{0},\mathbf{1} \in E$;
- $+$, called the additive operation, is closed, commutative and associative. Its unit element is $\mathbf{0}$;
- $\times$, called the multiplicative operation which is closed and associative. Its unit element is $\mathbf{1}$ and its absorbing element is $\mathbf{0}$; and
- $\times$ distributes over $+$, i.e. $a \times (b+c) = (a \times b) + (a \times c)$.

Definition C.2 (c-semiring) A **c-semiring** is a semiring such that $+$ is idempotent ($a \in E$ implies $a+a=a$), $\times$ is commutative, and $\mathbf{1}$ is the absorbing element of $+$.

Let us consider the relation $\leq_S$ over E such that $a \leq_S b$ if and only if $a+b=b$. This relation defines a partial ordering over the set E, which will enable to compare different elements of the semiring. Intuitively, $a \leq_S b$ means that a is *better* than b. Such a relation will be used to choose the *best* solution of a SCSP.

Semiring-based CSP framework is based on a c-semiring structure, where the set of semiring specifies the values to be associated with each tuple of values, and the two semiring operations $(+,\times)$ model constraint projection and combination respectively.

The definition of the semiring-based CSP framework is as follows:

Definition C.3 (Semiring-based Constraint System) A **semiring-based constraint system** is a tuple $CS = \langle S, D, V \rangle$, where S is a c-semiring, D is a finite set (the domain of the variables), and V is an ordered set of variables.

Definition C.4 (Semiring-based Constraint) Given a semiring $S = (E, +, \times, \mathbf{0}, \mathbf{1})$ and a constraint system $CS = \langle S, D, V \rangle$, a **semiring-based constraint** is a pair $\langle def, con \rangle$ where $con \subseteq V$ and it is called the *type* of the constraint, and $def : D^k \to E$ (where k is the size of con, that is, the number of variables in it), and it is called the *value* of the constraint.

Definition C.5 (Semiring-based CSP (SCSP)) A **Semiring-based CSP** P is a pair $P = \langle C, con \rangle$ over a constraint system $CS = \langle S, D, V \rangle$, where $con \subseteq V$ and C is a set of constraints.

Note that con is the set of variables of interest for the set of constraints C, which however may concern also variables not in con. In many approaches, all the variables are of interest, *i.e.*, all the variables are implied in the solutions of the problem.

In order to define the notion of solution for the SCSP framework, the concepts of constraint combination and constraint projection must be defined:

Definition C.6 (Semiring-based Constraint Combination) Given two constraints $c_1 = \langle def_1, con_1 \rangle$ and $c_2 = \langle def_2, con_2 \rangle$, their **combination**, noted $c_1 \otimes c_2$, is the constraint $\langle def, con \rangle$, where:

- $con = con_1 \cup con_2$, and
- $def(t) = def_1(t \downarrow^{con}_{con_1}) \times def_2(t \downarrow^{con}_{con_2})$

In an informal manner: combining two constraints means to build a new constraint involving all the variables of the original ones, and associating to each value tuple over such variables a semiring element, which is obtained by multiplying the elements associated by the original constraints on the appropriate sub-tuples. Due to commutativity and associativity of the multiplicative operation, combination of multiple constraints can be noted as:

$$c_1 \otimes \cdots \otimes c_n = \bigotimes_{i=1}^{n} c_i = \bigotimes C \text{ for } C = \{c_1, \ldots, c_n\}$$

Definition C.7 (Semiring-based Constraint Projection) Given a constraint $c = \langle def, con \rangle$ and a subset W of V, the **projection** of c over I, noted $c \Downarrow_I$, is the constraint $\langle def', con' \rangle$, where:

- $con' = con \cap I$, and

- $def'(t') = \sum_{t \mid t\downarrow^{con}_{con'} = t'} def(t)$

Informally, projecting a constraint over a subset of variables means to eliminate some variables of the constraint. This is done by associating to each tuple over the remaining variables a semiring element which is the sum of the elements associated by the original constraint to all the extensions of this tuple over the eliminated variables.

Constraint combination is performed via the multiplicative operation, while constraint projection is performed via the additive operation.

Finally, the notion of semiring-based solution and its degree of satisfaction can be defined as follows:

Definition C.8 (Semiring-based Satisfaction Degree and Solution) The **solution** of a SCSP $P = \langle C, con \rangle$ is the constraint:

$$Sol(P) = \left(\bigotimes C\right) \Downarrow_{con}$$

Thus, a solution to a SCSP is a constraint resulting from the combination of all the constraints of the problem, and then projected over the variables in *con* (usually all the variables of the problem). Such solution constraint provides, for each tuple of values of D (the search space of the problem), an associated element of the semiring. This constraint distributes by its def function an evaluation on each tuple, thus it can be seen as the **satisfaction degree**.

Informally, maximal, with respect to $\leq_S$, semiring value of tuples in solution $Sol(P)$ corresponds to the best level of consistency. The tuples with such a best level of consistency are the *set of optimal assignments*, *i.e.*, the optimal solutions to a SCSP.

C.5 Valued constraint satisfaction problems (VCSP)

Valued Constraint Satisfaction Problem (VCSP) [215, 17, 18, 214] is a CSP meta-framework defined over valuations over constraints. VCSPs are specified by a general structure based on totally ordered monoid over valuations.
The definition of a valuation structure is given as follows:

Definition C.9 (Valuation Structure) A **valuation structure** is defined by $(E, \circledast, \succ, \top, \bot)$, where:

- E is a set whose elements are called *valuations*,
- $\succ$ is a total ordering over E,
- $\top$ and $\bot$ are maximum and minimum elements of E given by $\succ$,
- $\circledast$ is a commutative, associative binary operation on E that satisfies:
  - identity: $\forall a \in E : a \circledast \bot = a$, and
  - monotonicity: $\forall a, a', b \in E : (a \succeq a') \Rightarrow ((a \circledast b) \succeq (a' \circledast b))$

From these axioms, it may be also inferred that the element $\top$ is an absorbing element, *i.e.*, $\forall a \in E : (a \circledast \top) = \top$.

The ordered set E allows to express different levels of violations. Commutativity and associativity guarantee that the valuation of an assignment only depends on the set of valuation of the violated constraints, and not on the way they are combined. Monotonicity guarantees that the valuation of an assignment that satisfies a set C of constraints will always be as good as the valuation of any assignment which satisfies a subset of C'.

Definition C.10 (Valued Constraint) A **valued constraint** is a tuple $(c, \varphi(c))$, where c is a classical constraint and φ is a function from a set of constraints C to a set of valuations E. φ is called valuation function.

According to the valued constraint definition, a valued CSP can be defined as follows:

Definition C.11 (Valued CSP (VCSP)) A **valued CSP** (VCSP) is defined by a classical CSP (V, D, C), a valuation structure $S = (E, \circledast, \succ, \top, \bot)$, and by a valuation of all constraints φ. Each assignment will be valuated by combining the valuations of all violated constraints using the operator $\circledast$.

Definition C.12 (Valued Satisfaction Degree) Given a valued CSP $P = (V, D, C, S, \varphi)$ and an assignment t of a subset of variables $X \subseteq V$, the **valuation of assignment** (or its satisfaction degree) t is defined by: $\circledast_{c \in C} \varphi(c)$.

Definition C.13 (Valued Consistency Degree and Solution) A **solution** of valued CSP $P = (V, D, C, S, \varphi)$ is an assignment to all variables having a minimal valuation with respect to the ordering $\succ$. This minimal valuation will be called **consistency degree** of the problem P.

Note that since solutions are assignments with minimal valuation computed by combining violated constraints with $\circledast$, the element $\top$ corresponds to unacceptable violation and is used to express hard constraints, while $\bot$ element corresponds to the concept of complete satisfaction.

C.6 Valuations for constraints and tuples

In all the CSP models described in this chapter, one can consider to associate valuations to constraints (as in the case of VCSPs) or to associate valuations to the tuples in the constraints (as in the case of SCSPs). Actually, both approaches are equivalent (for finite variable domains).

Valuations associated to each constraint can be noted: (c, w), where c is a classical constraint and w its valuation. On the other hand, valuations associated to tuples for each constraint can be noted by a valuation function: $c : D_1 \times \cdots \times D_k \to W$.

The transformation from valued-constraints (c, w) to valued-tuples is straight forward: the valuation w is assigned to all tuples which satisfy the constraint c, while the remaining tuples are associated with the complement valuation of w.

Transformation from valued-tuples $c : D_1 \times \cdots \times D_k \rightarrow W$ to valued-constraints can be done by replacing the constraint c by a set of constraints $c_1, \ldots, c_k$, where k is the cardinality of the product domain of the variables implied in c:

$$k = card\{w \in W \mid \exists t \in D_1 \times \cdots \times D_k : c(t) = w\}$$

C.7 Main properties of SCSP and VCSP

The most relevant properties of the two CSP meta-frameworks (SCSP and VCSP) can be summarized in the following way:

1. **Orderings**: SCSPs deal with partial orders whereas VCSP represent totally ordered sets of valuations.

2. **Equivalence**: SCSPs and VCSP are equivalent from the expressiveness point of view, under the total order assumption on the valuation set. Semiring and valuation structure are related together through replacement of additive semiring operation by *min* operation which is applied in valued CSP to compare valuations on different assignments. SCSP with partial ordered set as valuation set is suitable to express multi-objective cominatorial problems, and therefore SCSP is a more general model than VCSP.

3. **Valued constraints vs valued tuples**: while SCSPs associate valuations to tuples, VCSPs associate valuations to constraints. However, both approaches are theoretically equivalent (see Section C.6), but with different ease of use depending on the semantics of problem.

4. **SCSP** main properties [167]:

   - Local and global consistency: best level of consistency of a subproblem is higher than the best level of consistency of the whole problem.
   - Equivalence: after appying local consistency to a problem, it is euivalent to the original problem if $\times$ is idempotent.
   - Termination: a local consistenc procedure terminates in a finite number of steps, if the values of E used in the problem constraints form a finite set I, and operations $+$ and $\times$ are closed in I.
   - Order independence: two different applications of local consistency procedures to the same problem produce equal results if $\times$ is idempotent.

5. **VCSP** main properties [167]:

   - If $\otimes$ is idempotent, *i.e.*, $a \otimes a = a$, $\forall a \in E$, local consistency mehtods make sense. The only idempotent operator is *max*, used in classical CSP (logical and $\wedge$ can be considered as a *max* operator) and FCSP models, for which local consistency enforcing algorithms can be defined. For the other models, local consistency can be enforced by mehtods not involving constraint propagation.

See [18] for more details and justifications of the aforementioned properties.

C.8 Casting CSP frameworks into CSP meta-frameworks

The interesting point of meta-frameworks (SCSP and VCSP) of constraint satisfaction problems is that they accept many different instances of CSPs. Therefore, properties and theories of such meta-frameworks can be directly applied to specific instances of such meta-frameworks.

Table C.1 and Table C.2 show how to map classical CSPs and soft CSPs into SCSPs and VCSPs respectively.

CSP framework	E	$+$	$\times$	$\mathbf{0}$	$\mathbf{1}$
classical CSP	$\{0,1\}$	$\vee$	$\wedge$	0	1
MAX-CSP	$\{0,1\}$	*min*	$+$	$+\infty$	0
weighted CSP	$\mathbb{R}^+$	*min*	$+$	$+\infty$	0
probabilistic CSP	$[0,1]$	*max*	$\times$	0	1
possibilistic CSP	$[0,1]$	*min*	*max*	0	1
fuzzy CSP	$[0,1]$	*max*	*min*	0	1
lexicographic CSP	$\mathbb{N}^{\langle 0,1\rangle} \cup \{\bot\}$	max_{lex}	$\wedge$	0	1

Table C.1: Specifications $(E, +, \times, \mathbf{0}, \mathbf{1})$ of SCSP for different soft CSPs frameworks.

CSP framework	E	$\succ$	$\circledast$	$\top$	$\bot$
classical CSP	$\{0,1\}$	$<$	$\wedge$	0	1
MAX-CSP	$\{0,1\}$	$>$	$+$	$+\infty$	0
weighted CSP	$\mathbb{N} \cup \{+\infty\}$	$>$	$+$	$+\infty$	0
probabilistic CSP	$[0,1]$	$<$	$\times$	0	1
possibilistic CSP	$[0,1]$	$>$	*max*	1	0
fuzzy CSP	$[0,1]$	$>$	*max*	1	0

Table C.2: Specifications $(E, \succ, \circledast, \top, \bot)$ of VCSP for different soft CSPs frameworks.

Appendix D

Java Constraint Library

Man is a tool-using animal. Without tools he is nothing, with tools he is all.

Thomas Carlyle, 1795-1881.

This chapter describes the Java Constraint Library[1] (JCL) which has been implemented at the Artificial Intelligence Lab.[2] of the Swiss Federal Institute of Technology. JCL allows us to package constraint satisfaction problems and their solvers in compact autonomous agents suitable for transmission on the Internet. On the other hand, the Choice Constraint Language[3] (CCL) is an agent content language designed to explicitly support agent communication about CSPs.

JCL combined with CCL are suitable for building agent-based information systems on the web, designed to solve problems and to not just provide information to the user. Such systems would assist the user who is often faced to the problem of being overloaded with the available information on the Web, specially in complex domains (as the ones described in Section 1.3.1).

D.0.1 Historical background of the JCL

The evolution of the JCL can be described as follows:

- **Initial idea (1996).** The initial idea was developed during a semester project of Erik Bruchez under the supervision of Rainer Weigel and Boi Faltings. Erik Bruchez, during his semester project, implemented a graphical user interface for solving discrete and binary constraint satisfaction problems by a simple backtracking algorithm.

- **First version (1997-2000).** Within the diploma project of Marc Torrens, again under the supervision of Rainer Weigel and Boi Faltings, the first version of the JCL came out. This version contained about 15 different constraint solving algorithms

[1] `http://liawww.epfl.ch/JCL`.
[2] `http://liawww.epfl.ch`.
[3] `http://liawww.epfl.ch/CCL/`.

and a more complete GUI. On top of that, an application for scheduling meetings was implemented in order to show the applicability of the JCL to solve problems through the Web. The JCL was, for the first time, freely available to download from its homepage.

- **Second version (2000-present).** JCL was extended with methods for dealing with exclusion constraints (unary constraints) and with binary relations in order to support agent communication about CSP by using CCL (see Section D.2). Several people contributed directly to this achievement: Rajat Bhattacharjee and Santiago Macho. JCL 2.1 is currently distributed as an open source software under the terms of the LGPL[4]. Santiago Macho is currently the lead of the JCL project.

- **Next version** would include the use of soft constraint satisfaction problems. Clearly, this improvement will allow the development of more real applications using the JCL. Santiago Macho and Nicoleta Neagu are currently working on that.

The JCL has been downloaded by thousands of users world-wide during these last 5 years. It has mainly been used for educational purposes, to show the power and insights of constraint satisfaction reasoning.

D.1 Java Constraint Library

Basically, the JCL [242, 243, 244] provides services for:

- creating and managing discrete and binary CSPs, and

- applying preprocessing and search algorithms to CSPs.

The JCL can be used either in a stand-alone Java application or in an applet[5]. The purpose of the JCL is to provide a framework for easily building agents that solve CSPs on the Web. The JCL allows the development of portable applications and applets using constraint satisfaction techniques. The JCL package is divided into two parts, namely: a constraint library and a shell with a graphical user interface built on the top of the library (as shown in Figure D.1).

D.1.1 The constraint library

The constraint library provides methods for building CSPs from scratch and solving them. The user only has to model the problem she/he wants to solve and then use the methods from the JCL to build the corresponding CSP. Once the CSP is built, any of the solving algorithms implemented in the JCL can be used for solving the problem. These algorithms can search *one*, *all* or *n* solutions depending on one parameter that the user can set. Once the solving algorithm finishes, the JCL reports an enumeration of the solutions found and

[4]GNU Lesser General Public License (LGPL) Version 2.1.

[5]An applet is an application designed to be transmitted over the Internet and executed by a Java-compatible Web browser.

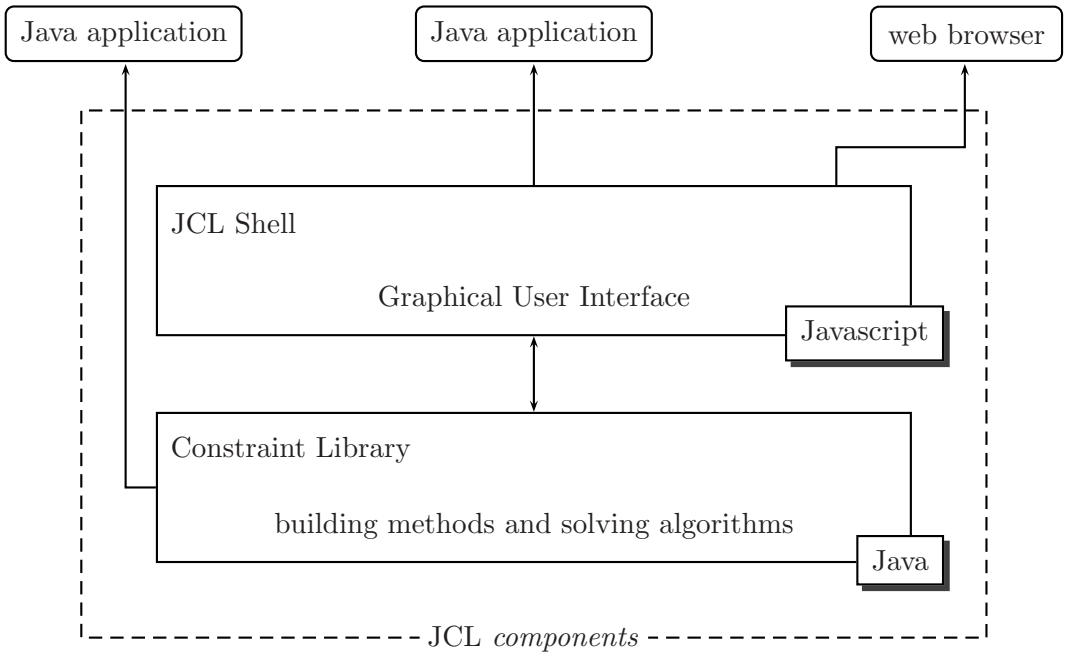


Figure D.1: The components of the JCL environment. The JCL is composed by two main elements: the constraint library and the graphical user interface. Java applications can directly use the constraint library or the GUI provided by the JCL Shell. Users can also directly build, save, and solve CSPs through a web browser using the JCL Shell.

some statistics on the search. These statistics include the number of backtracks, the number of consistency checks, the number of instantiations, and so forth. The library provides both search and preprocessing algorithms. Search algorithms allow us to find solutions of a CSP, while preprocessing algorithms are used to simplify a CSP by eliminating values and compound labels that cannot be part of any solutions.

The solving methods implemented in the JCL are:

- Chronological Backtracking (BT)
- Backjumping (BJ)
  - Constraint-directed Backjumping
  - Graph-based Backjumping
- Backmarking (BM)
  - Backmarking with Backjumping
  - Backmarking with constraint-directed Backjumping
  - Backmarking with graph-based Backjumping
- Forward Checking (FC)
  - Forward Checking with Backmarking

- Forward Checking with constraint-directed Backjumping
- Forward Checking with graph-based Backjumping
- Forward Checking using arc consistency (MAC)

For more details on the combination of the above algorithms, the reader is referred to [189]. Some of these algorithms are also implemented in the Peter van Beek's CSPLib[6] and were just adapted in the JCL. Two preprocessing algorithms are also implemented in the JCL: Arc-consistency (AC) and Path-consistency (PC) (see [158] for details of these consistency algorithms).

Methods for building classical random problems such as the n-queens problem, the confused n-queens problem, and the graph coloring problem have been provided within JCL. A random generator of constraint satisfaction problems is included as well in the JCL package.

D.1.2 The graphical user interface

The Graphical User Interface (GUI) has been implemented for facilitating the integration of the constraint library into Java applications and applets where the results of the algorithms must be displayed graphically. A java application and an applet for editing and solving CSPs in a user-friendly interface have also been implemented.

D.2 Choice Constraint Language (CCL)

This section is based on the text in the CCL homepage provided by Steve Willmott.

The CCL is an agent content language designed to explicitly support agent communication about CSPs [261, 262]. It has been developed at the Artificial Intelligence Lab.õf the Swiss Federal Institute of Technology since 1998. The contributors of the CCL are Steve Willmott, Monique Calisti, Boi Faltings, Santiago Macho, Omar Belakdhar and Marc Torrens. CCL is designed to be used directly with the FIPA[7] standard Agent Communication Language (ACL), and it has been incorporated in the FIPA 1999 standard as content language `FIPA-CCL`.

CCL uses CSP formalism as an underlying model for choice problems. In this way, CCL addresses a large class of choice problems with explicit support for notions such as choices, options for choices and relationships among choices.

The CSP model which underlies FIPA CCL has three restrictions imposed which have been made to make the model minimal and more suitable for a communication language:

1. **Binary Constraints**. All constraints expressed must be binary. This restriction is often made in the CSP field, since most powerful solving techniques only apply to CSPs with arity 2 constraints. Furthermore, for discrete CSPs, any CSP represented

[6] CSPLib: routines for solving binary constraint satisfaction problems by Peter van Beek. It can be freely downloaded at `ftp://ftp.cs.ualberta.ca:/pub/ai/csp`.

[7] FIPA, Federation for Intelligent Physical Agents, `http://www.fipa.org`.

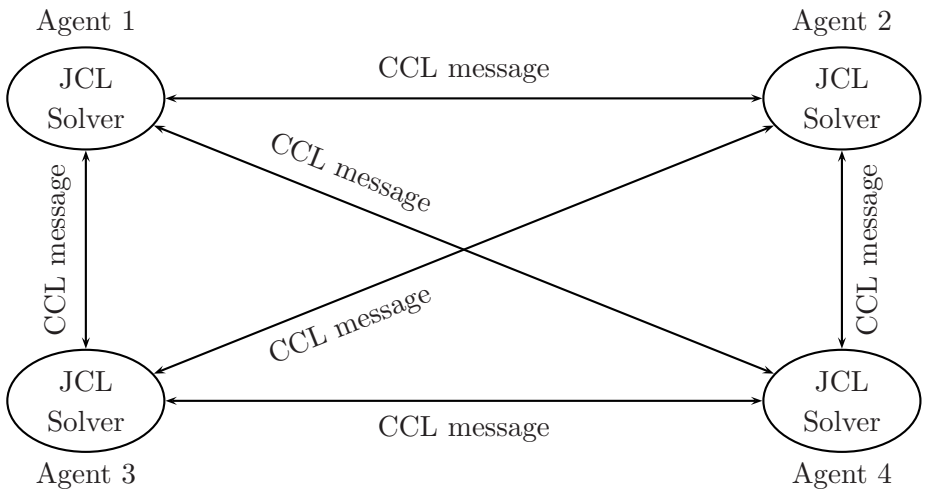


Figure D.2: JCL and CCL provide the needed tools to build multi-agent applications for solving choice problems. In this example, only 4 agents are considered.

in a form using n-ary constraints can be transformed into an equivalent CSP using only binary (2-ary) constraints. The language therefore looses none of its expressive power with this restriction.

2. **Discrete Variable Domains**. In practice, CSPs requiring continuous values can often be formulated by discretizing the continuous domain (so that discrete CSP solving techniques can be applied, see [207]).

3. **Intensional Relations**. There are two main ways of representing constraints for CSPs, as *extensional* relations (consisting of a list of the valid combinations of values for a pair or tuple of variables) and as *intensional* relations (consisting of relations such as equals, greater-than etc. which do not rely on an explicit list). FIPA CCL excludes the use of extensional relations because this makes CSPs expressed in FIPA CCL much easier to compose (merge) when fusing information from several sources.

There are also several implicit constraints which arise out of the fact that that CSPs represented in FIPA CCL must be contained in a single message. Concretely, the number of variables and the number of constraints of a CSP expressed in FIPA CCL must be finite.

D.3 JCL with CCL

Currently, many applications of agents involve to reason and communicate about multiple interrelated choices. In that sense, JCL provides the reasoning engines based on constraint satisfaction problems, and CCL provides a communication language about constraints, as shown in Figure D.2. Therefore, the combination of JCL and CCL can support agent-based applications which deal with problems with constrained choices, for instance electronic catalogs.

Bibliography

[1] F. Ben Abdelaziz, J. Chaouachi, and S. Krichen. A hybrid heuristic for multiobjective knapsack problems. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-heuristics; advances and trends in local search paradigms for optimization*, pages 205–212. Kluwer Academic Publishers, Dordrecht, 1999.

[2] Mohamed-Salah Affane and Hachemi Bennaceur. A Weighted Arc Consistency Technique for Max-CSP. In *Proceedings of the 13th European Conference on Artificial Intelligence, ECAI-98*, Brighton, United Kingdom, 1998.

[3] Christopher Ahlberg and Ben Shneiderman. Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays. In *Human Factors in Computer Systems: Proceedings of the CHI'94 Conference*, pages 313–317, New York, 1994. ACM.

[4] M. Joao Alves and Joao Clímaco. An interactive method for 0-1 multiobjective problems using simulated annealing and tabu search. *Journal of Heuristics*, 2000.

[5] T. M. Apostol. *Calculus, One-Variable Calculus, with an introduction to Linear Algebra*, volume 1. John Wiley & Sons, New York, USA, 2nd. edition, June 1967.

[6] Homa Atabakhsh. A Survey of Constraint Based Scheduling Systems Using an Artificial Intelligence Approach. *Artificial Intelligence in Engineering*, 6(2):58–73, 1991.

[7] Fahiem Bacchus and Paul van Run. Dynamic Variable Ordering in CSPs. In *First International Conference on Principles and Practice of Constraint Programming (CP-95)*, pages 258–275, 1995.

[8] Egon Balas and Paolo Toth. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, chapter Branch and Bound methods. John Wiley & Sons, New York, 1985.

[9] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The Quickhull Algorithm for Convex Hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, December 1996.

[10] Roman Barták. Constraint Programming: In Pursuit of the Holy Grail. In *Proceedings of the Week of Doctoral Students (WDS99), Part IV*, pages 555–564, Prage, June 1999. MatFyzPress.

[11] Ashok D. Belegundu, D. V. Murthy, Raviprakash R. Salagame, and Edward W. Constant. Multi-objective Optimization of Laminated Ceramic Composites Using Genetic algorithms. In *Fifth AIAA/USAF/NASA Symposium on Multidisciplinary Analysis and Optimization*, pages 1015–1022, Panama City, Florida, USA, 1994.

[12] Bertran Cabon and Simon de Givry and Gérard Verfaillie. Anytime lower bounds for constraint violation minimization problems. In *Fourth International Conference on Principles and Practice of Constraint Programming (CP98)*, volume 1520 of *Lectures Notes in Computer Science*, Pisa, Italy, October 1998. Springer Verlag.

[13] Christian Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65:179–190, 1994.

[14] Christian Bessière, Eugene C. Freuder, and Jean-Charles Régin. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107:125–148, 1999.

[15] Christian Bessière and Jean-Charles Régin. MAC and combined heuristics; two reasons to forsake FC (and CBJ?) on hard problems. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, CP-96*, pages 61–75, 1996.

[16] Stefano Bistarelli, Boi Faltings, and Nicoleta Neagu. A Definition of Interchangeability for Soft CSPs. In *ERCIM Working Group on Constraints on Constraint Solving and Constraint Logic Programming*, University College Cork, Ireland, 19th-21st of June 2002.

[17] Stefano Bistarelli, Hélène Fargier, Ugo Montanari, Francesca Rossi, Thomas Schiex, and Gérard Verfaillie. Semiring-based CSPs and Valued CSPs: Basic Properties and Comparison. *CONSTRAINTS: an international journal*, 4(3), September 1999.

[18] Stefano Bistarelli, Hélène Fargier, Ugo Montanari, Francesca Rossi, Thomas Schiex, and Gérard Verfaillie. Semiring-based csps and valued csps: Basic properties and comparison. In Michael Jampel, Eugene C. Freuder, and Michael Maher, editors, *Over-constrained Systems*, volume 1106 of *Lecture Notes on Computer Science*, pages 111–150. Springer-Verlag, August 1996.

[19] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Constraint solving over semirings. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 624–630. Morgan Kaufmann, 1995.

[20] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint logic programming. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 352–357. Morgan Kaufmann, 1997.

[21] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint solving and optimization. *Journal of ACM*, 44(2):201–236, March 1997.

[22] James R. Bitner and Edward M. Reingold. Backtrack programming techniques. *Communications of the ACM*, 18(11):651–656, November 1975.

[23] Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. Constraint Hierarchies. *Lisp and Symbolic Computation: An International Journal*, 5(3):223–270, September 1992.

[24] L. M. Boychuk and V. O. Ovchinnikov. Principal methods of solution of multicriterial optimization problems. *Soviet Automatic Control*, 6(1-4), 1973.

[25] Keith Bradley, Rachael Rafter, and Barry Smyth. Case-Based User Profiling for Content Personalization. In *Proceedings of the International Conference on Adaptive Hypermedia and Adaptive Web-based Systems (AH)*, Trento, Italy, August, 28-30 2000.

[26] I. Buchanan. Decision Theory, Planning and Constraint Satisfaction. In AAAI Press, editor, *Proceedings of the 1994 Spring AAAI Symposium on Decision-Theoretic Planning*, page 218, Menlo Park, California, US, 1994.

[27] Bertran Cabon, Gérard Verfaillie, David Martinez, and P. Bourret. Using mean field methods for boosting backtrack search in constraint satisfaction problems. In *Proceedings of the 12th European Conference on Artificial Intelligence, ECAI-96*, pages 165–169, Budapest, Hungary, 1996.

[28] D. G. Carmichael. Computation of Pareto Optima in Structural Design. *International Journal for Numerical Methods in Engineering*, 15:925–952, 1980.

[29] V. Cerny. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization, Theory and Applications*, 45:41–55, 1985.

[30] A. Charnes, W. W. Cooper, R. J. Niehaus, and A. Stedry. Static and dynamic assignment models with multiple objectives and some rearks on organization design. *Management Science*, 15(8):365–375, 1969.

[31] David N. Chin and Asanga Porage. Acquiring User Preferences for Product Customization. In Springer Verlag, editor, *Eighth International Conference on User Modeling (UM)*, Sonthofen, Germany, July 2001.

[32] Berthe Y. Choueiry. *Abstraction Methods for Resource Allocation.* PhD thesis, Swiss Federal Institute of Technology in Lausanne (EPFL), Lausanne, Switzerland, 1994.

[33] P. Chu and J. E. Beasley. Genetic algorithms for the generalized assignment problem. *Computers and Operations Research*, 24:17–23, 1997.

[34] Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry. *Discrete Computational Ceometry*, 4:387–421, 1989.

[35] Carlos Artemio Coello. *An empirical study of evolutionary techniques for multiobjective optimization in engineering design*. PhD thesis, Tulane University, Department of Computer Science, New Orleans, LA, USA, 1996.

[36] Carlos Artemio Coello and Alan D. Christiansen. Two new GA-based methods for multiobjective optimization. *Civil Engineering Systems*, 15(3):207–243, 1998.

[37] Martin Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41:89–95, 1989.

[38] Dragan Cvetković, Ian Parmee, and Eric Webb. Multi-objective Optimisation and Preliminary Airframe Design. In Ian Parmee, editor, *The Integration of Evolutionary and Adaptive Computing Technologies with Product/System Design and Realisation*, pages 255–267, Plymouth, United Kingdom, April 1998. Plymouth Engineering Design Centre, Springer-Verlag.

[39] Piotr Czyzak and Andrzej Jaszkiewicz. A multi-objective metaheuristic approach to the location of a petrol station by the capital budgeting model. *Control and Cybernetics*, 25:177–187, 1996.

[40] Piotr Czyzak and Andrzej Jaszkiewicz. Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7:34–47, 1998.

[41] Geir Dahl, Kurt Jörnsten, and Atne Lokketangen. A tabu search approach to the channel minimization problem. In *Proceedings of International Conference on Optimization Techniques and Applications (ICOTA'95)*, Chengdu, China, July 1995.

[42] Indraneel Das. Optimizing Large Systems via Optimal Mulicriteria Component Assembly. In *Proceedings of 7th AIAA/USAF/NASA/ISSMO Symposium on Multi-Disciplinary Analysis & Optimization*, pages 661–669, St. Loius, MO, USA, August 1998.

[43] Indraneel Das. An Improved Technique for Choosing Parameters for Pareto Surface Generation Using Normal-Boundary Intersection. In *Proceedings of the Third World Congress of Structural and Multidisciplinary Optimization*, Buffalo, NY, USA, March 1999.

[44] Indraneel Das. On characterizing the 'knee' of the Pareto curve based on Normal-Boundary Intersection. *Structural Optimization*, 18(2-3):107–115, October 1999.

[45] Indraneel Das and Sharmistha Das. Multicriteria Optimization of Error Probabilities in Digital Communications. In *Proceedings of the Third World Congress of Structural and Multidisciplinary Optimization*, Buffalo, NY, USA, March 1999.

[46] Indraneel Das and John Dennis. A Closer Look at Drawbacks of Minimizing Weighted Sums of Objectives for Pareto Set Generation in Multicriteria Optimization Problems. *Structural Optimization*, 14(1):63–69, 1997.

[47] Indraneel Das and John Dennis. Normal-Boundary Intersection: A new method for generating Pareto optimal points in multicriteria optimization problems. *SIAM Journal on Optimization*, 8(3):631–657, 1998.

[48] Andrew Davenport, Edward Tsang, Chang Wang, and Kangmin Zhu. GENET: A Connectionist Architecture for Solving Constraint Satisfaction Problems by Iterative Improvement. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-94*, pages 287–309, Seattle, WA, 1994. AAAI Press.

[49] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-88*, pages 49–54, St. Paul, MN, August 1988. AAAI Press.

[50] Rina Dechter. Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition. *Artificial Intelligence*, 41(3):273–312, January 1990.

[51] Rina Dechter and Dan Frost. Backtracking algorithms for Constraint Satisfaction Problems - A tutorial survey. Technical report, Department of Information and Computer Science, University of California, Irvine, 1998.

[52] Rina Dechter and Itay Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence*, 68:211–241, 1994.

[53] Rina Dechter and Judea Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1988.

[54] Rina Dechter and Judea Pearl. Tree Clustering Schemes for Contraint-Processing. *Artificial Intelligence*, 38:353–366, 1989.

[55] Didier Dubois, Hélène Fargier, and Henri Prade. The Calculus of Fuzzy Restrictions as a Basis for Flexible Constraint Satisfaction. In IEEE, editor, *Proceedings IEEE International Conference on Fuzzy Systems*, 1993.

[56] Didier Dubois, Hélène Fargier, and Henri Prade. Possibility Theory in Constraint Satisfaction Problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6:287–309, 1996.

[57] Matthias Ehrgott and Xavier Gandibleux. An Annotated Bibliography of Multiobjective Combinatorial Optimization. Technical Report 62, Universitat Kaiserslautern, Fachbereich Mathematik, Postfach 3049, D-67663, Kaiserslautern, Germany, April 2000.

[58] Ágoston Eiben, Paul-Erik Raué, and Zsófia Ruttkay. Solving constraint satisfaction problems using genetic algorithms. In *Proceedings of the first IEEE Conference on Evolutionary Computing*, pages 543–547, 1994.

[59] V. A. Emelichev and V. A. Perepelitsa. Complexity of vector optimization problems on graphs. *Optimization*, 22:903–918, 1991.

[60] Boi Faltings and Eugene C. Freuder. Guest Editor's Introduction: Configuration. In Eugene C. Freuder and Boi Faltings, editors, *IEEE Intelligent Systems, Special issue on Configuration*, volume 13(14), pages 32–33. IEEE, July/August 1998.

[61] Boi Faltings and Santiago Macho. Open Constraint Satisfaction. In *Proceedings of the AAMAS 2002, Workshop on Distributed Constraint Satisfaction*, 2002.

[62] Hélène Fargier and Jérôme Lang. Uncertainty in Constraint Satisfaction Problems: a Probabilistic Approach. In IEEE, editor, *Proceedings of European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty*, pages 97–104. Springer-Verlag, 1993.

[63] Hélène Fargier, Jérôme Lang, and Thomas Schiex. Selecting Preferred Solutions in Fuzzy Constraint Satisfaction Problems. In *Proceedings of the First European Congres on Fuzzy and Intelligent Technologies*, 1993.

[64] Daniel Felix, Christoph Niederberger, Patrick Steiger, and Markus Stolze. Feature-oriented vs. Needs-oriented Product Access for Non-Expert Online Shoppers. In *Proceedings first IFIP Conference on e-commerce, e-business, and e-government*, pages 399–406, 2001.

[65] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, California, USA, 2000.

[66] A. R. Finkel and J. L. Bentley. Quad trees. A data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[67] Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forest, editor, *Genetic Algorithms: Proceedings of Fifth International Conference*, pages 416–423, San Mateo, CA, USA, 1993. Morgan Kaufmann, San Francisco, USA.

[68] Carlos M. Fonseca and Peter J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3:1–16, 1995.

[69] Carlos M. Fonseca and Peter J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. I. A unified formulation. *Transactions on Systems, Man & Cybernetics, Part A*, 28(1):26–37, 1998.

[70] Carlos M. Fonseca and Peter J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. II. Application example. *Transactions on Systems, Man & Cybernetics, Part A*, 28(1):38–47, 1998.

[71] Mark Fox. *Constraint Directed Search: A Case Study of Job-Shop Scheduling*. Morgan and Kaufmann, Los Altos, California, US, 1987.

[72] Mark Fox. Why is Scheduling Difficult? A CSP Perspective. In *Proceedings of the 9th European Conference on Artificial Intelligence, ECAI-90*, pages 754–758, Stockholm, Sweden, 1990.

[73] Maria Sole Franzin, Eugene C. Freuder, Francesca Rossi, and Ricahrd J. Wallace. Multi-agent meeting scheduling with preferences: efficiency, privacy loss, and solution quality. In *Working Notes of the Workshop on Preferences in AI and CP: symbolic approaches,*, Edmonton, Alberta, Canada, July 28 - August 1 2002. AAAI Press.

[74] Bjorn Freeman-Benson, John Maloney, and Alan Borning. An Incremental Constraint Solver. *Communications of the ACM*, 33(1):54–63, January 1990.

[75] Christian Frei. *Abstraction Techniques for Resource Allocation in Communication Networks*. PhD thesis, Swiss Federal Institute of Technology in Lausanne (EPFL), Lausanne, Switzerland, 2000.

[76] Eugene C. Freuder. Synthesizing constraint expressions. *Communications ACM*, 21(11):958–966, 1978.

[77] Eugene C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

[78] Eugene C. Freuder. The Role of Configuration Knowledge in the Business Process. In Eugene C. Freuder and Boi Faltings, editors, *IEEE Intelligent Systems, Special issue on Configuration*, volume 13(14), pages 29–31. IEEE, July/August 1998.

[79] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1):21–70, 1992.

[80] Samuel Fricker. Extracting Constraints for Database Access from Natural Language. Swiss Federal Institute of Technology in Lausanne (EPFL). Internship, semestral project, March 1999. Supervised by Jean-Cédric Chappelier.

[81] Dan Frost. *Algorithms and Heuristics for Constraint Satisfaction Problems*. PhD thesis, ICS, University of California, Irvine, October 1997.

[82] Daniel Frost and Rina Dechter. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence, IJCAI-95*, pages 572–578, Montreal, Canada, 1995.

[83] International Galileo. EDIFACT Select. Functional Despcription. Fare Display and Quote Definition. Technical Manual, June 2000.

[84] International Galileo. EDIFACT Select. Functional Despcription. Product Offering Definition. Technical Manual, June 2000.

[85] Xavier Gandibleux, Nazik Mezdaoui, and Arnaud Fréville. A tabu search procedure to solve multiobjective combinatorial optimization problems. In R. Caballero, F. Ruiz, and Ralph E. Steuer, editors, *Advances in Multiple Objective and Goal Programming*, volume 455 of *Lecture Notes in Economics and Mathematical Systems*, pages 291–300. Springer Verlag, Berlin, 1997.

[86] Xavier Gandibleux, Nazik Mezdaoui, and Ekunda L. Ulungu. Evaluation of multiobjective tabu search procedure on a multiobjective permutation problem. In *OR-BEL*, Namur, Belgium, January 1997.

[87] John Gaschinig. Experimental case studies of backtrack vs. Waltz-type vs. new algorithms for stisfising assignment problems. In *Proceedings Second National Conference of the Canadian Society for computational Studies of Intelligence*, pages 268–277, 1978.

[88] John Gaschnig. Performance Measurement and Analysis of Certain Search Algorithms. Technical report.

[89] John Gaschnig. A general backtrack algorithm that eliminates most redundant checks. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence, IJCAI-77*, page 457, Cambridge, MA, 1977.

[90] Marco Gavanelli. Partially ordered constraint optimization problems. In Toby Walsh, editor, *Principles and Practice of Constraint Programming, 7th International Conference - CP 2001*, volume 2239 of *Lecture Notes in Computer Science*, page 763, Paphos, Cyprus, November 26 – December 1 2001. Springer Verlag.

[91] Marco Gavanelli. An Algorithm for Multi-Criteria Optimization in CSPs. In Frank van Harmelen, editor, *ECAI 2002. Proceedings of the 15th European Conference on Artificial Intelligence*, Lyon, France, July 21-26 2002. IOS Press. To Appear.

[92] Marco Gavanelli. An implementation of Pareto optimality in CLP(FD). In Narendra Jussien and François Laburthe, editors, *CP-AI-OR - International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems*, pages 49–64, Le Croisic, France, March 25 – 27 2002. Ecole des Mines de Nantes.

[93] Pieter A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of the 10th European Conference on Artificial Intelligence, ECAI-92*, pages 31–35, Vienna, Austria, 1992.

[94] Matthiew L. Ginsberg. A New Algorithm for Generative Planning. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pages 186–197, San Francisco, California, US, 1996.

[95] Simon de Givry, Gérard Verfaillie, and Thomas Schiex. Bounding the Optimum of Constraint Optimization Problems. In *Third International Conference on Principles*

and Practice of Constraint Programming, CP-97, volume 1330 of *Lecture Notes in Computer Science*, Linz, Austria, 1997. Springer Verlag.

[96] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computer Operation Research journal*, 5:533–549, 1986.

[97] Fred Glover. Tabu search - Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[98] Fred Glover. Tabu search - Part II. *ORSA Journal on Computing*, 2:4–32, 1990.

[99] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.

[100] David E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, 1987.

[101] S. W. Golumb and L. D. Baumert. Backtrack programming. *Journal of ACM*, 12:516–524, 1965.

[102] David Gralan and Mary Shaw. An Introduction to Software Architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, volume 1. World Scientific Publishing Company, New Jersey, 1994. Also appears as CMU Software Engineering Institute Technical Report.

[103] B. Grünbaum. Measures of symmetry for convex sets. In *Proceedings of the seventh Symposium in Pure Mathematics of the American Mathematical Society, Symposium on Convexity*, pages 233–270, 1961.

[104] Robert H. Guttman, Alexandros G. Moukas, and Pattie Maes. Agent-mediated Electronic Commerce: A Survey. *Knowledge Engineering Review*, 13(2):146–160, 1998.

[105] W. Habenicht. Quad trees, A data structure for discrete vector optimization problems. *Lecture Notes in Economics and Mathematical Systems*, 209:136–145, 1982.

[106] Michael Pilegaard Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimisation*, Capri, Italy, 1986.

[107] Michael Pilegaard Hansen. Tabu Search in Multiobjective Optimisation : MOTS. In *Multiple Criteria Decision Making: Theory, and applications. Proceedings of International Conference on MCDM'97*, Cape Town, South Africa, 1997.

[108] Michael Pilegaard Hansen. *Metaheuristics for multiple objective combinatorial optimization*. PhD thesis, Institute of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 1998.

[109] R. M. Haralick and G. L. Elliott. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14:263–313, 1980.

[110] Associates Harrel. The Internet Travel Industry: What Consumers Should Expect and Need to Know and Options for a Better Marketplace. New York City, June, 6 2002. A Report Prepared for Consumer WebWatch.

[111] Alois Haselböck, Markus Stumptner, and Gerhard Friedrich. COCOS, A Tool for Constraint-Based Dynamic Configuration. In *Proceedings of the 10th IEEE Conference on AI Applications*, pages 373–380, San Antonio, Texas, US, 1994.

[112] Jonathan Herlocker, Joseph Konstan, and John Riedl. Explaining Collaborative Filtering Recommendations. In *Proceedings of the ACM 2000 Conference on Computer Supported Collaborative Work*. ACM Press, December 2000.

[113] Enric Hernández Jiménez. Presonalización: de visitantes a compradores. White Paper, Intelligent Software Components SA, `www.isoco.com`, October 2001.

[114] Alain Hertz, Brigitte Jaumard, Celso C.. Ribeiro, and W. Formosinho Filho. A multi-criteria tabu search approach to cell formation problems in group technology with multiple objectives. *RAIRO - Recherche Opérationelle/Operations Research*, 28(3):303–328, 1994.

[115] J. Holland. *Adaptation in natural and artificial systems*. University of Michigan press, Ann Arbor, MI, USA, 1975.

[116] Jeffrey Horn. Multicriteria decision making. In T. Back, editor, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[117] Jeffrey Horn, Nicolas Nafpliotis, and David E. Goldberg. A niched Pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, pages 82–87, New Jersey, NY, USA, June 1994. IEEE.

[118] Eric Horvitz. Principles of Mixed-Initiative User Interfaces. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 159–166, Pittsburgh, Pennsylvania, USA, May, 15-20 1999. ACM Press.

[119] C. L. Hwang, Sudhakar R. Paidy, and K. Yoon. Mathematical programming with multiple objectives: A tutorial. *Computing and Operational Research*, 7:5–31, 1980.

[120] T. Ibaraki, S. Muro, T. Murakami, and T. Hasegawa. Using branch and bound algorithms to obtain suboptimal solutions. *Zeitchrift für Operations Research*, 27:177–202, 1983.

[121] James P. Ignizio. *Goal Programming and Extensions*. Lexington Books, Lexington, KY, 1976.

[122] Yuji Ijiri. *Management Goals and Accounting for Control.* North Holland, Chicago, USA, 1965.

[123] A. Inselberg. The plane with parallel co-ordinates. *The Visual Computer*, 1:69–91, 1985.

[124] Anthony Jameson, Joseph Konstan, and John Riedl. AI Techniques for Personalized Recommendation. Tutorial at 18th National Conference on Artificial Intelligence, Edmonton, Alberta, Canada, July/August 2002.

[125] Andrzej Jaszkiewicz. On the copmutational effectiveness of multiple objective metaheuristics. In *Proceedings of the Fourth International Conference on the Multi-Objective Programming and Goal Programming, MOPGP-00. Theory & Applications*, Ustroń, Poland, May 29 - June 1, 2000. Springer Verlag, Berlin.

[126] Andrzej Jaszkiewicz and Roman Slowiński. The LBS-Discrete Interactive Procedure for Analyisis of Decision Problems. In J. Climaco, editor, *Multiple Criteria Decision Making: Theory, and applications. Proceedings of the Eleventh International Conference on MCDM*, pages 320–330, Coimbra, Portugal, August 1997. Springer Verlag, Berlin.

[127] P.C. Kanellakis and C. H. Papadimitriou. Local search for the asymmetric traveling salesman problem. *Operations Research*, 28:1086–1099, 1980.

[128] Kalev Kask and Rina Dechter. Stochastic local search for bayesian networks. In *Proceedings of the International Workshop on AI and Statistics*, 1999.

[129] Naiping Keng and David Yun. A planning/scheduling methodology for the constrained resource problem. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, IJCAI-89*, pages 998–1003, 1989.

[130] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[131] Joshua D. Knowles. *Local-Search and Hybrid Evolution Algorithms for Pareto Optimization.* PhD thesis, Department of Computer Science, University of Reading, Reading, UK, January 2002.

[132] J. Koenemann and N. J. Belkin. A case for interaction: A study of interactive information retrieval behaviour and effectiveness. In *Proceedings of the Human Factors in Computing Systems Conference (CHI)*, pages 205–212, Vancouver, Canada, April 1996.

[133] Murat Köksalan, M. H. Karwan, and Stanley Zionts. An Approach for Solving Discrete Alternative Multiple Criteria Problems Involving Ordinal Criteria. *Naval Research Logistics*, 35(6):625–642, 1988.

[134] Grzegorz Kondrak and Peter van Beek. A Theoretical Evaluation of Selected Backtracking Algorithms. *Artificial Intelligence*, 89:365–387, 1997.

[135] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. GroupLens: applying collaborative filtering to Usenet news. *Communication of the ACM*, 40(3):77–87, March 1997.

[136] R. E. Korf. A complete anytime algorithm for number partitioning. *Artificial Intelligence*, 105:133–155, 1998.

[137] Pekka Korhonen. A Visual Reference Direction Approach to Solving Discrete Multiple Criteria Problems. *European Journal of Operational Research*, 34(2):152–159, 1988.

[138] Pekka Korhonen, Jyrki Wallenius, and Stanley Zionts. Solving Discrete Multiple Criteria Problem Using Convex Cones. *Management Science*, 30(11):1336–1345, 1984.

[139] A. Kosba. Personalized Hypermedia and International Privacy. *Communications of the ACM*, 45(5):64–67, 2002.

[140] Vipin Kumar. Algorithms for Constraint Satisfaction Problems: A Survey. *AI Magazine*, 13(1):32–44, 1992.

[141] Denis Lalanne. *Conception créative assistée par ordinateur: un modèle d'intelligence interactive*. PhD thesis, Swiss Federal Institute of Technology in Lausanne (EPFL), Lausanne, Switzerland, 1998.

[142] Amy L. Lansky and Andrew G. Philpot. COLLAGE: A Diversified Constraint-Based Planning Architecture. In AAAI Press, editor, *Proceedings of the 1993 AAAI Spring Symposium on Foundations of Automatic Planning: The Classical Approach and Beyond*, Standford, California, US, 1993.

[143] Javier Larrosa. *Algorithms and Heuristics for Total and Partial Constraint Satisfaction.* PhD thesis, Institut d'Investigació en Intelligencia Arftificial, Bellaterra, Catalonia, Spain, 1998.

[144] Javier Larrosa and Rina Dechter. On the dual representation of non-binary semiring-based CSPs. In *Modelling and Solving Soft Constraints Workshop, CP-2000*, Singapore, September 2000.

[145] Javier Larrosa and Pedro Meseguer. Exploiting the use of DAC in MAX-CSP. In *Second International Conference on Principles and Practice of Constraint Programming (CP96)*, volume 1118 of *Lecture Notes in Computer Science*, Cambridge, USA, 1996. Springer Verlag.

[146] Javier Larrosa and Pedro Meseguer. Partition-Based Lower Bound for Max-CSP. In *Fifth International Conference on Principles and Practice of Constraint Programming (CP-99)*, volume 1118 of *Lecture Notes in Computer Science*, Alexandria, Virginia, USA, October 1999. Springer Verlag.

[147] Javier Larrosa, Pedro Meseguer, and Thomas Schiex. Maintaining Reversible DAC for MAX-CSP. *Artificial Intelligence. An International Journal*, 107(1):149–163, 1999.

[148] Javier Larrosa, Pedro Meseguer, Thomas Schiex, and Gérard Verfaillie. Reversible DAC and Other Improvements for Solving MaxCSP. In *The Fifteenth National Conference on Artificial Intelligence*, Madison, Wisconsin, USA, 1998.

[149] E. L. Lawler and D. E. Wood. Branch-and-bound methods: a survey. *Operational Research*, 14:699–719, 1966.

[150] Sang M. Lee. *Goal Programming for decision analysis.* Auberach Publications, Philadelphia, PA, USA, 1972.

[151] Sang M. Lee and V. Jaaskelainen. Goal programming: Management's math model. *Industrial Engineering*, pages 30–35, February 1971.

[152] Y. H. Li. *Directed Annealing Search in Constraint Satisfaction and Optimization.* PhD thesis, Imperial College of Science, Department of Computing, London, United Kingdom, 1997.

[153] Greg Linden, Steve Hanks, and Neal Lesh. Interactive Assessment of User Preference Models: The Automated Travel Assistant. In *Proceedings of the sixth International Conference on User Modeling*, Chia Laguna, Sardinia, Italy, June, 2-5 1997.

[154] Vahid Lofti, Theodor J. Stewart, and Stanley Zionts. An aspiration-level interactive model for multiple criteria decision making. *Computing and Operational Research*, 16:677–681, 1992.

[155] Zoubir Lounis and Mircea Z. Cohn. Multiobjective optimization of prestressed concrete structures. *Journal of Structural Engineering*, 119:794–808, March 1993.

[156] M. Dell'Amico and F. Maffioli and S. Martello, editor. *Annotated Bibliographies in Combinatorial Optimization.* J. Willey & Sons, Chichester, 1997.

[157] Santiago Macho, Marc Torrens, and Boi Faltings. A Multi-Agent System for Planning Meetings. In *Proceedings of the fourth International Conference on Autonomous Agents, Workshop on Agent-based Recommender Systems (WARS-2000*, Barcelona, Catalonia, Spain, June 4 2000.

[158] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[159] Alan K. Mackworth. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–7, 1985.

[160] Alan K. Mackworth. *Encyclopedia of AI*, chapter Constraint Satisfaction, pages 205–211. Springer Verlag, 1988.

[161] Samir W. Mahfoud. *Niching Methods for Genetic Algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, USA, 1995.

[162] Behnam Malakooti. Theories and an Exact Interactive Paired-Comparison Approach for Discrete Multiple Criteria Problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(2):365–378, 1989.

[163] James J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, pages 229–250, 1979.

[164] Ammon Meisels, Ehud Gudes, and Gadi Solotorevsky. Combining Rules and Constraints for Employee Timetabling. *International Journal Intelligent Systems*, 12:419–439, 1997.

[165] Francisco Menezes, Pedro Barahona, and Philippe Codognet. An Incremental Constraint Solver Applied to a Timetabling Problem. In *Proceedings of the 13th Conference on Expert Systems*, pages –, Avignon, France, 1993.

[166] Pedro Meseguer. Lower bounds for non-binary Max-CSP. In *Modelling and Solving Constraint Problems Workshop, ECAI-2000*, Berlin, Germany, August 2000.

[167] Pedro Meseguer, Noureddine Bouhmala, Taoufik Bouzoubaa, Morten Igrens, and Martí Sánchez. Current Approaches for Solving Over-Constrained Problems. *CONSTRAINTS: an international journal*, In Press.

[168] Pedro Meseguer, Javier Larrosa, and Martí Sànchez. Lower Bounds for Non-binary Constraint Optimization Problems. In Toby Walsh, editor, *Principles and Practice of Constraint Programming, 7th International Conference - CP 2001*, volume 2239 of *Lecture Notes in Computer Science*, pages 317–331, Paphos, Cyprus, 2001. Springer Verlag.

[169] Nicolas C. Metropolis, Arianna Rosenbluth, Marshal Rosenbluth, Augusta Teller, and Edward Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[170] Steve Minton, Mark D. Johnson, Andy Philips, and Philip Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-90*, pages 17–24, Boston, MA, 1990. AAAI Press.

[171] Steve Minton, Mark D. Johnson, Andy Philips, and Philip Laird. Minmizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[172] Sanjay Mittal and Brian Falkenhainer. Dynamic Constraint Satisfaction Problems. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-90*, pages 25–32, Boston, MA, 1990. AAAI Press.

[173] Sanjay Mittal and Felix Freyman. Towards a Generic Model of Configuration Tasks. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, IJCAI-89*, pages 1395–1401, Detroit, MI, 1989.

[174] Roger Mohr and Thomas C. Henderson. Arc and Path Consistency Revised. *Artificial Intelligence*, 28:225–233, 1986.

[175] Ugo Montanari. Networks of constraints: Fundamental properties and application to picture processing. *Information Science*, 7, 1974.

[176] Joan Morris and Pattie Maes. Negotiating Beyond the Bid Price. In *CHI, Workshop on Designing Interactive Systems for 1-to-1 E-commerce*, The Hague, The Netherlands, April 2000.

[177] Paul Morris. The breakout method for escaping from local minima. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-93*, pages 40–45, Washington, DC, 1993. AAAI Press.

[178] Tadahiko Murata and Hisao Ishibuchi. MOGA: Multi-objective genetic algorithms. In *Proceedings of the Second IEEE International Conference on Evolutionary Computing*, pages 289–294, Perth, Australia, 1995.

[179] Mark O'Connor, Dan Cosley, Joseph A. Konstan, and John Riedl. PolyLens: A Recommender System for Groups of Users. In *Proceedings of the Seventh European Conference on Computer Supported Cooperative Work (ECSCW)*, pages 199–218, Bonn, Germany, September, 16-20 2001.

[180] Andrzej Osyczka. *Multicriterion Optimization in Engineering with FORTRAN programs*. Ellis Horwood Limited, 1984.

[181] Y. Deville P. Van Hentenryck and C.-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

[182] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1982.

[183] Claude Le Pape. Implementation of Resource Constraints in ILOC SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems. *Intelligent Systems Engineering*, 3(2):55–66, 1994.

[184] Vilfredo Pareto. *Cours d'économie politique professé à l'université de Lausanne*, volume 1. F. Rouge, Lausanne, 1896-1897.

[185] Judea Pearl. *Heuristics*. Addison-Wesley, 1984.

[186] Don Peppers and Martha Rogers. Has the one-to-one future arrived? *Inside 1to1*, October 2002.

[187] Patrick Prosser. Domain filtering can degrade intelligent backtracking search. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence, IJCAI-93*, pages 262–267, Chambéry, France, 1993.

[188] Patrick Prosser. Forward Checking with Backmarking. Technical report, Department of Comnputer Science, University of Strathclyde, September 1993. Research Report, AISL-48-93.

[189] Patrick Prosser. Hybrid Algorithmns for the Constraint Satisfaction Problem. *Computational Intelligence*, 9(3):268–299, 1993.

[190] Patrick Prosser. MAC-CBJ: maintaining arc-consistency with conglict-directed backjumping. Technical report, Department of Computer Science, University of Strathclyde, 1995. Research Report 95-177.

[191] Patrick Prosser and James T. Buchanan. Intelligent Scheduling: past, present, and future. *Intelligent Systems Engineering*, 3(2):67–78, 1994.

[192] Pear Pu and Boi Faltings. Enriching buyers' experiences: the SmartClient approach. In *Proceedings of the CHI 2000 conference on Human factors in computing systems*, The Hague, The Netherlands, 2000.

[193] Pearl Pu and Boi Faltings. Effective Interaction Principles for User-Involved Constraint Problem Solving. In *Second International Workshop on User-Interaction in Constraint Satisfaction. Eight International Conference on Principles and Practice of Constraint Programming (CP)*, Ithaca, New York, USA, September 2002.

[194] Pearl Pu and Denis Lalanne. Interactive Problem Solving via Algorithm Visualization. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 145–154, Salt Lake City, Utah, October 2000.

[195] Pearl Pu and Denis Lalanne. Designing Visual Thinking tools for Mixed Initiative Systems. In *Proceedings of the International conference on Intelligent User Interfaces (IUI)*, Miami, Florida, USA, January 2002.

[196] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1977.

[197] Paul Resnick and Hal R. Varian. Recommender Systems. *Communications of the ACM*, 40(3):56–58, 1997.

[198] Azriel Rosenfeld, Robert A. Hummel, and Steven W. Zucker. Scene Labelling by Relaxation Operations. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6:420–433, 1976.

[199] Richard E. Rosenthal. Principles of Multiobjective Optimization. *Decision Sciences*, 16:133–152, 1985.

[200] Hana Rudová. *Constraint Satisfaction with Preferences*. PhD thesis, Masaryk University Brno, Faculty of Informatics, January 2001.

[201] Zsófia Ruttkay. Fuzzy Constraint Satisfaction. In IEEE, editor, *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, pages 1263–1268, Orlando, 1994.

[202] Zsófia Ruttkay, Ágoston Eiben, and Paul-Erik Raué. Improving the performance of GAs on a GA-hard CSP. In *Proceedings, CP-95 Workshop on Studying and Solving Really Hard Problems*, pages 157–171, 1995.

[203] Daniel Sabin and Eugene C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of the 11th European Conference on Artificial Intelligence, ECAI-94*, pages 191–196, Amesterdam, The Netherlands, 1994.

[204] Daniel Sabin and Eugene C. Freuder. Configuration as Composite Constraint Satisfaction. In *Proceedings of the Artificial Intelligence and Manufacturing Research Planning Workshop*, pages 153–161, 1996.

[205] Daniel Sabin and Rainer Weigel. Product Configuration Frameworks - A Survey. In Eugene C. Freuder and Boi Faltings, editors, *IEEE Intelligent Systems, Special issue on Configuration*, volume 13(14), pages 42–49. IEEE, July/August 1998.

[206] Mindia E. Salukvadze. On the existence of solution in problems of optimization under vector valued criteria. *Journal of Optimization Theory and Applications*, 12(2):203–217, 1974.

[207] Djamila Sam-Haroud. Consistency Techniques for Continuous Constraints. *CONSTRAINTS: an international journal*, 1(1):85–118, 1996.

[208] Bardul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of Recommendation Algorithms for E-Commerce. In *Proceedings of the second ACM Conference on Electronic Commerce*, pages 158–167, Minneapoplis, Minnestota, USA, October 17-20 2000. ACM Press.

[209] Arvind Sathi and Mark S. Fox. Constraint-Directed Negotiation of Resource Allocations. In L. Gasser and M. Huhns, editors, *Distributed Artificial Intelligence Volume II*, pages 163–194. Pitman Publishing: London and Morgan Kaufmann, San Mateo, California, US, 1989.

[210] J. Ben Schafer, Joseph A. Konstan, and John Riedl. Recommender Sysems in e-commerce. In *Proceedings of the first ACM Conference on Electronic Commerce*, pages 158–166, Denver, CO, USA, 1999. ACM Press.

[211] J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Third International Conference on Genetic Algorithms*, pages 93–100, Hillsdale, NJ, USA, 19985. Lawrence Erlbaum.

[212] Thomas Schiex. Possibilistic Constraint Satisfaction Problems or "How to handle soft constraints?". In *Proceedings of the Eighth Conference of Uncertainty in Artificial Intelligence*, pages 296–275, Standford (CA), USA, July 1992.

[213] Thomas Schiex. Maximizing the reversible DAC lower bound in MaxCSP is NP-complete. In *Rapport technique INRA BIA*, volume 2, 1998.

[214] Thomas Schiex. Valued csps. In *Workshop on Soft Constraints: Theory and Practice, CP-99*, 2000.

[215] Thomas Schiex, Hélène Fargier, and Gérard Verfaillie. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence, IJCAI-95*, pages 631–637, Montreal, Canada, 1995.

[216] Beat Schmid. Requirements for Electronic Markets Architecture. *Electronic Markets Newsletter*, 7(1):3–6, 1997.

[217] Raimund Seidel. *Handbook of Discrete and Computational Geometry*, chapter 19, Convex Hull Computations, pages 361–375. Boca Raton, FL: CRC Press, 1997.

[218] Bart Selman and Henry A. Kautz. Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence, IJCAI-93*, Chambéry, France, 1993.

[219] Paolo Serafini. Some considerations about computational complexity for multi objective combinatorial problems. In J. Jahn and W. Krabs, editors, *Recent advances and historical development of vector optimization*, volume 294 of *Lecture Notes in Economics and Mathematical Systems*. Springer Verlag, Berlin, 1986.

[220] Paolo Serafini. Simulated annealing for multiobjective optimization problems. In *Proceedings of the Tenth International Conference on Multiple Criteria Decision Making*, volume 1, pages 87–96, Taipei, Taiwan (ROC), 1992.

[221] Paolo Serafini. Simulated annealing for multiple objective optimization problems. In G. H. Tzeng, H. F. Wang, V. P. Wen, and P. L. Yu, editors, *Multiple Criteria Decision Making. Expand and Enrich the Domains of Thinking and Application*, pages 283–292. Springer Verlag, 1994.

[222] Sybil Shearin and Henry Leberman. Intelligent Profiling by Example. In *Proceedings of the Intelligent User Interfaces (IUI)*, Santa Fe, New Mexico, USA, January, 14-17 2001.

[223] W. S. Shin and A. Ravindran. Interactive multiple objective optimization: survey I -continuous case. *Computing and Operational Research*, 18:97–114, 1991.

[224] Ben Shneiderman. Dynamic Queries for Visual Information Seeking. *IEEE Software*, 11(6):70–77, 1994.

[225] Nitesh Shrestha and Tomasz Janowski. Model-Based Travel Planning. Technical report, The United Nations University, International Institute for Software Technolgoy, Macau, November 2000. Report No. 219.

[226] Steven Skiena. *The Algorithm Design Manual*. Springer-Verlag, New York, 1997.

[227] Peter Spiller and Garald Lohse. A classification of Internet retail stores. *International Journal of Electronic Commerce*, 2(2), 1998.

[228] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[229] Mark Stefik. Planning with Constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16(2):111–140, 1981.

[230] Ralph E. Steuer. *Multi Criteria Optimization: Theory, Computation, and Application*. Wiley, New York, 1986.

[231] Markus Stolze. Soft Navigation in Product Catalogs. In *Proceedings Second European Conference on Research and Advanced Technology for Digital Libraries*, pages 385–396, Heraklion, Germany, September 1998. Springer Verlag, Berlin.

[232] Markus Stolze and Walid Rjaibi. Towards Scalable Scoring for Preference-based Item Recommendation. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(3):42–49, 2001.

[233] Markus Stolze and Michael Ströbel. The Shopping Gate - Enabling Role- and Preference-Specific e-Commerce Shopping Experiences. *Web Intelligence: Research and Development*, 2198:549–561, 2001.

[234] Markus Stolze and Michael Ströbel. Utility-based Decision Tree Optimization: A Framework for Adaptive Interviewing. In *Proceedings of the 8th International Conference on User Modeling*, pages 105–116, 2001.

[235] Minghe Sun and Ralph E. Steuer. InterQuad: An Interactive Quad Tree Based Procedure for Solving the Discrete Alternative Multi Criteria Problem. *European Journal of Operational Research (EJOR)*, 89(3):462–472, 1996.

[236] Orhan V. Taner and Murat Köksalan. Experiments and an Improved Method for Solving the Discrete Alternative Multi-Criteria Problem. *Journal of the Operational Research Society*, 42(5):383–392, 1991.

[237] Cynthia A. Thompson, Mehmet H. Göker, and Pat Langley. A personalized system for conversational recommendations. *Journal of Artificial Intelligence Research*, 2002. Submitted.

[238] Marc Torrens and Boi Faltings. SmartClients: Constraint Satisfaction as a Paradigm for Scaleable Intelligent Information Systems. In *Working Notes of the Workshop: Artificial Intelligence for Electronic Commerce, Technical Report WS-99-01, AAAI-99*, pages 10–15, Orlando, Florida, USA, July 1999. AAAI Press.

[239] Marc Torrens and Boi Faltings. Constraint Satisfaction for Modelling Scalable Electronic Catalogs. In Carles Sierra and Frank Dignum, editors, *Agent Mediated Electronic Commerce. A European Perspective*, number 1991 in Lecture Notes on Artificial Intelligence (LNAI), pages 217–232. Springer Verlag, December 2000.

[240] Marc Torrens, Boi Faltings, and Pearl Pu. Smartclients: Constraint satisfaction as a paradigm for scaleable intelligent information systems. *CONSTRAINTS: an international journal*, 7:49–69, 2002.

[241] Marc Torrens, Patrick Hertzog, Loic Samson, and Boi Faltings. *reality*: a Scalable Intelligent Travel Planer, Decision Support for the Business Traveler. In *Proceedings of the Eighteenth Annual ACM Symposium on Applied Computing*, Melbourne, Florida, USA, March 2003. In Press.

[242] Marc Torrens, Rainer Weigel, and Boi Faltings. Java Constraint Library: bringing constraints technology on the Internet using the Java language. In *Working Notes of the Workshop on Constraints and Agents, Technical Report WS-97-05, AAAI-97*, Providence, Rhode Island, USA, July 1997. AAAI Press.

[243] Marc Torrens, Rainer Weigel, and Boi Faltings. Java Constraint Library: bringing constraints technology on the Internet using the Java language. In *Working Notes of the CP-97 Workshop on Constraint Reasoning on the Internet. Third International Conference on Prinicples and Practice of Constraint Programming*, Linz, Austria, November 1997.

[244] Marc Torrens, Rainer Weigel, and Boi Faltings. Java Constraint Library (JCL). In *Working Notes of the Swiss Workshop on Collaborative and Distributed Systems*, Lausanne, Switzerland, May 1997.

[245] Marc Torrens, Rainer Weigel, and Boi Faltings. Distributing Problem Solving on the Web Using Constraint Technology. In *Proceedings of the 10th International Conference on Tools and Artifitial Intelligence, ICTAI-98*, pages 42–49, Taipei, Taiwan, November 1998.

[246] Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, London, UK, 1993.

[247] Edward Tsang, Chang Wang, Andrew Davenport, Christos Voudouris, and Tung L. Lau. A Family of Stochastic Methods for Constraint Satisfaction and Optimization. In *The first International Conference on The Practical Application of Constraint Technologies and Logic Programming*, London, 1999.

[248] Ekunda L. Ulungu. *Optimisation Combinatoire multicritère: Détermination de l'ensemble des solutions efficaces et méthodes interactives.* PhD thesis, Faculté des Sciences, Université de Mons-Hainaut, Mons, Belgium, 1993.

[249] Ekunda L. Ulungu and Jacques Teghem. MOSA method: a tool for solving multi-objective combinatorial optimization problems. *Journal of Multi-Criteria Decision Analysis*, 8:221–236, 1999.

[250] Ekunda L. Ulungu and Jaques Teghem. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3:83–104, 1994.

[251] Ekunda L. Ulungu, Jaques Teghem, and Philippe Fortemps. Heuristics for multi-objective combinatorial optimization by simulated annealing. In Q. Wei, J. Gu, G. Chen, and S. Wang, editors, *Multiple Criteria Decision Making: Theory, and applications. Proceedings of the Sixth International Conference on MCDM*, pages 228–238, Beijing, China, 1995. SCI-TECH Information Services.

[252] Gérard Verfaillie, Michel Lemaître, and Thomas Schiex. Russian Doll Search for Solving Constraint Optimization Problems. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-96*. AAAI Press, 1996.

[253] Christos Voudouris. *Guided Local Search for Combinatorial Optimisation Problems.* PhD thesis, Department of Computer Science, Univerity of Essex, Essex, United Kingdom, 1997.

[254] Benjamin W. Wah and Tao Wang. Simulated Annealing with Asymptotic Convergence for Nonlinear Constrained Global Optimization. In *Fifth International Conference on Principles and Practice of Constraint Programming (CP-99)*, volume 1118 of *Lecture Notes in Computer Science*, pages 461–475, Alexandria, Virginia, USA, October 1999.

[255] Richard J. Wallace. Directed Arc Consistency Preprocessing as Strategy for Maximal Constraint Satisfaction. In M. Meyer, editor, *ECAI-94, Workshop on Constraint Processing*, pages 69–77, Amesterdam, The Netherlands, 1994.

[256] Richard J. Wallace. Enhancements of branch and bound methods for the maximal constraint satisfaction problem. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-96*, pages 188–195. AAAI Press, 1996.

[257] Richard J. Wallace and Eugene C. Freuder. Conjunctive width heuristics for maximal constraint satisfaction. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-93*, pages 762–768, Washington, DC, 1993. AAAI Press.

[258] Richard J. Wallace and Eugene C. Freuder. Heuristics Methods for Over-Constrained Constraint Satisfaction Problems. In Ugo Montanari and Francesca Rossi, editors, *First International Conference on Principles and Practice of Constraint Programming, CP-95*, volume 976 of *Lecture Notes in Computer Science*, Cassis, France, 1995. Springer Verlag.

[259] L. N. Van Wassenhove and L. F. Gelders. Solving a bicriterion scheduling problem. *European Journal of Operational Research*, 4(1):42–48, 1980.

[260] Rainer Weigel. *Dynamic Abstractions and Reformulation in Constraint Satisfaction Problems*. PhD thesis, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, 1998.

[261] Steve Willmott, Monique Calisti, Boi Faltings, Santiago Macho Gonzalez, Omar Belakdhar, and Marc Torrens. CCL: Expressions of Choice in Agent Communication. In *The Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, pages 7–12, Boston, MA, USA, July 2000.

[262] Steve Willmott, Boi Faltings, Monique Calisti, Santiago Macho Gonzalez, Omar Belakhdar, and Marc Torrens. Constraint Choice Language (CCL), Language Specification v2.01. Technical Report 99/320, Ecole Polytechnique Federal de Lausanne (EPFL). Departement d'Informatique, Lausanne, Switzerland, November 1999.

[263] Steven Willmott. *Coordination Structures for the Intelligent Control of Dynamic Distributed Systems*. PhD thesis, Swiss Federal Institute of Technology in Lausanne (EPFL), Lausanne, Switzerland, 2002.

[264] Makoto Yokoo. Weak-commitment search for solving constraint satisfaction problems. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-94*, pages 313–318, Seattle, WA, 1994. AAAI Press.

[265] Ramin Zabih. Some applications of graph bandwidth to constraint satisfaction problems. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-90*, pages 46–51, Boston, MA, 1990.

[266] Milan Zeleny. Compromise programming. In M. K. Starr and M. Zeleny, editors, *Multiple Criteria Decision Making*. University of South Carolina Press, Columbia, South Carolina, USA, 1973.

[267] Milan Zeleny. *Multiple Criteria Decison Making*. McGraw-Hill Book Company, New York, 1982.

[268] Weixiong Zhang. Depth-First Branch-and-Bound versus Local Search: A Case Study. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-2000*, pages 930–935, Austin, Texas, 2000. AAAI Press.

[269] Stanley Zionts. A Multiple Criteria Method for Choosing among Discrete Alternatives. *European Journal of Operational Ressearch (EJOR)*, 7(1):143–147, 1981.

Index

reality, **151**
- architecture, 152
- scenario, 165
- solution display, 155

Choice Constraint Language (CCL), 206
classical CSP, **32**, 181
- algorithms
  - Arc-consistency, 206
  - Backjumping, 183, 205
  - Backmarking, 184, 205
  - Chronological Backtracking, 182, 205
  - Connectionist approach, 189
  - consistency, 182
  - Fast Local Search, 189
  - Forward Checking, 185, 205
  - Generate and Test, 181
  - Genetic Algorithms, 190
  - Guided Local Search, 189
  - Hill climbing, 188
  - look-ahead strategies, 185
  - look-back strategies, 183
  - Maintaining Arc Consistency, 186
  - Min-conflicts, 189
  - Path-consistency, 206
  - Random walk, 189
  - Simulated Annealing, 190
  - stochastic, 188
  - systematic search, 181
  - Tabu search, 190
  - variable and value ordering, 186
- conventions, 32
- definitions, 33

configuration, 7
- case-based reasoning, 11
- constraint-based reasoning, 10
- customization, 12
- electronic catalog, 11, 14
- logic-based reasoning, 10
- model-based reasoning, 10
- problem, 8
- ruled-based reasoning, 10

constraint
- configuration, 41, 42
- optimization criteria, 41, 43, 60
- user preference, 41, 42, 60

electronic catalog, 4
- architecture, 4
- business logic layer, 5
- complex industries, 15
- complexity, 14
- configuration, 7, 11
- persistence layer, 4
- presentation layer, 6
- travel industry, 15

electronic commerce, 2
- catalog, 4
- configurable goods, 4
- tangible goods, 4

extended CSP, 38, **193**
- constraint hierarchies, 45, **193**
- fuzzy (FCSP), 45, **194**
- lexicographic, 45, **196**
- MAX-CSP, 45, **194**
- partial, **194**
- possibilistic, 45, **195**
- probabilistic, 45, **195**
- semiring-based (SCSP), 45, **196**
- valued (VCSP), 198
- weighted (WCSP), 45, **196**

information systems, 2

Java Constraint Library (JCL), 203

mixed-initiative system, 24

online interaction, 28

personalization, 11, 21
  customization, 12
  electronic catalog, 14
planning travel problem, 139
  model, 143
    hard constraints, 144
    optimization constraints, 146
    soft constraints, 144
    values, 143
    variables, 143
  standard architecture, 137
preference elicitation
  modify, 26
preference elicitation, 25
  negative, 26
  positive, 26
  post, 26
  retract, 26

qualitative approach, 48
qualitative approach, **54**
  algorithms
    algorithms, 89
    constraint weight distribution, 93
    convex-hull, 111
    Gavanelli's algorithm, 106
    genetic, 110
    global criterion method, 108
    goal programming, 108
    hierarchical method, 108
    iterative method, 92, 109
    normal-boundary intersection method, 106
    parametric scalarization, 107
    simulated annealing, 109
    tabu search, 109
    trade-off method, 108
  convex-hull, 60, 61, 99
  MCOP, 47
  optimization criteria, 60
  Pareto, 56
  Pareto-optimality, 56
  solution dominance, 55
  solution ordering, 55
  user preference, 60
quantitative approach
  algorithms
    approximative methods, 104
    genetic algorithms, 104
quantitative approach
  algorithms
    depth first branch and bound, 74
quantitative approach, 48, **49**, 71
  algorithms
    anytime algorithm, 87
    backjumping, 84
    backmarking, 85
    depth first branch and bound, 71, **72**, 75
    lower bound cost function, 72
    min conflicts, 104
    partial forward checking, 77
    preprocessing look-ahead, 81
    prospective strategies, 77
    search ordering heuristics, 82
    simple method, 90
    simulated annealing, 104
    upper bound cost function, 72
    value ordering, 84
    variable ordering, 83
  assignment valuation, 49
  constraint weight vector, 49
  WCSP, 50

recommender system, 12
  attribute-based, 13
  item-to-item, 14
  non-personalized, 13
  people-to-people, 14

SmartClient, 123

software architecture, 114
    client-server, 115
    client-stateful-server, 116, 122
    client-stateless-server, 116, 122
    layered, 116
    SmartClient, 123
    three-tier, 117
    two-tier, 117
solution space, 43

traditional commerce, 2, 21
travel industry, 131
    web-based, 134
        CRS, 134
        GDS, 134

user profiles, 29

Vilfredo Pareto, 56

Curriculum Vitae and Publications

Personal Data

- **Name:** Marc Torrens i Arnal
- **Date and Place of Birth:** March 27th 1972 in Barcelona, Catalonia, Spain.
- **Nationality:** Spanish.
- **Languages:** Catalan, Spanish, French and English.
- **Military Service:** Social Service in Croatia (1992).

Education

1999 - 2002	Swiss Federal Institute of Technology in Lausanne (EPFL) **Ph.D.** *Scalable Intelligent Electronic Catalogs*
1998 - 1999	Swiss Federal Institute of Technology in Lausanne (EPFL) **Postgraduate course** *Intelligent Agents*.
1996 - 1997	Swiss Federal Institute of Technology in Lausanne (EPFL) **Erasmus european exchange**. Diploma work on *Air Travel Planning System on the Web*.
1991 - 1997	Technical University of Catalonia (UPC), Barcelona, Spain. **Computer Science Engineering**.

Professional Experience

July 2001 - present	i:FAO Switzerland (formerly Iconomic Systems S.A.) **Project Manager** for i:FAO Future Lab.
August 2000 - June 2001	Iconomic Systems S.A. **Co-founder** and **Technology Officer**.
September 1997 - July 2000	Swiss Federal Institute of Technology in Lausanne (EPFL) **Research Assistant** for Prof. Boi Faltings.

Research Publications

1. *reality: a Scalable Intelligent Travel Planner, Decision Suppport for the Business Traveler*, Marc Torrens, Patrick Hertzog, Loic Samson and Boi Faltings. Proceedings of the Eighteenth Annual ACM Symposium on Applied Computing. March 2003. Melbourne, Florida, USA. *In Press*.

2. *Multi-criteria Optimization in Constraint Satisfaction Problems*, Marc Torrens and Boi Faltings. Working Notes of the Workshop on preferences in AI and CP: symbolic approaches, National Conference on Artificial Intelligence (AAAI-02), Edmonton, Alberta, Canada, July 2002.

3. *Smart Clients: Constraint satisfaction as a paradigm for scaleable intelligent information systems*, Marc Torrens and Boi Faltings. Special Issue on Constraints and Agents of the International Journal on Constraints, volume 7, pag. 49-69. 2002. Kluwer Academic Publishers.

4. *Constraint Satisfaction for Modelling Scalable Electronic Catalogs*, Marc Torrens and Boi Faltings. In Carles Sierra and Frank Dignum, editors, Agent Mediated Electronic Commerce: An European Perspective. Lecture Notes on Artificial Intelligence, LNAI 1991, pag. 217-232. Springer-Verlag. December 2000.

5. *Smart Clients: Constraint satisfaction as a paradigm for scaleable intelligent information systems*, Marc Torrens and Boi Faltings. Working Notes of the Workshop *Artificial Intelligence for Electronic Commerce*, Technical Report WS-99-01, pag. 10-15, National Conference on Artificial Intelligence (AAAI-99), Orlando, Florida, USA, July 1999.

6. *Distributing Problem Solving on the Web using Constraint Technology.* Marc Torrens and Boi Faltings. Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI-98), pag. 42-49, Taipei, Taiwan (ROC), November 1998.

7. *Java Constraint Library.* Marc Torrens, Rainer Weigel and Boi Faltings. Working Notes of the Swiss Workshop on Collaborative and Distributed Systems, Lausanne, Switzerland, May 1997.

8. *Java Constraint Library: bringing constraints technology on the Internet using Java language.* Marc Torrens, Rainer Weigel and Boi Faltings. Working Notes of the Workshop on Constraints and Agents, Technical Report WS-97-05, National Conference on Artificial Intelligence (AAAI-97), Providence, Rhode Island, USA, July 1997.

9. *Java Constraint Library: bringing constraints technology on the Internet using Java language.* Marc Torrens, Rainer Weigel and Boi Faltings. Working Notes of the Workshop on Constraint Reasoning on the Internet, Third International Conference on Principles and Practice of Constraint Programming (CP-97). Linz, Austria, November 1997.

Co-authored papers

1. *A Multi-Agent Recommender System for Planning Meetings.* Santiago Macho, Marc Torrens and Boi Faltings. Workshop on Recomender Systems. Fourth International Conference on Autonomous Agents (Agents 2000). Barcelona, Spain, June 2000.

2. *CCL: Expressions of Choice in Agent Communication.* Steven Willmott, Monique Calisti, Boi Faltings, Santiago Macho, Omar Belakdhar and Marc Torrens. The Fourth International Conference on MultiAgent Systems (ICMAS-2000), July 2000. Boston, USA.

3. *Constraint Choice Language (CCL), Language Specification v2.01.* Steven Willmott, Monique Calisti, Boi Faltings, Santiago Macho, Omar Belakdhar and Marc Torrens. EPFL, Département d'Informatique. Technical Report No. 99/320. November 1999.

4. *Interchangeability for Case Adaptation in Configuration Problems.* Rainer Weigel, Boi Faltings and Marc Torrens. Working Notes of the Workshop: Case-based Reasoning Integrations, Technical Report WS-98-15, pag. 166-171, National Conference on Artificial Intelligence (AAA-98), Madison, Wisconsin, USA, July 1998.