DISTRIBUTED CONSTRAINT OPTIMIZATION:

PRIVACY GUARANTEES AND STOCHASTIC UNCERTAINTY

Thèse n. 5197 2011
présentée le 29 septembre 2011
à la Faculté Informatique et Communications
Laboratoire d'Intelligence Artificielle
programme doctoral en Informatique et Communications
École Polytechnique Fédérale de Lausanne

pour l'obtention du grade de Docteur ès Sciences
par

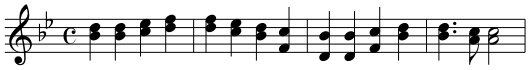
Thomas Léauté

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

acceptée sur proposition du jury :

Prof. Karl Aberer, président du jury
Prof. Boi Faltings, directeur de thèse
Prof. Jean-Pierre Hubaux, rapporteur
Prof. Makoto Yokoo, rapporteur
Dr. Juan A. Rodríguez-Aguilar, rapporteur

Lausanne, EPFL, 2011



— Ludwig van Beethoven (1770–1827)

To Julie…

Acknowledgements

I would like to start by expressing my warm acknowledgments to my thesis director, Prof. Boi Faltings, for his research supervision over the last five years, his support, and his very accommodating management style. My acknowledgements extend to the members of my thesis jury, for their expert and valuable feedback. I would also like to thank the two successive lab assistants I have got the chance to meet: Marie Decrauzat for her admirable dedication, and Sylvie Thomet for her support and joie de vivre.

Next, I would like to attempt to write a few meaningful(?) words about each of the co-workers at the Artificial Intelligence Lab whom I have got the pleasure to work with. In alphabetical order of their first names: Adrian Petcu, for his monumental work upon which I built this thesis, for setting an unachievable precedent for productivity, and for being his invaluable self; Brammert Ottens, for his major contribution to FRODO and our fruitful discussions; Christos Dimitrakakis, whom I have not had the time to get to know better yet; David Portabella, for his friendliness; Florent Garcin, for being his invaluable self too, and making a decent replacement for Adrian; Immanuel Trummer, for having such an unusual number of *m*'s in his name (I only just noticed); Jason Li, for his cute kids; Jean-Cedric Chappelier, for his occasional advice; Li Pu, for heroically surviving Florent; Ludek Cigler, for enriching my Google results with Harvard-hosted presentations; Marita Ailomaa, for her friendliness, and her long-missed contribution to the female-to-male ratio in the lab; Martin Rajman, for his critical thinking; Martin Veselý, for his support as system admin; Michael Schumacher, for his collaboration on the A.I. course; Mirek Melichar, whom I did not get the chance to know so well; Paolo Viappiani, for recommending me places in Toronto; Radek Szymanek, for his advice on redesigning FRODO, and for sharing my office (and effectively, part of my life) for so many years; Radu Jurca, for his inspiring thoughts about what one should do with one's life; Vincent Schickel, for his priceless help with computers and with the A.I. course; and Walter Binder, for allowing me to share his office for a very short period of time, and for putting an end to the longest sentence of this thesis.

During these five years of work towards my PhD, I have spent a measurable amount of time developing the FRODO software platform, which really was a collaborative effort that involved a certain number of people whom I would like to thank for their

contributions. Let me first mention the students I supervised or co-supervised, again in alphabetical order of their first names: Achraf Tangui, Alexandra Olteanu, Andreas Schädeli, Arnaud Jutzeler, Cécile Grometto, Éric Zbinden (with a special mention for his remarkable work on P-DPOP and P^2-DPOP), Jonas Helfer, Nacereddine Ouaret, and Stéphane Rabie. As FRODO was getting more traction, we also received a number of contributions from external users; this includes most noticeably Xavier Olive, and Prof. George Vouros and Sokratis Vavilis. I would also like to thank the following people for their constructive feedback: Abhishek Singh, Fredrik Heintz, Kathryn Macarthur, Roie Zivan, Sankalp Khanna, and Prof. Youssef Hamadi.

My acknowledgements also go to the researchers at EPFL and outside with whom I have had occasional enriching interactions; this includes my collaborators at the Nokia Research Center in Lausanne, Gian Paolo Perrucci and Imad Aad.

Almost last, but not almost not least, I would like to sincerely thank Julie for her love, her patience, and for tolerating my too often work-focused mindset. This also applies to the other members of my close family, whom I did not get to see so often.

Finally, I would like to apologize to anyone I might have unforgivably omitted.

Lausanne, August 20, 2011 T. L.

Abstract

Distributed Constraint Satisfaction (DisCSP) and *Distributed Constraint Optimization (DCOP)* are formal frameworks that can be used to model a variety of problems in which multiple decision-makers cooperate towards a common goal: from computing an equilibrium of a game, to vehicle routing problems, to combinatorial auctions. In this thesis, we independently address two important issues in such multi-agent problems: 1) how to provide strong guarantees on the protection of the privacy of the participants, and 2) how to anticipate future, uncontrollable events.

On the privacy front, our contributions depart from previous work in two ways. First, we consider not only *constraint privacy* (the agents' private costs) and *decision privacy* (keeping the complete solution secret), but also two other types of privacy that have been largely overlooked in the literature: *agent privacy*, which has to do with protecting the identities of the participants, and *topology privacy*, which covers information about the agents' co-dependencies. Second, while previous work focused mainly on quantitatively measuring and reducing privacy loss, our algorithms provide stronger, qualitative guarantees on what information will remain secret. Our experiments show that it is possible to provide such privacy guarantees, while still scaling to much larger problems than the previous state of the art.

When it comes to reasoning under uncertainty, we propose an extension to the DCOP framework, called *DCOP under Stochastic Uncertainty (StochDCOP)*, which includes uncontrollable, random variables with known probability distributions that model uncertain, future events. The problem becomes one of making "optimal" offline decisions, before the true values of the random variables can be observed. We consider three possible concepts of optimality: minimizing the *expected* cost, minimizing the *worst-case* cost, or maximizing the probability of a-posteriori optimality. We propose a new family of StochDCOP algorithms, exploring the tradeoffs between solution quality, computational and message complexity, and privacy. In particular, we show how discovering and reasoning about co-dependencies on common random variables can yield higher-quality solutions.

Keywords: Distributed Constraint Satisfaction (DisCSP), Distributed Constraint Optimization (DCOP), Privacy, Uncertainty, P-DPOP, $\mathbb{E}$[DPOP], FRODO

Résumé

La *satisfaction distribuée sous contraintes (DisCSP)* et l'*optimisation distribuée sous contraintes (DCOP)* sont des formalismes qui peuvent être utilisés pour modéliser une large gamme de problèmes dans lesquels plusieurs décideurs coopèrent pour atteindre un objectif commun : du calcul d'un équilibre d'un jeu stratégique, au routage de véhicules, en passant par les enchères combinatoires. Dans cette thèse, nous adressons indépendamment deux problématiques importantes dans de tels systèmes multi-agents : 1) comment apporter des garanties quant à la protection de la confidentialité des participants, et 2) comment anticiper des événements futurs incontrôlables.

Sur le front de la confidentialité, nos contributions se démarquent des précédents travaux de deux points de vue. Premièrement, nous considérons non seulement la *confidentialité des contraintes* (c'est-à-dire, les coûts et préférences privés des agents) et la *confidentialité décisionnelle* (garder secrète la solution complète), mais aussi deux autres types de confidentialité qui ont été généralement ignorés dans la littérature : la *confidentialité des agents*, qui implique la protection des identités des participants, et la *confidentialité topologique*, qui couvre les informations concernant les co-dépendances entre les agents. Deuxièmement, alors que les précédents travaux se sont concentrés principalement sur la mesure et la réduction des pertes de confidentialité, nos algorithmes apportent des garanties qualitatives plus fortes en ce qui concerne les informations qui restent secrètes. Nos expériences prouvent qu'il est possible d'apporter de telles garanties, tout en rendant aussi possible la résolution de problèmes beaucoup plus grands que précédemment.

En ce qui concerne l'incertitude, nous proposons une extension du formalisme DCOP, appelée *DCOP avec incertitude stochastique (StochDCOP)*, qui inclut des variables aléatoires incontrôlables, dont les distributions probabilistes sont connues, et qui modélisent des événement futurs incertains. Le problème consiste alors à prendre à l'avance des décisions “optimales", avant de pouvoir observer les valeurs effectives des variables aléatoires. Nous considérons trois concepts possibles d'optimalité : la minimisation de l'espérance du coût, la minimisation du pire coût, et la maximisation de la probabilité d'optimalité a posteriori. Nous proposons une nouvelle famille d'algorithmes de StochDCOP, qui explore les compromis entre la qualité de la solution, la

complexité de calcul et de communication, et la confidentialité. En particulier, nous montrons comment découvrir et raisonner sur l'existence de co-dépendances à des variables aléatoires communes peuvent générer des solutions de meilleure qualité.

Mots-clés : satisfaction distribuée sous contraintes (DisCSP), optimisation distribuée sous contraintes (DCOP), confidentialité, incertitude, P-DPOP, $\mathbb{E}$[DPOP], FRODO

Contents

Acknowledgements **v**

Abstract (English/Français) **vii**

Table of Contents **xiv**

List of Figures **xvi**

List of Tables **xvii**

List of Algorithms **xix**

List of Symbols **xxiii**

1 Introduction **1**

2 Preliminaries **7**
2.1 The DisCSP and DCOP Formalisms 7
2.2 Examples of Applications 9
2.2.1 Graph Coloring 10
2.2.2 Meeting Scheduling 10
2.2.3 Sensor Networks 11
2.2.4 Resource Allocation 13
2.2.5 Distributed Vehicle Routing Problems 14
2.2.6 Equilibria in Graphical Games 16
2.3 Complete DisCSP Algorithms 19
2.3.1 ABT: Asynchronous Backtracking 19
2.3.2 AWC: Asynchronous Weak Commitment 20
2.3.3 AAS: Asynchronous Aggregation Search 21
2.3.4 AFC: Asynchronous Forward Checking 23
2.3.5 APO: Asynchronous Partial Overlay 23
2.3.6 ConcDB: Concurrent Dynamic Backtracking 24
2.3.7 MPC-DisCSP4: Secure Multiparty Computation 24
2.4 Complete DCOP Algorithms 24
2.4.1 SynchBB: Synchronous Branch and Bound 24

2.4.2 ADOPT: Asynchronous Distributed OPTimization 25
2.4.3 OptAPO: Optimization by Asynchronous Partial Overlay 29
2.4.4 DPOP: Dynamic Programming Optimization Protocol 29
2.4.5 AFB: Asynchronous Forward Bounding 32
2.4.6 NCBB: No-Commitment Branch and Bound 33
2.4.7 ConcFB: Concurrent Forward Bounding 33
2.5 Incomplete DCOP Algorithms 33
2.5.1 DBA: Distributed Breakout Algorithm 33
2.5.2 DSA: Distributed Stochastic Algorithm 34
2.5.3 MGM: Maximum Gain Message 35
2.5.4 A-DPOP: Approximate DPOP 35
2.5.5 The Max-Sum Algorithm 35
2.5.6 DaCSA: Divide and Coordinate Subgradient Algorithm 36
2.5.7 DALO: Distributed Asynchronous Local Optimization 37

3 Strong Privacy Guarantees in DCOP 39
3.1 Privacy Definitions 39
3.2 Related Work 41
3.2.1 Cooperative ElGamal Homomorphic Encryption 41
3.2.2 Privacy in DisCSP and DCOP 44
3.2.3 Privacy in Linear Programming 49
3.3 The P-DPOP Algorithm 50
3.3.1 Overview of the Algorithm 51
3.3.2 Root Variable Election 51
3.3.3 UTIL Propagation 53
3.3.4 VALUE Propagation 56
3.3.5 Algorithm Properties 56
3.3.6 P-DPOP$^-$: Trading off Topology Privacy for Performance 58
3.4 Full Decision Privacy: the P$^{3/2}$-DPOP Algorithm 58
3.4.1 Overview of the Algorithm 59
3.4.2 Assigning Unique IDs to Variables 60
3.4.3 Routing of Messages along a Circular Variable Ordering 60
3.4.4 Choosing a New Root Variable 62
3.4.5 Algorithm Properties 64
3.5 Full Constraint Privacy: the P^2-DPOP Algorithm 65
3.5.1 Encrypted UTIL Propagation along a Linear Variable Order . . . 66
3.5.2 Decrypting an Optimal Value for the Root Variable 67
3.5.3 Algorithm Properties 68
3.6 Privacy Analysis 69
3.6.1 Agent Privacy 69
3.6.2 Topology Privacy 73
3.6.3 Constraint Privacy 81

3.6.4 Decision Privacy . . . 84
3.7 Experimental Results . . . 85
3.7.1 Graph Coloring . . . 86
3.7.2 Meeting Scheduling . . . 88
3.7.3 Resource Allocation . . . 91
3.7.4 Equilibria in Graphical Games . . . 93
3.7.5 Vehicle Routing Problems . . . 96
3.8 Summary . . . 98

4 StochDCOP: DCOP under Stochastic Uncertainty 99
4.1 DCOP under Stochastic Uncertainty . . . 99
4.1.1 StochDCOP Formalism . . . 99
4.1.2 Evaluation Functions . . . 101
4.2 Examples of Applications . . . 104
4.2.1 Stochastic Graph Coloring . . . 104
4.2.2 Sensor Networks with Moving Targets . . . 105
4.2.3 Stochastic Vehicle Routing . . . 107
4.2.4 The Distributed Kidney Exchange Problem . . . 109
4.3 Related Work . . . 110
4.3.1 Distributed Coordination of Exploration and Exploitation . . . 111
4.3.2 DCOP with Utility Distributions . . . 111
4.3.3 Quantified DCOP with Virtual Adversaries . . . 112
4.4 Complete, Centralized Reasoning: Comp-$\mathbb{E}$[DPOP] . . . 115
4.4.1 General Approach Based on Virtual Agents . . . 115
4.4.2 Distributed Pseudo-tree Generation . . . 117
4.4.3 Generating a *Consistent* Pseudo-tree . . . 119
4.4.4 Algorithm Properties . . . 121
4.5 Incomplete, *Local* Reasoning: Local-$\mathbb{E}$[DPOP] . . . 123
4.6 Incomplete, *Global* Reasoning: Global-$\mathbb{E}$[DPOP] . . . 127
4.6.1 Sharing Information about Dependencies on Random Variables 128
4.6.2 Limiting the Propagation of Information . . . 129
4.6.3 Related Work on Resource-Constrained DCOP . . . 131
4.6.4 Influence of the Choice of the Pseudo-tree . . . 132
4.6.5 Hybrid $\mathbb{E}$[DPOP] for Non-commensurable Evaluation Functions 134
4.6.6 Complexity Analysis . . . 135
4.7 Equivalence Theorems . . . 135
4.7.1 Global-$\mathbb{E}$[DPOP] $\equiv$ Local-$\mathbb{E}$[DPOP] . . . 135
4.7.2 Central-$\mathbb{E}$[DPOP] $\equiv$ Global-$\mathbb{E}$[DPOP] . . . 137
4.8 Approximations by Sampling Scenarios . . . 138
4.8.1 Motivation and Challenges . . . 138
4.8.2 Independent Sampling . . . 140
4.8.3 Collaborative Sampling . . . 141

4.8.4 Consensus vs. Expectation 142
4.9 Experimental Results 144
4.9.1 Solution Quality/Complexity Tradeoff 146
4.9.2 Partial Centralization and Privacy Loss 152
4.9.3 Effects of Sampling 153
4.10 Summary 156

5 The FRODO 2 Platform **159**
5.1 Architecture 160
5.1.1 Communications Layer 160
5.1.2 Solution Spaces Layer 161
5.1.3 Algorithms Layer 163
5.2 Features 164
5.2.1 Performance Metrics 165
5.2.2 Graphical User Interface 165
5.2.3 Websites 166
5.3 Contributors 167

6 Conclusion **169**

Bibliography **188**

Index **189**

Curriculum Vitae **193**

List of Personal Publications **195**

List of Figures

2.1 The constraint graph for a simple graph coloring problem. 10
2.2 The constraint graph for a simple meeting scheduling problem. 11
2.3 Sample sensor network problem. 12
2.4 Constraint graph for target k (in the middle). 13
2.5 A simple DisSDMDVRP instance. 15
2.6 The DCOP constraint graph corresponding to Figure 2.5. 16
2.7 An instance of the party game. 18
2.8 The problem in Figure 2.1 and a possible pseudo-tree. 26
2.9 DPOP's pseudo-tree and UTIL messages on the problem in Figure 2.8. 30
2.10 The factor graph for the problem in Figure 2.8. 35

3.1 The graph coloring problem from Figure 2.1. 40
3.2 The pseudo-tree from Figure 2.9. 52
3.3 The UTIL message $x_1 \rightarrow x_4$ in Figure 3.2. 53
3.4 The UTIL message received by agent $a(x_3)$ in Figure 3.2. 56
3.5 The (counter-clock-wise) circular ordering based on Figure 3.2. 62
3.6 UTIL propagation (in cleartext) in P^2-DPOP for Figure 2.1. 66
3.7 Two indistinguishable timelines from x's point of view. 75
3.8 Runtime performance on graph coloring problems. 87
3.9 Communication performance on graph coloring problems. 87
3.10 Runtime performance on small meeting scheduling problems. 88
3.11 Communication performance on small meeting scheduling problems. 88
3.12 Runtime performance on large meeting scheduling problems. 90
3.13 Communication performance on large meeting scheduling problems. 90
3.14 DisCSP formulation for a resource allocation problem. 92
3.15 Runtime performance on resource allocation problems. 93
3.16 Communication performance on resource allocation problems. 93
3.17 The constraint graph for the game in Figure 2.7. 94
3.18 Runtime performance on party games. 95
3.19 Communication performance on party games. 96

4.1 Sample stochastic graph coloring problem. 104
4.2 Allowed target moves. 105

4.3 Partial constraint graph for target k. 106
4.4 Some remarkable *on/off* sensor configurations. 107
4.5 A StochDisSDMDVRP instance. 108
4.6 The StochDCOP constraint graph for Figure 4.5. 108
4.7 A *consistent* pseudo-tree for the constraint graph in Figure 4.1. 116
4.8 Rob-Local-$\mathbb{E}$[DPOP]'s UTIL messages for the problem in Figure 4.1. . . 124
4.9 Possible pseudo-trees for a sensor network problem. 124
4.10 Rob-Global-$\mathbb{E}$[DPOP]'s UTIL messages for the problem in Figure 4.1. . . 130
4.11 Runtime on random graph coloring problems. 146
4.12 Runtime on random graph coloring problems. 147
4.13 Amount of information exchanged on graph coloring problems. 147
4.14 Optimality gap for Cons and Cons/Exp on graph coloring problems. . 148
4.15 Optimality gap for Rob-$\mathbb{E}$[DPOP] on graph coloring problems. 149
4.16 Optimality gap for Cons and Cons/Exp on kidney exchange problems. 150
4.17 Level of centralization of Comp-$\mathbb{E}$[DPOP] on graph coloring problems. 153
4.18 Solution quality convergence with the number of samples. 154
4.19 Solution quality convergence with the number of samples. 155

5.1 General FRODO software architecture. 160
5.2 FRODO's main GUI window. 166
5.3 FRODO's auxiliary GUI windows. 166

List of Tables

3.1 Privacy guarantees of various algorithms. 69
3.2 Experimental results on DisSDMDVRP benchmarks. 97

4.1 Optimal colorings for the problem in Figure 4.1. 104
4.2 Colorings found for the problem in Figure 4.1. 126
4.3 Pseudo-trees for a sensor network problem, varying the heuristic. . . . 133
4.4 The $\mathbb{E}$[DPOP] family of algorithms. 139
4.5 Solution quality and performance on VRP benchmarks. 151

List of Algorithms

1 The ABT algorithm for variable x. 20
2 Backtracking in AWC. 21
3 The AFC algorithm for variable x_i. 22
4 The SynchBB algorithm for variable x_i. 25
5 Root election algorithm for variable x. 27
6 Pseudo-tree generation algorithm for variable x 28
7 DPOP's UTIL propagation for variable x 30
8 The DBA algorithm for variable x. 34
9 The DSA algorithm (replacement for lines 11 to 24 in Algorithm 8) . . . 34
10 The Max-Sum algorithm. 36
11 The MPC-DisCSP4 algorithm for agent a_i. 47
12 MULTIPLY(V_1, V_2) for agent a_i. 48
13 Overal P-DPOP algorithm, for variable x 51
14 Anonymous root election algorithm for variable x. 52
15 P-DPOP's obfuscated UTIL propagation, for variable x 54
16 Overall P$^{3/2}$-DPOP algorithm with full decision privacy, for variable x . 59
17 Pseudo-tree and unique ID generation algorithm for variable x 61
18 Sending a message M clock-wise in the circular variable ordering. . . . 62
19 Algorithm to choose a new root, for variable x 63
20 Collaborative decryption of a multiply-encrypted cyphertext e 64
21 P^2-DPOP's UTIL propagation procedure, for variable x 66
22 Finding an optimal value for x in the encrypted cost matrix $m(x)$ 67
23 Finding a feasible value in the encrypted Boolean matrix $m(x)$ 68
24 Pseudo-tree generation algorithm for variable x 118
25 Local-$\mathbb{E}$[DPOP]'s UTIL propagation for variable x, with parent y 123
26 Central-$\mathbb{E}$[DPOP]'s UTIL propagation for variable x, with parent y . . . 128
27 Global-$\mathbb{E}$[DPOP]'s UTIL propagation for variable x, with parent y 130
28 Distributed computation of the lcas, for each decision variable x 131
29 Collaborative sampling procedure, for each decision variable x 141
30 Computational steps for the Exp-Local- approach. 143
31 Computational steps for the Cons/Exp-Local- approach. 144

List of Symbols

$\mathcal{A}$	a list of agents
a_i	the ith agent
$a(x)$	the agent owning the decision variable x
$\mathcal{C}$	a list of constraints
c_i	the ith constraint
c^X	agent X's internal constraint
$c_{\max}$	the maximum feasible cost value of a constraint
C	the set of customers in a VRP
c^i	the ith customer in a VRP
$\mathcal{D}$	a list of domains for decision variables
D_i	the domain of the ith decision variable x_i
D_x	the domain of the decision variable x
$D_{\max}$	the size of the largest variable domain
$\tilde{D_y^x}$	the obfuscated representation of variable y's domain used by variable x
Δ	a list of domains for random variables
Δ_i	the domain of the ith random variable
Δ_r	the domain of the random variable r
d_i	the ith depot in a VRP
δ_X	$= 1$ if X is true, 0 otherwise
$\mathbb{E}_r[c]$	the expectation of the cost function c with respect to the random variable r

e_c	the evaluation function with respect to the cost function c
e_c^r	the evaluation function with respect to the cost function c, restricted to the random variable r
$E(m)$	the ElGamal encryption of a message m
H	the visibility horizon of each depot, in a VRP
id_x	the unique identifier associated to variable x by Algorithm 17
id_x^+	an upper bound on id_x (Algorithm 17, line 5)
$incr_{\min}$	a free parameter of Algorithm 17 (line 5)
key_x^y	the key used by variable y to obfuscate the dependency of its subtree's cost/utility on variable x, in P-DPOP (Algorithm 13, line 13)
$lca(r)$	the lowest common ancestor of all decision variables in the pseudo-tree that must enforce a constraint involving the random variable r
$L_{\max}$	the maximum route length of a vehicle, in a VRP
m_i	the ith meeting, in a meeting scheduling problem
m_i^X	the decision variable modeling the time at which agent X schedules the ith meeting, in a meeting scheduling problem
n	the number of decision variables
n^+	an upper bound on n, computed by Algorithm 17
N_v	the list of neighbors of the vertex v in a graph
n_e	the number of edges in the constraint graph
n_V	the number of vehicles controlled by each depot, in a VRP
$\mathcal{P}$	a list of probability distributions
π_i	the probability distribution for the ith random variable
π_r	the probability distribution for the random variable r
$\mathbb{P}$	the *consensus* evaluation function
$\phi_{\max}$	an upper bound on the diameter of the constraint graph
Q	the set of customer demands in a VRP
q^i	the demand of the ith customer, in a VRP

$Q_{\max}$	the maximum vehicle load of a vehicle, in a VRP
$\mathcal{R}$	a list of random variables
r_i	the ith random variable
σ	a mixed strategy in a game
σ_y^x	the permutation applied by variable x to variable y's domain
S	the set of pure strategies in a game
$s(c)$	the size of the scope of the cost function c
S_r	a list of sampled values for the random variable r
sep_x	the separator of variable x in the pseudo-tree
$sep_{\max}$	the size of the largest separator
u	a constraint (or a value thereof), expressed in terms of utility
$vector_x$	the (encrypted) rerooting vector for variable x (Algorithm 16, line 3)
vrp_i	the VRP constraint of the ith agent
w	the treewidth of the constraint graph
w_i	the induced width of the pseudo-tree
$\mathcal{X}$	a list of decision variables
x_i	the ith decision variable
x^*	the optimal value chosen for the decision variable x
$\tilde{y^x}$	the codename used by variable x to represent variable y

1 Introduction

In many real-world decision problems one might be faced with, the optimal decision to make depends on simultaneous decisions made by others; in such settings, assuming that the common goal of the decision-makers is to find a joint choice of decisions that maximizes the social welfare, they need to exchange information about their respective constraints and preferences in order to coordinate their decisions. Consider for instance two delivery companies working under the same franchise, which must decide how to serve customers who have posted requests for various quantities of goods, so that the overall delivery cost is minimized. In the Operations Research literature, this problem is known as the *Multiple-Depot, Vehicle Routing Problem (MDVRP)*, in which each company controls a depot of goods and a fleet of delivery vehicles; the decision problem is which depot should serve which customers, with which vehicles, and in which order, so as to minimize the total route length, under the constraint that the vehicles have limited capacities. None of the two companies' knowledge about the overall problem is sufficient for the company to solve it on its own: typically, no company knows all the details of the other company's vehicle fleet, internal costs, and previous delivery commitments — in fact, even if they are business partners, the other company will probably want to keep some of this information private. Therefore, in order to solve the problem optimally, the agents need to follow some sort of decision protocol that involves exchanging ideally just enough information to find the optimal solution.

Another every-day example of such a decision problem requiring the coordination of multiple participants is the problem of *meeting scheduling*. Consider a group of friends who would like to choose a date and time for a gathering, such that they can all attend. One simple way to proceed is to designate one organizer, to whom all other attendees should report their respective availabilities and preferences, and who is then responsible for picking a time that suits all. One typical issue with this *centralized* solution process is that of privacy: the participants are not necessarily willing to reveal their full lists of available time slots to anyone. Another issue arises when some

participants' availabilities actually depend on the dates and times of other events with other people; it then becomes cumbersome and often undesirable to centralize at one participant all the necessary information about all events to be scheduled, even those to which she is not invited.

In order to be able to mathematically formalize such distributed decision problems and develop general algorithms to solve them, previous work in Distributed Artificial Intelligence has proposed the framework of *Distributed Constraint Satisfaction Problems (DisCSPs)*. In a DisCSP, the decision-makers (or the computers acting on their behalves) are called *agents*, and each decision that a given agent must make is modeled by a *decision variable*, which must be assigned a *value* taken among a set of possible values corresponding to the possible choices for that decision. The agents also express *constraints* on subsets of decision variables, which specify which combinations of value assignments to these decision variables are allowed or disallowed. A solution to the problem is a value assignment to all decision variables that satisfies all the constraints. In a generalization of the DisCSP formalism, called *Distributed Constraint Optimization (DCOP)*, the constraints that the agents express are also allowed to specify *costs* or *utilities* that variable-value assignments incur to the agents; an optimal solution to the problem is then a value assignment to all decision variables such that the total sum of all costs across all agents is minimized (or the total utility is maximized).

Over the recent years of research in this field, a number of distributed algorithms have been proposed to solve DisCSPs and DCOPs, without having to centralize all information about the problem into one agent. Typically, many of these algorithms are based on a general problem-solving technique called *search*, in which agents make tentative decisions (i.e. tentative assignments of values to their decision variables), then communicate these tentative decisions to other agents, which must attempt to make their own decisions accordingly, or send notifications if previous tentative decisions made by other agents have made their own local problem infeasible (or sub-optimal).

Novel Contributions on the Topic of Privacy

In this thesis, we advance the state of the art in the research on DisCSP and DCOP by addressing two issues that previous algorithms have overlooked, or only partially considered. The first issue is that of the privacy of the agents: while most existing algorithms make use of distributed computation to make sure that the agents do not need to reveal all their private constraints and preferences to a central solver, most of these algorithms still implicitly or explicitly leak a lot of this private information through the messages exchanged by the agents. The research that has been carried out so far on this topic of privacy in DisCSP/DCOP has produced algorithms that can

be classified in two categories. The first category is that of modifications of existing algorithms, which have to be “patched" in order to partially prevent some of their privacy leaks. Unfortunately, in most cases those patches are only partial fixes, and the resulting algorithms still leak private information, often in such a way that cannot be easily predicted: depending on the problem instance, any part of the problem initially known to only one agent may be revealed to one or more other agents with some non-zero probability. The second category contains new, often very different algorithms that have been inspired by research in the field of Cryptography, and that attempt to provide strong guarantees that at least some information about the problem will not be leaked, because it is only communicated to other agents in encrypted form. However, these guarantees usually come at a prohibitive cost in terms of computational and/or communication complexity, and often only apply to some types of information about the problem.

We propose three new algorithms that also make use of known cryptographic techniques, but in such a way that they can scale to larger problems than before, while providing strong privacy guarantees that cover more types of information about the problem. In particular, we identify four types of information that agents might want to protect, and we define four corresponding types of privacy that our algorithms are the first to address simultaneously. The first such type is *agent privacy*, which involves hiding the presence and the identity of each agent from other agents that it does not directly share constraints with. For instance, if a journalist is trying to schedule an interview with the CEO of a company, who simultaneously wants to choose a date for a business lunch with the CEO of another company, then the involvement of the second CEO in the meeting scheduling problem should be hidden from the journalist, who could otherwise then infer that the two companies might have plans to merge. We also define *topology privacy*, which goes one step further, preventing the journalist from discovering the existence of any other meeting in the problem to which she is not invited. More generally, this second type of privacy covers the knowledge about the structure of the problem, in terms of which decision is constrained with which other decision. These first two types of privacy have been largely overlooked in the DisCSP/DCOP literature so far.

The third type is the one that is most often addressed by previous algorithms, and which we call *constraint privacy*; it involves hiding from other agents how must cost or utility a given agent is assigning to various variable-value assignments through the constraints it expresses. For instance, this includes hiding from the journalist the details of her interviewee's availability schedule and preferences. Finally, *decision privacy*, which is more rarely addressed in the literature, deals with only revealing to each agent the part of the solution to the problem that concerns its own decisions, while keeping secret the choices made by the other agents.

Novel Contributions on the Topic of Uncertainty

The second issue that we address in this thesis, and which has been largely ignored in the DisCSP/DCOP literature up until 2009, is the issue of uncertainty in the problem knowledge. In many multi-agent optimization problems, the agents must agree on and commit to decisions before a certain deadline, but the effective costs or utilities that are derived from these decisions actually depend on uncertain, uncontrollable events that may happen after the decisions have been made. For instance, meeting attendees might have to choose a time for a meeting, before they know exactly how much time it might take each attendee to arrive to the meeting location, because this time depends on road traffic or on airplane delays. While such sources of uncertainty are not controllable by the agents, it is often the case that they are partly *predictable*: typically, the agents might have models of the uncertainty that estimate the probabilities of various events, which can be learned from previous experiences or historical data.

In this thesis, we propose a novel extension of the DCOP formalism to include descriptions of such sources of uncertainty; we call this extension *DCOP under Stochastic Uncertainty (StochDCOP)*. In a StochDCOP, in addition to the decision variables that model the decisions that have to be made by the agents, *random variables* are used to model sources of uncertainty, or, in other words, decisions that are made by Nature *after* the agents have made their decisions, and which affect the costs or utilities that are incurred to the agents.

StochDCOP instances typically exhibit two interesting properties, which give rise to two contributions of this thesis. The first property is that a given source of uncertainty often only has a *limited* influence on the overall problem, only affecting a small number of the agents. In this thesis, we study whether it is possible to exploit the locality of uncertainty, in order to produce algorithms that gain in efficiency by localizing the reasoning about uncertainty. The second property is that several agents might express constraints that depend on a common source of uncertainty, but in a different way. Consider for instance a vehicle routing problem in which the location of a customer may be uncertain; the cost for a delivery company to serve the customer will be higher if the customer is further away from its depot, while the same move might take the customer closer to another company's depot, hereby lowering the cost of delivery incurred to that other company. In this thesis, we also investigate whether solutions of higher quality can be obtained if agents explicitly exchange information about how their respective local problems depend on various sources of uncertainty, in order to make better-informed decisions.

Outline of this Thesis

In Chapter 2, we first recall some background knowledge that is necessary to understand our contribution to the state of the art. We formally define the DisCSP and DCOP frameworks, and we illustrate how they can be used to model various examples of multi-agent decision problems. We then briefly review many of the algorithms that have been proposed in the literature to solve DisCSPs and DCOPs, with a special emphasis on the *DPOP* algorithm upon which we base our own new algorithms.

Chapter 3 is the first technical chapter, in which we describe three novel privacy-preserving algorithms, named respectively P-DPOP, $P^{3/2}$-DPOP, and P^2-DPOP. We provide formal proofs of the various privacy guarantees they provide, and we report empirical results of thorough experiments on various classes of problem benchmarks, in which we have compared our algorithms with the previous state of the art in privacy-preserving DisCSP/DCOP.

Our contributions on the second topic of uncertainty are described in Chapter 4. After formally defining our new StochDCOP formalism and illustrating it on various multi-agent decision problems under uncertainty, we propose a family of algorithms, which we call $\mathbb{E}$[DPOP]. We show how these algorithms compare to each other both theoretically, and empirically on new classes of StochDCOP benchmarks.

Finally, in Chapter 5, we describe how all these algorithms (and many others) have been implemented as part of a new software platform, called *FRODO 2*, which has been made available open-source for the benefit of the research community and of potential users of the DisCSP/DCOP technology.

2 Preliminaries

This preliminary chapter presents the general background knowledge that is at the foundation of our work. In particular, Section 2.1 defines the two classes of mathematical problems that are addressed in this thesis: Distributed Constraint Satisfaction Problems (DisCSPs), and their generalization to optimization problems in the form of Distributed Constraint Optimization Problems (DCOPs). Section 2.2 illustrates how these two mathematical formalisms can be used to model a wide variety of multi-agent decision problems. We then survey the different algorithms that have been proposed in the literature to solve DisCSPs and DCOPs, and in particular the DPOP algorithm [Petcu and Faltings, 2005b], on which all algorithms proposed in this thesis are based.

2.1 The DisCSP and DCOP Formalisms

A Distributed Constraint Satisfaction Problem [Yokoo et al., 1992] can be formally defined as follows.

Definition 1 (DisCSP)**.** *A discrete* Distributed Constraint Satisfaction Problem *is a tuple* $< \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C} >$ *such that:*

- $\mathcal{A} = \{a_1, ..., a_k\}$ *is a set of* agents*;*
- $\mathcal{X} = \{x_1, ..., x_n\}$ *is a set of* decision variables*, such that each variable* x_i *is controlled by a given agent* $a(x_i)$*;*
- $\mathcal{D} = \{D_1, ..., D_n\}$ *is a set of finite variable* domains*, variable* x_i *taking values in* D_i*;*
- $\mathcal{C} = \{c_1, ..., c_m\}$ *is a set of* hard constraints*, where each* c_i *is a* $s(c_i)$*-ary function of scope* $(x_{i_1}, \cdots, x_{i_{s(c_i)}})$*,* $c_i : D_{i_1} \times .. \times D_{i_{s(c_i)}} \rightarrow \{\texttt{false}, \texttt{true}\}$*, assigning* `false` *to infeasible tuples, and* `true` *to feasible ones.*

A solution *is a complete assignment such that the conjunction* $\bigwedge_{c_i \in \mathcal{C}} c_i = \texttt{true}$*, which is the case exactly when the assignment is consistent with all constraints.*

Some of the important assumptions of the DisCSP framework are the following. First, we assume that all the details of a given constraint c_i are known to all agents involved; if an agent wants to keep some constraints private, it should formulate them in such a way that they only involve variables it controls. Furthermore, we assume that two neighboring agents (i.e. agents that share at least one constraint) are able to communicate with each other directly, and that messages are delivered in FIFO order and in finite time. On the other hand, we assume that two non-neighboring agents initially ignore everything about each other, even including their involvement in the problem. For clarity and simplicity, we will also assume that a DisCSP is not composed of completely disconnected subproblems; otherwise, each subproblem can be solved independently. In a slight abuse of notation, the agent $a(x)$ owning a variable x will often be referred to simply as x.

A DisCSP is distributed in nature, in two respects:

1. The *decision power* is distributed: each agent is in charge of choosing values for the decision variables it controls;
2. The *knowledge* is distributed: each agent only knows the constraints over the decision variables it controls, and knows all the variables involved in these constraints.

In some problem domains, specifying whether an assignment is feasible or infeasible is not enough: agents might want to express *soft* preferences over the set of feasible solutions. Distributed Constraint Optimization Problems (DCOPs) are a generalization of DisCSPs, in which the constraints associate *costs* to variable assignments, and the goal is to minimize the sum of all costs.

Definition 2 (DCOP)**.** *A discrete* Distributed Constraint Optimization Problem *is defined as a tuple* $< \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C} >$*, such that* $\mathcal{A}$*,* $\mathcal{X}$ *and* $\mathcal{D}$ *are the same as in Definition 1:*

- $\mathcal{A} = \{a_1, ..., a_k\}$ *is a set of* agents*;*
- $\mathcal{X} = \{x_1, ..., x_n\}$ *is a set of* decision variables*, such that each variable* x_i *is controlled by a given agent* $a(x_i)$*;*
- $\mathcal{D} = \{D_1, ..., D_n\}$ *is a set of finite variable* domains*, variable* x_i *taking values in* D_i*;*

and furthermore:

- $\mathcal{C} = \{c_1, \ldots, c_m\}$ *is a set of* soft constraints*, where each constraint is a* $s(c_i)$*-ary function* $c_i : D_{i_1} \times \ldots \times D_{i_{s(c_i)}} \to \mathbb{R} \cup \{+\infty\}$ *that assigns a* cost $c_i(x_{i_1}, \ldots, x_{i_{s(c_i)}})$ *to combinations of assignments to a subset of decision variables; infinite costs correspond to infeasible assignments.*

A solution *to the DCOP is an assignment to all decision variables that minimizes the sum of all costs:*

$$(x_1^*, \ldots, x_n^*) = \arg\min_{\mathcal{X}} \sum_i c_i \, .$$

Notice that Definition 2 is formulated in terms of cost minimization; equivalently, a DCOP can be seen as a maximization problem in which the constraints define rewards, or *utilities*. In fact, within the same given DCOP instance, some constraints may express costs by taking on non-negative values, while other constraints may express utilities by taking on non-positive values. In other words, the DCOP formalism can be used to model multi-agent optimization problems that involve *balancing costs and utilities.*

Not all algorithms that claim to be DCOP algorithms actually apply to this general DCOP definition; in fact, most algorithms (with the noticeable exception of the *DPOP* algorithm in Section 2.4.4) assume that constraints can only take on values in $[0, +\infty]$, because they are designed to exploit *monotonicity* properties. If this is not the case, the DCOP needs to be preliminarily *rescaled* by a sufficiently large constant. Some algorithms additionally assume that constraints can only take on *integer* values, or even values only in $\{0, 1\}$. The following two definitions put names on two particular sub-classes of DCOP that make such restrictive assumptions on the allowed range of cost values.

Definition 3 (DisWCSP)**.** *A Distributed, Weighted CSP is a DCOP in which cost values are restricted to* $\mathbb{N} \cup \{+\infty\}$ *(non-negative unbounded integers).*

Definition 4 (Max-DisCSP)**.** *A Max-DisCSP is a DCOP in which cost values are restricted to* $\{0, 1\}$*. A Max-DisCSP can also be seen as the relaxation of a DisCSP in which the solution is not necessarily feasible, but minimizes the number of constraint violations.*

2.2 Examples of Applications

The DisCSP and DCOP formalisms are abstract, very general mathematical problems; they can be used to describe a wide variety of problems in which multiple agents need to coordinate their decisions. This section presents a number of such problem classes that can be formulated as DisCSPs and/or DCOPs.

2.2.1 Graph Coloring

The *graph coloring* problem is often presented as the problem of assigning colors to countries on a world map, such that no two neighboring countries have the same color. In fact, many multi-agent decision problems can been seen as distributed graph coloring problems, in which each agent needs to pick a color different from those of its neighboring agents. An example of a concrete application is the allocation of time schedules in a Time Division Multiple Access (TDMA) scheme for an ad-hoc wireless network.

Figure 2.1 introduces a simple graph coloring problem instance that will be used to illustrate the algorithms presented in the next chapter. We assume that the five nodes in the graph correspond to five different agents, which must each choose a color among *red*, *blue* and *green* (noted R, B and G respectively). These decisions are modeled by the five DisCSP variables $x_1, \ldots, x_5$ with common domain $\{R, B, G\}$. Each agent may express a *unary* constraint on its variable; for instance, x_1 does not want to be assigned the color *red*. Binary, inequality constraints are imposed between each pair of neighboring nodes.

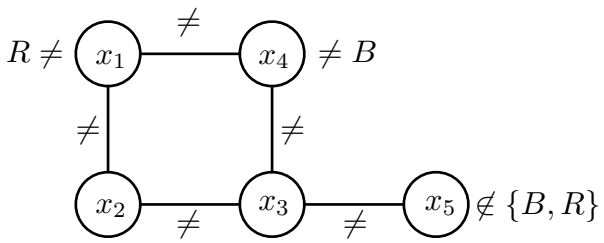


Figure 2.1: The constraint graph for a simple graph coloring problem.

An optimization variant of the graph coloring problem can be produced by replacing all hard constraints by soft constraints in which a cost of 0 indicates feasibility, and a constraint violation corresponds to a cost of 1. The problem then becomes a Max-DisCSP, for which the optimal solution is the one that minimizes the number of pairs of neighboring nodes that have been assigned the same color.

2.2.2 Meeting Scheduling

Another class of multi-agent decision problems that can be modeled as DisCSPs is multi-agent meeting scheduling, defined informally as follows. Several participants want to schedule meetings with each other. Each meeting involves a subset of the participants. Each participant has her own, private availabilities and preferences regarding her meetings: typically, a participant cannot be attending two meetings at the same time, and she may have additional constraints regarding the order of the

meetings, or unavailabilities imposed by pre-existing meetings. The final goal is to find a feasible time slot for each meeting.

Various DisCSP formulations for this problem class have been proposed and studied in [Maheswaran et al., 2004b]. The *PEAV* formulation (*Private Events As Variables*) is the one that has the nicest privacy properties, and proceeds as follows. Each participant X owns one variable m_i^X per meeting m_i she has to attend, and expresses her availabilities and preferences regarding her meetings as an intra-agent constraint c^X over these variables. For each meeting, inter-agent equality constraints enforce that all attendees agree on the time for the meeting. Figure 2.2 illustrates the constraint graph of the resulting DisCSP for a three-agent, three-meeting problem instance in which Agents A and B want to schedule a meeting m_1 between each other, B and C want to schedule m_2, and A and C, m_3.

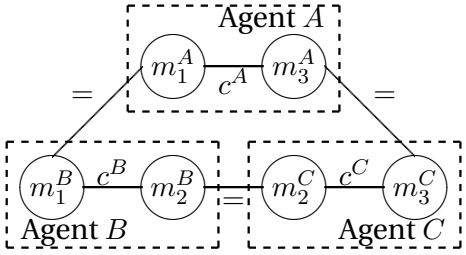


Figure 2.2: The constraint graph for a simple meeting scheduling problem.

2.2.3 Sensor Networks

There is a large amount of previous work on applying DisCSP and DCOP to various sensor network problems; see [Vinyals et al., 2011a] for a survey of general Multi-Agent Systems techniques that have been applied to this domain. Most of the DisCSP/DCOP literature considered problems in which sensors are directional, and must decide in which direction to point in order to track a number of targets. In [Zhang et al., 2005], one sensor is assumed to be enough to track any given target, and the authors cast the problem into a graph coloring problem, because sensors must coordinate so that each area under surveillance is only tracked by one sensor. Such a casting can no longer be done if several sensors are necessary to track a target; for instance, in [Béjar et al., 2005], three sensors are needed, and the authors model the problem as a DisCSP in which the agents are the targets, which is not necessarily very realistic. In contrast, in [Modi et al., 2001; Maheswaran et al., 2004b; Zivan et al., 2009], the agents are the sensors, which must coordinate in order to track targets with unknown and changing positions.

In this section, we describe a slightly different problem, in which sensors are omnidirectional, and target positions are known. Section 4.2 then describes an extension of

this problem, in which the targets' initial positions are known, but the targets can move in various directions with known probability distributions. The goal for this variant of the problem is not so much realism, but rather didacticism in explaining the properties of the algorithms presented in Chapter 4.

Problem Definition

Consider a square grid of sensors, whose task is to observe a number of targets with known, constant positions. Each sensor is omnidirectional, and can observe an unlimited number of targets, but has a very limited view of the world: it is initially only aware of the existence of, and can only communicate directly with its eight closest neighboring sensors. It also has a limited sensing range, such that any given target is only visible to its 4 closest neighboring sensors. This is illustrated in Figure 2.3, in which sensor x_2^2's limited view of the world is highlighted in grey and bold.

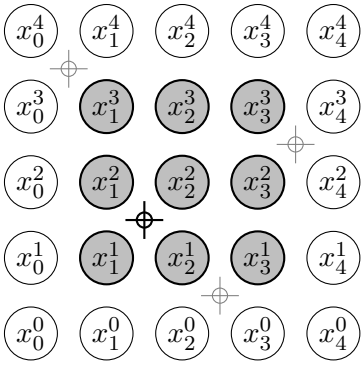


Figure 2.3: Sample sensor network problem.

The goal of the distributed decision problem is for each sensor to decide whether it should turn itself *on* or *off*, in order to maximize all sensors' overall utility gained by observing the targets. This utility is defined as the sum of two competing terms:

- Each sensor has to pay a cost of 1 (i.e. receives a utility of -1) when it is *on*;
- For each target, if 3 or more neighboring sensors are *on*, the sensors share a utility of 12. If only 2 sensors are observing the target, this utility drops to 6, and down to 1 if only one sensor observes it.

Formulation as a DCOP

We propose the following DCOP formulation for the sensor network problem, illustrated in Figure 2.4. The set of agents $\mathcal{A}$ is the set of all sensors that are in view of at least one target. For any sensor that cannot observe any target, its optimal decision is

obviously to turn itself *off*, and therefore it does not need to participate in the DCOP. Each sensor agent in position (i, j) owns a single binary variable $x_i^j \in \{0, 1\}$, where $x_i^j = 1$ if and only if the sensor is *on*. The constraints are the following:

- One unary soft constraint per sensor (i, j), expressing the negative utility (i.e. positive cost) incurred by turning the sensor *on*:

$$u_{i,j} : x_i^j \to -x_i^j \; ; \tag{2.1}$$

- One 4-ary soft constraint per target k, modeling the utility gained by observing the target, as a function of the number of neighboring *on* sensors involved:

$$u_k : (x_{i_k}^{j_k}, x_{i_k}^{j_k+1}, x_{i_k+1}^{j_k}, x_{i_k+1}^{j_k+1}) \to \begin{cases} 0 & \text{if} \quad \sum x_i^j = 0 \\ 1 & \text{if} \quad \sum x_i^j = 1 \\ 6 & \text{if} \quad \sum x_i^j = 2 \\ 12 & \text{if} \quad \sum x_i^j \geq 3 \end{cases} . \tag{2.2}$$

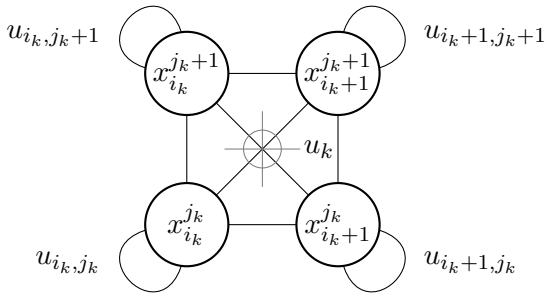


Figure 2.4: Constraint graph for target k (in the middle).

2.2.4 Resource Allocation

A fair amount of research has been carried out on the topic of Multi-Agent Resource Allocation [Chevaleyre et al., 2006]; however, much of the previous work that claims to apply DisCSP or DCOP to resource allocation problems actually talk about meeting scheduling [Maheswaran et al., 2004b] or sensor network problems [Modi et al., 2001], in which the “resources" are the agents themselves. Arguably, from the point of view of the agents (which, in the end, are the users of the system), these are not really resource allocation problems, but rather scheduling and task allocation problems, respectively. In this section, we are interested in problems in which the agents are

the resource *consumers*, and must coordinate their access to the limited-capacity resources. The approach to sensor networks described in [Béjar et al., 2005] is closer to this problem class, since the agents are the targets, which request access to a number of resources (the sensors); however, their formulation as a DisCSP is quite specific to their *SensorDCSP* domain. Besides the graph coloring problem, which can be seen as a special case of a resource allocation problem in which the resources are the colors, we are not aware of any previous work (apart from our own; see Section 3.7.3) that applied DisCSP to address general resource allocation problems, in which the agents may request combinations of resources.

However, there is some sparse previous work on applying DCOP to the optimization variant of the resource allocation problem, which corresponds to winner determination in Combinatorial Auctions (CAs) [Parsons et al., 2011]. In CAs, the agents are bidders who express utilities over being assigned bundles of items. While CAs have been the subject of a considerable amount of work, most of the algorithms proposed are specific to auctions, and very little research has actually been carried out that formalized the winner determination problem as a DCOP. We are only aware of two exceptions: Silaghi [2005b] proposed a DCOP formulation, but used different assumptions about the knowledge of agents, assuming that variables are public and constraints are private (see Section 3.2.2 for a discussion of these different assumptions). Finally, Petcu [2007] proposed a DCOP formulation in which each agent owns one variable per item it is interested in, and the domain for the variable is the list of bidders who are interested in the item. In Section 3.7.3, we propose a novel formulation that has better privacy properties.

2.2.5 Distributed Vehicle Routing Problems

Vehicle Routing Problems (VRPs) [Toth and Vigo, 2001] are a class of problems in which customers want to be delivered certain quantities of goods that are stored in a depot. A fleet of vehicles with limited capacities is available to deliver the goods; the problem is that of optimally routing the vehicles so as to serve all customers at minimal cost, where the cost is often taken as the sum of the vehicles' route lengths. Multiple variants of the VRP have been proposed and studied in the literature [Gendreau et al., 2007]; here we consider a variant involving multiple depots, each controlled by a separate agent, and in which each customer's demand can be split among multiple depots.

Definition 5 (DisSDMDVRP)**.** *The* Distributed, Split-Delivery, Multiple-Depot, Vehicle Routing Problem *is defined as a tuple* $< D, n_V, Q_{\max}, L_{\max}, H, C, Q >$*, where:*

- $D = \{d_1, \ldots, d_{n_D}\}$ *is a set of depots in the Euclidian plane, each controlled by a different delivery company;*
- n_V *is the number of vehicles at each depot;*

- $Q_{\max}$ *and* $L_{\max}$ *are respectively the maximum load and maximum route length of any vehicle;*
- $H \leq L_{\max}/2$ *is the visibility horizon of each company, which defines the boundaries of its knowledge of the overall problem. Depot* d_i *is only aware of and can only serve the customers that are within distance* H*;*
- $C = \{c^1, \ldots, c^{n_C}\}$ *is a set of customers with known locations;*
- $Q = \{q^1, \ldots, q^{n_C}\} \in \mathbb{N}^{n_C}$ *are the customer demands, which can be split among multiple companies.*

The goal is for the companies to agree on who should serve which customers, using which vehicle routes, so as to fully serve all visible customers, at minimal total route length.

Figure 2.5 shows a DisSDMDVRP instance with 4 depots and 4 customers. Each depot is at distance 1 of its two closest customers, and owns $n_V = 2$ vehicles, with $Q_{\max} = 1$ and $L_{\max} = 2.5$, so that a vehicle can only serve one customer, of common demand $q = 1$.

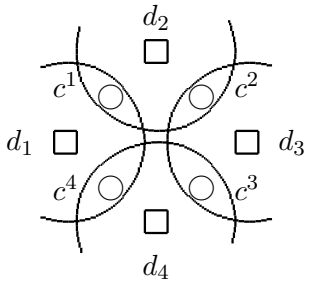


Figure 2.5: A simple DisSDMDVRP instance.

The visibility radius is $H = 1.25$, such that, for instance, depot d_1 only knows customers c^1 and c^4, and ignores the existence of c^2 and c^3. It also knows the two companies d_2 and d_4, and will have to coordinate with each of them to decide who serves c^1 and c^4, but d_2 and d_4 do not know about each other, and neither do d_1 and d_3.

In [Léauté and Faltings, 2011a], we have shown how a DisSDMDVRP can be reformulated as a DCOP, with one agent per company/depot. Depot d_i owns one integer variable $x_i^j \in [0, q^j]$ for each visible customer c^j, modeling how much of the customer's demand q^j it serves. Variables need not be created for customers that can only be served by a single depot, since these variables would necessarily take their maximum values. The DCOP constraints are of two types:

1. For each customer c^j, if the set D^j of depots within visible distance H of c^j is non-empty, then the following $|D^j|$-ary hard constraint enforces that c^j's demand must be fully served:
$$\sum_{d_i \in D^j} x_i^j = q^j \ .$$

2. For each depot d_i, if the set $C_i = \{c^{j_1}, \ldots, c^{j_n}\}$ of visible customers is non-empty, then the following $|C_i|$-ary soft constraint represents the cost of the optimal solution to d_i's own VRP, as a function of how much of the request of each customer in C_i it serves:
$$vrp_i(x_i^{j_1}, \ldots, x_i^{j_n}) = \text{optimal cost of } d_i\text{'s VRP} \in [0, \infty] \ .$$

Notice that evaluating a constraint vrp_i on a given assignment of values to its variables requires solving a (centralized) single-depot VRP. Such an evaluation can been seen as a call by agent d_i to a local subroutine, which does not require exchanging additional information with other agents, and can be implemented using any existing VRP algorithm. This modeling approach decouples the distributed, master problem of assigning customers to agents, solved using a DCOP algorithm, from the local, slave problem of routing a given agent's vehicles, solved using a traditional VRP algorithm.

The DCOP model for the DisSDMDVRP instance in Figure 2.5 is illustrated in Figure 2.6. In this example, each agent owns two variables, whose respective values model the quantities of goods the company is going to deliver to the two customers in its area of visibility. Internally, each agent has to solve a VRP problem, represented by a vrp_i constraint over its variables. Inter-agent *sum* constraints enforce that each customer's demand must be fully served.

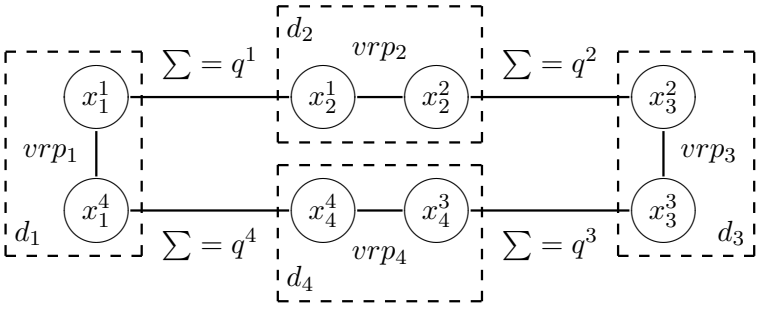


Figure 2.6: The DCOP constraint graph corresponding to Figure 2.5.

2.2.6 Equilibria in Graphical Games

Graphical games are strategic games in which each player's reward only depends on the actions of a limited subset of the players, called *neighbors*. The game can be

seen as being composed of coupled, *local* games played by subsets of players. The neighborhood relationships in the game can be represented by a graph, in which nodes are players, and edges represent the influences of agents' actions on other agents' rewards/utilities. Several formal definitions can be found in the literature; Mura [2000] proposed the concept of *Game Networks (G nets)*; we adopt the following definition from [Kearns et al., 2001]:

Definition 6 (graphical game)**.** *An n-player* graphical game *can be defined as a tuple (G, S, U) such that:*

- *G is an undirected graph in which each vertex v is a player, and edges define the player's neighborhood N_v;*
- *S is a finite set of strategies, assumed to be the same for all players for simplicity;*
- *$U = \{u_1, \ldots, u_n\}$ defines the utility u_i of each player i, as a function of the strategies of i and her neighbors: $u_i : S^{|N_i \cup \{i\}|} \to \mathbb{R}$.*

The solution of a game is typically a choice of strategies for the players such that they play at some form of desirable *equilibrium*. Two types of equilibria are usually considered. A *pure-strategies Nash equilibrium (NE)* is a choice of a strategy s^i for each player i, such that s^i is a *best response* to her neighbors $n_i \in N_i$ playing their respective best responses s^{n_i}:

$$\forall s \in S \quad u_i(s^{n_i \in N_i}, s^i) \geq u_i(s^{n_i \in N_i}, s) \; . \tag{2.3}$$

While a pure-strategy equilibrium might not always exist, there always exists a *mixed-strategies equilibrium* [Nash, 1950], where a mixed strategy σ for agent i is a probability distribution over the pure strategy space S. An ϵ-approximation of such a *mixed Nash equilibrium (mNE)* is defined by each player i choosing a mixed strategy σ^i that is an ϵ-best response in expectation, assuming that her neighbors do the same:

$$\forall \sigma \quad \mathbb{E}\left[u_i(\sigma^{n_i \in N_i}, \sigma^i)\right] + \epsilon \geq \mathbb{E}\left[u_i(\sigma^{n_i \in N_i}, \sigma)\right] \; . \tag{2.4}$$

Such ϵ-mNEs can be obtained by first discretizing the probability space $[0, 1]$ into $\{0, \tau, 2\tau, \ldots, 1\}$, where τ is the *discretization parameter*. Soni et al. [2007] proposed a formula for this parameter τ as a function of ϵ that guarantees the existence of an ϵ-mNE in the discretized problem.

If the players can play the game repeatedly, choosing in turn the best-response strategies that maximize utility based on their respective neighbors' current strategies, then the game should converge to an equilibrium. However, in the case of one-shot,

simultaneous-move games, it is not possible to repeat the game until an equilibrium is reached. It can then make sense for the agents to preliminarily compute such an equilibrium before they choose their respective strategies.

To illustrate this problem, let us consider a particular example of a graphical game. In the *party game*[Singh et al., 2004], the players are invited to a common party and need to decide whether to attend ($S = \{0, 1\}$). The graph G is a social interaction graph between players, such that the neighbors N_i of player i are her acquaintances. Player i's cost function is defined as follows:

$$-u_i = \begin{cases} a_i + \Sigma_{n \in N_i^{s=1}} l_n & \text{if } i \text{ attends} \\ 0 & \text{else} \end{cases} \tag{2.5}$$

where $a_i \in [-1, 1]$ is the cost that the player assigns to attending the party, $N_i^{s=1}$ is the subset of i's neighbors that attend, and $l_n \in \{-1, 1\}$ describes whether player i likes ($l_n = -1$) or dislikes ($l_n = 1$) her neighbor n.

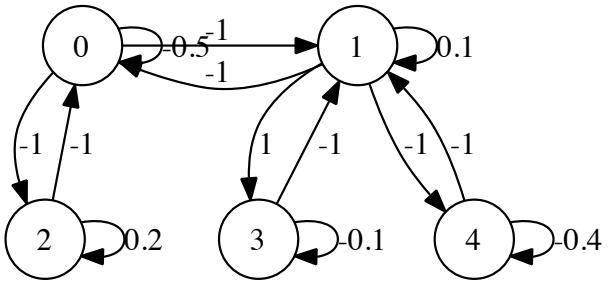


Figure 2.7: An instance of the party game.

Instances of the party game can be illustrated as labelled, bi-directional graphs (Figure 2.7), in which the nodes are the players (like in the underlying graph G), and each edge corresponds to one direction of an edge in the underlying graph G, and is labelled by the cost incurred to the tail by the attendance of the head (the case head = tail corresponds to a_i). A pure Nash equilibrium for the game in Figure 2.7 involves all players deciding to attend the party. This can be seen as follows. Players 3 and 4's preferred strategy is to attend the party, because they both assign a negative cost (i.e. a positive utility) to attending, and they both like all their respective neighbors (i.e. player 1); therefore they are better off attending, regardless of whether player 1 attends. On the other hand, player 1 likes player 4 but dislikes player 3; since they are both attending, their respective influences on player 1's preferred strategy cancel each other out. Player 1 also assigns a positive cost to attending the party, so she will only want to attend if player 0, which she likes, also decides to attend. Player 0, on the other hand, assigns a negative cost to attending, and likes all her neighbors; therefore she will

attend, and so will player 1. Finally, player 2 will also attend since player 0's attendance compensates the positive cost that she assigns to attending.

There can be multiple ways to formalize the problem of finding an equilibrium to a graphical game as a DisCSP. In the most natural DisCSP formulation, proposed in [Vickrey and Koller, 2002], each agent/player owns one single variable, representing the player's strategy. Each player expresses one constraint over her variable and those of her neighbors, enforcing that her strategy must be a best response to her neighbors' strategies (Equation (2.3) or (2.4)). Soni et al. [2007] proposed a dual DisCSP formulation, in which each agent's single variable represents the *joint* strategies of her whole neighborhood in the graph, including herself. In Section 3.7.4, we describe a novel DisCSP formulation that has better privacy properties.

2.3 Complete DisCSP Algorithms

Historically, the first algorithms proposed were DisCSP algorithms that were complete, i.e. were guaranteed to find a feasible solution when there exists one.

2.3.1 ABT: Asynchronous Backtracking

The very first algorithm was *Asynchronous Backtracking (ABT)* [Yokoo et al., 1992], which is a distributed, asynchronous adaptation of the traditional *backtracking search* that has been applied successfully to centralized CSPs. In ABT, each variable $x \in \mathcal{X}$ is assigned a fixed *priority* $p(x) \in \mathbb{N}$, which defines a *total order* on the variables. Each agent $a(x)$ is then responsible for enforcing the constraints that involve x and *only* higher-priority neighbors (Algorithm 1, line 2). Each agent performs assignments to its variables concurrently and asynchronously (lines 4 and 15), notifying its lower-priority neighbors of its decisions using OK? messages (lines 5 and 16).

Lower-priority agents that receive these OK? messages update their *agent views* of the current assignments to their higher-priority neighbors (line 9), and attempt to find assignments to their own variables that satisfy the constraints they share with higher-priority neighbors, based on their agent views (lines 13 to 16). If no such feasible assignment is found (line 17), the agent initiates a backtrack, by responding with NOGOOD messages that contain reasons for the backtrack. Each such *nogood*, i.e. inconsistent partial assignment to higher-priority variables (line 19), is sent to the lowest-priority variable involved (lines 21 to 23), except if the empty nogood is generated, in which case the DisCSP has been found infeasible (line 20). When it receives a NOGOOD message, an agent converts the nogood into a new constraint that it must enforce (lines 10 to 12).

On feasible DisCSP instances, the algorithm terminates when all agents are idle, which

Algorithm 1 The ABT algorithm for variable x.

Require: a fixed, injective, priority function $p : \mathcal{X} \to \mathbb{N}$ of the variables

1: // Join all constraints involving x and only higher-priority variables:
2: $c_x(x,\cdot) \leftarrow \bigwedge_{c \in \{c' \in \mathcal{C} \mid p(x)=\min_{y \in scope(c')} p(y)\}} c(x,\cdot)$
3: // Initialize value for x:
4: $x \leftarrow$ value x^* from x's domain D_x chosen according to some heuristic
5: Send message (OK?, $x = x^*$) to all lower-priority neighbors $\in N_x$
6: **loop**
7: Wait for a message M
8: **if** $M =$ (OK?, $y = y^*$) **then**
9: $y \leftarrow y^*$
10: **else if** $M =$ (NOGOOD, $\{y_1 = y_1^c, \ldots, y_k = y_k^c\}$) **then**
11: $c_x(x,\cdot) \leftarrow c_x(x,\cdot) \wedge (y_1 \neq y_1^c \vee \ldots \vee y_k \neq y_k^c)$ // record the nogood
12: $N_x \leftarrow N_x \cup \{y_1, \ldots, y_k\}$ // add links if they do not already exist
13: **if** $c_x(x,\cdot) =$ `false` **then** // conflict; try to find a new consistent value for x
14: **if** $\exists x^* \in D_x \mid c_x(x^*,\cdot) =$ `true` **then**
15: $x \leftarrow x^* \in D_x$ chosen according to some heuristic
16: Send message (OK?, $x = x^*$) to all lower-priority neighbors $\in N_x$
17: **else** // backtrack
18: // All subsets of higher-priority variables with inconsistent values:
19: $\mathcal{N}_x^c \leftarrow \{\{y_1, \ldots, y_k\} \subset scope(c_x) \backslash \{x\} \mid c_x(y_1 = y_1^*, \ldots, y_k = y_k^*) \equiv$ `false`$\}$
20: **if** $\emptyset \in \mathcal{N}_x^c$ **then** broadcast message (INFEASIBLE) and **exit**
21: **for** $\{y_1, \ldots, y_k\} \in \mathcal{N}_x^c$ **do**
22: $y_{\min} \leftarrow \arg\min_{y=y_1,\ldots,y_k} p(y)$ // lowest-priority variable
23: Send message (NOGOOD, $\{y_1 = y_1^*, \ldots, y_k = y_k^*\}$) to $y_{\min}$

has to be detected by a separate algorithm such as the one in [Chandy and Lamport, 1985]. The processing of nogoods can require *adding links* to the constraint graph (line 12); some later work has looked into removing this requirement, such as in the *Distributed Dynamic Backtracking (DisDB)* [Bessiere et al., 2001], *ABT$_{not}$* [Bessiere et al., 2005] and *Distributed Backtracking with Sessions (DBS)* [Monier et al., 2009] algorithms. The *Distributed Forward Checking (DisFC)* algorithm [Brito and Meseguer, 2003] is another later variant of ABT that addresses the issue of privacy; privacy-related discussions are postponed to Section 3.2.2.

2.3.2 AWC: Asynchronous Weak Commitment

Asynchronous Weak Commitment (AWC) [Yokoo, 1995] introduces two performance improvements to ABT. First, when an agent has a choice between multiple feasible values for its variable (Algorithm 1, line 15), it chooses the one that satisfies as many constraints with lower-priority agents as possible. In other words, the value ordering

heuristic used in AWC is the *min-conflict* heuristic.

Second, each time a backtrack is initiated, the total variable order is revised so as to assign the highest priority to the initiator variable following Algorithm 2, which replaces lines 19 to 23 in Algorithm 1. This is achieved by allowing the priority values $p : \mathcal{X} \rightarrow \mathbb{N}$ to change dynamically; the updated values of p are communicated via the OK? messages. This dynamic variable ordering mechanism makes it possible to quickly revise a "hopeless" assignment to a high-priority variable, while ABT would only revise it after having formally proven that it is hopeless (i.e. it cannot lead to any feasible solution), which can require an exponential amount of computation.

Algorithm 2 Backtracking in AWC.

1: $\mathcal{NG}_x^c \leftarrow \{\{y_1 = y_1^*, \ldots, y_k = y_k^*\} \subset scope(c_x) \backslash \{x\} \mid c_x(y_1 = y_1^*, \ldots, y_k = y_k^*) \equiv \texttt{false}\}$
2: **if** $\emptyset \in \mathcal{NG}_x^c$ **then** broadcast message (INFEASIBLE) and **exit**
3: **if** $\mathcal{NG}_x^c \cap \mathcal{NG}_x^{sent} = \emptyset$ **then**
4: $\quad \mathcal{NG}_x^{sent} \leftarrow \mathcal{NG}_x^{sent} \cup \mathcal{NG}_x^c$
5: $\quad$ **for** $nogood \in \mathcal{NG}_x^c$ **do** send message (NOGOOD, $nogood$) to all $y_i \in nogood$
6: $\quad p(x) \leftarrow 1 + \max_y p(y)$
7: $\quad x \leftarrow x^*$ chosen according to the *min-conflict* heuristic
8: $\quad$ Send message (OK?, $x = x^*, p(x)$) to all lower-priority neighbors $\in N_x$

While AWC was shown to outperform ABT empirically on small problems, contrary to ABT, the AWC algorithm must record all nogoods (Algorithm 2, line 4), which results in a worst-case exponential space complexity. Silaghi et al. [2001] later proposed a modification to ABT, called *ABT with asynchronous Reordering (ABTR)*, which implements a reordering mechanism comparable to that of AWC, but with polynomial space complexity.

2.3.3 AAS: Asynchronous Aggregation Search

Departing from previous conventions, Silaghi et al. [2000] proposed to look at a DisCSP variant in which the variables are public, and the constraints are private (see Section 3.2.2), and introduced the *Asynchronous Aggregation Search (AAS)* algorithm. This algorithm proceeds as ABT, except that instead of reasoning about assignments of single values to single variables in the form $x = x_0$, AAS reasons about *aggregates*, which are assignments of multiple values to multiple variables, in the form $(x, y) \in \{x_0, x_1\} \bigotimes \{y_0, y_1\}$. To detect termination, AAS uses a special *system agent* that is responsible for keeping track of whether the proposals contained in OK? messages have been accepted or not. Silaghi et al. investigated three variants of AAS that differ in the number of nogoods they record; the version that records all nogoods (AAS-2) was shown to outperform ABT, at a cost of a worst-case exponential memory complexity.

Algorithm 3 The AFC algorithm for variable x_i.

Require: a fixed, known, linear order on the variables: $x_1 \succ \ldots \succ x_n$

// Join all constraints involving x_i and only previous variables:
$c_i(x_i, \cdot) \leftarrow \bigwedge_{c \in \{c' \in \mathcal{C} \mid x_i \in scope(c') \wedge \forall y \in scope(c'),\, y \succ x_i\}} c(x_i, \cdot)$
$t_i \leftarrow 0$ // timestamp
$ok_i \leftarrow$ `true` // whether my current agent view is consistent
$D'_i \leftarrow D_i$ // a copy of x_i's domain D_i
if $i = 1$ **then** // the first variable x_1 starts the search
    $x_1 \leftarrow$ choose $x_1^* \in D'_1$
    Send messages (CPA, $t_1, (x_1^*)$) to x_2 and (FC, $t_1, (x_1^*)$) to all $x_{i>1}$
for each received message M **do**
    if $M =$ (CPA **or** BACK, $t, (x_1^*, \ldots, x_{i-1}^*)$) **then** // look for a value for x_i
        if $ok_i =$ `false` **then** // my current agent view is inconsistent
            if $x_1 = x_1^* \wedge \ldots \wedge x_{i-1} = x_{i-1}^*$ **then** BACKTRACK() **and continue**
            else $ok_i \leftarrow$ `true`
        if M is of type BACK **then** $D'_i \leftarrow D'_i \backslash \{x_i^*\}$ // $x_i = x_i^*$ is inconsistent
        else $(x_1, \ldots, x_{i-1}) \leftarrow (x_1^*, \ldots, x_{i-1}^*)$ **and** $D'_i \leftarrow D_i$
        ASSIGNCPA()
    else if $M =$ (FC, $t > t_i, (x_1^*, \ldots, x_k^*)$) **then** // perform forward checking
        $D'_i \leftarrow D_i$
        if $ok_i =$ `false` **and** $x_1 \neq x_1^* \vee \ldots \vee x_k \neq x_k^*$ **then** $ok_i \leftarrow$ `true`
        if $ok_i =$ `true` **then**
            $(x_1, \ldots, x_k) \leftarrow (x_1^*, \ldots, x_k^*)$ **and** $D'_i \leftarrow D'_i \backslash \{x_i^* \in D'_i \mid c_i(x_i^*) \equiv$ `false`$\}$
            if $D'_i = \emptyset$ **then** send message (NOT_OK, $t_i, (x_1^*, \ldots, x_k^*)$) to all $x_{j>k}$
    else if $M =$ (NOT_OK, $t, (x_1^*, \ldots, x_k^*)$) **then** // failed forward checking
        if $x_1 = x_1^* \wedge \ldots \wedge x_k = x_k^*$ **or** $t > t_i$ **then**
            $(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n) \leftarrow (x_1^*, \ldots, x_k^*, \bot, \ldots, \bot)$ **and** $ok_i \leftarrow$ `false`
    $t_i \leftarrow \max\{t_i, t\}$ // keep timestamp up to date

Procedure: BACKTRACK()
if $i > 1$ **then**
    $ok_i \leftarrow$ `false` **and** $j_{\min} \leftarrow \min\{j \in [1, n] \mid c_i(x_1^*, \ldots, x_j^*, \bot, \ldots \bot) \equiv$ `false`$\}$
    $(x_1, \ldots, x_{j_{\min}}, x_{j_{\min}+1}, \ldots, x_n) \leftarrow (x_1^*, \ldots, x_{j_{\min}}^*, \bot, \ldots, \bot)$
    Send message (BACK, $t_j, (x_1^*, \ldots, x_{j_{\min}-1}^*)$) to $x_{j_{\min}}$
else broadcast message (INFEASIBLE) **and exit**

Procedure: ASSIGNCPA()
$x_i \leftarrow x_i^* \in D'_i$ such that $c_i(x_i^*, \cdot) =$ `true`
if there exists such a x_i^* **then**
    if $i < n$ **then**
        $t_i \leftarrow t_i + 1$
        Send (CPA, $t_i, (x_1^*, \ldots, x_i^*)$) to x_{i+1} and (FC, $t_i, (x_1^*, \ldots, x_i^*)$) to all $x_{j>i}$
    else broadcast message (SOLUTION, $x_1^*, \ldots, x_n^*$) **and exit**
else BACKTRACK()

2.3.4 AFC: Asynchronous Forward Checking

All algorithms mentioned so far were fully asynchronous algorithms; *Asynchronous Forward Checking (AFC)* [Meisels and Zivan, 2003, 2007] is, contrary to what its name might suggest, the first successful complete DisCSP algorithm that performs *synchronous search*; only the *forward checking* computation is done asynchronously. Given a total ordering of variables, the first variable starts the synchronous search by choosing a value for itself, and communicates this tentative decision to the second variable via a CPA (*Current Partial Assignment*) message (Algorithm 3, lines 6 to 8). It also sends the current CPA to all other subsequent variables via an FC (*Forward Checking*) message.

These FC messages tell all other agents to temporarily remove from their respective variables' domains the values that have already become inconsistent, even if not all previous variables have been instantiated yet (line 21). If this results in a variable's domain becoming empty, the agent notifies all unassigned variables of the inconsistency via NOT_OK messages (line 22). When receiving such a notification (lines 23 to 26), the agents record the fact that their current agent views are inconsistent, unless the NOT_OK message is obsolete and refers to a branch of the search tree that has already been abandoned; obsolescence is detected using a timestamp mechanism.

The single agent that receives the current CPA message extends it with a consistent assignment to its own variable (lines 15 to 16), unless no such consistent assignment exists (line 38) or the agent had previously been notified by a NOT_OK message that the current CPA will eventually become inconsistent (lines 11 to 12). In these two cases, the agent initiates a backtrack by sending a BACK message to the last variable in the smallest inconsistent partial assignment (lines 27 to 30), effectively performing *backjumping*. AFC was shown to outperform ABT and DisDB. Ezzahir et al. [2009] later proposed two performance improvements to AFC: *NoGood-based AFC (AFC-ng)*, and *Asynchronous Inter-Level Forward-Checking (AILFC)*, which runs AFC-ng on a *pseudo-tree* ordering of the variables (see Section 2.4.2).

2.3.5 APO: Asynchronous Partial Overlay

An unusual member in the family of DisCSP algorithms is the *Asynchronous Partial Overlay (APO)* algorithm [Mailler and Lesser, 2003]. In this algorithm, some agents are designated as *mediators*, which are made responsible for partially centralizing sub-problems of the full DisCSP. Grinshpoun and Meisels [2008] later showed that the algorithm was actually incomplete, and that its complete version, *CompAPO*, had a tendency to centralize most of the DisCSP into one mediator, and was empirically clearly outperformed by ABT and AFC.

2.3.6 ConcDB: Concurrent Dynamic Backtracking

Also departing from previous work on DisCSP, Zivan and Meisels [2004, 2006] proposed to exploit the inherent parallel computing capabilities of the agents by running multiple search processes in parallel, on non-intersecting sub-problems of the overall DisCSP. They showed that their resulting *Concurrent Dynamic Backtracking (ConcDB)* algorithm outperformed ABT.

2.3.7 MPC-DisCSP4: Secure Multiparty Computation

An even more exotic DisCSP algorithm is MPC-DisCSP4 [Silaghi, 2005a], which, like AAS, assumes public variables and private constraints, and then applies *secure Multi-Party Computation (MPC)* to solve the DisCSP, while providing privacy guarantees. There also exists an extension of this algorithm to DisWCSPs with bounded costs, in the form of *MPC-DisWCSP4* [Silaghi and Mitra, 2004; Silaghi, 2005a]. The discussion of MPC-Dis(W)CSP4 is delayed to Section 3.2.2.

2.4 Complete DCOP Algorithms

This section now presents complete algorithms that have been proposed in the literature to solve DCOPs.

2.4.1 SynchBB: Synchronous Branch and Bound

Probably the simplest complete DCOP algorithm that exists is *Synchronous Branch and Bound (SynchBB*[1]*)* [Hirayama and Yokoo, 1997], which is a straightforward distributed adaptation of the well-known centralized *Branch and Bound* mechanism. Originally a Max-DisCSP algorithm, Meisels [2008] showed how it can be generalized to a DCOP algorithm (still assuming non-negative costs), and how the resulting Algorithm 4 compares with the (later) AFC algorithm. Like in AFC, the variables are arranged along a total order, and the agents pass around a single CPA message (which Hirayama and Yokoo called a *PATH* message) that includes the current partial assignment to the highest-priority variables, and, additionally, the current associated cost (line 14). When a full assignment has been generated by the last agent (line 23), its corresponding total cost is broadcasted to all other agents (line 24), which can use this cost as an upper bound on the optimal cost (lines 5 and 12), so as to trigger a backtrack whenever a partial assignment is found to be sub-optimal (lines 18 and 30).

[1] Hirayama and Yokoo initially called their algorithm *SBB*, but almost all recent literature consistently refers to it as *SynchBB*.

Algorithm 4 The SynchBB algorithm for variable x_i.

Require: a fixed, known, linear order on the variables: $x_1 \succ \ldots \succ x_n$

// Join all constraints involving x_i and only previous variables:
$c_i(x_i, \cdot) \leftarrow \bigwedge_{c \in \{c' \in \mathcal{C} \mid x_i \in scope(c') \wedge \forall y \in scope(c'),\, y \succ x_i\}} c(x_i, \cdot)$
$D'_i \leftarrow D_i$ // a copy of x_i's domain D_i
$c_i^- \leftarrow 0$ // cost of the CPA up to and including x_{i-1}
$c^* \leftarrow \infty$ // cost of the best solution found so far
if $i = 1$ **then** // the first variable x_1 starts the search
    $x_1 \leftarrow$ first $x_1^* \in D'_1$ such that $c_1(x_1^*) < \infty$
    if there exists such a x_1^* **then** send message (CPA, $(x_1^*), c_1(x_1^*)$) to x_2
    else broadcast message (INFEASIBLE)
for each received message M **do**
    if $M =$ (UB, $(x_1^*, \ldots, x_n^*), c$) **then**
        $c^* \leftarrow c$ **and** record $(x_1^*, \ldots, x_n^*)$ as the best solution found so far
        continue
    if $M =$ (CPA, $(x_1^*, \ldots, x_{i-1}^*), c$) **then**
        $D'_i \leftarrow D_i$ **and** $(x_1, \ldots, x_{i-1}) \leftarrow (x_1^*, \ldots, x_{i-1}^*)$ **and** $c_i^- \leftarrow c$
    else if $M =$ (BACK) **then** $D'_i \leftarrow D'_i \backslash \{x_i^*\}$
    // Look for a (better) value for x_i
    $x_i \leftarrow$ first $x_i^* \in D'_i$ such that $c_i^- + c_i(x_i^*, \cdot) < c^*$
    if there exists such a x_i^* **then**
        if $i = n$ **then** // last variable
            $x_n \leftarrow x_n^* = \arg\min_{x'_n \in D'_n} \{c_n(x'_n, \cdot)\}$
            $c^* \leftarrow c_n^- + c_n(x_n^*, \cdot)$
            Record $(x_1^*, \ldots, x_n^*)$ as the best solution found so far
            Broadcast message (UB, $(x_1^*, \ldots, x_n^*), c^*$)
            if $c^* = 0$ **then** broadcast message (TERMINATE)
            else send message (BACK) to x_{n-1}
        else send message (CPA, $(x_1^*, \ldots, x_i^*), c_i^- + c_i(x_i^*, \cdot)$) to x_{i+1}
    else
        if $i = 1$ **then** broadcast message (TERMINATE)
        else send message (BACK) to x_{i-1}

2.4.2 ADOPT: Asynchronous Distributed OPTimization

The first asynchronous, complete algorithm for DCOP was *Asynchronous Distributed OPTimization (ADOPT)* [Modi et al., 2005]. One of the key, novel features in ADOPT is that it does not use a total ordering of variables; instead, variables are ordered along a *pseudo-tree*.

Pseudo-tree Generation

The ADOPT algorithm first organizes the decision variables in the DCOP along a hierarchical structure called a *pseudo-tree*, defined a follows.

Definition 7 (pseudo-tree)**.** *A* pseudo-tree *is a generalization of a tree, in which a node is allowed to have additional links (called* back-edges*) with remote ancestors (called* pseudo-parents*) and with remote descendants (called* pseudo-children*), but never with nodes in other branches of the tree.*

Such variable hierarchies make it possible to use a *divide-and-conquer* approach to solving the DCOP, by splitting the responsibility for enforcing constraints (and hereby, the computation load) among agents. Following this approach, a given agent will only have to enforce constraints that involve its variables and higher variables in the hierarchy; the responsibility for enforcing the other constraints is delegated to its neighboring agents lower in the hierarchy. Furthermore, compared to simpler, linear variable orderings, pseudo-trees have the advantage of better exploiting the topological structure of the problem: two loosely coupled parts of the DCOP will correspond to two different branches in the pseudo-tree, on which computation will be performed in parallel.

A possible pseudo-tree arrangement of the constraint graph in Figure 2.1 is illustrated in Figure 2.8. This pseudo-tree naturally decomposes the original problem into two, loosely coupled subproblems, corresponding to the two branches. The dashed edges represent back-edges with remote ancestors (i.e. pseudo-parents) in the pseudo-tree.

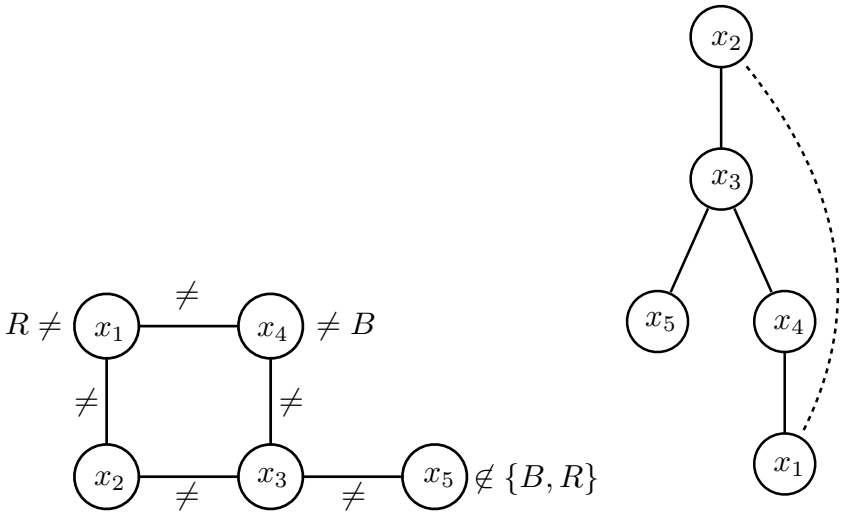


Figure 2.8: The problem in Figure 2.1 and a possible pseudo-tree.

Root Election Most of the literature on pseudo-tree-based algorithms for DCOP completely ignores the issue of selecting a root variable for the pseudo-tree. Algorithm 5 describes a simple procedure to achieve this, based on a viral propagation of variable scores. Initially, each variable is assigned a score (line 1) that can be chosen following any heuristic. The most common heuristic is the *most connected* heuristic, which assigns to each variable its number of neighbors. At the end of the algorithm, each variable discovers the maximum score, and the variable whose score is equal to this value is chosen as the root of the pseudo-tree (line 7). Score tie breaking should be performed based on the variable name, to make sure that only one variable has the highest score. Furthermore, to guarantee that one and only one variable is elected, the agents require the knowledge of an upper bound $\phi_{\max}$ on the diameter of the constraint graph, i.e. the maximum distance (in number of edges) between any pair of variables. If the constraint graph is disconnected, this algorithm will elect one root per connected component of the graph.

Algorithm 5 Root election algorithm for variable x.

Require: upper bound $\phi_{\max}$ on the constraint graph diameter
1: $score_x \leftarrow$ heuristic score for variable x
2: $max \leftarrow score_x$
3: **for** $\phi_{\max}$ times **do**
4: Send max to all neighbors
5: Get $max_1 \dots max_k$ from all neighbors
6: $max \leftarrow \max(max, max_1, \dots, max_k)$
7: **if** $max = score_x$ **then** x is the elected root

Constraint Graph Traversal Once it has been chosen, the root variable then initiates a distributed, depth-first traversal of the constraint graph, with itself as the root. This is done using Algorithm 6, originally proposed by Cheung [1983]. During this traversal, each variable x maintains the following variables:

- $parent_x$ is x's parent in the pseudo-tree (if x is not the elected root);
- $children_x$ is the list of x's children;
- $pseudo\_parents_x$ is the list of x's pseudo-parents;
- $pseudo\_children_x$ is the list of x's pseudo-children;
- $open_x$ is the list of x's neighboring variables that remain to be visited.

Variable x heuristically chooses a neighbor y_0 as its first child (line 4) and sends it a CHILD message (line 5). The most common heuristic is again the *most connected*

Algorithm 6 Pseudo-tree generation algorithm for variable x

```
1:  if x has at least one neighbor then
2:      if x is the root then
3:          open_x ← all neighbors of x
4:          Heuristically remove a neighbor y_0 from open_x and add it to children_x
5:          Send a CHILD message to y_0
6:      loop
7:          Wait for an incoming message of type type from a neighbor y_i
8:          if open_x = ∅ then // first time x is visited
9:              open_x ← all neighbors of x except y_i
10:             parent_x ← y_i
11:         else if type = CHILD and y_i ∈ open_x then
12:             Remove y_i from open_x and add it to pseudo_children_x
13:             Send PSEUDO message to y_i
14:             next
15:         else if type = PSEUDO then
16:             Remove y_i from children_x and add it to pseudo_parents_x
17:         // Forward the CHILD message to the next open neighbor:
18:         Heuristically choose a neighbor y_j ∈ open_x
19:         if there exists such a y_j then
20:             Remove y_i from open_x and add it to children_x
21:             Send a CHILD message to y_j
22:         else
23:             if x is not the elected root then // backtrack
24:                 Send a CHILD message to parent_x
25:             return
```

heuristic, which picks the neighbor with the highest degree. Notice that implementing this heuristic requires that each variable preliminarily send its degree to all its neighbors.

When variable x receives a message from a neighbor y_i (line 7), if it is the first message received, then the sender y_i is identified as the parent of x (lines 8 to 10). Else, if it is a CHILD message received from an *open* neighbor, then y_i is identified as a pseudo-child, which variable x reveals to y_i by responding with a PSEUDO message (lines 11 to 16). The CHILD message is then recursively passed from *open* neighbor to *open* neighbor, until no *open* neighbor remains (lines 18 to 22).

When variable x has no more open neighbors, it initiates a traversal backtrack by returning a CHILD message to its parent (lines 23 to 24). The algorithm terminates when the root variable receives a CHILD message from a child and has no more *open* neighbors.

Complexity Analysis The root election algorithm involves exchanging $O(nd\phi_{\max}) \in O(n^3)$ messages, where n is the number of decision variables, d is the degree of the constraint graph, and $\phi_{\max}$ is an upper bound on its diameter. Each message only contains a score (and a destination variable). Once the root variable has been elected, the constraint graph traversal algorithm exchanges $O(n)$ tokens: one CHILD token down and up each tree-edge, plus one CHILD token up and one PSEUDO token down each back-edge.

Asynchronous Search

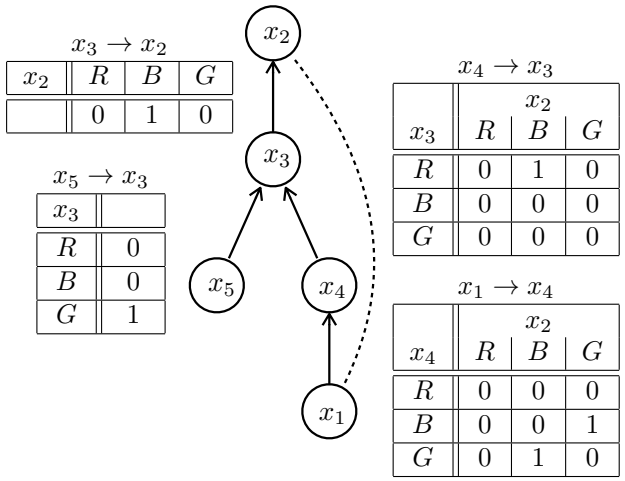
The agents then asynchronously perform best-first search, higher-priority agents sending down candidate assignments in *VALUE* messages, and lower-priority agents responding with *COST* messages that contain lower and upper bounds on costs. *THRESHOLD* messages are also exchanged to reduce redundant search. ADOPT was shown to outperform SynchBB in terms of number of *communication cycles*; however, it has the tendency to exchange a very large number of messages (exponential in the depth of the pseudo-tree). Various performance improvements to ADOPT have since been proposed, such as *ADOPT-ng* [Silaghi and Yokoo, 2006] and *BnB-ADOPT* [Yeoh et al., 2008].

2.4.3 OptAPO: Optimization by Asynchronous Partial Overlay

Most of the DCOP algorithms subsequently proposed are optimization variants of existing DisCSP algorithm; this is the case of *OptAPO* [Mailler and Lesser, 2004] (later made complete as *CompOptAPO* by Grinshpoun and Meisels [2008]), which, as the same suggests, is a generalization of APO. The OptAPO algorithm was shown to outperform ADOPT, but has the same tendency as APO to centralize most of the problem into one mediator agent.

2.4.4 DPOP: Dynamic Programming Optimization Protocol

Petcu and Faltings [2005b] introduced the *Dynamic Programming Optimization Protocol (DPOP)*, which, contrary to most other complete DCOP algorithms generally based on *search*, performs *dynamic programming* along a pseudo-tree variable ordering. DPOP is an instance of the general bucket elimination scheme by Dechter [2003], which is adapted for the distributed case. Once a pseudo-tree has been constructed, DPOP runs in two phases: 1) variables first propagate aggregated costs (or utilities) up the pseudo-tree; and 2) variables then propagate decisions down the pseudo-tree.



x_2	R	B	G
	0	1	0

x_3	
R	0
B	0
G	1

x_3	x_2 R	B	G
R	0	1	0
B	0	0	0
G	0	0	0

x_4	x_2 R	B	G
R	0	0	0
B	0	0	1
G	0	1	0

Figure 2.9: DPOP's pseudo-tree and UTIL messages on the problem in Figure 2.8.

Algorithm 7 DPOP's UTIL propagation for variable x

1: // Join local constraints:
2: $p_x \leftarrow parent_x$
3: $m(x, p_x, \cdot) \leftarrow \Sigma_{c \in \{c' \in \mathcal{C} \mid x \in scope(c') \wedge scope(c') \cap (children_x \cup pseudo\_children_x) = \emptyset\}} c(x, \cdot)$
4: // Join with received messages:
5: **for** each $y_i \in children_x$ **do**
6: Wait for the message (UTIL, $m_i(x, \cdot)$) from y_i
7: $m(x, p_x, \cdot) \leftarrow m(x, p_x, \cdot) + m_i(x, \cdot)$
8: // Project out x:
9: **if** x is not the root variable **then**
10: $x^*(p_x, \cdot) \leftarrow \arg\min_x \{m(x, p_x, \cdot)\}$
11: $m(p_x, \cdot) \leftarrow \min_x \{m(x, p_x, \cdot)\}$
12: Send the message (UTIL, $m(p_x, \cdot)$) to p_x
13: **else** $x^* \leftarrow \arg\min_x \{m(x)\}$ // $m(x, p_x, \cdot)$ actually only depends on x

UTIL Propagation

During the first UTIL propagation phase (Algorithm 7), messages travel up the pseudo-tree, propagating information about the aggregated optimal cost. Each variable x first computes the sum (also called the *join*) of the constraints involving x (line 3). However, since a given constraint may involve multiple variables, but must be counted only once in the overall sum of all constraints that is being minimized, variable x ignores all constraints that involve at least one descendant of x in the pseudo-tree. In other words, the responsibility of enforcing a given constraint is assigned to its lowest variable in the pseudo-tree (like in ADOPT).

Variable x then waits for the reception of a UTIL message from each of its children (if

any), and *joins* them all together with its constraints (lines 4 to 7). Finally, variable x *projects* itself out of the join (lines 8 to 13), and sends the resulting cost to its parent p_x. This cost corresponds to the minimum achievable cost for the whole subtree rooted at x, as a function of the values for p_x and possibly also other variables higher in the pseudo-tree. More precisely, the set of all variables in the UTIL message sent by x is called its *separator*, and includes x's parent and pseudo-parents, as well as all of x's descendants' pseudo-parents that are above x in the pseudo-tree. Variable x also records its optimal value $x^*(p_x, \cdot)$, which is a function of its separator.

Consider for instance the message sent by agent $a(x_5)$ to its parent agent $a(x_3)$ in Figure 2.9. This message is the result of the *projection* of variable x_5 out of the conjunction of x_5's two (soft) constraints $x_5 \neq x_3$ and $x_5 \notin \{B, R\}$, and summarizes $a(x_5)$'s minimal number of constraint violations, as a function of the ancestor variable x_3. More generally, each message sent by a variable x expresses the minimal number of constraint violations for the entire subtree rooted at x, as a function of x's separator. For instance, the message $x_4 \rightarrow x_3$ contains the minimal number of constraint violations for the entire subtree rooted at x_4, as a function of x_4's separator $\{x_2, x_3\}$. Notice that the separator of a variable x can contain variables that are not neighbors of x, just like x_2 is in x_4's separator because a descendent of x_4 has a constraint with x_2. Upon receiving the messages $x_5 \rightarrow x_3$ and $x_4 \rightarrow x_3$, agent $a(x_3)$ *joins* them (i.e. adds them up) with its constraint $x_3 \neq x_2$. Variable x_3 is then projected out of the resulting joint table, which produces the message $x_3 \rightarrow x_2$.

Conceptually, the DPOP algorithm can be seen as a method of reformulating the computation of $\min_{x_1,\ldots,x_n}\{c_1(\cdot) + \ldots + c_m(\cdot)\}$ by appropriately distributing the $\min$ over the sum. For instance, the pseudo-tree in Figure 2.9 corresponds to reformulating the following computation:

$$\min_{x_1 \ldots x_5} \{(x_1 \neq R) + (x_1 \neq x_2) + (x_1 \neq x_4) + (x_2 \neq x_3) + (x_3 \neq x_4) + (x_3 \neq x_5) + (x_4 \neq B) + (x_5 = G)\}$$

into the following tree of operations:

$$\min_{x_2}\left\{\min_{x_3}\left\{(x_2 \neq x_3) + \left(\begin{array}{l}\min_{x_5}\{(x_3 \neq x_5) + (x_5 = G)\} \\ + \\ \min_{x_4}\{(x_3 \neq x_4) + (x_4 \neq B) + \min_{x_1}\{(x_1 \neq R) + (x_1 \neq x_2) + (x_1 \neq x_4)\}\}\end{array}\right)\right\}\right\}.$$

VALUE Propagation

At the end of the UTIL propagation phase, the root variable has obtained its optimal value, which is not a function of any other variable since its separator is empty (it is the highest variable in the pseudo-tree). It sends this optimal value to each of its children through VALUE messages. Recursively, each variable x, when receiving the message $VALUE_{p_x \rightarrow x}$ from its parent p_x, is able to look up its corresponding optimal value x^* computed during the UTIL propagation phase. To each of its children y_i,

variable x sends not only x^*, but also the optimal values for all the variables in y_i's separator, which are contained in the VALUE message it received. Optimal decisions are hereby propagated down the pseudo-tree, until all variables have been assigned optimal values.

Complexity Analysis

The complexity bottleneck of DPOP in terms of amount of information exchanged lies in the UTIL propagation phase. The algorithm only exchanges $(n-1)$ UTIL messages (each variable sends one to its parent, except the root), but these messages can be exponentially large. More precisely, the largest UTIL message contains $O(D_{\max}^{w_i})$ cost values, where $D_{\max}$ is the size of the largest domain, and w_i is the *induced width* of the pseudo-tree used, which is bounded above by the number of variables n, and bounded below by the *treewidth* w of the constraint graph.

The VALUE propagation phase also exchanges $(n-1)$ VALUE messages, but each contains at most w_i optimal assignments to variables. Therefore, overall, the complexity of DPOP in terms of information exchanged is $\Omega(nD_{\max}^{w})$ and $O(nD_{\max}^{w_i})$.

DPOP was shown to largely outperform ADOPT in runtime and number of messages; however, contrary to ADOPT, its space complexity is exponential, which makes it inapplicable to DCOPs with large treewidths. A non-denumerable number of variants of DPOP have been proposed, including a memory-bounded version (*MB-DPOP*) [Petcu and Faltings, 2007]. Other authors have also proposed generalizations of DPOP using different variable orderings; this includes the *Distributed Cross-edged Pseudo-tree Optimization Procedure (DCPOP)* [Atlas and Decker, 2007], *Distributed Cluster Tree Elimination (DCTE)* [Brito and Meseguer, 2010], and *Action-GDL* [Vinyals et al., 2010b].

2.4.5 AFB: Asynchronous Forward Bounding

Asynchronous Forward Bounding (AFB) [Gershman et al., 2006] is an optimization variant of AFC, in which search is performed in a synchronous manner like in SynchBB, but *forward bounding* is performed in parallel (like forward checking in AFC). Gershman et al. showed that, when using a lexicographic variable ordering, AFB outperforms SynchBB and ADOPT in terms of *Non-Concurrent Constraint Checks (NCCCs)* [Gershman et al., 2008]. However, we have recently shown that, when smarter variable ordering heuristics are used, such as the *min-width* heuristic, AFB is outperformed by SynchBB on dense problems, and by both ADOPT and SynchBB on sparse problems [Olteanu et al., 2011]. Variants of AFB have also been proposed, such as to include *conflict-directed back-jumping (AFB-BJ)* [Gershman et al., 2009].

2.4.6 NCBB: No-Commitment Branch and Bound

No-Commitment Branch and Bound (NCBB) [Chechetka and Sycara, 2006] is an algorithm that, like ADOPT and DPOP, is based on a pseudo-tree ordering of the variables. Like ADOPT, it is also based on search; however, contrary to ADOPT, in NCBB a variable x that has multiple children in the pseudo-tree may communicate a different candidate assignment $x = x^*$ to each of its children, which results in various branches of the pseudo-tree exploring in parallel various, non-intersecting parts of the solution space in parallel. The algorithm was shown to outperform ADOPT, but not DPOP; however, contrary to DPOP, NCBB has a polynomial-space complexity. NCBB was also later shown to be outperformed by BnB-ADOPT [Yeoh et al., 2008].

2.4.7 ConcFB: Concurrent Forward Bounding

A similar concept of multiple parallel searches over non-intersecting subspaces is used in the *Concurrent Forward Bounding (ConcFB)* algorithm [Netzer et al., 2010], which is an optimization variant of ConcDB, and was shown to outperform SynchBB and BnB-ADOPT.

2.5 Incomplete DCOP Algorithms

The majority of incomplete algorithms are based on some form of local search, in which agents start from an initial, probably infeasible (or sub-optimal) solution, and attempt to repair it by making local changes, based only on local information from their neighbors.

2.5.1 DBA: Distributed Breakout Algorithm

In the Max-DisCSP *Distributed Breakout Algorithm (DBA)* [Yokoo and Hirayama, 1996], neighboring agents exchange information about how much the solution could be improved if a local variable assignment were changed (Algorithm 8, lines 8 to 12). Like in all local search algorithms, the agents can get stuck in local minima, in which no single local change seems to improve the solution; to get out of such local optima, DBA increases the weights of the current assignment (line 23), effectively casting the Max-DisCSP into a DisWCSP. Hirayama and Yokoo [1997] later proposed the *Iterative Distributed Breakout (IDB)* variant, and several more variants were proposed in [Hirayama and Yokoo, 2005].

Algorithm 8 The DBA algorithm for variable x.

Require: a fixed, known, linear order on the variables: $x_1 \succ \ldots \succ x_n$
Require: an upper bound $\phi_{\max}$ on the constraint graph diameter
1: $c_x(x,\cdot) \leftarrow \bigwedge_{c \in \{c' \in \mathcal{C} \mid x \in scope(c')\}} c(x,\cdot)$ // join all constraints involving x
2: $x \leftarrow$ random $x^* \in D_x$
3: $t_x \leftarrow 0$ // termination counter
4: Send message (OK?, x^*) to all neighbors
5: **loop**
6: // Wait for all OK? messages
7: Wait for one message (OK?, y^*) from each neighbor y **and** record $y \leftarrow y^*$
8: $c_x^- \leftarrow c_x(x,\cdot)$ // current local cost
9: $x^* \leftarrow \arg\min_{x' \in D_x} c_x(x',\cdot)$ // candidate better value for x
10: $\Delta c_x \leftarrow c_x^- - c_x(x^*,\cdot)$ // cost improvement
11: **if** $c_x^- > 0$ **then** $t_x \leftarrow 0$
12: Send message (IMPROVE, $\Delta c_x, c_x^-, t_x$) to all neighbors
13: // Wait for all IMPROVE messages
14: **for** each message (IMPROVE, $\Delta c, c^-, t$) from a neighbor y **do**
15: $t_x \leftarrow \min(t_x, t)$
16: **if** $\Delta c > \Delta c_x$ **then**
17: $\Delta c_x \leftarrow -\Delta c < 0$ // x must not move; not a quasi-local minimum
18: **else if** $\Delta c = \Delta c_x \ \wedge \ y \succ x$ **then** $\Delta c_x \leftarrow -\Delta c_x \leq 0$ // x must not move
19: $c_x^- \leftarrow \max(c_x^-, c^-)$
20: // Send OK? messages
21: **if** $c_x^- = 0$ **then** $t_x \leftarrow t_x + 1$
22: **if** $t_x \geq \phi_{\max}$ **then** broadcast message (TERMINATE) **and exit**
23: **if** $\Delta c_x = 0$ **then** $c_x(x^*,\cdot) \leftarrow c_x(x^*,\cdot) + 1$ // quasi-local min; increase weight
24: **else if** $\Delta c_x > 0$ **then** $x \leftarrow x^*$ // x moves
25: Send message (OK?, x) to all neighbors

Algorithm 9 The DSA algorithm (replacement for lines 11 to 24 in Algorithm 8)

1: $p \leftarrow$ random number $\in [0,1]$
2: **if** variant $A \ \wedge \ \Delta c_x > 0 \ \wedge \ p < p^*$ **then** $x \leftarrow x^*$
3: **if** variant $B \ \wedge \ (\Delta c_x > 0 \ \vee \ c_x(x,\cdot) > 0) \ \wedge \ p < p^*$ **then** $x \leftarrow x^*$
4: **if** variant $C \ \wedge \ p < p^*$ **then** $x \leftarrow x^*$
5: **if** variant $D \ \wedge \ (\Delta c_x > 0 \ \vee \ (c_x(x,\cdot) > 0 \ \wedge \ p < p^*))$ **then** $x \leftarrow x^*$
6: **if** variant $E \ \wedge \ (\Delta c_x > 0 \ \vee \ p < p^*)$ **then** $x \leftarrow x^*$

2.5.2 DSA: Distributed Stochastic Algorithm

The *Distributed Stochastic Algorithm (DSA)* [Fitzpatrick and Meertens, 2001] performs local search on Max-DisCSPs in a similar manner, except that it has a different strategy to escape from local minima. When no single local change seems to improve solution quality, with some probability p^* and in some circumstances which depend on the

variant of the algorithm (A, B, C, D or E), the agents will perform a non-strictly-improving change (Algorithm 9). Zhang et al. [2005] showed how both DSA and DBA could be generalized to DCOPs, and compared their respective performances.

2.5.3 MGM: Maximum Gain Message

Maheswaran et al. [2004a] proposed the *Maximum Gain Message (MGM)* algorithm, which is only a simplification of DBA, and then introduced the *MGM-2* variant, in which local search is performed over slightly larger neighborhoods, allowing *two* local changes to occur simultaneously instead of only one in DBA and DSA. The resulting algorithm generates 2-*optimal* solutions, according to the definition of k-*optimality* in [Pearce and Tambe, 2007].

2.5.4 A-DPOP: Approximate DPOP

Departing from the local search approach, Petcu and Faltings [2005a] proposed the *A-DPOP* algorithm, which is derived from the complete DPOP algorithm (Section 2.4.4) by *dropping dimensions* from utility messages using approximate projections. Propagating lower and upper bounds on utilities then makes it possible to obtain *a posteriori* a guarantee on the maximum distance from optimality of the solution found.

2.5.5 The Max-Sum Algorithm

Instead of reasoning on the constraint graph of the DCOP, the *Max-Sum* algorithm [Farinelli et al., 2008] introduced the novel (to the DCOP literature) idea of reasoning on a *factor graph* representation of the problem, which is a bipartite graph in which *function nodes* stand for constraints, and *variable nodes* stand for the decision variables in the DCOP (Figure 2.10).

Each variable node x initiates the algorithm by sending to its neighboring function

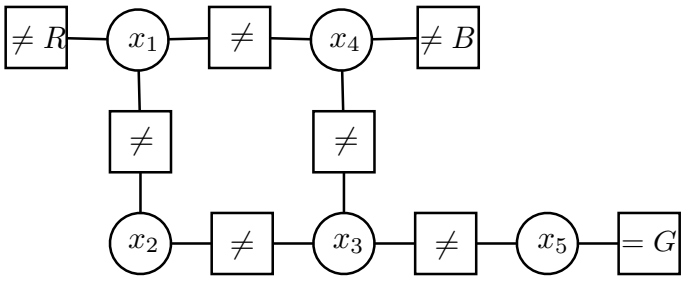


Figure 2.10: The factor graph for the problem in Figure 2.8.

nodes a zero utility function of itself $q : x \to 0$ (Algorithm 10, lines 3 to 6). Each function node reacts by sending to each of its neighboring variable nodes x a utility function $r(x)$ that is equal to the marginal, maximum, global utility achievable, based on the utility function $q(z)$ it has received from each neighboring variable node z (lines 19 to 21). This makes it possible for each variable node to compute an assignment to itself, as the one that maximizes an approximation of the marginal global utility (line 10). This approximation is guaranteed to become exact after a certain number of iterations if the factor graph is free of cycles. Farinelli et al. showed that Max-Sum outperforms DSA.

Algorithm 10 The Max-Sum algorithm.

Procedure: executed by every variable node x
1: $\mathcal{C}_x \leftarrow \{u' \in \mathcal{C} \mid x \in scope(u')\}$ // all contraints involving x
2: $u_x(x,\cdot) \leftarrow \bigwedge_{u\in\mathcal{C}_x} u(x,\cdot)$ // join all constraints involving x
3: $q : x \to 0$
4: **for** $u \in \mathcal{C}_x$ **do**
5: $\quad r_u^- : x \to \infty$ // last function received from function node u
6: $\quad$ Send function $q(x)$ to function node u
7: **loop**
8: $\quad$ Wait for a function $r(x)$ from a neighboring function node u
9: $\quad$ **if** $r \equiv r_u^-$ **then continue else** $r_u^- \leftarrow r$
10: $\quad x^* \leftarrow \arg\max_{x'\in D_x} \left\{\sum_{u\in\mathcal{C}_x} r_u^-(x')\right\}$
11: $\quad$ **for** $u \in \mathcal{C}_x$ **do**
12: $\quad\quad q : x \to \sum_{u'\in\mathcal{C}_x\setminus\{u\}} r_{u'}^-(x)$
13: $\quad\quad q : x \to q(x) - \sum_{x'\in D_x} q(x')$ // rescale so that $\sum_{x\in D_x} q(x) = 0$
14: $\quad\quad$ Send function $q(x)$ to function node u

Procedure: executed by every function node u
15: **for** $x \in scope(u)$ **do** $q_x^- : x \to \infty$ // last function received from variable node x
16: **loop**
17: $\quad$ Wait for a function $q(x)$ from a neighboring variable node x
18: $\quad$ **if** $q \equiv q_x^-$ **then continue else** $q_x^- \leftarrow q$
19: $\quad$ **for** $x \in scope(u)$ **do**
20: $\quad\quad r : x \to \max_{y\in scope(u)\setminus\{x\}} \left\{u(x,\cdot) + \sum_{z\in scope(u)\setminus\{x\}} q_z^-(x)\right\}$
21: $\quad\quad$ Send function $r(x)$ to variable node x

2.5.6 DaCSA: Divide and Coordinate Subgradient Algorithm

In the *Divide and Coordinate Subgradient Algorithm (DaCSA)* [Vinyals et al., 2010a], agents iteratively interleave two stages: during the *divide* stage, the agents break down the DCOP into a set of simpler, local subproblems, which they solve. Then, during the *coordinate* stage, the agents exchange information on their disagreements about the

solution. DaCSA exhibits an *any-time* behavior during which it continuously reports solutions of improving quality, along with estimates of their distances to optimality.

2.5.7 DALO: Distributed Asynchronous Local Optimization

Finally, Kiekintveld et al. [2010] proposed the *Distributed Asynchronous Local Optimization (DALO)* algorithm, which revisits the notion of k-optimality, and can also use the new notion of t*-distance optimality* to produce approximate solutions. Recently, Vinyals et al. [2011b] further extended DALO to also support their new concept of *region optimality*.

3 Strong Privacy Guarantees in DCOP

This chapter presents novel DCOP algorithms that provide strong guarantees on what part of any given agent's knowledge about the problem will *provably not* be revealed to other agents by the messages exchanged during the algorithm execution. In other words, we consider that agents are *curious* and we want to prevent them from discovering critical pieces of information about other agents' sub-problems. Still, we assume that agents honestly follow the protocol. In this thesis, we do not investigate byzantine behaviors or self-interested agents; instead, we focus on preventing private information leaks to other agents by the algorithm. Such a cooperative behavior can be justified by the fact that, if the algorithm fails to find a solution to the problem, all agents lose (for instance, no feasible meeting schedule is found).

In Section 3.1, we first identify and formally define four types of information about the problem that agents may want to protect. Section 3.2 then describes some related work that, beyond the previous work already presented in Chapter 2, addressed in particular the issue of privacy in multi-agent decision making. Sections 3.3 to 3.5 introduce three novel DCOP algorithms[1] that provide various levels of privacy guarantees, at various costs in terms of complexity. Section 3.6 then formally proves these privacy guarantees, and Section 3.7 finally reports empirical evaluations of the performance properties of our algorithms against the state of the art.

3.1 Privacy Definitions

Before addressing what and how private information loss can be prevented in DCOP algorithms, it is important to realize that there can exist private information that may *inevitably* be leaked by *any* DCOP algorithm. This is the purpose of the following definition.

[1] We have published these algorithms in the following workshop and conference papers: [Faltings et al., 2008; Léauté and Faltings, 2009b; Léauté et al., 2010; Léauté and Faltings, 2011a].

Definition 8 (Semi-private information)**.** Semi-private information *refers to information about the problem and/or its solution that an agent might consider private, but that can* inevitably *be leaked to other agents by their views of the chosen solution to the DCOP. No distributed algorithm can guarantee the protection of such information.*

In other words, everything a given agent can discover about other agents by making inferences simply based on its initial knowledge of the problem and on the values its variables take in the solution is considered semi-private information. For instance, in a distributed graph coloring problem involving only two colors, each node can infer the color of each of its neighbors from the color it was assigned in the chosen feasible solution, regardless of the (correct) algorithm used to produce this solution. Excluding semi-private information, we now distinguish four types of private information that agents may desire to protect.

Definition 9 (Agent privacy)**.** *No agent should be able to discover the identity, or even the existence of non-neighboring agents, i.e. agents it does not share constraints with. A particular consequence of this type of privacy is that two agents should only be required to communicate directly if they share a constraint.*

Figure 3.1 recalls the graph coloring example introduced in Chapter 2. In this example, agent privacy means for instance that agent $a(x_1)$ should not be able to discover the existence and identities of agents $a(x_3)$ and $a(x_5)$. This type of privacy cannot be protected by linear-ordering-based algorithms, because they may require, for instance, that $a(x_1)$ and $a(x_3)$ directly exchange messages, while pseudo-tree-based algorithms like DPOP do not suffer from this flaw. However, agent privacy might still be leaked by the contents of messages; in this chapter we propose a method based on *codenames* to fully protect agent privacy.

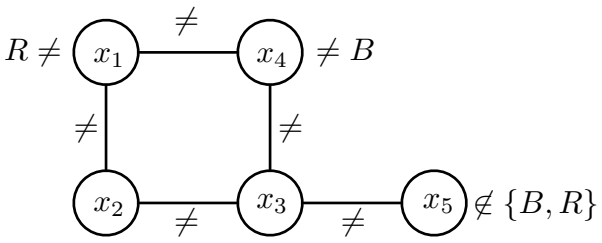


Figure 3.1: The graph coloring problem from Figure 2.1.

Definition 10 (Topology privacy)**.** *No agent should be able to discover the existence of topological constructs in the constraint graph, such as nodes (i.e. variables), edges (i.e. constraints), or cycles, unless it owns a variable involved in the construct.*

In Figure 3.1, topology privacy means for instance that agent $a(x_1)$ should not discover how many other neighbors x_2 has besides itself. However, $a(x_1)$ might discover the

existence of a cycle involving x_1, x_2 and x_4. This is tolerated because x_1 is involved in this cycle, but $a(x_1)$ should not discover the length of the cycle (i.e. that x_2 and x_4 share a neighbor). The use of a pseudo-tree ordering of the variables also has advantages in terms of topology privacy, because an agent in one branch of the pseudo-tree will not be able to discover anything about the topology of other branches.

Definition 11 (Constraint privacy)**.** *No agent should be able to discover the nature of a constraint that does not involve a variable it owns.*

In Figure 3.1, an example of a breach in constraint privacy would be if agent $a(x_1)$ were able to discover that agent $a(x_4)$ does not want to be assigned the color *blue*. This is the type of privacy that the DisCSP and DCOP literature mostly focuses on.

Definition 12 (Decision privacy)**.** *No agent should be able to discover the value that another agent's variable takes in the solution chosen to the problem (modulo semi-private information).*

In a distributed graph coloring problem like in Figure 3.1, this means that no agent can discover the color of any neighbor (let alone any non-neighboring agent) in the solution chosen to the problem. Notice that this is only possible in the presence of at least three colors, otherwise each neighbor's color is semi-private information, as already mentioned.

3.2 Related Work

This section first presents some background knowledge about ElGamal homomorphic encryption (Section 3.2.1), which we will make use of in the privacy-preserving algorithms presented in this chapter. Section 3.2.2 then surveys the literature on privacy in DisCSP and DCOP, and describes how our work relates to this previous work.

3.2.1 Cooperative ElGamal Homomorphic Encryption

Homomorphic encryption is a crucial building block of the privacy-preserving algorithms introduced in this thesis. *Encryption* is the process by which a message — in this section, a Boolean or an integer — can be turned into a *cyphertext*, in such a way that decrypting the cyphertext to retrieve the initial cleartext message is impossible (or, in this case, computationally very hard in the worst case) without the knowledge of the secret encryption key that was used to produce the cyphertext. An encryption scheme is said to be *homomorphic* if it is possible to perform operations on cyphertexts that translate to operations on the initial cleartext messages, without the need to know the encryption key. *ElGamal encryption* [Elgamal, 1985] is one such encryption scheme that possesses this homomorphic property.

Basic ElGamal Encryption of Booleans

We first describe how ElGamal encryption can be used to encrypt Booleans, in such a way that carrying out the following two operations on encrypted Booleans is possible without the knowledge of the secret encryption key:

- the AND of an encrypted and a cleartext Boolean;
- the OR of two encrypted Booleans.

ElGamal encryption is a homomorphic, public key cryptography system based on the intractability of the Diffie-Hellman problem [Tsiounis and Yung, 1998], which proceeds as follows. Let p be a *safe prime* of the form $2rt + 1$, where r is a large random number, and t is a large prime. All numbers and all computations will be modulo p. Let g be a *generator* of $\mathbb{Z}_p^*$, i.e. g is such that its powers cover $[1, p-1]$. With p and g assumed public knowledge, the ElGamal private key is a chosen random number $x \in [1, p-2]$, and the associated public key is $y = g^x$. A cleartext number m is then encrypted as follows:

$$E(m) = (\alpha, \beta) = (my^r, g^r) \tag{3.1}$$

where r is an arbitrary random number chosen by the encryptor. The message is decrypted by computing:

$$\frac{\alpha}{\beta^x} = \frac{my^r}{(g^r)^x} = m\,.$$

A useful feature of ElGamal encryption is that it allows to *randomize* an encrypted value to generate a new encryption of the same value that has no similarity with the original value. Randomizing $E(m)$ in Eq. (3.1) yields

$$E^2(m) = (\alpha y^{r'}, \beta g^{r'}) = (my^{r+r'}, g^{r+r'})$$

which still decodes to m.

To encrypt Booleans, we represent `false` by 1, and `true` by a value $z \neq 1$, which allows us to compute the AND and OR operations:

- AND operation with a cleartext boolean:

$$E(m) \wedge \texttt{false} = E(1) \qquad E(m) \wedge \texttt{true} = E^2(m)$$

- OR operation over two encrypted booleans:

$$E(m_1) \vee E(m_2) = (\alpha_1 \cdot \alpha_2, \beta_1 \cdot \beta_2) = E(m_1 \cdot m_2) \tag{3.2}$$

Cooperative ElGamal Encryption

In the basic ElGamal encryption scheme previously described, decryption can be performed in a single step, using the private key, which is a secret of the agent that originally encrypted the message. However, it is also possible to perform ElGamal encryption in such a way that *all* agents need to cooperate in order to perform decryption. This is possible through the use of a *compound* ElGamal key (x, y) that is generated cooperatively by all agents [Pedersen, 1991]:

Distributed Key Generation The ElGamal key pairs (x_i, y_i) of n agents can be combined in the following fashion to obtain the compound key pair (x, y):

$$x = \Sigma_{i=1}^{n} x_i \qquad y = \Pi_{i=1}^{n} y_i \; .$$

Distributed Decryption If each agent publishes its decryption share β^{x_i}, the message can be decrypted as follows:

$$\frac{\alpha}{\Pi_{i=1}^{n} \beta^{x_i}} = \frac{\alpha}{\beta^{x}} = m \; .$$

Generalization to Integers

In the context of DisCSPs, being able to encrypt Booleans is sufficient, since all constraints assign either `true` (for feasibility) or `false` (for infeasibility) to variable assignments. *Joining* two feasibility values corresponds to computing their AND, and *projecting* a variable corresponds to performing an OR operation. However, for this technique to be applied to optimization problems, it is necessary to be able to encrypt cost values (which we will assume are integers); joining then corresponds to a sum, and projection to a $\min$. This can be performed as follows [Yokoo and Suzuki, 2002].

We first restrict ourselves to integer cost values in $[0, c_{\max}] \cup \{+\infty\}$, where $c_{\max} \in \mathbb{N}$ is a known upper bound on the cost of the optimal solution. A cost c is represented with the following cleartext vector of size $(c_{\max} + 1)$:

$$c \rightarrow [\overbrace{1, \ldots, 1}^{c}, \overbrace{z, \ldots, z}^{c_{\max}+1-c}] \; .$$

Cost values greater than $c_{\max}$ (including $+\infty$) are represented with vectors full of 1's, and cannot be discerned from $+\infty$.

Computing the $\min$ of encrypted costs is performed by applying the same operation as in Eq. (3.2), element-wise on the vectors. The sum of an encrypted cost with a cleartext

cost c is computed by shifting the vector by c as follows:

$$[E_0, \ldots, E_{c_{\max}}] + c = [\overbrace{E(1), \ldots, E(1)}^{c}, E_0, \ldots, E_{c_{\max}-c}] \,.$$

3.2.2 Privacy in DisCSP and DCOP

Before discussing what information may be leaked by a given DCOP algorithm, and how to prevent it, it is important to clarify what information is assumed to be initially known to each agent.

Initial Knowledge Assumptions

In this thesis, we use the following assumptions, which are currently the most widely used in the DisCSP and DCOP literature.

1. Each agent a knows all agents that own variables that are neighbors of a's variables, but does not know any of the other agents (not even their existence);
2. A variable and its domain are known only to its owner agent and to the agents owning neighboring variables, but the other agents ignore the existence of the variable;
3. A constraint is fully known to all agents owning variables in its scope, and no other agent knows anything about the constraint (not even its existence).

Not all previous work on DisCSP and DCOP use exactly the same assumptions. Most noticeably, all algorithms that involve exchanging messages between non-neighboring agents relax Assumption 1: they assume that each agent knows the full list of agents. The algorithms that use this modified assumption include all algorithms based on a linear ordering of variables, with the exception of ABT$_{not}$ [Bessiere et al., 2005]. These algorithms violate agent privacy by design.

When it comes to Assumption 3, Brito and Meseguer [2003] introduced the notion of *Partially Known Constraints (PKCs)*, in which the scope of a constraint is known to all agents involved, but the knowledge of its nature (i.e. which assignments are allowed or disallowed) is distributed among these agents. This generalization actually does not bring any additional expressiveness, since any PKC can be decomposed into a number of constraints over copy variables, such that Assumption 3 holds. However, the introduction of copy variables can be detrimental to decision privacy. Grubshtein et al. [2009] later proposed the similar concept of *asymmetric* constraints, which can also be reformulated as symmetric constraints over copy variables.

Other previous work adopted a dual approach, assuming that variables are public and known to all agents, but each constraint is known to only one agent, which does not necessarily own any of the variables in its scope [Silaghi et al., 2000; Yokoo et al., 2002; Silaghi, 2005a]. Silaghi [2005b] even proposed a framework in which the constraints are secret to everyone. This dual approach has the disadvantage of necessarily violating topology privacy, since all variables are public.

Measuring Constraint Privacy Loss

The majority of the literature on privacy in DisCSP and DCOP focuses on constraint privacy. Metrics have been proposed to evaluate constraint privacy loss in various algorithms, in particular in the context of distributed meeting scheduling, as in [Franzin et al., 2004; Wallace and Freuder, 2005]. Maheswaran et al. [2006] designed a framework called *Valuation of Possible States (VPS)* that they used to measure constraint privacy loss in the OptAPO and SynchBB algorithms, and they considered the impact of whether the problem topology is public or only partially known to the agents. Greenstadt et al. [2006] also applied VPS to evaluate DPOP and ADOPT on meeting scheduling problems, under the assumption that the problem topology is public. Doshi et al. [2008] proposed to consider the cost of privacy loss in optimization problems, in order to elegantly balance privacy and optimality.

Preventing Constraint Privacy Loss

Some previous work also proposed approaches to partially reduce constraint privacy loss. For instance, Brito and Meseguer [2007] described a modification of the *Distributed Forward Checking (DisFC)* algorithm for DisCSPs in which agents are allowed to *lie* for a finite time in order to achieve higher levels of privacy. However, most search-based algorithms like DisFC exhibit a typical *easy-hard-easy* transition: for low constraint tightness, they easily find a solution to the DisCSP, while for high tightness they easily prove that no solution exist; the hard problems lie in the middle. Silaghi and Mitra [2004] made the important observation that this phase transition is undesirable when one's goal is to protect privacy, because it makes it possible to make inferences on the constraint tightness by observing the runtime or the amount of information exchanged. To avoid this subtle privacy leak, one must either perform full exhaustive search, which is the option chosen by Silaghi, or resort to dynamic programming (i.e. use a variant of DPOP), which is the option we have chosen in this thesis. It is worth noting that, while our contributions focus particularly on DPOP, much of this work could be adapted and generalized to other DCOP algorithms.

The cryptographic technique of *secret sharing* [Shamir, 1979; Ben-Or et al., 1988] was also applied in [Silaghi et al., 2006; Greenstadt et al., 2007] to lower constraint privacy in DPOP, but the authors always assumed that the constraint graph topology was public

knowledge. In fact, many techniques from cryptography have been applied to provide strong guarantees on constraint privacy preservation in multi-agent decision making. For instance, Yokoo and Suzuki [2002]; Yokoo et al. [2002, 2005] showed how a public key encryption scheme can be used to solve DisCSPs using multiple servers, while protecting both constraint privacy and decision privacy. Bilogrevic et al. [2011] solved single-meeting scheduling problems using similar techniques, and one semi-trusted server. In this thesis however, we only consider algorithms that do not make use of third parties, as such third parties might not be available. Herlea et al. [2001] showed how to use *Secure Multiparty Computation (SMC)*[2] [Yao, 1986; Goldreich et al., 1987] to securely schedule a single meeting, without relying on servers. The idea behind SMC is for agents to collaboratively compute the value of a given, publicly known function on private inputs, without revealing the inputs. In [Herlea et al., 2001], the private inputs are each participant's availability at a given time, and the function outputs whether they are all available.

The MPC-Dis(W)CSP4 Algorithms

Silaghi and Mitra [2004]; Silaghi [2005a] applied a similar approach to output a solution to general DisCSPs and DisWCSPs, where the private inputs are the agents' constraint valuations, and the function returns a randomly chosen solution. Their *MPC-DisCSP4* algorithm for DisCSP is given in Algorithm 11. Each agent a_i first creates a vector F_i with one entry per candidate solution to the DisCSP, equal to 1 if the candidate solution satisfies a_i's private constraints, and to 0 otherwise (lines 1 to 4). To reduce the size of F_i, the candidate solutions may be filtered through publicly known constraints, if there exist any. Using Shamir's polynomial secret sharing technique, agent a_i then sends one secret share $F_{i\to j}$ of its vector F_i to each other agent a_j, and receives corresponding secret shares $F_{j\to i}$ of their respective vectors (lines 8 to 10). Agent a_i then multiplies together all the secret shares it received (lines 11 and 12), so that each entry in its vector F_i becomes a secret share of 1 if and only if the corresponding candidate solution satisfies all agents' private constraints (otherwise, it is a secret share of 0).

Unfortunately, unlike addition, the multiplication of Shamir secret shares is a non-trivial operation, because each secret share is the value of a polynomial, and multiplying two polynomials increases the degree of the output, which must always remain lower than the number of agents to be resolvable. Therefore, after each multiplication of two secret shares V_1 and V_2 (Algorithm 12, line 1), agent a_i must perform a complex sequence of operations involving the exchange of messages in order to reduce the degree of the output V. The agent first exchanges and adds secret shares of zero to V to randomize it (lines 2 to 7). Once V has been randomized, the agent can share it with other agents, which do the same (lines 8 to 13). The agent can then produce reduced secret shares using the function REDUCEDEGREE (line 15), which involves

[2] Silaghi uses the different acronym *MPC* for the same concept.

Algorithm 11 The MPC-DisCSP4 algorithm for agent a_i.

$c_i(x_1, \ldots, x_n) = \bigwedge_{c \in \{c' \in \mathcal{C} \mid a_i = owner(c')\}} c(\cdot)$ // join all of a_i's private constraints
$c_0(x_1, \ldots, x_n) = \bigwedge_{c \in \{c' \in \mathcal{C} \mid c' \text{ is publicly known}\}} c(\cdot)$ // join all public constraints (if any)
// For each publicly feasible solution, 1 if it is also privately feasible, 0 else:
$F_i \leftarrow \left(\delta_{c_i(x'_1,\ldots,x'_n)=\texttt{true}} \mid x'_1 \in d_1 \ \wedge \ \ldots \ \wedge \ x'_n \in d_n \ \wedge \ c_0(x'_1, \ldots, x'_n) = \texttt{true}\right)$
// For each publicly feasible solution, the assignment indexes of all variables:
$I_i \leftarrow ((\text{index of } x'_1 \in d_1, \ldots, \text{index of } x'_n \in d_n) \mid c_0(x'_1, \ldots, x'_n) = \texttt{true})$
$\sigma_i : [1, |F_i|] \mapsto [1, |F_i|]$ // secret random permutation
for $a_j \in \mathcal{A}$ **do** // secretly share all the F vectors
    Send message (SHARE, secret share of F_i for agent a_j) to agent a_j
Wait for all (SHARE, $F_{j \to i}$) messages and record each $F_{j \to i}$
$F_i \leftarrow F_{1 \to i}$
for $a_j \in \mathcal{A} \backslash \{a_1\}$ **do** $F_i \leftarrow \text{MULTIPLY}(F_i, F_{j \to i})$ // as in Algorithm 12
// Shuffle all F vectors (mix-net):
for $a_j \in \mathcal{A}$ **do** $Z_j \leftarrow E(\text{secret share of } (0, \ldots, 0) \text{ for agent } a_j)$ // Paillier encryption
Send (MIX, $a_i, E(F_i)$) to the first agent a_1 // Paillier encryption
for each received message (MIX, a_j, F) **do**
    $F \leftarrow \sigma_i(F) + Z_j$ // shuffle and re-encrypt
    if $i < |\mathcal{A}|$ **then** send message (MIX, a_j, F) to the next agent a_{i+1}
    else send message (MIXED, F) to a_j
Wait for message (MIXED, F) **and** $F_i \leftarrow E^{-1}(F)$ // Paillier decryption
// Isolate the first globally feasible solution:
$h_i \leftarrow 1$ **and** $f_i \leftarrow F_i[0]$
for $k \in [1, |F_i|]$ **do**
    $h_i \leftarrow \text{MULTIPLY}(h_i, 1 - f_i)$ // as in Algorithm 12
    $f_i \leftarrow F_i[k]$ **and** $F_i[k] \leftarrow \text{MULTIPLY}(F_i[k], h_i)$ // as in Algorithm 12
// Un-shuffle all F vectors (inverse mix-net):
for $a_j \in \mathcal{A}$ **do** $Z_j \leftarrow E(\text{secret share of } (0, \ldots, 0) \text{ for agent } a_j)$ // Paillier encryption
Send (UNMIX, $a_i, E(F_i)$) to the last agent $a_{|\mathcal{A}|}$ // Paillier encryption
for each received message (UNMIX, a_j, F) **do**
    $F \leftarrow \sigma_i^{-1}(F) + Z_j$ // un-shuffle and re-encrypt
    if $i > 0$ **then** send message (UNMIX, a_j, F) to the previous agent a_{i-1}
    else send message (UNMIXED, F) to a_j
Wait for message (UNMIXED, F) **and** $F_i \leftarrow E^{-1}(F)$ // Paillier decryption
// Reveal the secret shares of the variables' chosen assignment indexes:
for $a_j \in \mathcal{A}$ **do**
    for each variable x_k owned by a_j **do**
        $\tau_i^k \leftarrow -1 + \sum_{s \in [1, |F_i|]} (1 + I_i[s][k]) F_i[s]$
        Send message (SOLUTION, x_k, τ_i^k) to agent a_j
Wait for all (SOLUTION, x_k, τ_j^k) messages and record each τ_j^k
for each variable x_k owned by a_i (myself) **do**
    $x_k \leftarrow x_k^* = d_x \left[\text{RESOLVE}(\tau_1^k, \ldots, \tau_{|\mathcal{A}|}^k)\right]$

several (communication-free) multiplications with constant projection and Vandermonde matrices (refer to [Ben-Or et al., 1988] for the gory details). After exchanging the resulting reduced secret shares (lines 16 to 17), agent a_i can resolve its reduced vector V using a (communication-free) operation involving the Lagrange coefficients; these details are also not shown for conciseness.

Algorithm 12 MULTIPLY(V_1, V_2) for agent a_i.

1: $V \leftarrow V_1 \times V_2$ // element-wise multiplication
2: // Exchange and add secret shares of a vector full of zeros to randomize V:
3: **for** each other agent $a_j \in \mathcal{A}\backslash\{a_i\}$ **do**
4: $\quad Z_{i\to j} \leftarrow$ secret share of $(0, \ldots 0)$ for agent a_j
5: $\quad$ Send message (SHARE, $Z_{i\to j}$) to agent a_j
6: $V \leftarrow V +$ secret share of $(0, \ldots, 0)$ for agent a_i (myself)
7: **for** each received message (SHARE, $Z_{j\to i}$) **do** $V \leftarrow V + Z_{j\to i}$
8: // Secretly share all the V vectors:
9: **for** each other agent $a_j \in \mathcal{A}\backslash\{a_i\}$ **do**
10: $\quad V_{i\to j} \leftarrow$ secret share of V for agent a_j
11: $\quad$ Send message (SHARE, $V_{i\to j}$) to agent a_j
12: $V_{i\to i} \leftarrow$ secret share of V for agent a_i (myself)
13: Wait for all (SHARE, $V_{j\to i}$) messages and record each $V_{j\to i}$
14: // Reduce the polynomial degree of the shares and exchange the reduced shares:
15: $(V_{1\to i}, \ldots, V_{|\mathcal{A}|\to i}) \leftarrow$ REDUCEDEGREE$(V_{1\to i}, \ldots, V_{|\mathcal{A}|\to i})$
16: **for** each other agent $a_j \in \mathcal{A}\backslash\{a_i\}$ **do** send message (SHARE, $V_{j\to i}$) to agent a_j
17: Wait for all (SHARE, $V_{i\to j}$) messages and record each $V_{i\to j}$
18: $V \leftarrow$ RESOLVE$(V_{i\to 1}, \ldots, V_{i\to|\mathcal{A}|})$
19: **return** V

Going back to Algorithm 11: after performing $(|\mathcal{A}| - 1)$ such pairwise multiplications of secret shares, agent a_i's vector F_i contains secret shares of 1 at the entries corresponding to globally feasible solutions. Agent a_i is then going to perform a transformation on F_i so that only *one* such secret share of 1 remains, identifying one particular feasible solution (if there exists one). Just selecting the first such entry would a posteriori reveal that all previous entries are secret shares of 0, i.e. correspond to infeasible solutions to the DisCSP. To prevent this privacy leak, the vector F_i is first randomly permuted using a *mix-net*. Agent a_i first encrypts its vector F_i using additively-homomorphic Paillier encryption [Paillier, 1999], and then sends it to the first agent a_1 (line 15). Agent a_1 shuffles the vector (line 17) using a secret random permutation (line 7), and then re-encrypts the result to hide the permutation used by adding encrypted shares of zero (line 14). Agent a_i's shuffled vector F_i is passed further on through all agents in sequence so that they can apply their respective secret random permutations, until F_i finally returns fully shuffled to a_i, which decrypts it (lines 18 to 20).

Agent a_i then performs a sequence of iterative operations on F_i (lines 21 to 25) to set all

its entries to secret shares of 0, except one secret share of 1 corresponding to the chosen solution to the DisCSP (if any). The vector F_i is then un-shuffled by re-traversing the mix-net in reverse (lines 26 to 33). Finally, agent a_i can compute secret shares of the domain index of each variable's chosen assignment, and reveal these secret shares only to the owner of the variable (lines 6 and 34 to 41).

Silaghi and Mitra [2004] described how this algorithm can be modified to produce the *MPC-DisWCSP4* algorithm, for DisWCSPs with a known upper bound $c_{\max} < \infty$ on the cost of any full variable assignment. This can be achieved by repeatedly running MPC-DisCSP4 in order to find a solution of a given cost $c \in [0, c_{\max}]$.

The two MPC-Dis(W)CSP4 algorithms have numerous drawbacks. First, agents must be able to communicate pairwise securely, which violates agent privacy. Second, the list of all variables must be publicly known, which violates topology privacy. Third, Shamir's secret sharing scheme is a majority threshold scheme, which means that if at least half of the agents collude, they can discover everyone's private information. These undesirable privacy properties are summarized in Table 3.1. Finally, these algorithms are often only practical for small problems, because they perform full exhaustive search; this is demonstrated by our experimental results in Section 3.7.

3.2.3 Privacy in Linear Programming

Some research has also been carried out to protect privacy in *Linear Programs (LPs)*, which form a different class of optimization problems in which all decision variables have unbounded real-valued domains (rather than finite domains), and all (hard) constraints as well as the objective function (i.e. the single soft constraint) are linear formulae. When the knowledge of the constraints and/or the objective function is *partitioned* among two or more parties, the question that arises is whether the LP can be solved without each party revealing its knowledge of the problem to any other party. Three types of knowledge partitioning can exist: 1) *horizontal partitioning* refers to the case when each constraint is fully known to exactly one party; 2) *vertical partitioning* corresponds to when the knowledge of each constraint is split among all the parties; and 3) *hybrid partitioning* is actually a generalization of the two.

Du and Atallah [2001] proposed a privacy-preserving algorithm for the general, hybrid partitioning case, but restricting themselves to LPs involving only equality constraints, partitioned among only two parties. Their algorithm uses a transformation approach, following which the first party's constraint matrix is first decomposed as a sum of random matrices, and each such random matrix is then further hidden among a large enough set of random matrices, which are communicated to the second party for it to perform the algorithm on the transformed data.

Vaidya [2009] later claimed that Du and Atallah's algorithm was incorrect when applied to more general LPs involving inequalities, and proposed a slightly different algorithm, restricting himself to a special partitioning case in which the objective function is known to one party, and the constraints to another party. But Bednarz et al. [2009] showed that Vaidya failed to fully fix the incorrectness of Du and Atallah's algorithm, and that both approaches still suffer from a flaw that can result in outputting a solution that the algorithm would claim is optimal, while it is actually infeasible.

Mangasarian [2010b] described another transformation algorithm to solve general LPs that are vertically partitioned among possibly more than two parties. In conclusion, he pointed out that his algorithm unfortunately could not work for horizontally partitioned LPs. Shortly thereafter, he proposed in [Mangasarian, 2010a] another algorithm for such horizontally partitioned LPs, but restricting himself to LPs with only equality constraints and non-negativity constraints. He concluded that the more general case of horizontally partitioned LPs with inequality constraints was still an open problem. This open problem was then very recently solved by Li et al. [2011], who proposed to reformulate inequality constraints as equality constraints by introducing slack variables with random positive coefficients, after Mangasarian pointed out that using slack coefficients equal to 1 (as is traditionally done) would violate privacy.

Several questions remain however open. First, while presumably correct algorithms have been proposed respectively for the horizontal and vertical partitioning cases, to our knowledge there still does not exist any correct, privacy-preserving algorithm for the general case of hybrid partitioning. Second, further investigations must be carried out on the current existing algorithms to study their empirical performance on linear programming formulations of realistic, industrial problems. The exact privacy guarantees provided by these algorithms also still remain vague; related work such as [Pedersen and Savaş, 2009] suggest that it may be impossible to produce information-theoretically secure algorithms for Linear Programming.

Recently, Kerschbaum [2011]; Wang et al. [2011] also described how transformation techniques could be used to obfuscate an LP, and then pass it on to one or more third-party servers to solve it.

3.3 The P-DPOP Algorithm

This section describes a first DCOP algorithm that is a variant of the DPOP algorithm, called *Private DPOP (P-DPOP)*, which guarantees full agent privacy. It also partially protects topology, constraint, and decision privacy.

3.3.1 Overview of the Algorithm

The high-level pseudo-code for the algorithm is presented in Algorithm 13. First (lines 1 and 2), a pseudo-tree ordering on the variables is generated. This initial step is essentially the same as in DPOP, except that we provide a new root variable election mechanism with improved privacy properties. A pseudo-tree for the problem in Figure 3.1 is illustrated in Figure 3.2, which is a copy of Figure 2.9 for convenience.

Algorithm 13 Overal P-DPOP algorithm, for variable x

1: Elect a root variable (Algorithm 14, Section 3.3.2)
2: Construct a pseudo-tree rooted at the elected root (Algorithm 6)
3: // Choose and exchange codenames for x and its domain D_x:
4: Wait for messages (CODES, $\tilde{y_i^x}, \tilde{D_{y_i}^x}, \sigma_{y_i}^x$) from all $y_i \in \{parent_x\} \cup pseudo\_parents_x$
5: **for** each $y_i \in children_x \cup pseudo\_children_x$ **do**
6: $\quad x^{\tilde{y_i}} \leftarrow$ large random number
7: $\quad \tilde{D_x^{y_i}} \leftarrow$ list of $|D_x|$ random, unique identifiers
8: $\quad \sigma_x^{y_i} \leftarrow$ random permutation of $[1, \ldots, |D_x|]$
9: $\quad$ Send message (CODES, $x^{\tilde{y_i}}, \tilde{D_x^{y_i}}, \sigma_x^{y_i}$) to y_i
10: // Choose and exchange obfuscation key for x:
11: Wait for and record a message (KEY, $key_{y_i}^x$) from each $y_i \in pseudo\_parents_x$ (if any)
12: **for** each $y_i \in pseudo\_children_x$ **do**
13: $\quad key_x^{y_i} \leftarrow$ vector of large random numbers, indexed by D_x
14: $\quad$ Send message (KEY, $key_x^{y_i}$) to y_i
15: Propagate cost values up the pseudo-tree (Algorithm 15, Section 3.3.3)
16: // Propagate decisions top-down along the pseudo-tree (Section 3.3.4):
17: **if** x is not the root **then**
18: $\quad$ Wait for message (VALUE, $\tilde{p_x^*}, \cdot$) from parent p_x
19: $\quad x^* \leftarrow x^*(\tilde{p_x} = \tilde{p_x^*}, \cdot)$ // where $x^*(\cdot)$ was computed in Algorithm 15, line 22
20: **for** each $y_i \in children_x$ **do**
21: $\quad$ Send message (VALUE, $\tilde{sep_{y_i}^*}$) to y_i, with sep_{y_i} from Algorithm 15, line 13

P-DPOP's new UTIL propagation phase is described in Section 3.3.3; in particular, it is necessary to obfuscate the messages presented in cleartext in Figure 3.2 in order to protect the participants' privacy. At the end of this UTIL propagation, the root variable x_2 can choose a value for itself that is optimal for the entire subproblem (for instance, $x_2 = R$). This decision can then be propagated downwards along tree-edges until all variables have been assigned optimal values (lines 16 to 21, Section 3.3.4).

3.3.2 Root Variable Election

The first step in generating a pseudo-tree ordering of the variables consists in choosing a root for this pseudo-tree. In P-DPOP, this is achieved using Algorithm 14, which is an

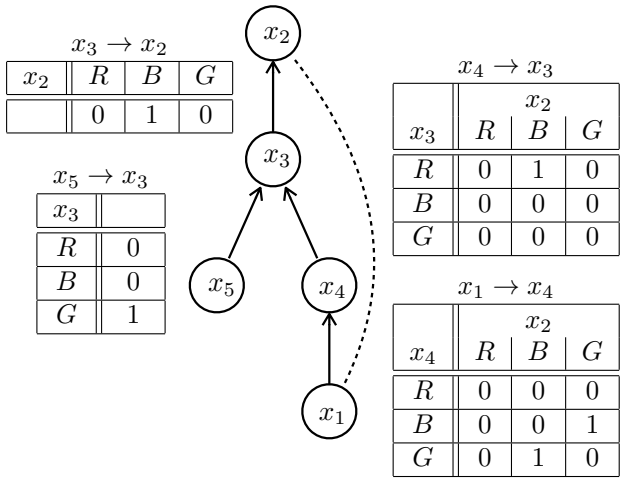


Figure 3.2: The pseudo-tree from Figure 2.9.

Algorithm 14 Anonymous root election algorithm for variable x.

Require: upper bound $\phi_{\max}$ on the constraint graph diameter

1: $score_x \leftarrow$ large random number
2: $max \leftarrow$ random number $\leq score_x$
3: $n_{lies} \leftarrow \text{rand}(\phi_{\max} \dots 2\phi_{\max})$
4: // Propagate under-estimates of the maximum score:
5: **for** n_{lies} times **do**
6: Send max to all neighbors
7: Get $max_1 \dots max_k$ from all neighbors
8: $max\_tmp \leftarrow \max(max, max_1, \dots, max_k)$
9: $max \leftarrow \text{rand}(max\_tmp \dots \max(score_x, max\_tmp))$
10: // Propagate the true maximum score:
11: $max \leftarrow \max(max, score_x)$
12: **for** $(3\phi_{\max} - n_{lies})$ times **do**
13: Send max to all neighbors
14: Get $max_1 \dots max_k$ from all neighbors
15: $max \leftarrow \max(max, max_1, \dots, max_k)$
16: **if** $max = score_x$ **then** x is the elected root

improvement over DPOP's root election mechanism (Algorithm 5). There are two differences with Algorithm 5: first, P-DPOP uses a random scoring heuristic (rather than the *most connected* heuristic, for instance), so as not to leak topological information. Second, if a naive viral propagation were used like in Algorithm 5, each variable would be able to observe the converging sequence of tentative maximum scores received from each neighbor, and to infer the distance between that neighbor and the elected root, and also whether that neighbor has other neighbors. In order to prevent such

inferences and guarantee topology privacy, the agents first lie a random, bounded number of times, by understating the true maximum score (lines 2 to 9). Upon termination, only the elected variable knows that it is the root.

3.3.3 UTIL Propagation

As already illustrated in Section 3.3.1, the agents then proceed to a bottom-up propagation of costs (or utilities) along the pseudo-tree. The corresponding algorithm is given in Algorithm 15. The following sections describe the obfuscation techniques used to protect the private information that could be leaked by the UTIL messages, using codenames and addition of random numbers. We choose the convention to describe the algorithm in terms of cost minimization.

Hiding Variable Names and Values using Codenames

Consider the UTIL message $x_1 \rightarrow x_4$ sent by agent $a(x_1)$ to its parent variable x_4 in Figure 3.2, recalled in Figure 3.3(a) for convenience. If this message were actually received in cleartext, it would breach agent privacy and topology privacy: agent $a(x_4)$ would be able to infer from the dependency of the message on variable x_2 both the existence of agent $a(x_2)$ (which violates agent privacy) and the fact that x_2 is a neighbor of one or more unknown nodes below (and including) x_1, which would violate topology privacy.

$x_1 \rightarrow x_4$

	x_2		
x_4	R	B	G
R	0	0	0
B	0	0	1
G	0	1	0

(a) in cleartext

$x_1 \rightarrow x_4$

	928372		
x_4	α	β	γ
R	0	0	0
B	0	0	1
G	0	1	0

(b) partly obfuscated

$x_1 \rightarrow x_4$

	928372		
x_4	α	β	γ
R	620961	983655	534687
B	620961	983655	534688
G	620961	983656	534687

(c) fully obfuscated

Figure 3.3: The UTIL message $x_1 \rightarrow x_4$ in Figure 3.2.

In order to patch these privacy leaks, variable x_2 and its domain $D_2 = \{R, B, G\}$ are replaced with random codenames $\tilde{x_2^{x_1}} = 928372$ and $\tilde{D_2^{x_1}} = \{\alpha, \beta, \gamma\}$, as shown in Figure 3.3(b). These codenames are preliminarily generated by $a(x_2)$ and communicated directly to the leaf of the corresponding back-edge (Algorithm 13, lines 4 to 9). The leaf of the back-edge applies these codenames to its messages before sending it (Algorithm 15, line 6), and they are only resolved once the propagation reaches the root of the back-edge (Algorithm 15, line 15). As a result, without the knowledge of these codenames, all agents in between (such as $a(x_4)$) are only able to infer the existence of a cycle in the constraint graph involving some unknown ancestor and some unknown

Algorithm 15 P-DPOP's obfuscated UTIL propagation, for variable x

1: // Join local constraints:
2: $p_x \leftarrow parent_x$
3: $m(x, p_x, \cdot) \leftarrow \Sigma_{c \in \{c' \in \mathcal{C} \mid x \in scope(c') \wedge scope(c') \cap (children_x \cup pseudo\_children_x) = \emptyset\}} c(x, \cdot)$
4: // Apply codenames:
5: **for** each $y_i \in \{p_x\} \cup pseudo\_parents_x$ **do**
6: $m(x, \tilde{p_x}, \cdot) \leftarrow$ replace (y_i, D_{y_i}) in $m(x, p_x, \cdot)$ with $(\tilde{y_i^x}, \tilde{D_{y_i}^x})$ from Algorithm 13, line 4, and apply the permutation $\sigma_{y_i}^x$ to $\tilde{D_{y_i}^x}$
7: // [Only for Max-DisCSPs] Obfuscate infeasible entries:
8: $r \leftarrow$ small, positive, random number
9: $m(x, \tilde{p_x}, \cdot) \leftarrow \begin{cases} m(x, \tilde{p_x}, \cdot) & if \quad m(x, \tilde{p_x}, \cdot) = 0 \\ m(x, \tilde{p_x}, \cdot) + r & if \quad m(x, \tilde{p_x}, \cdot) > 0 \end{cases}$
10: // Join with received messages:
11: **for** each $y_i \in children_x$ **do**
12: Wait for the message (UTIL, $m_i(\tilde{x}, \cdot)$) from y_i
13: $sep_{y_i} \leftarrow scope(m_i)$
14: **for** each $z \in children_x \cup pseudo\_children_x$ **do**
15: $m_i(x, \cdot) \leftarrow$ identify $(\tilde{x^z}, \tilde{D_x^z})$ as (x, D_x) in $m_i(\tilde{x}, \cdot)$ (if $\tilde{x^z}$ is present)
16: $m(x, \tilde{p_x}, \cdot) \leftarrow m(x, \tilde{p_x}, \cdot) + m_i(x, \cdot)$
17: // De-obfuscate costs with respect to x:
18: **for** each $y_i \in pseudo\_children_x$ **do**
19: $m(x, \tilde{p_x}, \cdot) \leftarrow m(x, \tilde{p_x}, \cdot) - key_x^{y_i}(x)$ // with $key_x^{y_i}$ from Algorithm 13, line 13
20: // Project out x:
21: **if** x is not the root variable **then**
22: $x^*(\tilde{p_x}, \cdot) \leftarrow \arg\min_x \{m(x, \tilde{p_x}, \cdot)\}$
23: $m(\tilde{p_x}, \cdot) \leftarrow \min_x \{m(x, \tilde{p_x}, \cdot)\}$
24: // Obfuscate costs:
25: **for** each $y_i \in pseudo\_parents_x$ **do**
26: $m(\tilde{p_x}, \cdot) \leftarrow m(\tilde{p_x}, \cdot) + key_{y_i}^x(\tilde{y_i^x})$ // with $key_{y_i}^x$ from Algorithm 13, line 11
27: Send the message (UTIL, $m(\tilde{p_x}, \cdot)$) to p_x
28: **else** $x^* \leftarrow \arg\min_x \{m(x)\}$ // $m(x, \tilde{p_x}, \cdot)$ actually only depends on x

descendent, which is tolerated by the definition of topology privacy (Definition 10) since they are also involved in this cycle. Furthermore, a secret, random permutation $\sigma_2^{x_1}$ is also applied to the obfuscated domain $\tilde{D_2^{x_1}}$; this is useful for problem classes in which variable domains are public knowledge.

Obfuscating Costs

Hiding variable names and values using codenames addresses the leaks of agent and topology privacy. However, this does not address the fact that the costs in the UTIL

message $x_1 \rightarrow x_4$ in Figure 3.3(b) violate constraint privacy, because they reveal to x_4 that its subtree can always find a feasible solution to its subproblem when $x_4 = R$, regardless of the value of the obfuscated variable 928372. To patch this privacy leak, costs are obfuscated by adding large, random numbers that are generated by the root of the back-edge (x_2) and sent over a secure channel to the leaf of the back-edge (Algorithm 13, lines 11 to 14). The obfuscation is performed in such a way that a different random number is added to all costs associated with each value of x_2, as in Figure 3.3(c), using the obfuscation key $[620961, 983655, 534687]$. These random numbers are added by the leaf of the back-edge to its outgoing message (Algorithm 15, line 26), and they are only eventually subtracted when the propagation reaches the root of the back-edge (Algorithm 15, line 19).

Important note: For the addition of random numbers to be a valid obfuscation scheme, the UTIL messages must not contain any infinite costs/utilities, whose obfuscations would otherwise remain infinite. Therefore, for P-DPOP to be applicable, the input DCOP must first be formulated as a *pure soft* DCOP, in which all $\pm\infty$ costs/utilities must be replaced with $\pm M$ entries, with M sufficiently large, but still smaller than the random numbers used for obfuscation. Similarly, pure DisCSPs must first be reformulated as Max-DisCSPs.

Notice that this obfuscation scheme achieves two objectives: 1) it hides from x_4 the absolute costs of its subtree, and 2) it hides the relative dependencies of these costs on the obfuscated variable 928372, because different random numbers were used for each value in its obfuscated domain $\{\alpha, \beta, \gamma\}$. Agent $a(x_4)$ is still able to infer the relative dependencies on its own variable x_4, which is necessary to perform the projection of this variable, but it is unable to tell, for each value of the other (obfuscated) variable, what the corresponding cost is for its subtree. Notice in particular that, for a given value of the obfuscated variable, agent $a(x_4)$ does not know whether any of the corresponding entries was initially equal to 0, and therefore it would be incorrect to simply assume that the lowest of the obfuscated entries decrypts to 0.

Notice also that this obfuscation scheme is only applicable in the presence of a back-edge, i.e. when the message contains more than just the parent variable. Consider for instance the single-variable message $x_5 \rightarrow x_3$ in Figure 3.2, recalled for convenience in Figure 3.4(a). If agent $a(x_3)$ knew that x_5 is a leaf of the pseudo-tree, the cleartext message would reveal agent $a(x_5)$'s private local constraint $x_5 \notin \{B, R\}$ to agent x_3, and the previous obfuscation scheme does not apply because of the absence of back-edges. Notice that this threat to constraint privacy is tempered by the fact that P-DPOP's guarantees in terms of topology privacy prevent agent $a(x_3)$ from discovering that x_5 is indeed a leaf. From $a(x_3)$'s point of view, a larger subproblem might be hanging below variable x_5 in Figure 3.2, and the message could actually be an aggregation of multiple agents' subproblems.

Further Obfuscating Numbers of Conflicts in Max-DisCSP

In a Max-DisCSP, whose objective is to find a feasible solution of cost 0, the aforementioned privacy leak in single-variable UTIL messages can be partly patched by adding small, positive, random numbers to non-zero entries, in order to obfuscate the true numbers of constraint violations (Algorithm 15, line 9), as illustrated in Figure 3.4(b), where the random number 41 has been added to the non-zero entries of the cleartext message in Figure 3.4(a). Because these random numbers are never subtracted back, they must not be added to zero entries, otherwise the algorithm would fail to find a solution with no violation. This means that feasible entries are still revealed, but the number of constraint violations for infeasible entries remain obfuscated. This improved privacy is not applicable to DCOPs, in which it is not possible *a priori* to identify costs that are guaranteed to be sub-optimal.

$x_5 \rightarrow x_3$

x_3	# conflicts
R	0
B	0
G	1

(a) in cleartext

$x_5 \rightarrow x_3$

x_3	# conflicts
R	0
B	0
G	42

(b) obfuscated

Figure 3.4: The UTIL message received by agent $a(x_3)$ in Figure 3.2.

3.3.4 VALUE Propagation

Once the costs have propagated up all the way to the root of the pseudo-tree, and an optimal assignment to this root variable has been found, this assignment is propagated down the pseudo-tree (Algorithm 13, lines 16 to 21). Each variable uses the assignments contained in the message from its parent, in order to look up a corresponding optimal assignment for itself (line 19). It then sends to each child the assignments for the variables in its separator (line 21), using the same codenames as before so as to protect agent and topology privacy. Decision privacy is only partially guaranteed, because each variable learns the values chosen for its parent and pseudo-parents — but not for other, non-neighboring variables in its separator because they are hidden by unknown codenames.

3.3.5 Algorithm Properties

This section formally proves that the overall algorithm is complete. Proofs about the other properties of the algorithm in terms of privacy guarantees will be given in Section 3.6.

Theorem 1. *P-DPOP (Algorithm 13) terminates with a probability that can be made arbitrarily close to* 1*. Upon termination, it returns an optimal solution to the DCOP.*

Proof. The root election procedure in Algorithm 14 is a viral propagation procedure that is guaranteed to terminate in exactly $3\phi_{\max}$ steps, after which a single variable (per connected component of the constraint graph) has been elected, unless a collision occurs in which two or more variables have been assigned the maximum score (line 1). If scores are drawn uniformly from $[0, s_{\max})$, then the probability of such a collision can be made arbitrarily small by choosing $s_{\max}$ sufficiently large.

Once a root variable has been chosen, the subsequent pseudo-tree construction (Algorithm 6) terminates after the exchange of exactly $2n_e$ sequential messages, where n_e is the number of edges in the pseudo-tree: two CHILD messages down and up each tree-edge, plus one CHILD message up and one PSEUDO message down each back-edge.

After exchanging codenames and obfuscation keys, which is guaranteed to require a number of messages linear in the number n of variables, the bottom-up UTIL propagation (Algorithm 15) terminates after sending exactly $(n-1)$ messages (one up each tree-edge). One can prove by induction (left to the reader) that this multi-party dynamic programming computation almost surely correctly reveals to each variable x the (obfuscated) optimal cost of its subtree's subproblem, as a function of x and possibly of ancestor variables in the pseudo-tree. This process may only fail in case of collisions of codenames, when the roots of two overlapping back-edges choose the same codenames. This can be made as improbable as desired by augmenting the size of the codename space.

Finally, the top-down VALUE propagation phase (Algorithm 13, lines 16 to 21) is guaranteed to yield an optimal assignment to each variable, after the exchange of exactly $(n-1)$ messages (one down each tree-edge). □

When it comes to the complexity of the algorithm in terms of number of messages exchanged, the bottleneck is in the election of the root variable, which requires exactly $6nn_e\phi_{\max} \in O(n^3)$ messages. However, the $(n-1)$ UTIL messages can be exponentially large: the message sent by variable x is expressed over $|sep_x|$ variable codenames (Algorithm 15, line 13), and therefore contains $O(D_{\max}^{|sep_x|})$ costs, where $D_{\max}$ is the size of the largest variable domain. The overall complexity in terms of information exchange is therefore $O(n \cdot D_{\max}^{sep_{\max}})$, where $sep_{\max} = \max_x |sep_x|$. In the worst case, the pseudo-tree consists of a single branch (necessarily involving $(n-1)$ tree-edges), and all remaining $(n_e - n + 1)$ back-edges are rooted strictly above the sender variable x and leaved below x (possibly including x). The separator of x then contains the codenames for the roots of all these back-edges, plus the parent variable of x, and therefore we

have $sep_{\max} \leq n_e - n + 2 \leq \frac{1}{2}n^2 - \frac{3}{2}n + 2$. Notice that, contrary to DPOP, in P-DPOP each UTIL message can contain multiple times the same variable, obfuscated using different codenames; therefore, the number of variables in the largest UTIL message is no longer bounded above by the induced width of the pseudo-tree.

3.3.6 P-DPOP$^-$: Trading off Topology Privacy for Performance

As previously explained, the bottleneck of the P-DPOP algorithm in terms of information exchange lies in the sizes of the UTIL messages, which are exponential in the numbers $|sep_{x_i}|$ of variable codenames they contain. It is possible to reduce the sizes of the separators, by enforcing that each agent $a(x)$ send the *same* codename $\tilde{x}$ for x to *all* of x's (pseudo-)children, contrary to was is prescribed in Algorithm 13, lines 4 to 9. Algorithms including this modification will be identified by a minus sign in exponent; P-DPOP$^-$ is actually the version of P-DPOP that we initially proposed in [Faltings et al., 2008].

As a result of this change, variables that previously may have occurred multiple times in the same UTIL message under different codenames can now only appear at most once. The worst-case complexity of P-DPOP$^-$ in terms of information exchange then becomes the same as that of the original DPOP algorithm (Section 2.4.4), in which $sep_{\max}$ is equal to the induced width of the pseudo-tree, which is bounded above by n and below by the treewidth of the constraint graph.

While the complexity of P-DPOP$^-$ is hereby decreased compared to P-DPOP, sending the same codename $\tilde{x}$ for variable x to all its (pseudo-)children also has drawbacks in terms of topology privacy. One variable y might receive a UTIL message expressed over a codename $\tilde{x}$ that it knows corresponds to x because x is a (pseudo-)parent of y, and x has also communicated the same codename $\tilde{x}$ to y. This enables y to infer that x has at least one pseudo-child below y in the pseudo-tree. Consider for instance the case of variable x_3 in Figure 3.2, which receives from x_4 a message $x_4 \rightarrow x_3$ that contains variable x_2, represented by a codename. Because x_3 is a child of x_2, it will have received the same codename for x_2, and will therefore be able to infer that x_2 has a pseudo-child below (and possibly including) x_4. This privacy leak is analyzed in more detail in Theorem 7.

3.4 Full Decision Privacy: the P$^{3/2}$-DPOP Algorithm

This section presents another variant of the P-DPOP algorithm (Section 3.3) that achieves *full* decision privacy. This results in a hybrid algorithm between the P-DPOP and P^2-DPOP (Section 3.5) algorithms, and is therefore called *P$^{3/2}$-DPOP*. Unfortunately, full decision privacy comes at a minor loss in topology privacy, as will be

discussed in Section 3.6.

3.4.1 Overview of the Algorithm

As already explained in Section 3.3.4, the P-DPOP algorithm partially violates decision privacy, because the final top-down propagation of decisions reveals to each variable the values chosen for its parent and pseudo-parent variables. The variant Algorithm 16 patches this privacy leak by removing this VALUE propagation. As a result, only the root variable is assigned a value, and for all variables to be assigned values, it is necessary to make each variable root in turn (unless the first UTIL propagation has revealed that the problem is infeasible, in which case the algorithm can terminate early).

Algorithm 16 Overall P$^{3/2}$-DPOP algorithm with full decision privacy, for variable x

1: Elect a root variable (Algorithm 14)
2: Construct a pseudo-tree and choose unique variable IDs (Algorithm 17)
3: $vector_x \leftarrow [\overbrace{1,\dots,1}^{id_x},0,\overbrace{-1,\dots,-1}^{id_x^+-id_x},1,\dots,1]$ // with id_x, id_x^+ and n^+ from Algorithm 17 (underbrace: n^+)
4: // Exchange public key shares:
5: $private_x \leftarrow$ generate a private ElGamal key for x
6: $public_x \leftarrow$ generate a set of $(id_x^+ - id_x + 1)$ public key shares for $private_x$
7: **for each** $share \in public_x$ **do** TOPREVIOUS((SHARE, $share$)) as in Algorithm 18
8: **for** $i = 1 \dots n^+$ **do**
9: Wait for and record one message (SHARE, $share$)
10: **if** $share \notin public_x$ **then** TOPREVIOUS((SHARE, $share$)) as in Algorithm 18
11: Generate the compound ElGamal public key based on all the public key shares
12: **while** $vector_x \neq \emptyset$ **do**
13: Choose a new root (Algorithm 19, Section 3.4.4)
14: Construct a new pseudo-tree rooted at the new root (Algorithm 6)
15: Exchange codenames for x and its domain D_x (Algorithm 13, lines 4 to 9)
16: Exchange obfuscation key for x (Algorithm 13, lines 11 to 14)
17: Propagate costs up the pseudo-tree (Algorithm 15)
18: **if** x is root **then** Add local constraint $x = x^*$, with x^* from Algorithm 15, line 28

To iteratively reroot the pseudo-tree, we proceed as follows. The procedure requires that each of the n variables be assigned a unique ID, which is achieved via a modification of the pseudo-tree generation Algorithm 6; this modified algorithm is presented in Section 3.4.2. Each variable x then creates a Boolean vector $vector_x$ with a single zero entry at the index corresponding to its unique ID id_x (line 3). Each vector is then

encrypted using ElGamal homomorphic encryption (Section 3.2.1). All these vectors are then passed from variable to variable in a round-robin fashion, using a circular message routing algorithm presented in Section 3.4.3, each variable applying a secret permutation to each vector and systematically re-encrypting the result to hide the permutation used (Section 3.4.4). Whenever the pseudo-tree needs to be rerooted, each variable removes the first element in its vector; the only one that decrypts to 0 identifies the new root. This approach ensures that only the newly elected root knows that it is the root, and no coalition of agents (except the grand coalition) can influence the order in which the variables are made roots, which is a desirable feature because agents might otherwise compete to be root first, in order to be the first to find assignments for their variables.

3.4.2 Assigning Unique IDs to Variables

The assignment of unique IDs to the n variables is an instance of the well-known *renaming* problem, for which multiple algorithms have been proposed in the literature on distributed algorithms. However, to our knowledge, all these algorithms focus on robustness to failures, and ignore the issue of privacy. On the contrary, in this paper we do not consider agent failures, and we rather need an algorithm that protects agent and topology privacy. To this purpose, we propose Algorithm 17, which is a modification of the pseudo-tree generation Algorithm 6, and is an improved version of the algorithm we initially proposed in [Léauté and Faltings, 2009b]. Each variable x is assigned a unique number id_x that corresponds to the order in which it is first visited during the distributed traversal of the constraint graph (or, more precisely, an upper bound thereon). This is done by appending to each CHILD message the number id of variables visited so far (lines 8, 29 and 31). Each variable adds a random number to id so as not to leak any useful upper bound on its number of neighbors (lines 5 and 15). At the end of this algorithm, the root variable discovers an upper bound n^+ on the total number of variables, and reveals it to everyone (lines 35 and 22 to 24).

3.4.3 Routing of Messages along a Circular Variable Ordering

In order to implement the round-robin exchange of vectors briefly mentioned in Section 3.4.1, the variables are ordered along a circular ordering that is mapped to the chosen pseudo-tree, as illustrated in Figure 3.5. Each variable needs to be able to send a message to the previous variable (i.e. clock-wise) in the ordering, which is a challenge in itself because of the assumption that only neighboring variables can communicate directly. Furthermore, to protect agent and topology privacy, no agent should know the overall circular ordering. Algorithm 18 is a routing algorithm that solves this issue.

Consider for instance a message M that agent $a(x_1)$ wants to send to the previous

Algorithm 17 Pseudo-tree and unique ID generation algorithm for variable x

```
1:  if x has at least one neighbor then
2:      open_x ← ∅
3:      if x has been elected as the root then
4:          id_x ← 0
5:          id_x^+ ← id_x + rand(incr_min ... 2incr_min)
6:          open_x ← all neighbors of x
7:          Remove a random neighbor y_0 from open_x and add it to children_x
8:          Send the message (CHILD, id_x^+ + 1) to y_0
9:      loop
10:         Wait for an incoming message (type, id) from a neighbor y_i
11:         if open_x = ∅ then // first time x is visited
12:             open_x ← all neighbors of x except y_i
13:             parent_x ← y_i
14:             id_x ← id
15:             id_x^+ ← id_x + rand(incr_min ... 2incr_min)
16:         else if type = CHILD and y_i ∈ open_x then
17:             Remove y_i from open_x and add it to pseudo_children_x
18:             Send message (PSEUDO, id) to y_i
19:             next
20:         else if type = PSEUDO then
21:             Remove y_i from children_x and add it to pseudo_parents_x
22:         else if type = NBRVARS then
23:             n^+ ← id // upper bound on the true number of variables n
24:             break
25:         // Forward the CHILD message to the next open neighbor:
26:         Choose a random y_j ∈ open_x
27:         if there exists such a y_j then
28:             Remove y_i from open_x and add it to children_x
29:             Send the message (CHILD, max(id, id_x^+ + 1)) to y_j
30:         else if x is not the elected root then // backtrack
31:             Send message (CHILD, max(id, id_x^+ + 1)) to parent_x
32:         else
33:             n^+ ← id // upper bound on the true number of variables n
34:             break
35:     Send message (NBRVARS, n^+) to all children of x
```

variable (which is x_4, but $a(x_1)$ does not know this piece of information). Agent $a(x_1)$ wraps M into a PREV message that it sends to its parent variable x_4 (line 2). Because the sender variable x_1 is x_4's first (and only) child, $a(x_4)$ infers that it should deliver M to itself (line 9). Consider that $a(x_4)$ now wants to forward M to its previous variable (x_5, which $a(x_4)$ does not know because they are not neighbors). Like before, $a(x_4)$ sends a message (PREV, M) to its parent variable x_3, which then reacts by sending

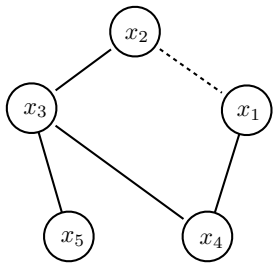


Figure 3.5: The (counter-clock-wise) circular ordering based on Figure 3.2.

Algorithm 18 Sending a message M clock-wise in the circular variable ordering.

Procedure: TOPREVIOUS(M) for variable x
1: **if** x is the root of the pseudo-tree **then** Send the message (LAST, M) to x's last child
2: **else** Send the message (PREV, M) to x's parent

Procedure: ROUTEMESSAGES() for variable x
3: **loop**
4: Wait for an incoming message ($type, M$) from a neighbor y_i
5: **if** $type =$ LAST **then**
6: **if** x is a leaf **then** Deliver message M to x
7: **else** Send the message (LAST, M) to x's last child
8: **else if** $type =$ PREV **then**
9: **if** y_i is x's first child **then** Deliver the message M to x
10: **else** Send the message (LAST, M) to the child before y_i in x's list of children

a message (LAST, M) to its last child preceding x_4 in its list of children, which is x_5 (line 10). LAST messages indicate that the payload M should be delivered to the last leaf of the current subtree (line 7); therefore, $a(x_5)$ delivers M to itself (line 6) since it has no children. If the root wants to send a message to its previous variable, it also uses a LAST message to forward it to the last leaf of the overall pseudo-tree (line 1).

3.4.4 Choosing a New Root Variable

When it comes to electing a new root, the agents proceed as in Algorithm 19. Each variable x first starts the procedure SHUFFLEVECTORS(), which is run only once — this is a performance improvement over the initial algorithm we proposed in [Léauté and Faltings, 2009b], in which the shuffling was performed at each iteration. When this initial procedure reaches line 17, x's vector $vector_x$ (Algorithm 16, line 3) will have been randomly shuffled, and will contain a single entry that decrypts to 0 at the position corresponding to the number of the iteration at which x will be the root.

The procedure SHUFFLEVECTORS() proceeds in four rounds. During round 1 (started on

Algorithm 19 Algorithm to choose a new root, for variable x

Procedure: SHUFFLEVECTORS() for variable x

1: $myID \leftarrow$ large random number
2: $p_x \leftarrow$ random permutation of $[1 \dots n^+]$
3: // Propagate x's encrypted vector backwards along the circular ordering
4: $vector_x \leftarrow E(vector_x)$ // encrypts the vector using the compound public key
5: TOPREVIOUS((VECT, $myID, vector_x$, 1)) as in Algorithm 18 and Section 3.4.3
6: // Process all received vectors
7: **while true do**
8: Wait for a message (VECT, $id, vector, round$) from the next variable
9: **if** $round = 1$ **then**
10: **if** $id \neq myID$ **then for** $j = (id_x + 1) \dots id_x^+$ **do** $vector[j] \leftarrow -1$
11: **else** $round \leftarrow round + 1$ // x's vector; move to next round
12: **if** $round > 1$ and x is the current root **then**
13: $round \leftarrow round + 1$ // the root starts each round except the first
14: **if** $round = 3$ **then**
15: $vector \leftarrow p_x(vector)$ // shuffle the vector
16: **else if** $round = 4$ and $id = myID$ **then** // done processing $vector_x$
17: $vector_x \leftarrow vector$
18: **continue**
19: // Pass on the vector backwards along the circular ordering
20: $vector \leftarrow E(vector)$ // re-encrypts the vector using the compound public key
21: TOPREVIOUS((VECT, $id, vector, round$)) as in Algorithm 18 and Section 3.4.3

Procedure: REROOT() for variable x

22: **repeat**
23: $entry \leftarrow$ DECRYPT($pop(vector_x)$) // as in Algorithm 20
24: **until** $entry \neq -1$
25: **if** $entry = 0$ **then** x is the new root

line 5), variable x's vector $vector_x$ makes a full round along the circular ordering, during which each other variable x' overwrites some of the entries with -1 (line 10), at the same positions it did for its own vector (Algorithm 16, line 3). These -1 entries account for the "phantom variables" that each variable implicitly introduces when overstating its id_x as id_x^+ (Algorithm 17). Once x has received back its own vector, it enters round 2 (line 11), which is an incomplete round during which the vector is passed on until it reaches the current root (line 12). The root then starts round 3 (line 13), during which each variable shuffles the vector using its secret permutation p_x (line 15), using a *mix-net* mechanism similar to the one in [Silaghi, 2005a]. The incomplete round 4 finally returns the fully shuffled vector to its owner (line 17). Each agent's vector therefore performs 3 full rounds, except for the root variable's vector, which performs only 2 full rounds, since the incomplete rounds 2 and 4 can be skipped.

When it is necessary to reroot the variable ordering at the beginning of each iteration of P$^{3/2}$-DPOP, each variable x then calls the procedure REROOT(), which consists in removing and decrypting the first element of its vector. Entries that decrypt to -1 correspond to phantom variables, and must be skipped. The single entry that decrypts to 0 identifies the new root. The decryption process, Algorithm 20, is a collaborative effort that involves each variable using its private ElGamal key to partially decrypt the cyphertext. The cyphertext travels around the circular variable ordering in the same way as the vectors, until it gets back to its sender variable, which can finally fully decrypt it.

Algorithm 20 Collaborative decryption of a multiply-encrypted cyphertext e

Procedure: DECRYPT(e) for variable x
1: $codename \leftarrow$ large random number
2: $codenames_x \leftarrow codenames_x \cup \{codename\}$
3: TOPREVIOUS((DECR, $codename$, e)) as in Algorithm 18
4: Wait for message (DECR, $codename$, e') from next variable in the ordering
5: **return** decryption of e using x's private key

Procedure: COLLABORATIVEDECRYPTION() for variable x
6: **loop**
7: Wait for a message (DECR, c, e) from next variable in the ordering
8: **if** $c \notin codenames_x$ **then**
9: $e' \leftarrow$ partial decryption of e using x's private key
10: TOPREVIOUS((DECR, c, e')) as in Algorithm 18

3.4.5 Algorithm Properties

The following theorem states that the P$^{3/2}$-DPOP algorithm (Algorithm 16) is complete. The analysis of the algorithm in terms of privacy guarantees is delayed to Section 3.6.

Theorem 2. *The P$^{3/2}$-DPOP algorithm terminates with a probability that can be made arbitrarily close to* 1. *Upon termination, it returns an optimal solution to the DCOP, if there exists one.*

Proof. On the basis of Theorem 1, it remains to prove that the rerooting Algorithm 19 terminates and is correct, and that the overall algorithm remains correct. The latter is easy to prove: at each iteration, an optimal value is found for the root variable, and that value is necessarily consistent with the chosen assignments to previous roots since these assignments are enforced by new, additional constraints (Algorithm 16, line 18).

When it comes to the termination and correctness of the rerooting procedure, Algorithm 17 ensures that each of the n variables gets a unique ID in $0 \ldots (n^+ - 1)$. Therefore, each variable has a 0 entry at a unique position in its vector (Algorithm 16, line 3). Round 1 of Algorithm 19 also makes sure that all vectors have -1 entries at the same

positions. This ensures that exactly one variable will become the new root at each iteration, since all vectors are applied the same sequence of permutations, and no variable will be root twice. □

In terms of complexity, P$^{3/2}$-DPOP proceeds in a similar way to P-DPOP (Section 3.3.5), except that the bottom-up UTIL propagation phase is repeated n times (each time with a different root variable). The overall complexity in information exchange therefore becomes $O(n^2 \cdot D_{\max}^{sep'_{\max}})$, where $sep'_{\max}$ is the maximum separator size over all variables, *and* over all iterations, which therefore is likely to be higher than the exponent for P-DPOP. The information exchanged by the rerooting protocol is negligible in comparison. In terms of number of ElGamal cryptographic operations, the rerooting procedure requires a total of $n(3n-1)n^+ \in O(n^3)$ encryptions: each of the n variables (re-)encrypts $(3n-1)$ vectors of size n^+ (each variable's vector performs 3 full rounds, except for the root's vector, which performs only 2 full rounds), with $n^+ \leq n + n \cdot 2incr_{\min}$, where $incr_{\min}$ is a constant input parameter of the algorithm. The procedure also requires a total of $n^2 n^+ \in O(n^3)$ collaborative decryptions: each of the n variables (partially) decrypts n vectors or size n^+.

3.5 Full Constraint Privacy: the P^2-DPOP Algorithm

The algorithms in the previous sections address constraint privacy through the obfuscation of costs by adding large random numbers (Section 3.3.3). As later discussed in Section 3.6.3, this method is not fully, cryptographically secure, and only guarantees partial constraint privacy.

We now describe how this obfuscation scheme can be replaced with ElGamal homomorphic encryption (Section 3.2.1) to achieve full constraint privacy, which corresponds to the P^2-DPOP algorithm we originally proposed in [Léauté and Faltings, 2009b]. An important limitation of the ElGamal cryptographic scheme is that it is not *fully homomorphic*: it is possible to compute the $\min$ of two encrypted numbers, but it is only possible to compute the sum of an encrypted number with a *cleartext* number. A consequence of this limitation is that the bottom-up UTIL propagation has to be performed on a variable ordering such that each variable can have only *one* child, i.e. a *linear* variable ordering. Otherwise, in a pseudo-tree variable ordering, a variable with two children would not be able to join the two encrypted UTIL messages sent by the children.

Another limitation of the ElGamal scheme as described in Section 3.2.1 is that, when applied to numbers (rather than Booleans), it only works on integers in $[0, c_{\max}] \cup \{+\infty\}$, where $c_{\max} \in \mathbb{N}$ is a free parameter. Therefore, when it comes to optimization, the P^2-DPOP algorithm is only going to be applicable to DisWCSPs for which an upper bound $c_{\max}$ on the cost of the optimal (feasible) solution is known; not to general

DCOPs. Notice however that, contrary to P$^{<2}$-DPOP, the input DisWCSP may include infinite costs, and, when applied to pure DisCSPs, the problems do not need to be first reformulated into Max-DisCSPs.

3.5.1 Encrypted UTIL Propagation along a Linear Variable Order

In contrast with Figure 3.2, which illustrates multi-party dynamic programming on a pseudo-tree variable ordering, Figure 3.6 shows (in cleartext) how the same computation can be performed on a linear ordering (corresponding to the circular variable ordering in Figure 3.5). This assumes that a circular communication structure has preliminarily been set up as described in Section 3.4.3.

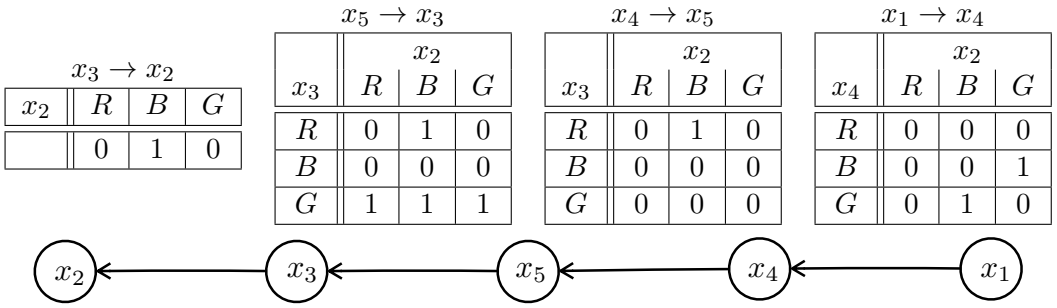


Figure 3.6: UTIL propagation (in cleartext) in P^2-DPOP for Figure 2.1.

Algorithm 21 P^2-DPOP's UTIL propagation procedure, for variable x

1: // Join local constraints:
2: $m(x,\cdot) \leftarrow \sum_{c \in \{c' \in \mathcal{C} \mid x \in scope(c') \wedge scope(c') \cap (children_x \cup pseudo\_children_x) = \emptyset\}} c(x,\cdot)$
3: // Apply codenames:
4: **for** each $y_i \in \{parent_x\} \cup pseudo\_parents_x$ **do**
5: $m(x,\cdot) \leftarrow$ replace (y_i, D_{y_i}) in $m(x,\cdot)$ with $(\tilde{y_i^x}, \tilde{D_{y_i}^x})$ from Algorithm 13, line 4, and apply the permutation $\sigma_{y_i}^x$ to $\tilde{D_{y_i}^x}$
6: // Join with received message:
7: Wait for the message (UTIL, $m'(\cdot)$) from the next variable in the ordering
8: **for** each $z \in children_x \cup pseudo\_children_x$ **do**
9: $m'(\cdot) \leftarrow$ identify $(\tilde{x^z}, \tilde{D_x^z})$ as (x, D_x) in $m'(\cdot)$ (if $\tilde{x^z}$ is present)
10: $m(x,\cdot) \leftarrow m(x,\cdot) + m'(\cdot)$
11: // Project out x:
12: **if** x is not the root variable **then**
13: $m(\cdot) \leftarrow \min_x m(x,\cdot)$
14: $m(\cdot) \leftarrow E(m(\cdot))$ // re-encrypts costs using the compound public key
15: TOPREVIOUS((UTIL, $m(\cdot)$)) as in Algorithm 18
16: **else** $(x^*, c^*) \leftarrow$ OPTIMIZE$(m(x,\cdot))$ as in Algorithm 22

Algorithm 21 gives the detailed pseudocode for this procedure, and is intended as a replacement for line 17 in Algorithm 16. The differences with the pseudo-tree-based Algorithm 15 are the following. First notice that, in the case of the linear ordering, a variable's local subproblem no longer necessarily involves its parent variable in the ordering (line 2), just like x_4 shares no constraint with x_5 in Figures 3.5 and 3.6. The next, more important difference is that variable x no longer partially de-obfuscates its cost matrix before projecting itself (Algorithm 15, line 19). The reason for this is that the ElGamal cryptographic scheme is homomorphic, and therefore it is no longer necessary to first decrypt the costs in order to project x using the operator $\min_x$. Only the root variable requires decryption (Algorithm 21, line 16), because it needs to find a value x^* for its variable x whose encrypted cost is the optimal one. This is described in the following section.

3.5.2 Decrypting an Optimal Value for the Root Variable

Algorithm 22 is a dichotomy procedure to find an optimal assignment to the root variable x within its domain D_x. Like for the vector heads in Section 3.4.4, the decryption of ElGamal-encrypted costs is a collaborative process that requires each variable to partially decrypt the cyphertext using its private key (Algorithm 20). This dichotomy algorithm involves at most $2\lceil \log_2 |D_x| \rceil$ such decryptions (assuming $|D_x| > 1$). In the DisCSP case, this worst-case number of decryptions can be brought down to $\lceil \log_2 |D_x| + 1 \rceil$, using Algorithm 23.

Algorithm 22 Finding an optimal value for x in the encrypted cost matrix $m(x)$

Procedure: OPTIMIZE($m\,(x = x_{i_l} \dots x_{i_r})$)
1: **if** $i_l < i_r$ **then** // cut in half the remaining subdomain:
2: $\quad I_l \leftarrow \left[i_l, \left\lfloor \frac{i_l+i_r}{2} \right\rfloor\right]$
3: $\quad I_r \leftarrow \left[\left\lfloor \frac{i_l+i_r}{2} \right\rfloor + 1, i_r\right]$
4: $\quad c_l^* \leftarrow$ DECRYPT($\min_{i\in I_l} m\,(x = x_i)$) as in Algorithm 20
5: $\quad c_r^* \leftarrow$ DECRYPT($\min_{i\in I_r} m\,(x = x_i)$) as in Algorithm 20
6: $\quad$ **if** $c_l^* \leq c_r^*$ **then**
7: $\quad\quad$ **if** $I_l = \{i_l\}$ **then return** (x_{i_l}, c_l^*)
8: $\quad\quad$ **else return** OPTIMIZE($m\,(x = x_{i\in I_l})$)
9: $\quad$ **else**
10: $\quad\quad$ **if** $I_r = \{i_r\}$ **then return** (x_{i_r}, c_r^*)
11: $\quad\quad$ **else return** OPTIMIZE($m\,(x = x_{i\in I_r})$)
12: **else** // only one value possible for x
13: $\quad$ **return** $(x_{i_l}$, DECRYPT $(m(x_{i_l})))$ as in Algorithm 20

Algorithm 23 Finding a feasible value in the encrypted Boolean matrix $m(x)$

Procedure: FEASIBLEVALUE($m\,(x = x_{i_l} \dots x_{i_r})$)

1: **if** $i_l < i_r$ **then** // cut in half the remaining subdomain:
2: $I \leftarrow \left[i_l, \left\lfloor \frac{i_l+i_r}{2} \right\rfloor\right]$
3: $feasible \leftarrow$ DECRYPT$\left(\bigvee_{i\in I} m\,(x = x_i)\right)$ as in Algorithm 20
4: **if** $feasible =$ `true` **then return** FEASIBLEVALUE($m\,(x = x_{i\in I})$)
5: **else return** FEASIBLEVALUE$\left(m\left(x = x_{i\in([i_l,i_r]-I)}\right)\right)$
6: **else** // only one value remains for x
7: $feasible \leftarrow$ DECRYPT($m\,(x = x_{i_l})$) as in Algorithm 20
8: **if** $feasible =$ **true then return** x_{i_l}
9: **else return null**

3.5.3 Algorithm Properties

Like for the previous algorithms, the discussion about the properties of P^2-DPOP in terms of privacy is postponed to Section 3.6. The following theorem focuses on the termination and completeness properties.

Theorem 3. *The P^2-DPOP algorithm terminates with a probability that can be made arbitrarily close to* 1. *Upon termination, it returns an optimal solution to the DCOP.*

Proof. The termination property follows directly from Theorem 2, and by the fact that the message routing procedure in Algorithm 18 guarantees that all UTIL messages eventually reach their destinations. When it comes to completeness, the homomorphic property of the ElGamal cryptographic scheme ensures that the projection of a variable x out of an encrypted cost matrix is correct, and that the UTIL message received by each variable in the linear ordering still summarizes the (encrypted) optimal cost of the lower agents' aggregated subproblems, as a function of variables higher in the ordering. In particular, the UTIL message received by the root allows it to find a value for its variable that is optimal for the overall problem. □

The analysis of the complexity of the algorithm in terms of information exchanged remains similar to the analysis in Section 3.4.5, i.e. it is $O(n^2 \cdot D_{\max}^{sep''_{\max}})$, but $sep''_{\max}$ is now the maximum separator size *along the successive linear variable orderings*, instead of along the pseudo-trees. The requirement that each variable may have at most one child is likely to make this exponent increase. In terms of number of ElGamal cryptographic operations, in addition to the cost of rerooting the variable ordering (Section 3.4.5), the algorithm also requires $O(n^2 \cdot D_{\max}^{sep''_{\max}})$ encryptions, and only $O(n \log D_{\max})$ collaborative decryptions.

3.6 Privacy Analysis

The following analysis of the privacy properties of the various algorithms proposed in this thesis is based on the standard assumption that *all participants are honest, but curious* [Goldreich, 2009]. This means that each agent is assumed to faithfully execute the corresponding algorithms, but is interested in learning as much as possible about other agents, based on the messages it receives during the algorithm execution. As already explained in Section 3.2.2, the initial knowledge of each agent is assumed to only include the following:

- the names and domains of all the variables it controls;
- all constraints involving any one of the variables it controls;
- the names, domains and owners of all variables that are constrained with any one of the variables it controls.

This section presents privacy guarantees about which *additional* information the agent *provably cannot* discover about the overall problem, excluding the *semi-private information* (Definition 8) that may inevitably be revealed by the values chosen for its variables. As described in Section 3.1, this information can be classified in four categories: *agent privacy* (Section 3.6.1), *topology privacy* (Section 3.6.2), *constraint privacy* (Section 3.6.3), and *decision privacy* (Section 3.6.4). Table 3.1 summarizes the privacy properties of each algorithm along these four categories.

Table 3.1: Privacy guarantees of various algorithms.

privacy type:	agent	topology	constraint	decision
P-DPOP$^{(-)}$	full	partial	partial	partial
P$^{3/2}$-DPOP$^{(-)}$	full	partial	partial	full
P^{2}-DPOP$^{(-)}$	full	partial	full	full
MPC-Dis(W)CSP4	-	partial	partial	partial

3.6.1 Agent Privacy

Recall (Definition 9) that *agent privacy* relates to no agent being able to discover the identities of non-neighboring agents (i.e. agents it does not share a constraint with). There are only two ways the identity of an agent A could be leaked to a non-neighbor B:

1. The algorithm can require A and B to exchange messages with each other;
2. Agent A can receive a message whose content refers identifiably to B.

The former case can never happen in any of our algorithms, because they only ever involve exchanging messages with neighboring agents. The latter case is addressed mainly through the use of *codenames*.

Agent Privacy in P-DPOP$^{(-)}$

Theorem 4. *The P-DPOP$^{(-)}$ algorithms (Algorithm 13 and its variant in Section 3.3.6) guarantee agent privacy.*

Proof. The P-DPOP$^{(-)}$ algorithms proceed in the following sequential phases:

Root variable election (Algorithm 14) The messages exchanged only contain large, random numbers, which cannot be linked to any specific agent. In particular, the elected root variable is only revealed to the agent that owns it; other agents cannot discover the identity of this owner agent.

Pseudo-tree construction (Algorithm 6) All messages are empty tokens that contain no payload other than their types (CHILD or PSEUDO).

Bottom-up UTIL propagation (Algorithm 15) Each UTIL message contains only a function (line 12) over a set of variables, whose names, if transmitted in clear text, could identify their owner agent to the recipient of the message. To patch this potential violation of agent privacy, the P-DPOP$^{(-)}$ algorithms replace all variable names with secret, random codenames, as follows.

Consider a variable x in the pseudo-tree. First note that no UTIL message sent by x or by any ancestor of x can be a function of x. The message sent by x is not a function of x, because x is projected out before the message is sent (line 23). Variable x cannot re-appear in any UTIL message higher in the pseudo-tree, because no agent's local problem can involve any variable lower in the pseudo-tree (line 3).

Similarly, consider now the UTIL message sent by a descendant y of x in the pseudo-tree, and assume first that y is a leaf of the pseudo-tree. Since y has no children, it will not receive any UTIL message, and therefore the UTIL message it sends can only be a function of the variables in its local problem. If this local problem involves x, y will replace x by its codename $\tilde{x^y}$ (line 6) before it sends its UTIL message. One can then prove by inference that no UTIL message sent by any variable between y and x will contain x either; it can only (and not necessarily) contain one or several of its codenames $\tilde{x^{y_i}}$.

Since the codenames $\tilde{x^{y_i}}$ are random numbers chosen by x (Algorithm 13, line 6), and only communicated (through channels that are assumed secure) to the respective neighbors y_i of x (Algorithm 13, line 9), no non-neighbor of x receiving a message involving any $\tilde{x^{y_i}}$ can discover the identity of its owner agent.

The domain D_x of variable x could also contain values that might identify its owner agent. To fix this privacy risk, x's domain is also replaced by obfuscated domains $\tilde{D_x^{y_i}}$ of random numbers, similarly to the way variable names are obfuscated. In this paper, we also make the simplifying assumption that all variables have the same domain size, so that the domain size of a variable does not give any information about the identity of its owner agent. This assumption naturally holds in many problem classes; when it does not hold, variable domains can be padded with fake values in order to make them all have the same size.

Top-down VALUE propagation (Section 3.3.4) The VALUE messages exchanged only contain assignments to variables (Algorithm 13, line 21), which are obfuscated using the same codenames as in the previous phase.

This concludes the proof that, in the P-DPOP$^{(-)}$ algorithms, no agent can receive any message from which it can infer the identity of any non-neighboring agent. Agent privacy is therefore guaranteed. □

Agent Privacy in P$^{3/2}$-DPOP$^{(-)}$

Theorem 5. *The P$^{3/2}$-DPOP$^{(-)}$ algorithms (Algorithm 16 and its variant in Section 3.3.6) guarantee agent privacy.*

Proof. Recall that the P$^{3/2}$-DPOP$^{(-)}$ algorithms differ from P-DPOP$^{(-)}$ in the introduction of a rerooting procedure, which replaces the top-down VALUE propagation in P-DPOP$^{(-)}$. This rerooting procedure itself requires the introduction of a modified pseudo-tree construction algorithm that also assigns unique IDs to variables, and of a message routing procedure along a circular variable ordering. The rerooting procedure also involves the collaborative decryption of ElGamal cyphertexts.

Unique variable ID assignment (Algorithm 17) Each message only contains an upper bound on the number of variables already visited, which cannot be used to make inferences about the identities of agents.

Circular message routing (Algorithm 18) The goal of this algorithm is precisely to address agent privacy issues in the pseudo-tree rerooting procedure, which involves each variable sending a message to the previous variable in a circular ordering of the variables. There is no guarantee that there exist a circular ordering such that any two consecutive variables are owned by neighboring agents, which is necessary to protect agent privacy. Therefore, Algorithm 18 is responsible for routing these messages through paths that only involve communication between neighboring agents.

The routing procedure itself only involves encapsulating the routed messages inside PREV or LAST messages, which do not contain any other payload. Therefore, as long as the routed messages do not contain information that can be used to identify a non-neighboring agent, the routing procedure guarantees agent privacy.

Pseudo-tree rerooting (Algorithm 19) The VECT messages exchanged by the preliminary SHUFFLEVECTORS() procedure contain three payloads: a variable ID, a vector of ElGamal encryptions of -1, 0 or 1 (Algorithm 16, line 3), and a round number. The variable ID is used by the recipient agent to detect whether the vector is its own (modified) vector, of whether it belongs to another agent. This ID is a large random number chosen by the owner agent (Algorithm 19, line 1), and therefore cannot be linked to the identity of this owner agent by any other agent. The ElGamal vector and the round number do not contain either any information that could be used to identify the agent. Also note that the procedure used to exchange ElGamal public key shares (Algorithm 16, lines 4 to 11) does not leak any information about agents' identities.

The SHUFFLEVECTORS() procedure is only run once, as a preliminary step to rerooting the pseudo-tree. Every time such a rerooting must be performed, it is the REROOT() procedure that is called. This procedure makes use of the collaborative decryption algorithm, whose properties in terms of agent privacy are discussed below.

Collaborative decryption (Algorithm 20) This procedure involves exchanging messages that only contain the ElGamal cyphertext to be decrypted, and a codename that saves the same purpose as the variable ID in Algorithm 19. This codename is similarly set to a large random number chosen by the current agent, and cannot be linked to the identity of this agent by any other agent.

This concludes the proof that the P$^{3/2}$-DPOP$^{(-)}$ algorithms guarantee agent privacy. □

Agent Privacy in P^2-DPOP$^{(-)}$

Theorem 6. *The P^2-DPOP$^{(-)}$ algorithms (Section 3.5 and the variant in Section 3.3.6) guarantee agent privacy.*

Proof. The only changes introduced in P^2-DPOP$^{(-)}$ with respect to P$^{3/2}$-DPOP$^{(-)}$ are a modified UTIL propagation, and an additional step to find an optimal value for the root variable.

ElGamal UTIL propagation (Algorithm 21) From the point of view of agent privacy, this is the same procedure as Algorithm 15, but using Algorithm 18 for message

routing, both of which algorithms have already been shown to guarantee agent privacy.

Root variable assignment (Algorithm 23) This consists in iteratively calling the procedure in Algorithm 20, which has also already been shown to guarantee agent privacy.

This concludes the proof that the P^2-DPOP$^{(-)}$ algorithms guarantee agent privacy. □

3.6.2 Topology Privacy

Recall (Definition 10) that topology privacy is about preventing any agent from discovering information about topological constructs of the constraint graph that it is not involved in.

Topology Privacy in P-DPOP$^-$

Theorem 7. *The P-DPOP$^-$ algorithm (Section 3.3.6) guarantees* partial *topology privacy. An upper bound on the diameter of the constraint graph is leaked, and a variable* might *also be able to discover:*

- *a lower bound on a neighbor variable's degree in the constraint graph;*
- *a lower bound on the total number of variables;*
- *bounds on the length of a cycle it is involved in.*

Proof. The following analyses the properties of the P-DPOP$^-$ algorithm, from the point of view of topology privacy.

Root variable election (Algorithm 14) Conceptually, this algorithm uses a viral propagation mechanism to compute and reveal to all agents the maximum variable score across the constraint graph. Each variable's score is chosen as a large random number (line 1), so that revealing the maximum score to all agents does not reveal any topological information about the variable that has been assigned this maximum score. To decide when to stop the viral propagation, the algorithm assumes that the agents know an upper bound $\phi_{\max}$ on the constraint graph diameter, which is a minor leak of topology privacy.

During each round of this viral propagation procedure, each variable receives scores $max_1, \ldots, max_k$ from its k neighboring variables, and then replies with a score max at least equal to $\max(max_1, \ldots, max_k)$ (the same max is sent to all

neighbors). By analyzing the history of scores received from its neighbors, an agent may be able to make inferences about the topology of the constraint graph; Algorithm 14 includes a number of mechanisms to prevent this leak of topology privacy.

Let us first consider whether it is possible for a variable x to discover the existence of non-neighboring variables. In a pure viral propagation procedure, during the first round, x would receive the scores of all its neighbors. Any message received during a later round and containing a new, different score would reveal the existence of a non-neighboring variable with that score. To prevent such an inference, agents are required to under-report their respective scores during a positive, bounded number of rounds. More precisely, if an agent's score is greater than the maximum score it has seen in previous rounds, instead of reporting its true score, it sends to its neighbors a random number lower than its score. This number is still required to be greater than the maximum score seen so far, so that neighbors cannot detect that the agent lied. As long as its neighboring variable y could still be lying, x cannot conclude whether an unknown max received from y is due to a lie by y or whether y received it from a third variable at the previous round. However, to guarantee termination, agents can only be allowed to lie during a bounded number of rounds (in this algorithm, at most $2\phi_{\max}$ rounds). Therefore, after $(2\phi_{\max}+1)$ rounds, if x receives an unknown max from y, it will still be able to infer that y has another neighbor, which cannot be a neighbor of x otherwise x would have received the max from that neighbor at the previous round. However, nothing else is leaked about the identity of that other variable.

Let us now consider whether a variable x can discover the existence of an edge in the constraint graph between two neighboring variables y and z, based on the respective max values it receives from them. The effect of the existence of an edge between y and z is that, if z sends $max_z^{(k)}$ at round k, x and y will both receive it at round $(k+1)$. Therefore, if x also receives $max_z^{(k)}$ from y at round $(k+2)$, x might be tempted to infer that y probably received it from z at iteration $(k+1)$, and therefore that y and z are neighbors. However this inference would be unfounded, because it is actually equally probable that y and z are non-neighbors, and that $max_z^{(k)}$ was originally sent at round $j < k$ by another variable that is at distance $(k-j)$ from z and $(k-j+1)$ from y. This is illustrated in Figure 3.7. To sum it up, x might be able to detect with some probability the existence of a cycle of length at most $2k+1$ involving x, y and z. In the most extreme case of $k=1$, in this context x might be able to discover that y and z are neighbors. Notice however that it might be possible to fix this privacy leak by having each agent report a *different* under-estimated $\max$ to each of its neighbors in a given round; the consequences of such a change to the algorithm remain to be investigated.

Finally, it is important to ask oneself whether the algorithm leaks any information

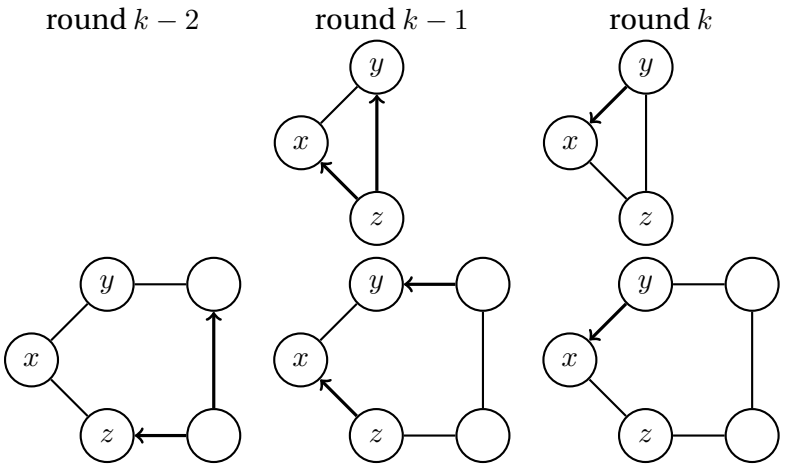


Figure 3.7: Two indistinguishable timelines from x's point of view.

about the *position* of the elected variable in the constraint graph. Notice that if such information were leaked, this would not necessarily be considered a violation of topology privacy (except if it revealed the existence of a non-neighbor variable); however, the knowledge of the position of the root of the pseudo-tree could be useful to the agents in order to make inferences during the following steps of the algorithm. In a pure viral propagation procedure, by observing the round at which the maximum score was received from each neighbor, an agent would be able to infer the distance (in number of edges) between that neighbor and the elected root. In particular, if the root were a neighbor, the agent would discover it. Algorithm 14 prevents this by requiring that agents lie at least $\phi_{\max}$ times, where $\phi_{\max}$ is an upper bound on the graph diameter, i.e. on the distance between any pair of variables. This way, the maximum score will only be received after at least $\phi_{\max}$ rounds, which therefore does not leak any useful information.

Pseudo-tree construction (Algorithm 6) This algorithm performs a depth-first traversal of the constraint graph, starting with the elected root. A single CHILD token is passed around by each variable to its unvisited neighbors. When the token is received by a variable that has already been visited, a back-edge is discovered, and the recipient replies with a PSEUDO token. When a variable has finished traversing its subtree, it returns the CHILD token to its parent.

After running the algorithm, each variable's knowledge of the chosen pseudo-tree is limited to the following:

- The variable knows its parent variable, which is necessarily a neighbor in the constraint graph, therefore this does not violate topology privacy.
- The variable knows its pseudo-parents, which are also neighbors. For each

such pseudo-parent, the variable discovers the existence of a cycle in the constraint graph, involving itself, its parent, and the given pseudo-parent. This is tolerated by the definition of topology privacy, since the variable is involved in this cycle.

Since there exists a treepath in the pseudo-tree from the root, through all its pseudo-parents, to itself, and since there exists a back-edge between itself and the highest of its pseudo-parents, the variable also discovers the existence of a long cycle involving all these variables. This is also tolerated by our definition of topology privacy, since the variable is also involved in this cycle. Furthermore, notice that the variable does not know the order in which its pseudo-parents are arranged in the pseudo-tree and in the cycle.

Notice also that the variable *does not* discover the existence of possible ancestors in the pseudo-tree other than its neighbors.

- The variable knows its children, which are also neighbors. Because each child is in a different branch of the pseudo-tree, the variable can infer that there does not exist any constraint between any of its children. This is tolerated by the definition of topology privacy.
- The variable knows its pseudo-children, which are also neighbors. Furthermore, it knows below which of its children each pseudo-child is situated in the pseudo-tree. Therefore, for each pseudo-child below any given child, the variable discovers the existence of a cycle involving itself, the child and the pseudo-child. This is tolerated since the variable is involved in this cycle.

  Moreover, like for its parent and pseudo-parents, for each child, the variable discovers the existence of a long cycle involving itself, the given child, and all pseudo-children below that child. And, unlike for pseudo-parents, the variable can infer the order of its pseudo-children in the cycle, based on the order in which the CHILD token has been received from the pseudo-children. This is also tolerated by the definition of topology privacy, since the variable is involved in that cycle.

  Finally, like for children, the variable discovers the absence of constraints between any two (pseudo-)children located below two different children, which is tolerated. The variable *does not* discover the existence of possible descendants other than its neighbors.

Bottom-up UTIL propagation (Algorithm 15) During this phase, each variable x receives one UTIL message from each of its children, containing a cost function, whose scope might reveal topological information. Notice that each variable y in this scope is represented by a secret codename $\tilde{y}$ (and so is the corresponding domain of values); however x may be able to decrypt the codename $\tilde{y}$, if and only if y is a neighbor of x (or is x itself), because y has then sent the *same* codename $\tilde{y}$ to all its neighbors. This results in a leak of topology privacy: x discovers, for each neighboring ancestor y, whether y has at least one other neighbor below a given

child of x. But it cannot discover exactly how many of these other neighbors there are; in particular, there might be only one, and it might be one of x's known pseudo-children (if any).

Furthermore, if x is unable to decrypt the codename $\tilde{y}$, it can infer the existence of another, non-neighboring ancestor that this codename corresponds to. This is another breach of topology privacy. Because y sent the same codename $\tilde{y}$ to all its neighbors, x is also able to discover whether that other ancestor (which it only knows as $\tilde{y}$) has at least one neighbor below each of x's children (and that neighbor might be one of x's known pseudo-children, if any). Moreover, since codenames are large, random numbers that are almost surely unique, x may discover the existence of *several, distinct* such non-neighboring ancestors.

Top-down VALUE propagation (Section 3.3.4) Each variable receives a message from its parent, which can only contain codenames for variables and variable values that were already present in the UTIL message received during the previous phase.

This concludes the proof that P-DPOP$^-$ only *partially* protects topology privacy. Notice that the limited topology information that is leaked to any variable can only concern its branch in the pseudo-tree; no information can be leaked about any other branch, not even their existence. □

Topology Privacy in P-DPOP

Theorem 8. *The use of different codenames for each (pseudo-)child improves the topology privacy in P-DPOP compared to P-DPOP$^-$, but P-DPOP* can *still leak the same bounds as P-DPOP$^-$.*

Proof. Consider a variable x that receives a UTIL message including a secret codename $\tilde{y}$ that corresponds to variable y (assumed $\neq x$). Because y now sends a *different* codename to each of its neighbors, x is no longer ever able to decrypt $\tilde{y}$, even if y is a neighbor of x. As a consequence, x is no longer able to infer whether $\tilde{y}$ refers to a known neighbor of x, or to an unknown, non-neighboring variable. However, since each codename now corresponds to a unique back-edge in the pseudo-tree, for each pair (α, β) of unknown codenames in x's received UTIL message (if such a pair exists), at least one of the following statements must hold:

- α and β refer to two different ancestors of x, and therefore x discovers at it has at least two ancestors (which it might not have known, if it has no pseudo-parent); *and/or*

- α and β were sent to two different descendants of x below (and possibly including) the sender child y, and therefore x discovers that it has at least two descendants below (and including) y (which it might not have known, if it has no pseudo-child below y).

Therefore x *might* be able to refine its known lower bound on the total number of variables in the problem. □

Topology Privacy in P$^{3/2}$-DPOP$^{(-)}$

Theorem 9. *The P$^{3/2}$-DPOP$^{(-)}$ algorithms (Algorithm 16 and its variant in Section 3.3.6) guarantee* partial *topology privacy. Each variable unavoidably discovers the total number of variables in the problem, an upper bound on the constraint graph diameter, and* might *also discover:*

- *a lower bound on a neighbor variable's degree in the constraint graph;*
- *bounds on the length of a cycle it is involved in;*
- *the fact that there exists at least one other branch in the constraint graph that it is not involved in.*

The advantages of P$^{3/2}$-DPOP over P$^{3/2}$-DPOP^{-} are the same as P-DPOP over P-DPOP^{-}.

Proof. The main difference with P-DPOP$^{(-)}$ is that there is no VALUE propagation, but rather one UTIL propagation phase per variable in the problem, each variable being root of the pseudo-tree in turn. Therefore, the total number of variables inevitably becomes public knowledge. The following paragraphs analyze the properties, in terms of topology privacy, of each phase of P$^{3/2}$-DPOP$^{(-)}$ that is not already present in the P-DPOP$^{(-)}$ algorithms.

Unique variable ID assignment (Algorithm 17) This is a modification of the depth-first traversal of the constraint graph in Algorithm 6, which has already been shown to guarantee full topology privacy. One difference is that the messages also carry an integer id, which is an upper bound on the number of variables visited so far.

- When receiving a CHILD message from its parent, variable x discovers that there exist at most id other variables in the problem that have already been visited. These variables are either ancestors of its parent, or descendants of its parent in another branch of the pseudo-tree. To make sure this bound is loose and uninformative, each variable adds a random number

in $[incr_{\min} \dots 2incr_{\min}]$ to its id, where $incr_{\min}$ is a free parameter of the algorithm. In particular, this prevents the case $id = 1$, which would allow x to infer that its parent is the root of the pseudo-tree, and that x is its first child. Variable x also discovers that there exist at least $\lfloor id/(2incr_{\min}) \rfloor$ already-visited variables. These two bounds are not informative about the topology of the constraint graph, since the total number of variables is revealed anyway.

- When receiving a CHILD message from a child y, variable x can compare the id it contains with the id it previously sent to y. The difference $\Delta id \geq incr_{\min}$ is an upper bound on the number of y's descendants; the parameter $incr_{\min}$ can be chosen as large as necessary to make this bound as loose as desired. Variable x also discovers that $\lfloor \Delta id/(2incr_{\min}) \rfloor$ is a lower bound on the number of y's descendants.
- The id contained in a PSEUDO message does not provide any information, as it is equal to the id in the CHILD message to which the PSEUDO message is a response. In fact, id could be removed from the PSEUDO message; it has only be left in Algorithm 17 for the sake of conciseness of the pseudo-code.

Once all variables have been assigned unique IDs, the last such ID is revealed to all variables (and is recorded as n^+). This reveals an upper bound on the total number of variables, which is useless since the exact total number of variables is later revealed anyway. However, it also reveals to each variable whether it is the last visited variable, i.e. the last leaf of the pseudo-tree. This is another, minor leak of topology privacy, since the leaves of the pseudo-tree that are not the last leaf discover that there exists at least one other branch in the pseudo-tree (and therefore in the constraint graph) that they are not involved in.

Exchange of ElGamal key shares (Algorithm 16, lines 4 to 11) The P$^{3/2}$-DPOP$^{(-)}$ algorithms involve each variable sharing a certain number of ElGamal key shares with all other variables. The corresponding SHARE messages do not contain any information that could be used to make inferences about the topology of the constraint graph. These messages are routed circularly through all variables using a mechanism whose properties in terms of topology privacy are discussed below.

Circular message routing (Algorithm 18) The purpose of this algorithm is for variables to be able to propagate messages along a circular variable ordering, without the need to know any additional topological information about the constraint graph, other than the knowledge of their respective (pseudo-)parents and (pseudo-) children in the pseudo-tree. In particular, the method ToPrevious() makes it possible to send a message to the previous variable in the circular variable ordering, without knowing which variable this is.

- The reception of a (PREV, M) message only indicates that the sender child wants the included message M to be delivered to its previous variable. This previous variable is either the recipient of the PREV message itself, or an unknown descendant thereof.
- The reception of a (LAST, M) message from one's parent indicates that an unknown variable (either the — unknown — root of the pseudo-tree, or the unknown child of an unknown ancestor, in another branch) wants M to be delivered to its previous variable, and that this previous variable is one's descendant in the pseudo-tree.

Pseudo-tree rerooting (Algorithm 19) The goal of the procedure SHUFFLEVECTORS() is for each variable to apply a given operation to a vector initially sent by each variable. VECT messages are propagated along a circular variable ordering using the previous message routing algorithm. A received VECT message contains the following information:

- id identifies the owner of the vector; being a secret, large random number, only the owner of the vector can identify itself;
- the vector is encrypted (and re-encrypted after each operation) and therefore cannot provide any topological information;
- the round number can take the following values:
  - $round = 1$ indicates that the vector is being modified, each variable setting in turn some of the values to -1;
  - $round = 2$ indicates that the vector is being sent to the root of the pseudo-tree. This does not happen for the vector belonging to the (unknown) root;
  - $round = 3$ indicates that the vector is being shuffled by each variable in turn;
  - $round = 4$ indicates that the vector is on its way back to its owner. This does not happen for the vector belonging to the (unknown) root.

Once the preliminary procedure SHUFFLEVECTORS() has been completed, rerooting the pseudo-tree actually consists in calling the procedure REROOT(), which makes use of the collaborative decryption algorithm whose properties are described below.

Collaborative decryption (Algorithm 20) This algorithm involves passing around a DECR message along the circular variable ordering. The message contains a secret codename identifying the original sender variable, which is the only variable capable of deciphering this codename, since it chose it itself as a large, random number. The last part of the message payload is an ElGamal cyphertext, which remains encrypted until it reaches back the original sender, and therefore does not leak any topological information.

This concludes the proof that the P$^{3/2}$-DPOP$^{(-)}$ algorithms guarantee *partial* topology privacy. □

Topology Privacy in P^2-DPOP$^{(-)}$

Theorem 10. *The P^2-DPOP$^{(-)}$ algorithms (Section 3.5 and the variant in Section 3.3.6) guarantee* partial *topology privacy. The limited leaks of topology privacy are the same as for P$^{3/2}$-DPOP$^{(-)}$.*

Proof. With respect to P$^{3/2}$-DPOP$^{(-)}$, there are only two differences in the algorithm: the UTIL propagation is performed along a linear variable ordering, and choosing a value for the root variable requires collaborative decryption.

ElGamal UTIL propagation (Algorithm 21) To exchange UTIL messages along a linear variable ordering, the algorithm makes use of the circular message routing procedure, which has already been shown to guarantee full topology privacy. However, the last variable in the linear ordering needs to know it is the last in order to initiate the UTIL propagation; therefore, by contraposition, non-last variables know they are not the last, and, in particular, non-last leaves of the pseudo-tree discover the existence of another branch. This minor leak of topology privacy is already present in the unique variable ID assignment algorithm anyway. Besides this, the topology privacy properties of the UTIL propagation phases in P^2-DPOP$^-$ and P^2-DPOP are the same as in P-DPOP$^-$ and P-DPOP, respectively.

Root variable assignment (Algorithm 23) This algorithm involves recursively calling the collaborative decryption procedure, which has been shown to guarantee full topology privacy.

This concludes the proof that the P^2-DPOP$^{(-)}$ algorithms guarantee *partial* topology privacy. □

3.6.3 Constraint Privacy

Recall (Definition 11) that constraint privacy is about protecting the knowledge of the costs of various assignments of values to variables, based on one or more constraints between these variables.

Constraint Privacy in P-DPOP$^{(-)}$

Theorem 11. *The P-DPOP$^{(-)}$ algorithms (Algorithm 13 and its variant in Section 3.3.6) guarantee* partial *constraint privacy. The local optimal cost of a subproblem with respect to a partial variable assignment may be revealed. In the Max-DisCSP case, the feasibility of such a partial assignment may be revealed, even though the partial assignment cannot be extended to an overall feasible solution (and therefore this is not semi-private information).*

Proof. Information about constraints is only transmitted during the UTIL propagation phase (Algorithm 15). With the help of the knowledge of the optimal variable assignments transmitted during the top-down VALUE propagation phase (Section 3.3.4), some of the cost information may be decrypted.

Single-variable UTIL messages In the DCOP case, all costs in the single-variable UTIL message received by a variable p_x remain in clear text. However, recall that the properties of P-DPOP$^{(-)}$ in terms of topology privacy guarantee that, if the sender of the UTIL message is a leaf of the pseudo-tree, p_x will not be able to discover it.[3] Therefore, p_x cannot conclude that the message contains constraint information about a single variable, instead of containing aggregated information about multiple variables' constraints.

In the Max-DisCSP case, the message has been obfuscated by adding secret random numbers only to its infeasible entries (line 9). Feasible entries are not obfuscated and remain equal to 0, and therefore variable p_x is able to identify which of the message entries refer respectively to feasible or infeasible assignment to p_x.

However, the addition of a secret, positive, random number to each infeasible entry makes sure that only a loose upper bound on the number of constraint violations is leaked, and the exact number of constraint violations remains secret. This upper bound can be made as loose as desired by choosing random numbers as large as necessary; however, these random numbers must remain significantly smaller than the ones used for obfuscating multi-variable messages (see below).

Multi-variable UTIL messages If the UTIL message involves at least one other variable $\tilde{y_i}$, then all entries of the message have been obfuscated by adding large random numbers $key^x_{y_i}(\tilde{y_i^x})$ (line 26). These random numbers should be significantly larger than the ones used to obfuscate infeasible entries in single-variable messages (in the Max-DisCSP case), so that obfuscated feasible and infeasible entries are indistinguishable. Furthermore, $key^x_{y_i}(\tilde{y_i^x})$ is only known to the sender x of the message and to its pseudo-parent y_i, but not to the recipient p_x, which therefore cannot subtract it back to de-obfuscate the feasibility entries.

[3] We have seen before that the reverse is false: p_x might be able to discover that a child is *not* a leaf.

Let us assume, for the sake of simplicity, that the message $m(\tilde{p_x}, \tilde{y_i^x})$ involves only the two variables $\tilde{p_x}$ and $\tilde{y_i^x}$; the argument extends easily to more variables. The recipient p_x might be able to make inferences in two different ways: 1) by fixing $\tilde{y_i^x}$ and comparing the obfuscated entries corresponding to different values for $\tilde{p_x}$; or 2) by fixing $\tilde{p_x}$ and varying $\tilde{y_i^x}$ instead.

1. For a given value of $\tilde{y_i^x}$, all cost entries have been obfuscated on line 26 by adding the same random number $key^x_{y_i}(\tilde{y_i^x})$. Therefore, the recipient is able to compute the relative differences of cost values for various assignments to $\tilde{p_x}$. However, it is unable to decrypt the absolute values, since it does not know $key^x_{y_i}(\tilde{y_i^x})$. In particular, it is incorrect to assume that the lowest obfuscated value for a given value of $\tilde{y_i^x}$ is equal to $key^x_{y_i}(\tilde{y_i^x})$, because the lowest cost value does not necessarily decrypt to 0.

   There is one exception to this argument, in the Max-DisCSP case: if a feasible solution is found to the problem in which $\tilde{y_i^x} = \tilde{y_i^x}^*$ and $\tilde{p_x} = \tilde{p_x}^*$, then it is necessarily the case that $m(\tilde{p_x}^*, \tilde{y_i^x}^*)$ decrypts to 0 (since the solution is feasible), and therefore the recipient of the UTIL message will be able to infer $key^x_{y_i}(\tilde{y_i^x}^*)$. After fixing $\tilde{y_i^x} = \tilde{y_i^x}^*$ in the message and subtracting $key^x_{y_i}(\tilde{y_i^x}^*)$, the same reasoning can be made as for the single-variable case, in which feasible and infeasible entries are identifiable, but the numbers of constraint violations for infeasible entries remain obfuscated.
2. For a given value of $\tilde{p_x}$, each cost value $m(\tilde{p_x}, \tilde{y_i^x})$ has been obfuscated by adding a different, secret random number $key^x_{y_i}(\tilde{y_i^x})$. Choosing these random numbers sufficiently large makes sure that no useful information (relative, or absolute) can be obtained by comparing the obfuscated cost values.

This concludes the proof that the P-DPOP$^{(-)}$ algorithms guarantee *partial* constraint privacy. □

Constraint Privacy in P$^{3/2}$-DPOP$^{(-)}$

The constraint privacy properties of the P$^{3/2}$-DPOP$^{(-)}$ algorithms differ slightly from those of P-DPOP$^{(-)}$, because the former protect decision privacy (which is indirectly beneficial to constraint privacy), but also reveal the total number of variables in the problem (which can hurt constraint privacy).

Theorem 12. *The P$^{3/2}$-DPOP$^{(-)}$ algorithms (Algorithm 16 and its variant in Section 3.3.6) guarantee* partial *constraint privacy. The leaks are the same as in P-DPOP$^{(-)}$, but they happen less frequently.*

Proof. Single-variable UTIL messages leak the same amount of constraint privacy as in P-DPOP$^{(-)}$; notice however that, since the P$^{3/2}$-DPOP$^{(-)}$ algorithms now reveal the

total number of variables, in some circumstances it may be possible for a variable to discover that a child is a leaf, and that the UTIL message it sends therefore contains information about its local subproblem only.

In the Max-DisCSP case, multi-variable UTIL messages now leak potentially much less information, for the following reason. Consider again the simpler and non-restrictive case of a two-variable message $m(\tilde{p_x}, \tilde{y_i^x})$ received by $\tilde{p_x}$. Because decision privacy is now guaranteed, $\tilde{p_x}$ no longer discovers the value $\tilde{y_i^x}^*$ chosen for $\tilde{y_i^x}$, and is therefore no longer able to infer which of the entries corresponding to $\tilde{p_x} = \tilde{p_x}^*$ decrypts to 0. One exception to this is when the following three conditions simultaneously hold: 1) P$^{3/2}$-DPOP$^-$ is used, 2) the codename $\tilde{y_i^x}$ refers to a variable y_i^x that is a neighbor of $\tilde{p_x}$, and 3) $\tilde{y_i^x}^*$ is semi-private information to $\tilde{p_x}$; then $\tilde{p_x}$ will still discover $\tilde{y_i^x}^*$, and will be able to make the same inferences as in P-DPOP$^{(-)}$. If the first condition is not satisfied, i.e. P$^{3/2}$-DPOP is used, then $\tilde{p_x}$ will not be able to link the codename $\tilde{y_i^x}$ to any known variable. This is also the case if P$^{3/2}$-DPOP$^-$ is used, but the second condition does not hold. Finally, if the first two conditions hold, $\tilde{p_x}$ will only be able to discover $\tilde{y_i^x}^*$ if it is semi-private information, i.e. if it can infer it only from its knowledge of the problem, and of its own chosen value $\tilde{p_x}^*$. □

Constraint Privacy in P^2-DPOP$^{(-)}$

Theorem 13. *The P^2-DPOP$^{(-)}$ algorithms (Section 3.5 and the variant in Section 3.3.6) guarantee (full) constraint privacy.*

Proof. The P^2-DPOP$^{(-)}$ algorithms fix all the leaks of constraint privacy in P$^{<2}$-DPOP$^{(-)}$, by replacing the cryptographically insecure obfuscation through addition of random numbers, by the cryptographically secure ElGamal encryption (Section 3.2.1). This makes it no longer possible to compare two encrypted cost values without decrypting them, which would require the collaboration of all agents. In particular, while it is possible to compute the $\min$ of two cyphertexts without decrypting them, the result remains encrypted, and cannot be compared to the two inputs to decide which one is the $\arg\min$. □

3.6.4 Decision Privacy

Recall (Definition 12) that decision privacy is about hiding the values assigned to variables in the chosen solution to the problem.

Theorem 14. *The P-DPOP$^{(-)}$ algorithms (Algorithm 13 and its variant in Section 3.3.6) guarantee* partial *decision privacy. The leak of decision privacy lies in the fact that a variable* might *discover the respective values chosen for some or all of its neighboring variables.*

Proof. First notice that the algorithm cannot leak any information about the chosen values for variables that are *lower* in the pseudo-tree, since these variables have been projected out of the UTIL messages received. However, during the VALUE propagation phase, each variable receives a message from its parent that contains the chosen values for its parent and pseudo-parents. The message may also contain the values chosen for other, non-neighboring variables, but the recipient of the message will not be able to decode the codenames used to obfuscate these values. Furthermore, variable domains are shuffled using secret permutations to make it impossible to decode the codename for the value of a non-neighboring variable from its index in the variable's domain. □

Theorem 15. *$P^{3/2}$-DPOP$^{(-)}$ and P^2-DPOP$^{(-)}$ (Algorithm 16 and Section 3.5, and the variants in Section 3.3.6) guarantee (full) decision privacy.*

Proof. The leak of decision privacy in P-DPOP$^{(-)}$ is fixed by removing the VALUE propagation phase. Instead, the variable ordering is rerooted, and the UTIL propagation phase is restarted. Notice that it is not possible to compare the UTIL messages received from one iteration to the next in an attempt to infer the decision that has been made at the previous iteration: the messages are not comparable, since different codenames and obfuscation keys are used. □

3.7 Experimental Results

This section reports the empirical performance of the various algorithms considered in this paper, on four classes of DisCSP benchmarks: graph coloring (Section 3.7.1), meeting scheduling (Section 3.7.2), resource allocation (Section 3.7.3), and game equilibrium (Section 3.7.4). We also report results on the DisWCSP class of multi-agent Vehicle Routing Problems (Section 3.7.5). For each problem class, the choice of the Dis(W)CSP formulation is crucial, because it dictates how the four types of privacy defined based on the problem constraint graph will relate to the actual privacy of the original problem. In particular, as discussed in Section 3.2.2, the P*-DPOP algorithms use the standard assumption that each constraint is known to all agents owning a variable in its scope. Therefore, when an agent wants to hide a constraint from neighboring agents, it must express its constraint over *copies* of its neighbors' variables. Additional equality constraints must be introduced to make copy variables equal to their respective original variables. On the other hand, the MPC-Dis(W)CSP4 algorithms do not make use of this assumption, and therefore do not need the introduction of copy variables.

The three performance metrics considered are *simulated time* [Sultanik et al., 2007], number of messages, and total amount of information exchanged; for each metric, we report the median over at least 100 problem instances, with 95% confidence intervals. For the obfuscation of cost values in P$^{<2}$-DPOP$^{(-)}$, we used 128-bit random numbers,

while P^2-DPOP$^{(-)}$ used 512-bit ElGamal encryption. MPC-Dis(W)CSP4 also used 512 bits for its Paillier encryption. For the unique variable ID generation procedure in P$^{>1}$-DPOP$^{(-)}$, the parameter $incr_{\min}$ was set to 10. All algorithms were implemented inside the FRODO platform (Chapter 5), which was coupled with the CSP solver JaCoP [2011] to speed up MPC-Dis(W)CSP4. The experiments were run on a 2.2-GHz, dual-core computer, with Java 1.6 and a Java heap space of 2 GB. The timeout was set to 10 min (wall-clock time).

3.7.1 Graph Coloring

We first report the respective performances of the algorithms on random, pure DisCSP, graph coloring problems. As mentioned in Section 2.2.1, one real-life example of such graph coloring problems is the allocation of time schedules in a Time Division Multiple Access (TDMA) scheme for an ad-hoc wireless network. The graphs were randomly generated with varying numbers of nodes, and an edge density fixed to 0.4. The number of colors was set to 3. Notice that in the DisCSP formulation, inter-agent constraints are binary inequality constraints, and therefore decision privacy is relevant to this problem class: knowing one's chosen color is insufficient to infer the respective colors of one's neighbors. To study the tradeoff between privacy and performance in MPC-DisCSP4, we also considered a variant that we denote MPC-DisCSP4$^-$, in which we assumed that all inter-agent inequality constraints are public — in other words, the neighborhoods of nodes are public, and only the final choices of colors are protected.

In Figures 3.8 and 3.9, one can observe that the MPC-DisCSP4 algorithm (denoted simply as *MPC* in these and all subsequent figures) scales very poorly: it times out on problems involving more than 6 nodes. MPC-DisCSP4$^-$ performs much better; however, as mentioned before, it only protects the final choices of colors. It also exhibits an interesting phase transition, which is due to the fact that its complexity is directly linked to the number of feasible solutions to the DisCSP. For small numbers of nodes, the total state space is small, and MPC-DisCSP4$^-$ performs relatively well. For numbers of nodes above 9, the problem instances are mostly infeasible (since the constraint density and the number of colors remain constant), and MPC-DisCSP4$^-$ quickly detects infeasibility without even having to exchange any message.

The most efficient algorithms by far are P-DPOP$^{(-)}$, whose curves are at least one order of magnitude below all other algorithms. In particular, P-DPOP$^-$'s runtime is sensibly the same as DPOP, and the overhead in terms of communication is almost solely due to the root election algorithm. The cost of improved topology privacy in P-DPOP vs. P-DPOP$^-$ only starts to show for problem sizes above 7. Full decision privacy comes at much higher costs: P$^{3/2}$-DPOP$^{(-)}$'s curve is between 1 and 3 orders of magnitude above P-DPOP$^{(-)}$'s, which suggests that rerooting the pseudo-tree (which involves expensive cryptographic operations) is by far the complexity bottleneck, even when

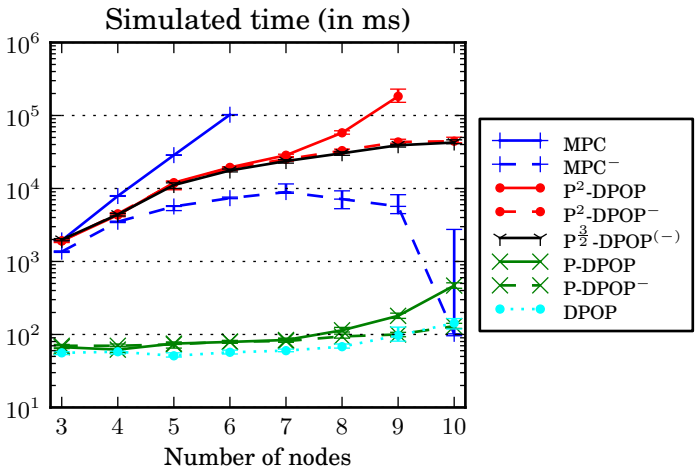


Figure 3.8: Runtime performance on graph coloring problems.

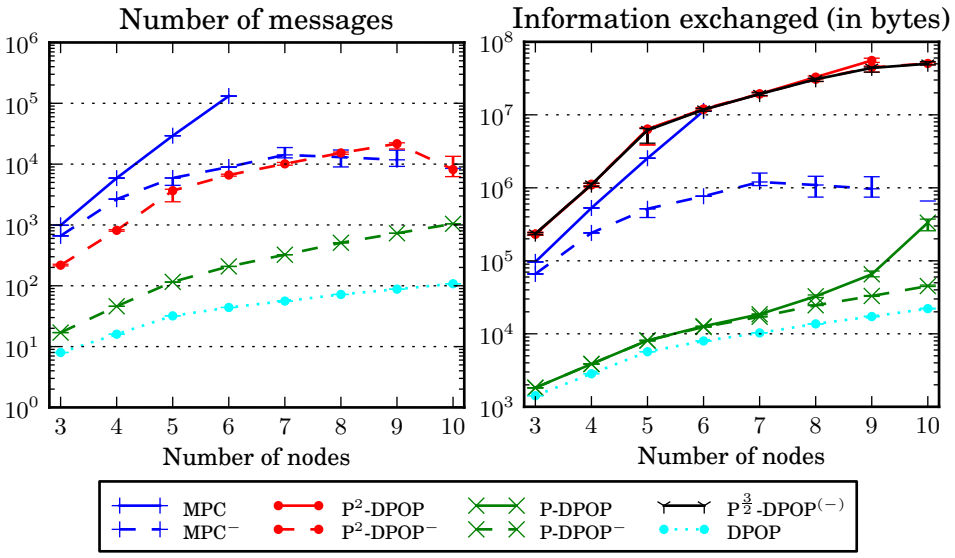


Figure 3.9: Communication performance on graph coloring problems.

full constraint privacy is additionally guaranteed, as in P^2-DPOP$^-$. Notice that the slope decreases as the problem size increases; this is due to the fact that more and more problems become infeasible, and the P$^{>1}$-DPOP$^{(-)}$ algorithms are able to terminate after the first iteration on infeasible problems. Like for P-DPOP vs. P-DPOP$^-$, the cost of improved topology privacy is only visible above 7 nodes; P^2-DPOP even timed out on problems of size 10. Finally, Figure 3.9 illustrates the fact that MPC-DisCSP4 tends to send large numbers of small messages, while the P$^{>1}$-DPOP$^{(-)}$ algorithms send lower numbers of larger messages.

3.7.2 Meeting Scheduling

We now report experimental results on random, pure DisCSP, meeting scheduling benchmarks (Section 2.2.2). We varied the number of meetings, while keeping the number of participants per meeting equal to 2. For each meeting, the participants were randomly drawn from a common pool of 3 agents. The pool of agents was deliberately chosen small to increase the complexity of the problems, by increasing the probability that each agent take part in multiple meetings. The number of available time slots was

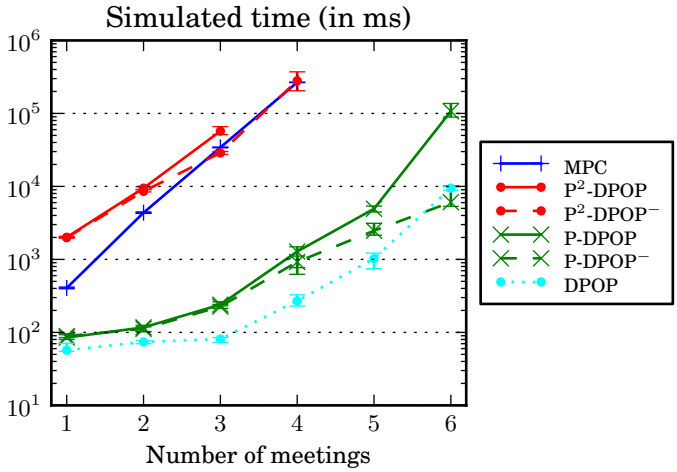


Figure 3.10: Runtime performance on small meeting scheduling problems.

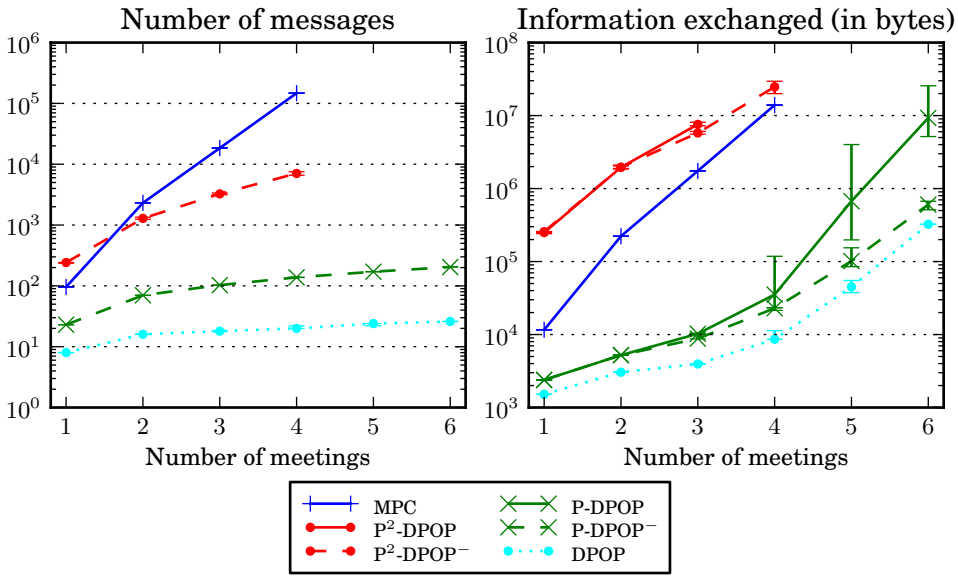


Figure 3.11: Communication performance on small meeting scheduling problems.

set to 8.

Notice that in the DisCSP formulation, all inter-agent constraints are binary equality constraints, and therefore the $P^{3/2}$-DPOP$^{(-)}$ algorithms do not bring any additional decision privacy compared to P-DPOP$^{(-)}$, since the values of neighboring variables are semi-private information; therefore, we do not report the performance of $P^{3/2}$-DPOP$^{(-)}$. For the MPC-DisCSP4 algorithm, we simplified the formulation by only introducing one variable per meeting, owned by the initiator of the meeting. This way, for each meeting, only its initiator is made public, but its exact list of participants remains secret (it is only revealed *a posteriori* to the participants of the meeting when they attend it).

As can be seen in Figures 3.10 and 3.11, P^2-DPOP$^-$'s performance is comparable to that of MPC-DisCSP4 (but with much stronger privacy guarantees), although the former sends significantly more information on the smallest problems, but significantly fewer messages on the largest problems they could solve within the timeout. On the other hand, because it is a majority threshold scheme, MPC-DisCSP4 actually could not provide any privacy guarantees on problems of size 1, since they only involved 2 agents. Both algorithms could only scale up to problems of size 4, and timed out on larger problems. P^2-DPOP's increased topology privacy comes at a price that made it timeout earlier than P^2-DPOP$^-$, on problems of size 4.

Like for graph coloring problems, the P-DPOP$^{(-)}$ algorithms remain the most efficient algorithms by far: they perform between 1 and 2 orders of magnitude better than all other algorithms, both in terms of runtime and information exchanged. And like for graph coloring, the improved topology privacy in P-DPOP comes at a price that is negligible for small problems, but can grow to one order of magnitude on problems of size 6. In terms of runtime and information exchange, P-DPOP$^-$ is only worse than DPOP by a small factor; however it sends approximately one order of magnitude more messages.

Figures 3.12 and 3.13 report the performances of the various algorithms on larger meeting scheduling problems, with 5 participants per meeting, and a pool of 10 agents. The performance of MPC-DisCSP4 is degraded by approximately one order of magnitude across all three metrics. On the other hand, the P*-DPOP$^{(-)}$ algorithms perform comparatively much worse than on smaller problem instances: P^2-DPOP was not able to solve even single-meeting problems, P^2-DPOP$^-$ could only solve single-meeting problems before it timed out, and P-DPOP was outperformed by MPC-DisCSP4 and could only scale up to problems of size 2. All algorithms timed out on problems involving 4 meetings, including DPOP.

This dramatic loss of performance of DPOP-based algorithms can be explained by the presence of 5-ary *AllEqual* constraints enforcing that all participants to each meeting

agree on the time for the meeting. As a result, many UTIL messages will be expressed over multiple variables which are in fact bound to be equal, which is very inefficient. One way to fix this inefficiency and to allow DPOP-based algorithm to scale to larger problems could be to decompose these *AllEqual* constraints into chains of binary equality constraints based on the positions of the variables in the pseudo-tree. This should make the DPOP-based algorithms much less sensitive to the arity of the *AllEqual* constraints, and is left for future work.

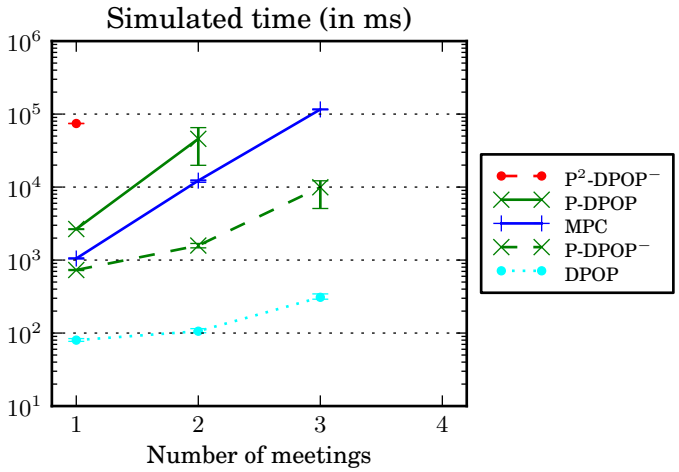


Figure 3.12: Runtime performance on large meeting scheduling problems.

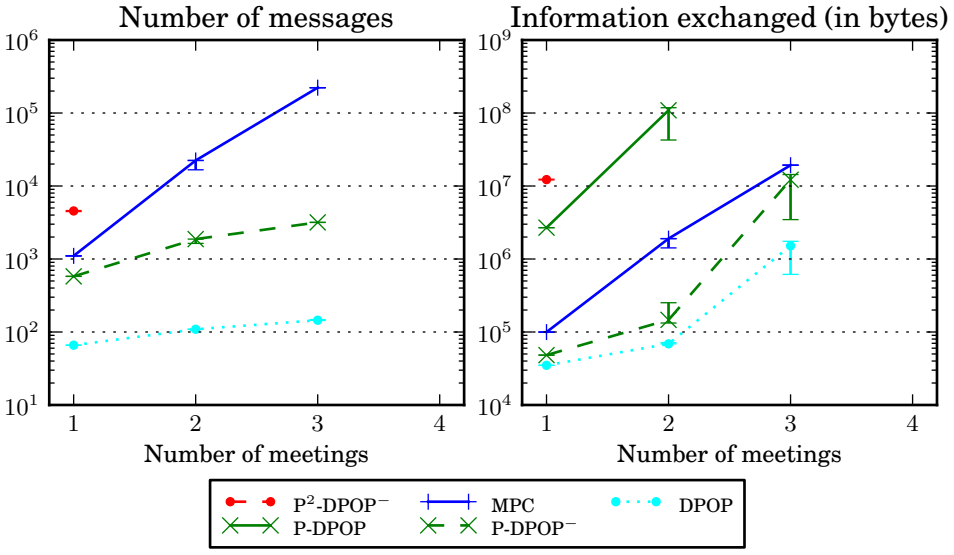


Figure 3.13: Communication performance on large meeting scheduling problems.

3.7.3 Resource Allocation

Next, we performed experiments on pure DisCSP, resource allocation benchmarks (Section 2.2.4). Random problem instances were produced using the combinatorial auction problem generator CATS [Leyton-Brown et al., 2000], ignoring bid prices. We used the *temporal matching* distribution, which models the allocation of takeoff and landing slots at airports, fixing the total number of goods (i.e. resources) to 8, and varying the numbers of bids. Each bid is a request for a bundle of exactly 2 resources (a takeoff slot and a corresponding landing slot). Two requests may be placed by the same airline company; each airline should get exactly one of its requests fulfilled.

Novel, Privacy-Friendly DisCSP Formulation

The DisCSP formulation proposed by Petcu [2007] for resource allocation problems (actually, for Combinatorial Auctions) has undesirable properties in terms of privacy. Since, for each given resource, the domain of each corresponding decision variable is the list of consumers interested in the resource, this list of consumers is inherently leaked by the problem formulation. To address this, we propose to use binary variables instead, which indicate whether or not the owner consumer gets allocated the resource. The list of consumers interested in a particular resource is also leaked by the scope of the inter-agent constraint enforcing that the resource be only allocated to one of them. To address this, we propose to introduce agents for the *resource providers*[4], which own copy variables over which this constraint is expressed.

To illustrate our DisCSP formulation, consider the simple example of three consumers A, B and C, interested in two resources y and z. Consumer A is interested in getting either one of the two resources (OR constraint), consumer B also wants either one, but not both (XOR constraint), and consumer C wants either both resources or none (equality constraint). We model this problem using two sets of decision variables x and $\hat{x}$. Variable x_a^b is owned by the agent providing resource b, and takes the value 1 if resource b is allocated to agent a, and 0 otherwise. Variable $\hat{x}_a^b$ is a copy variable owned by agent a, and constrained to be equal to x_a^b. Three types of constraints exist on these variables:

1. Each resource b can only be assigned to at most one consumer, which is enforced by the constraint $\sum_i x_i^b \leq 1$. This constraint is private to the resource provider for b.

2. Each consumer has private constraints on the feasible combinations of resource assignments. For example, since consumer A is interested in getting at least one

[4] CATS assumes there is a single auctioneer, and does not specify which slot is at which airport in the problem instances it generates; therefore in our experiments we have assumed that each resource was provided by a separate resource provider.

of the resources, this is expressed by the constraint $\hat{x}_A^y$ OR $\hat{x}_A^z$.

3. Corresponding x and $\hat{x}$ variables must have equal values.

This is illustrated in Figure 3.14. Like for the meeting scheduling problem class, all inter-agent constraints are equality constraints, therefore we do not report the performance of P$^{3/2}$-DPOP$^{(-)}$, whose decision privacy guarantees are the same as P-DPOP$^{(-)}$.

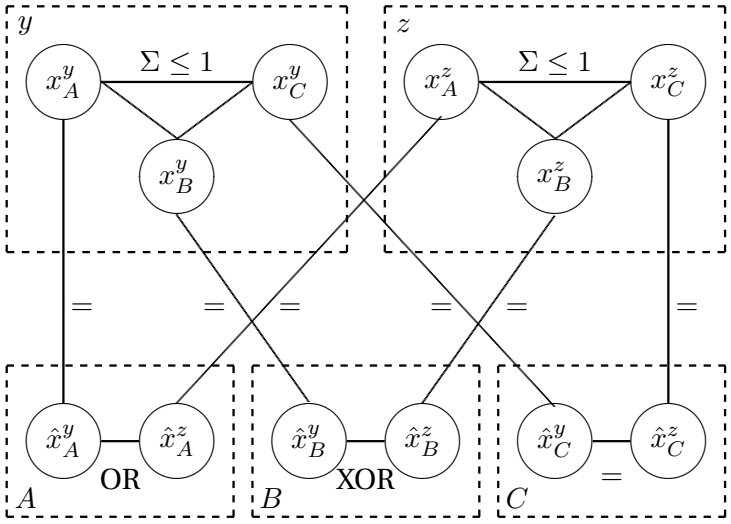


Figure 3.14: DisCSP formulation for a resource allocation problem.

For MPC-DisCSP4, the DisCSP formulation was simplified by not introducing copy variables hold by consumers, since they are not necessary to protect constraint privacy: consumers can request resources by expressing constraints directly over the variables owned by the resource providers, without revealing these constraints. However, since MPC-DisCSP4 assumes that all variables are public, in order to increase topology privacy we introduced, for each resource, as many variables as consumers, regardless of whether they are actually interested in the resource. To reduce the size of the search space, we assumed that the $\Sigma \leq 1$ constraints were public.

Experimental Results

Figures 3.15 and 3.16 show that the performance of MPC-DisCSP4 decreases very fast with the number of requests, such that the algorithm was not able to scale beyond problems of size 4. The P^2-DPOP$^{(-)}$ algorithms seem to scale better, and were able to solve problems involving 5 requests. On all three metrics, both algorithms were largely outperformed by P-DPOP$^{(-)}$, whose runtime curve is remarkably flat, and almost overlaps with the runtime curve of DPOP. The overhead of P-DPOP$^{(-)}$ compared to DPOP is slightly larger in terms of information exchanged, and goes up to one order of magnitude in terms of number of messages. On this problem class, and for these

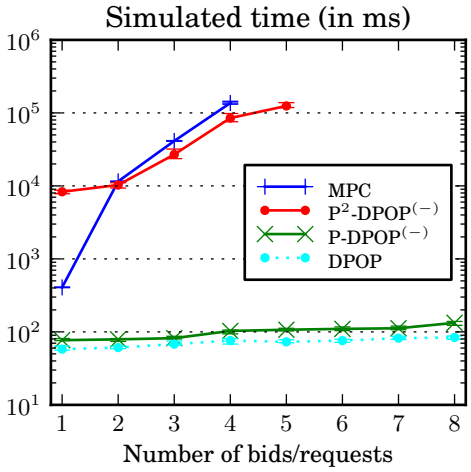


Figure 3.15: Runtime performance on resource allocation problems.

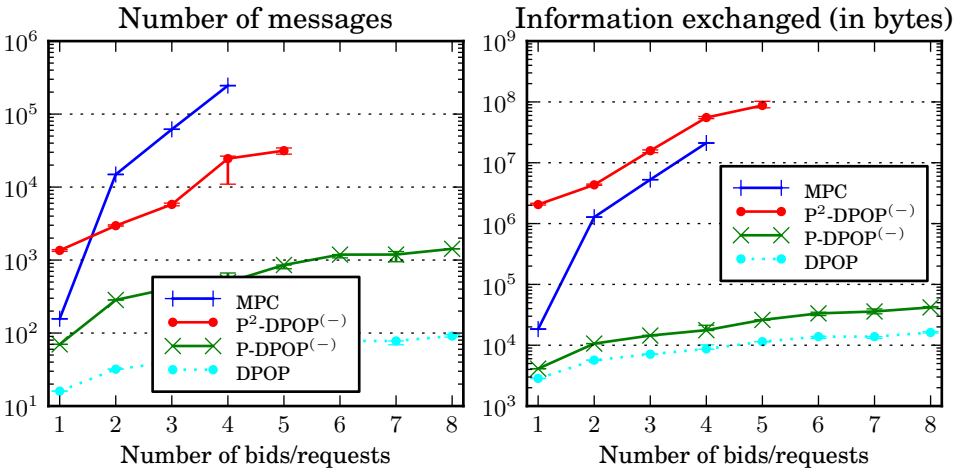


Figure 3.16: Communication performance on resource allocation problems.

problem sizes, P-DPOP and P^2-DPOP performed the same as their respective “minus" variants.

3.7.4 Equilibria in Graphical Games

The problem of computing equilibria in graphical games was already introduced in Section 2.2.6, in which we have also briefly described two DisCSP formulations proposed in previous work. We first go over the undesirable privacy properties of these two formulations, and we propose a novel, third one that makes it possible to better protect the privacy of the players in the game.

DisCSP Formulations

The formulation in [Vickrey and Koller, 2002] (variables as single-player strategies) has a remarkable disadvantage: players express their cost functions in Equation (2.5) as constraints involving their neighbors' variables. This implies that this information is shared with all neighbors, and this formulation would reveal upfront to player i's neighbors her list of neighbors N_i, whether she likes or dislikes each one of them ($l_{n \in N_i}$), and her private cost a_i.

While the formulation in [Soni et al., 2007] (variables as joint neighborhood strategies) has the distinct complexity disadvantage of dramatically expanding the variable domains (player i's variable can then take on $|S|^{|N_i \cup \{i\}|}$ values instead of just $|S|$), it partially addresses the privacy leaks in the previous formulation, because each best-response constraint is now a unary constraint, initially known only to the corresponding player. However, the binary constraints between each pair of neighbors, which are needed to enforce that the two players' variables take on consistent values (i.e. that the two players agree on their respective strategies and those of their common neighbors), necessarily reveal each agent's list of neighbors to each of her neighbors.

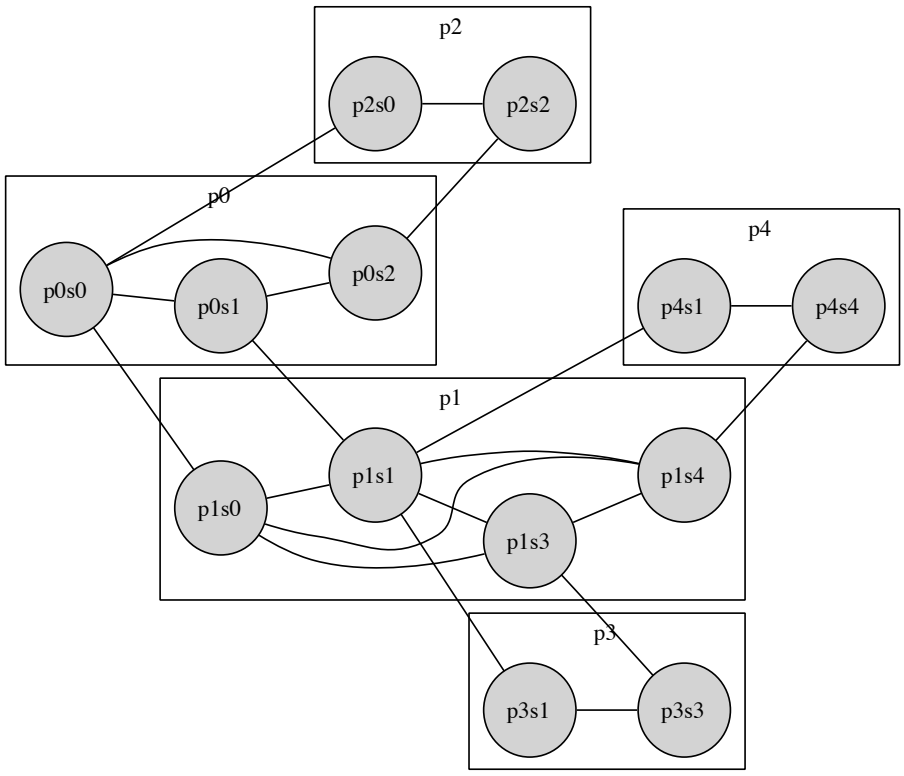


Figure 3.17: The constraint graph for the game in Figure 2.7.

Instead, we propose the following privacy-friendly DisCSP formulation, which is based on introducing copy variables. Each player i owns one variable $p_i s^{n_j}$ for each neighbor n_j's strategy, and for her own, as shown in Figure 3.17. Notice that the resulting constraint graph is not the same as the game graph, due to the presence of copy variables. The best-response constraints are the same as in [Vickrey and Koller, 2002], except that player i expresses her constraint exclusively on variables that she owns. Additionally, binary equality constraints enforce that each pair of neighbors agree on each other's strategy. This addresses the leak of topological information in [Soni et al., 2007], because each variable corresponds to a single player's strategy. However this comes at the price of decision privacy: each player discovers her neighbors' strategies at the equilibrium. Therefore, we do not report the performance of P$^{3/2}$-DPOP$^{(-)}$.

For MPC-DisCSP4, we used the DisCSP formulation from [Vickrey and Koller, 2002], i.e. without copy variables. An interesting consequence of this difference is that, contrary to P*-DPOP$^{(-)}$, MPC-DisCSP4 is then able to hide each player's chosen strategy from her neighbors. In the context of the party game, this is not very useful to players who decide to attend the party, since they will necessarily eventually discover whether their acquaintances also decided to attend or not. On the other hand, a player who declines the invitation does not *directly* discover anything about the list of attendees. However, she might still be able to make indirect inferences about the decisions of her acquaintances, based on the fact that her decision to decline is a best response to their respective chosen strategies.

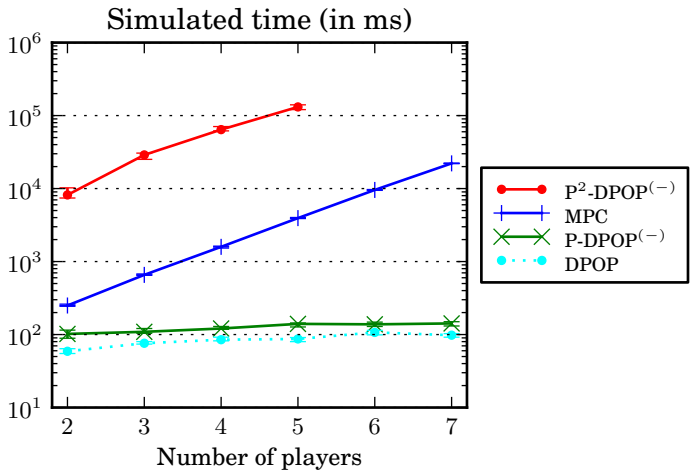


Figure 3.18: Runtime performance on party games.

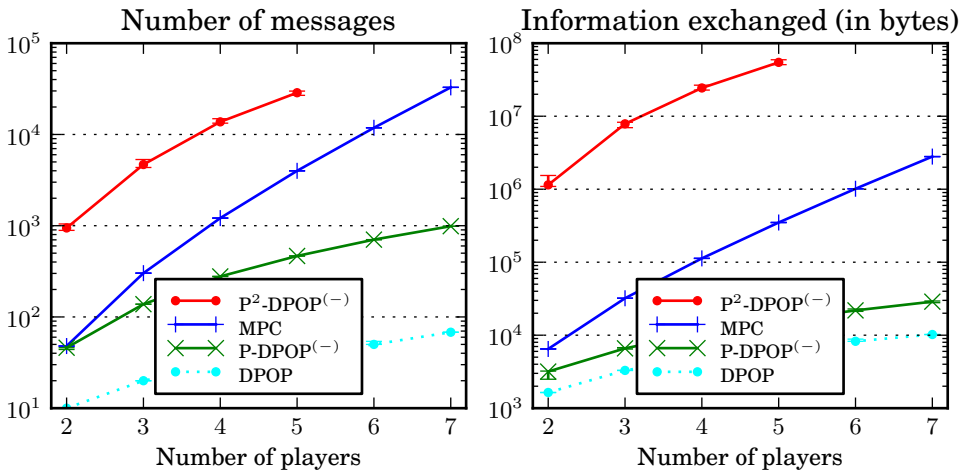


Figure 3.19: Communication performance on party games.

Experimental Results

Figures 3.18 and 3.19 report the performances of the algorithms on random party game instances based on acyclic game graphs of degree 2 (the graph is a tree in which each node has at most 2 children), for pure Nash equilibria, with varying numbers of players. The P^2-DPOP$^{(-)}$ algorithms were only able to scale up to problems of size 5, and were outperformed by MPC-DisCSP4 by at least one order of magnitude across all three metrics. Both algorithms still performed largely worse than the P-DPOP$^{(-)}$ algorithms, which seem capable of scaling to much larger problems. Like for most previous problem domains, the performance overhead in P-DPOP$^{(-)}$ compared to DPOP is minimal in terms of runtime, slightly larger in terms of information exchanged, and goes up to one order of magnitude in terms of number of messages.

3.7.5 Vehicle Routing Problems

Finally, to evaluate the performance of our algorithms on DisWCSP benchmarks, we carried out experiments on multi-agent VRPs (Section 2.2.5). DisSDMDVRP instances were generated based on the Cordeau MDVRP benchmark instances from [Dorronsoro, 2007], by varying the visibility horizon H of the depots. To solve each depot's local VRP, the FRODO platform was coupled with the OR-Objects Library [OR-Objects, 2010]. The VRP algorithm was set to the best out of Clarke and Wright's *savings* algorithm [1964] and Gillett and Miller's *sweep* algorithm [1974] as construction algorithm, with the 2-Opt TSP improvement algorithm [Croes, 1958]. To compensate for the expensiveness of the VRP constraint checks, the timeout was raised to 1 hour (wall clock time), and the experiments were run on a faster, 2.53-GHz computer. Each entry in Table 3.2 is the median over 43 runs. n_D is the number of depots that must coordinate their

Table 3.2: Experimental results on DisSDMDVRP benchmarks.

prob.	n_D	H	n_C	$q_{\max}$	P-DPOP		$P^{3/2}$-DPOP		P^2-DPOP		MPC-DisWCSP4	
					time	info	time	info	time	info	time	info
p01	4	13	2 / 27	25	395.0 ms	10 kB	582.0 ms	25 kB	26.5 s	10 MB	-	-
		14	4 / 30	25	2.1 s	27 kB	22.2 s	779 kB	-	-	-	-
p03	2	10	1 / 14	20	462.0 ms	5 kB	*756.0 ms*	*12 kB*	11.8 s	2 MB	*25.2 s*	*8 MB*
		12	2 / 16	20	2.1 s	21 kB	*4.4 s*	*98 kB*	5.0 min	39 MB	*6.2 min*	*130 MB*
p11	2	22	1 / 19	47	1.2 s	9 kB	*1.9 s*	*20 kB*	49.8 s	10 MB	*2.1 min*	*69 MB*
		24	2 / 23	47	11.3 s	80 kB	*37.2 s*	*334 kB*	51.6 min	372 MB	-	-
p12	2	65	2 / 72	1	324.0 ms	3 kB	*690.0 ms*	*23 kB*	1.5 min	10 MB	*1.5 min*	*414 MB*
		70	4 / 72	2	879.0 ms	7 kB	*5.0 s*	*110 kB*	31.5 min	191 MB	*11.1 min*	*2 GB*
		79	8 / 80	2	6.1 s	58 kB	*5.2 min*	*2 MB*	-	-	-	-
		80	10 / 80	4	2.1 min	1 MB	-	-	-	-	-	-
p15	4	60	8 / 144	1	943.0 ms	19 kB	13.0 s	1 MB	-	-	-	-
		70	16 / 144	2	9.3 min	17 MB	-	-	-	-	-	-
p18	6	60	14 / 215	1	2.6 s	86 kB	2.2 min	7 MB	-	-	-	-
p21	9	60	24 / 324	1	34.4 s	2 MB	-	-	-	-	-	-

decisions, and n_C is reported as *number of customers that can be served by at least two depots / total number of visible customers*. To solve each problem instance, P^2-DPOP and MPC-DisWCSP4 need an upper bound $c_{\max}$ respectively on the optimal cost and on the worst feasible cost; this bound was set to the sum of the worst costs of all VRPs.

Being a majority threshold scheme, MPC-DisWCSP4 could not provide any privacy guarantee on the problem instances that involve only 2 agents. These problem instances are indicated in italics in Table 3.2, and happen to be the only ones that MPC-DisWCSP4 could solve before the timeout. On these instances, P^2-DPOP had comparable performance in terms of runtime, but exchanged up to 40 times less information. Furthermore, where MPC-DisWCSP4's threshold scheme failed to guarantee any level of privacy, P^2-DPOP had an almost perfect score (Table 3.1). P^2-DPOP was also able to solve more instances.

Table 3.2 also shows that replacing ElGamal encryption with obfuscation by addition of random numbers, like in P-DPOP and P$^{3/2}$-DPOP, cut the runtime by 1 to 2 orders of magnitude, and the information exchange by 3 orders of magnitude, making it possible to solve many more instances.

Finally, comparing P$^{3/2}$-DPOP against P-DPOP shows that decision privacy can often be achieved at low performance losses. It is only when the number of customers on which depots need to coordinate (first number in column n_C) gets larger, that P$^{3/2}$-DPOP's performance starts to degrade significantly. This is consistent with the theoretical performance analysis in Section 3.4.5. Notice however that, on problem instances with only 2 depots (in italics), decision privacy actually cannot be guaranteed, since all decisions are semi-private information that cannot be protected.

3.8 Summary

In this chapter, we have introduced the definitions of four types of information about a DisCSP or a DCOP that agents solving the problem might want to keep secret. *Agent privacy* is about protecting the presence and identities of the agents; *topology privacy* deals with the presence of topological constructs in the constraint graph (nodes, edges, cycles); *constraint privacy* has to do with the nature of the constraints; and *decision privacy* involves only revealing to an agent the part of the solution that concerns its variables. After reviewing the literature on privacy in DisCSP/DCOP, we have described a new family of P-DPOP algorithms that attempt to provide strong guarantees on the protection of all four types of information. We have shown that our algorithms both score higher privacy-wise than the previous state of the art, and also tend to scale to larger problems on most benchmark problem classes we have investigated.

4 StochDCOP: DCOP under Stochastic Uncertainty

This chapter presents an extension of the traditional DCOP formalism that we call *StochDCOP*, which includes sources of uncertainty in the form of random, uncontrollable variables with known probability distributions. Section 4.1 first formally defines this extension, and illustrates it on various applications. Section 4.3 then presents some previously proposed extensions of DCOP to include uncertainty, and explains how this previous work relates to ours. Sections 4.4 to 4.6 then introduce three new algorithms[1] to solve StochDCOPs. They are all based on DPOP, but the general techniques we propose could easily be applied to other DCOP algorithm as well. Section 4.7 shows that, under certain conditions, some of our algorithms are actually equivalent, in that they produce the same solutions. Section 4.8 then shows how our algorithms can be adapted to use *sampling* when the uncertainty space is too large to be explored exhaustively. Finally, Section 4.9 compares the performances of the three algorithms on various StochDCOP classes.

4.1 DCOP under Stochastic Uncertainty

We first formally define our novel StochDCOP framework, as an extension of the DCOP formalism that includes random variables in addition to the traditional decision variables.

4.1.1 StochDCOP Formalism

The formal definition of a DCOP under Stochastic Uncertainty is an extension of the definition of a DCOP (Definition 2), in which uncontrollable, random variables are used to model the sources of uncertainty in the problem. Additionally, we introduce the notion of an *evaluation function* to measure the quality of a solution.

[1] We have published these algorithms in the following workshop and conference papers: [Léauté and Faltings, 2009a, 2011b].

Definition 13 (StochDCOP)**.** *We define a* Distributed Constraint Optimization Problem under Stochastic Uncertainty *as a tuple* $< \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R}, \Delta, \mathcal{P}, \mathcal{C}, e >$*, where* $\mathcal{A}$*,* $\mathcal{X}$ *and* $\mathcal{D}$ *are defined as in Definition 2:*

- $\mathcal{A} = \{a_1, ..., a_k\}$ *is a set of* agents*;*
- $\mathcal{X} = \{x_1, ..., x_n\}$ *is a set of* decision variables*, such that each variable* x_i *is controlled by a given agent* $a(x_i)$*;*
- $\mathcal{D} = \{D_1, ..., D_n\}$ *is a set of finite variable* domains*, variable* x_i *taking values in* D_i*;*

and furthermore:

- $\mathcal{R} = \{r_1, \ldots, r_q\}$ *is a set of* random variables *modeling future, uncontrollable events;*
- $\Delta = \{\Delta_1, \ldots, \Delta_q\}$ *is a set of (not necessarily finite) domains for the random variables such that* r_i *takes values in* Δ_i*;*
- $\mathcal{P} = \{\pi_1, \ldots, \pi_q\}$ *is a set of* probability distributions *for the random variables, where each distribution* $\pi_i : \Delta_i \rightarrow \mathbb{R}$ *defines the probability law for random variable* r_i*, with* $\int_{r \in \Delta_i} \pi_i(r) dr = 1$*;*
- $\mathcal{C} = \{c_1, \ldots, c_m\}$ *is a set of soft constraints over mixed subsets of decision and random variables. Any constraint is assumed to involve at least one decision variable (constraints over only random variables can be modeled as probability distributions);*
- e *is an* evaluation function *that assigns cost values to assignments to the decision variables in* $\mathcal{X}$*, based on a cost function* $c : \mathcal{X} \times \mathcal{R} \rightarrow \mathbb{R} \cup \{+\infty\}$*:*

$$e_c : D_1 \times \ldots \times D_n \rightarrow \mathbb{R} \cup \{+\infty\} .$$

The restriction of e *to a single random variable* r *will be denoted by* e^r*:*

$$e_c^r : D_1 \times \ldots \times D_n \rightarrow (\mathbb{R} \cup \{+\infty\})^{\bigotimes_{r_i \neq r} \Delta_i} .$$

A solution *is an assignment of values to the decision variables* only *that is independent of the random variables, and that minimizes the evaluation of the sum of all costs:*

$$(x_1^*, \ldots x_n^*) = \arg \min_{x_1 \ldots x_n} \left\{ e_{\sum_i c_i} (x_1, \ldots x_n) \right\} .$$

Random variables are not initially assigned to any specific agent and do not participate in the solution; this is because random variables model uncontrollable uncertainty in the agents' common environment. StochDCOPs are still distributed in nature:

1. The decision power is distributed like in DCOPs, but random variables are not under the control of the agents: their values are drawn by Nature from their respective probability distributions, independently of the decision variables, and after the agents must have chosen values for their decision variables. Solving a StochDCOP involves choosing values for the decision variables *only*, anticipating Nature's future decisions.
2. The knowledge of each random variable, its domain and its probability distribution is shared only among the agents in control of neighboring decision variables.

The evaluation function is used to summarize in one criterion the quality of the chosen solution, which would normally depend on the values of the random variables. A typical evaluation function is the *expectation*, in which case the solution is the assignment that minimizes the expected sum of all costs $\mathbb{E}_{\mathcal{R}}\left[\sum_i c_i\right]$. Other evaluation functions are proposed in the next section.

Notice that following Definition 13, multiple agents' costs can depend on a common random variable, without the agents initially being aware of it. The algorithms we propose differ about whether and how agents discover and reason about such co-dependencies. Furthermore, the probability distributions for the random variables are assumed independent of each other and of the decision variables, which amounts to assuming *independent* sources of *exogenous* uncertainty. Two sources of uncertainty with correlated probability distributions can be represented by a single random variable in order to enforce independence.

4.1.2 Evaluation Functions

As presented in Definition 13, a solution to a StochDCOP is an assignment $(x_1^*, \ldots, x_n^*)$ to all decision variables that is *independent* of the values of the random variables $\mathcal{R} = \{r_1, \ldots, r_q\}$. As a result, the quality of a given solution *does* necessarily depend on the random variables, and is equal to $\sum_i c_i(x_1^*, \ldots, x_n^*, r_1, \ldots, r_q)$. In order to evaluate this solution quality using a single criterion, the StochDCOP defines an *evaluation function* e, such that the optimal solution must minimize $e_{\sum_i c_i}(x_1^*, \ldots, x_n^*) \in \mathbb{R}$. This section presents examples of such evaluation functions, and highlights some of their properties.

Examples of Evaluation Functions

Consider a cost function $c(x, r)$ that depends on a decision variable x and on a random variable r. We propose several approaches to evaluating an assignment to x based on $c(x, r)$: the *expectation* approach, the *robust* approach, and the *consensus* approach.

Expected Solution Quality The *expectation* approach uses the evaluation function that consists in taking the *expected* value of $c(x,r)$, based on r's probability distribution π_r:

$$e_c : x \rightarrow \mathbb{E}_r\left[c(x,r)|x\right] = \int_{r\in\Delta_r} \pi_r(r)c(x,r)dr\ .$$

Using a sample list of values S_r for r instead of its true (possibly continuous) domain Δ_r, the expectation can be approximated as follows:

$$\mathbb{E}_r\left[c(x,r)|x\right] \approx \frac{1}{|S_r|}\sum_{r\in S_r} c(x,r)\ . \tag{4.1}$$

The prefix *Exp-* will denote algorithms using this approach.

***Consensus*: Maximum Probability of Optimality** Bent and van Hentenryck [2004] introduced a new approach for centralized stochastic problems called *consensus*, which differs from the traditional approach based on minimizing the expected cost. They showed that when the number of samples used to approximate the probability distributions of the random variables is limited, for instance by time constraints, it is better to choose a solution that is the optimal one for the largest number of samples, rather than one that minimizes the expected cost across samples. This amounts to maximizing the probability of optimality, which we denote by $\mathbb{P}$, rather than the expected solution quality, and corresponds to choosing the evaluation function that measures (minus) the probability that $c(x,r)$ is minimized:

$$e_c : x \rightarrow \mathbb{P}_r[c(x,r)|x] = -\int_{r\in\Delta_r} \pi_r(r)\cdot\delta_{x=\arg\min_{x'} c(x',r)}(x,r)dr$$

where $\delta_X = 1$ if X is true, 0 otherwise.

When the domain Δ_r of r is approximated by a list S_r of sample values for r, this evaluation function can be approximated as follows, modulo a constant multiplicative factor:

$$\mathbb{P}_r[c(x,r)|x] \approx -\left|\left\{r\in S_r \mid x = \arg\min_{x'} c(x',r)\right\}\right|\ . \tag{4.2}$$

This approach will be referred to using the prefix *Cons-*.

***Robust* Decision-making: Worst-case Solution Quality** In some problem domains, agents might not want to take the risk of choosing a solution that can have a very high cost with some non-zero probability, even if this solution gives minimum expected

cost. In such cases, it can be more desirable to choose the solution that minimizes the *worst-case* cost instead. This can be achieved by choosing the following evaluation function:

$$e_c : x \to \sup_{r \in \Delta_r} \{c(x, r)\} .$$

When a sample list of values S_r is used instead of r's full domain Δ_r, the previous formula can be replaced with the following:

$$e_c(x) \approx \max_{r \in S_r} \{c(x, r)\} .$$

The prefix *Robust-* will be used for algorithms based on this approach.

Properties of Evaluation Functions

We now define some properties that an evaluation function may or may not satisfy, which will have important consequences on the properties of the StochDCOP algorithms presented in this chapter.

Definition 14 (linearity)**.** *An evaluation function e is* linear *if and only if it satisfies the following property:*

$$\forall c_1, c_2, \ \forall x \quad e_{c_1+c_2}(x) = e_{c_1}(x) + e_{c_2}(x) .$$

Of all the examples of evaluation functions given in the previous section, only the *expectation* function is linear.

Definition 15 (commensurability)**.** *An evaluation function e is* commensurable *if and only if its output is commensurable with the output of the cost function c; in other words, if the evaluation $e_c(x)$ remains a cost.*

The *expectation* and *worst-case* evaluation functions are both commensurable, because $\mathbb{E}_r[c(x, r)|x]$ and $\sup_r\{c(x, r)\}$ are still costs. On the contrary, the *consensus* evaluation function is non-commensurable, because $\mathbb{P}[c(x, r)|x]$ is not a cost, but rather a *probability*.

Definition 16 (support)**.** *A commensurable evaluation function e is said to be* supported *if the following property holds:*

$$\forall c, \ \forall x, \ \exists r_c(x) \ \mid \ e_c(x) = c(x, r_c(x)) \tag{4.3}$$

and r_c is then called the support *of e with respect to c.*

Intuitively, whether an evaluation function e is supported is equivalent to whether there exists such a function as $\arg e$. For instance, if random variables have finite domains Δ, then $\sup \equiv \max$, and the *worst-case* evaluation function is clearly supported, with support $r_c(x) = \arg\max_{r\in\Delta}\{c(x,r)\}$, since Eq. (4.3) then holds:

$$\forall c,\ \forall x,\ c(x, r_c(x)) = c(x, \arg\max_{r\in\Delta}\{c(x,r)\}) = \max_{r\in\Delta}\{c(x,r)\} = e_c(x)\ .$$

On the other hand, the *expectation* evaluation function is unsupported, because $\mathbb{E}_r\left[c(x,r)|x\right]$ might not correspond to the value of $c(x,r)$ for any assignment to r.

4.2 Examples of Applications

This section illustrates the StochDCOP formalism on a set of application domains, most of which are stochastic adaptations of DCOP application domains from Section 2.2.

4.2.1 Stochastic Graph Coloring

Consider a stochastic variant of the Max-DisCSP graph coloring problem class introduced in Section 2.2.1, in which some nodes' colors are uncontrollable. Figure 4.1

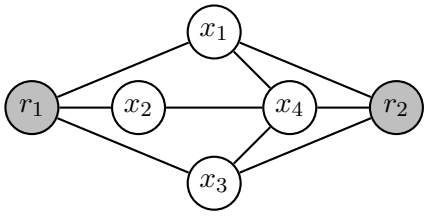
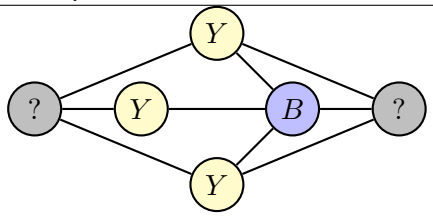
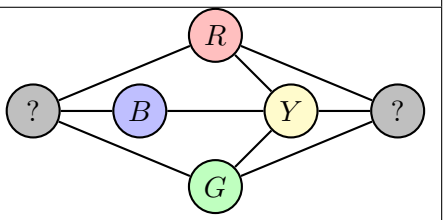


Figure 4.1: Sample stochastic graph coloring problem.

Table 4.1: Optimal colorings for the problem in Figure 4.1.

	expectation and *consensus*	*robustness*
		
$\mathbb{E}[c]$	0.7	1.7
$\max c$	5	2
$\mathbb{P}[c]$	63%	2%

shows a sample problem instance involving four agents $x_1 \dots x_4$ that can take four colors {red, green, blue, yellow}, and two uncontrollable nodes r_1 and r_2, with the same color probability distribution $\{R : 0.4, G : 0.3, B : 0.2, Y : 0.1\}$. Table 4.1 illustrates the optimal solutions to this problem, depending on whether we choose the *expectation* and *consensus* evaluation function (left) or the *robust* evaluation function (right).

4.2.2 Sensor Networks with Moving Targets

Section 2.2.3 introduced a class of sensor network problems, in which omni-directional sensors have to decide whether to be *on* or *off* in order to observe a set of targets at known locations. We now describe a stochastic variant of this problem class, in which only the *initial* positions of the targets are known, but their respective positions at the next time step are only known with some probabilities. The goal is now for the sensors to commit to being *on* or *off*, so as to maximize the reward of observing the targets in their initial positions *and* their next, uncertain positions (the sensors cannot change their own states in between, and only plan one step ahead). The utility obtained by tracking targets therefore becomes a sum of *three* terms, the first two being the same as in Section 2.2.3:

- Each sensor gets a negative utility of -1 when it is *on*;
- For each target, if 3 or more sensors close to its initial, known position are *on*, the sensors share a utility of 12. If only 2 sensors are observing the target in its initial position, this utility drops to 6, and down to 1 if only one sensor observes it.
- For each target, for each next uncertain position P it can move to, if 3 or more sensors close to P are *on*, and the target actually moves to P, the sensors share a utility of 12. The utility drops down to 6 for only 2 sensors, and 1 for only one sensor.

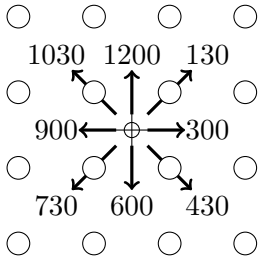


Figure 4.2: Allowed target moves.

The StochDCOP formulation for this stochastic problem is the same as for the deterministic problem introduced in Section 2.2.3, with two additional components:

- For each target k, the StochDCOP contains one random variable t_k that takes values in the domain $\Delta_{t_k} = \{130, 300, 430, 600, 730, 900, 1030, 1200\}$, and represents the next, uncertain position of the target, as illustrated in Figure 4.2. The value 900 corresponds to moving towards 9 o'clock, i.e. moving horizontally to the left.

- For each target k, and each possible next position $P \in \Delta_{t_k}$ for the target, the following 5-ary, soft constraint expresses the utility obtained by the four sensors around P if they eventually are able to observe the target:

$$u_k^P : (x_{i_P}^{j_P}, x_{i_P}^{j_P+1}, x_{i_P+1}^{j_P}, x_{i_P+1}^{j_P+1}, t_k) \rightarrow \begin{cases} u_k(x_{i_P}^{j_P}, x_{i_P}^{j_P+1}, x_{i_P+1}^{j_P}, x_{i_P+1}^{j_P+1}) & \text{if} \quad t_k = P \\ 0 & \text{if} \quad t_k \neq P \end{cases} \tag{4.4}$$

  where u_k is defined in Eq. (2.2). This constraint is illustrated in Figure 4.3, for the direction of motion $P = 900$. The constraints for the other directions of motion are defined similarly, all involving the *same* random variable t_k. The 4-ary constraint u_k is defined as in Figure 2.4, and encodes the utility obtained by tracking the target in its known, initial position.

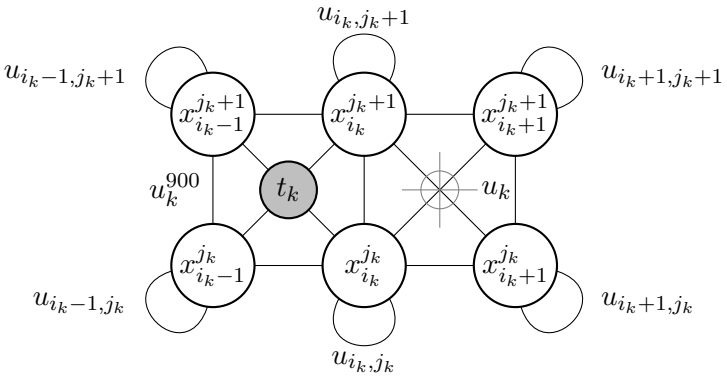
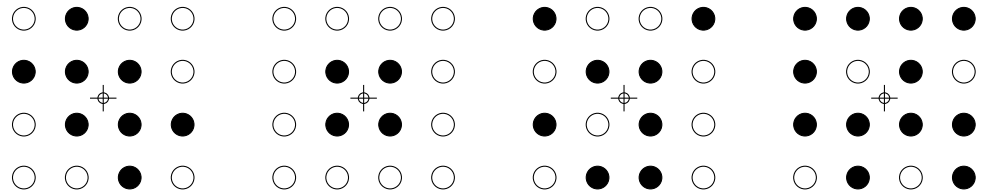


Figure 4.3: Partial constraint graph for target k.

This models decomposes the reward for tracking the target in its next, uncertain position into eight 5-ary constraints, which is much more tractable than using a single 17-ary constraint. However, as a consequence, no sensor initially has a global view of the impact of the target's move on the overall utility. We believe this short-sightedness is an important feature of many real-life, distributed problems. This thesis is precisely about algorithms that differ on whether and how agents acquire a global view of the impact of various sources of uncertainty on the overall utility. For instance, in the sensor network problem with moving targets, the sensors need to exchange a certain amount of information about uncertainty in order to discover that the target actually

cannot escape: regardless of its next, uncertain move, it will always be in sight of four sensors.

Figure 4.4(a) illustrates the optimal solution found using the *expectation* evaluation function for the single-target problem instance in Figure 4.2, when all target moves are assumed equiprobable (*on* sensors are in black). This is the configuration that maximizes the expected solution quality. Notice that, while the problem in Figure 4.2 has multiple symmetries, optimal solutions (according to some evaluation function) do not necessarily exhibit the same symmetries.



(a) maximum expected utility (b) optimal robust 1-configuration (c) optimal robust 2-configuration (d) optimal robust 3-configuration

Figure 4.4: Some remarkable *on/off* sensor configurations.

When using the *robust* evaluation function, the following definition singles out some remarkable robust solutions to the sensor network problem, which will be used in a later section to compare various algorithms. Such robust sensor configurations are illustrated in Figure 4.4.

Definition 17 (robust n-configuration)**.** *We call a* robust n-configuration *a solution to the sensor network problem with moving targets in which any given target will always remain in sight of at least* n on *sensors, regardless of the direction in which it moves. Such a configuration is called* optimal *if it also minimizes the number of* on *sensors.*

4.2.3 Stochastic Vehicle Routing

We now introduce a stochastic variant of the multi-agent VRP class described in Section 2.2.5.

Definition 18 (StochDisSDMDVRP)**.** *The* Stochastic, Distributed, Shared-Deliveries, Multiple-Depot VRP *is defined as a tuple* $< D, n_V, Q_{\max}, L_{\max}, H, C, R, Q >$*, where* D*,* n_v*,* $Q_{\max}$*,* H *and* Q *are defined as in Definition 5:*

- $D = \{d_1, \ldots, d_{n_D}\}$ *is a set of depots in the Euclidian plane, each controlled by a different delivery company;*
- n_V *is the number of vehicles at each depot;*

- $Q_{\max}$ *and* $L_{\max}$ *are respectively the maximum load and maximum route length of any vehicle;*
- $H \leq L_{\max}/2$ *is the visibility horizon of each company, which defines the boundaries of its knowledge of the overall problem. Depot* d_i *is only aware of and can only serve the customers that are within distance* H*;*
- $Q = \{q_1, \ldots, q_{n_C}\} \in \mathbb{N}^{n_C}$ *are the customer demands, which can be split among multiple companies.*

and furthermore:

- $C = \{c^1, \ldots, c^{n_C}\}$ *is a set of customers;*
- $R = \{r_1, \ldots, r_{n_C}\}$ *are the uncertain locations for the customers, with known probability distributions;*

The goal is for the companies to agree on who should serve which customers, using which vehicle routes, so as to fully serve all visible customers, at minimal evaluated total route length.

The StochDCOP formulation for this problem class is the same as in Section 2.2.5, except that it now also includes one random variable r_j for each customer c^j that corresponds to her uncertain location, and each depot's VRP constraint is now also

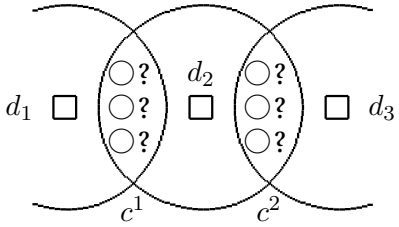


Figure 4.5: A StochDisSDMDVRP instance.

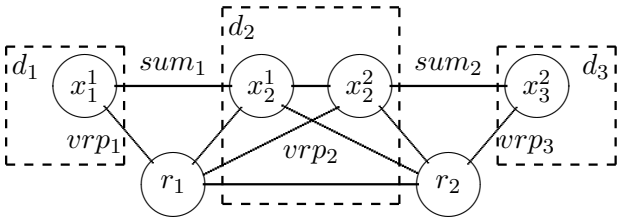


Figure 4.6: The StochDCOP constraint graph for Figure 4.5.

expressed over the random variables associated with the customers it can serve. This is illustrated in Figures 4.5 and 4.6, on an instance involving two customers c^1 and c^2 with each three possible uncertain locations, and three depots such that depot d_1 can only serve c^1, d_2 can serve both customers, and d_3 only c^2.

4.2.4 The Distributed Kidney Exchange Problem

The Kidney Exchange Problem introduced by Saidman et al. [2006] is the problem of matching potential kidney donors to patients, and is an example of a broader class of problems that involves looking for cycles in a directed graph. As a new class of benchmarks for DCOP and StochDCOP, we propose a distributed version of this problem, which can be defined as follows.

Definition 19 (DKEP)**.** *The* Distributed Kidney Exchange Problem *can be defined as a directed graph, in which each node is an agent acting on behalf of a pair of people formed by a patient awaiting a kidney transplant, and a friend or relative of the patient, who is willing to donate one kidney to save her friend, but who is unfortunately biologically incompatible. The directed edges in the graph indicate which donor (at the tail of the edge) can give a kidney to which patient (at the head).*

An optimal solution to the problem is a selection of non-intersecting directed cycles in the graph that has maximum total length, i.e. an assignment of donors to patients that maximizes the number of saved patients, while only allowing cycles of length 2 or 3. Larger cycles would involve at least 8 people (4 pairs) and are not logistically implementable, because all transplantations in a cycle must be executed simultaneously to make sure no donor changes her mind once her friend or relative has received a kidney.

In order to use this new problem class as a benchmark for our StochDCOP algorithms, we also propose the following stochastic version of the problem.

Definition 20 (StochDKEP)**.** *A* Stochastic Distributed Kidney Exchange Problem *is a DKEP in which each node in the directed graph is labeled with the vital prognosis for the corresponding patient, i.e. her probability to survive until a hypothetical transplantation.*

A StochDKEP can be represented as a StochDCOP in which each agent i (i.e. each patient-donor pair) owns two decision variables f_i and t_i whose values are the IDs of the agent j that it gets a kidney from, respectively the agent k it gives a kidney to in return. These two variables can also take the value 0, which indicates that the agent does not take part in any transplantation. We also introduce one binary random variable s_i per agent i, such that $s_i = 1$ if and only if the patient survives until the transplantation. The constraints are then of the following 4 types:

I give $\iff$ I receive: Each agent i expresses the following internal, binary, hard constraint over its two decision variables f_i and t_i:

$$f_i = 0 \iff t_i = 0$$

which is satisfied if and only if the variables are either both zero (the agent does not give nor receives a kidney) or both non-zero (the agent gives a kidney and receives one in return).

I give to you $\iff$ you get from me: For each edge $i \to j$, i.e. for each pair of agents (i, j) such that i can give a kidney to j, the following binary, hard constraint enforces the consistency of the two agents' decision variables:

$$f_i = j \iff t_j = i\,.$$

2-cycles: For each agent i, and for each other cross-compatible agent j (i.e. we have the bidirectional edge $i \leftrightarrow j$), the following 4-ary, soft constraint expresses the fact that agent i gets a utility of 10 if the two agents take part in a 2-cycle and both patients survive until the exchange; otherwise it gets a utility of 0:

$$u_2 : (f_i, t_i, s_i, s_j) \to \begin{cases} 10 & \text{if } f_i = t_i = j \wedge s_i = s_j = 1 \\ 0 & \text{else} \end{cases}\,.$$

3-cycles: For each agent i, if there exists a 3-cycle $i \to j \to k \to i$ in the graph, then agent i gets a utility of 9 if the 3-cycle is implemented, and all three patients survive until the exchange or one of the two other patients dies, but the two remaining agents can fall back to a 2-cycle; the utility is 0 otherwise.

$$u_3 : (f_i, t_i, t_j, s_i, s_j, s_k) \to \begin{cases} 9 & \text{if } \left(\begin{array}{ll} & f_i = k \wedge t_i = j \\ \wedge & t_j = k \wedge s_i = 1 \end{array} \right) \wedge \left(\begin{array}{ll} & s_j = s_k = 1 \\ \vee & (s_j = 1 \neq s_k \wedge \exists\, i \leftrightarrow j) \\ \vee & (s_k = 1 \neq s_j \wedge \exists\, i \leftrightarrow k) \end{array} \right) \\ 0 & \text{else} \end{cases}$$

The per-agent utility of a 3-cycle is set lower than the per-agent utility for a 2-cycle ($9 < 10$) to express the fact that 3-way exchanges are more complex to implement and therefore less desirable than 2-way exchanges.

4.3 Related Work

This section presents some previously proposed extensions of the DCOP framework to include uncertainty. Section 4.3.1 first briefly describes the *DCEE* framework, in which the utility values of some constraints may be unknown, and the corresponding decisions need to be tentatively made for these utilities to be observable. Section 4.3.2 then explains a DCOP extension in which uncertainty is modeled by *utility distributions*

rather than through the introduction of random variables. Section 4.3.3 recalls the *Quantified DCOP* framework, which introduces universally quantified variables, while normal decision variables are existentially quantified.

4.3.1 Distributed Coordination of Exploration and Exploitation

Jain et al. [2009] have proposed to extend the traditional DCOP formalism, by replacing the assumption that each constraint is fully known to all agents involved, by the much weaker assumption that each constraint's *scope* is known to all agents involved, but none of the *values* of the constraint (i.e. the costs) is initially known to *any* agent. The actual cost of a given assignment to decision variables can only be observed after the corresponding decisions have been collectively made and implemented. Given a fixed maximum number of decision rounds, the problem then becomes one of balancing *exploration* and *exploitation* [Taylor et al., 2010], which is why they call their framework *Distributed Coordination of Exploration and Exploitation (DCEE)*.

While this framework can be useful to model various real-world problems, it does not incorporate any model of uncertainty. In contrast, one of the most prominent features of the StochDCOP formalism is that it is able to model problems in which some probabilistic knowledge of uncertainty is available, and in which such knowledge should be exploited to produce better solutions.

4.3.2 DCOP with Utility Distributions

Atlas and Decker [2010] have proposed another extension of the traditional DCOP formalism, by adding a notion of stochastic uncertainty. Their framework (later revisited by Stranders et al. [2011]) can be formally defined as follows, rephrased in terms of cost minimization.

Definition 21. *A* Distributed Constraint Optimization Problem with Cost Distributions *is defined as a tuple* $< \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{E} >$*, where* $\mathcal{A}$*,* $\mathcal{X}$*, and* $\mathcal{D}$ *are defined as in Definition 2:*

- $\mathcal{A} = \{a_1, ..., a_k\}$ *is a set of* agents*;*
- $\mathcal{X} = \{x_1, ..., x_n\}$ *is a set of* decision variables*, such that each variable* x_i *is controlled by a given agent* $a(x_i)$*;*
- $\mathcal{D} = \{D_1, ..., D_n\}$ *is a set of finite variable* domains*, variable* x_i *taking values in* D_i*;*

and furthermore:

- $\mathcal{C} = \{c_1, \ldots, c_m\}$ *is a set of soft,* probabilistic *constraints, where each constraint is a* $s(c_i)$*-ary function* $c_i : D_{i_1} \times \ldots \times D_{i_{s(c_i)}} \rightarrow (\mathbb{R} \times [0,1])^{\mathbb{N}}$ *that assigns, to each com-*

bination of assignments to a subset $\{x_{i_1}, \ldots, x_{i_{s(c_i)}}\}$ *of decision variables, a discrete* cost distribution *that can be represented by a finite set of (value, probability) pairs:*

$$c_i(x_{i_1}, \ldots, x_{i_{s(c_i)}}) = \left\{ \left(c_i^k(x_{i_1}, \ldots, x_{i_{s(c_i)}}), \pi_i^k(x_{i_1}, \ldots, x_{i_{s(c_i)}}) \right) \right\}_{k \in \left[0, \delta_i(x_{i_1}, \ldots, x_{i_{s(c_i)}})\right]}$$

such that $\sum_{k \in \left[0, \delta_i(x_{i_1}, \ldots, x_{i_{s(c_i)}})\right]} \pi_i^k(x_{i_1}, \ldots, x_{i_{s(c_i)}}) = 1$*;*

- $\mathcal{E} = \{e_1, \ldots, e_n\}$ *is a set of* evaluation functions*, one per decision variable, where each evaluation function* $e_i : \mathbb{R}^{\Delta} \to \mathbb{R}$ *reduces a cost distribution to a single cost value.*

Intuitively, this framework extends the DCOP framework by having constraints return *cost distributions* instead of just single cost values. The resulting framework can be simultaneously seen as a generalization, and a particular case of our StochDCOP framework. First, it is a generalization, because each decision variable is allowed to have its own evaluation function, which is not necessarily the same for all decision variables. Furthermore, it allows *endogenous* uncertainty, i.e. probability distributions that may depend on the decision variables, while a StochDCOP as defined in Definition 13 only allows *exogenous* uncertainty. If one wanted to reformulate a DCOP with Cost Distributions into a StochDCOP, one would have to express each probabilistic constraint c_i as a StochDCOP constraint $\tilde{c}_i(x_{i_1}, \ldots, x_{i_{s(c_i)}}, r_i) = c_i^{r_i}(x_{i_1}, \ldots, x_{i_{s(c_i)}})$, whose scope includes an additional random variable r_i of domain $\Delta_i = \left[0, \delta_i(x_{i_1}, \ldots, x_{i_{s(c_i)}})\right]$, and of probability distribution $\pi_i(x_{i_1}, \ldots, x_{i_{s(c_i)}}, r_i) = \pi_i^{r_i}(x_{i_1}, \ldots, x_{i_{s(c_i)}})$, which depends on the decision variables $x_{i_1}, \ldots, x_{i_{s(c_i)}}$. The StochDCOP formalism could be extended to support these two features, if the application domain requires it.

On the other hand, this is a special case of StochDCOP, because no two constraints can depend on the same source of uncertainty. Each cost distribution of each constraint is based on its own probability distribution, which is independent of all other probability distributions. This is an important limitation of this formalism, because it cannot be used to model problems in which multiple agents' costs depend on *common* sources of uncertainty. For example, in Figure 4.1, it would not be possible to express the fact that the respective cost distributions of nodes x_1 and x_2 are correlated by the fact that they both need to have a color different from that of the uncontrollable node r_1.

4.3.3 Quantified DCOP with Virtual Adversaries

This section presents previous work on *Quantified DCOPs*, which are another extension to the traditional DCOP framework that bares similarities with our StochDCOP extension. Some of the approaches proposed to solve Quantified DCOPs can be adapted to solve StochDCOPs, as we show in Section 4.4.

Definition of Quantified DCOP (QDCOP)

The *QDCOP* formalism was proposed by Matsui et al. [2010]; we present below a formal definition that is a slight reformulation from their original definition, so as to highlight the similarities with our definition of a StochDCOP (Definition 13).

Definition 22 (QDCOP)**.** *A* QDCOP *can be defined as a tuple* $< \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R}, \Delta, \mathcal{C}, \mathcal{Q} >$*, such as* $\mathcal{A}$*,* $\mathcal{X}$*,* $\mathcal{D}$*,* $\mathcal{R}, \Delta$*, and* $\mathcal{C}$ *are defined as in Definition 13:*

- $\mathcal{A} = \{a_1, ..., a_k\}$ *is a set of* agents*;*
- $\mathcal{X} = \{x_1, ..., x_n\}$ *is a set of* decision variables*, such that each variable* x_i *is controlled by a given agent* $a(x_i)$*;*
- $\mathcal{D} = \{D_1, ..., D_n\}$ *is a set of finite variable* domains*, variable* x_i *taking values in* D_i*;*
- $\mathcal{R} = \{r_1, \ldots, r_q\}$ *is a set of* random variables *modeling future, uncontrollable events;*
- $\Delta = \{\Delta_1, \ldots, \Delta_q\}$ *is a set of (not necessarily finite) domains for the random variables such that* r_i *takes values in* Δ_i*;*
- $\mathcal{C} = \{c_1, \ldots, c_m\}$ *is a set of soft constraints over mixed subsets of decision and random variables. Any constraint is assumed to involve at least one decision variable (constraints over only random variables can be modeled as probability distributions);*

and furthermore:

- $\mathcal{Q}$ *is a partial order on the variables in* $\mathcal{X} \cup \mathcal{R}$ *defined as follows:*

$$\mathcal{X}_0 \succ_\mathcal{Q} \mathcal{R}_0 \succ_\mathcal{Q} \ldots \succ_\mathcal{Q} \mathcal{X}_i \succ_\mathcal{Q} \mathcal{R}_i \succ_\mathcal{Q} \ldots \succ_\mathcal{Q} \mathcal{X}_p \succ_\mathcal{Q} \mathcal{R}_p$$

  where $(\mathcal{X}_i)_{0 \leq i \leq p}$ *and* $(\mathcal{R}_i)_{0 \leq i \leq p}$ *are partitions respectively of* $\mathcal{X}$ *and* $\mathcal{R}$ *of same cardinality, and* $\mathcal{R}_p$ *is allowed to be empty.*

  Matsui et al. omitted to specify how the knowledge of $\mathcal{Q}$ *is distributed among agents; we assume each agent knows the relative orders of all variables it has constraints with.*

A solution $(x_1^*, \ldots, x_n^*)$ *is an assignment to each decision variable* $x_j \in \mathcal{X}$ *that is a function of the decision variables and random variables preceding* x_j *in* $\mathcal{Q}$*:*

$$x_j^* : \left(\bigotimes_{x_k \succ_\mathcal{Q} x_j} D_k \right) \times \left(\bigotimes_{r_k \succ_\mathcal{Q} x_j} \Delta_k \right) \to D_j$$

such that the sum of all constraints is equal to:

$$\sum_i c_i(x_1^*, \ldots, x_n^*) = \min_{\mathcal{X}_0} \left\{ \max_{\mathcal{R}_0} \left\{ \ldots \min_{\mathcal{X}_0} \left\{ \max_{\mathcal{R}_0} \sum_i c_i \right\} \right\} \right\} .$$

A QDCOP can be seen as a *multi-stage* StochDCOP in which the evaluation function is the worst-case function. This makes it natural and promising to attempt to modify existing QDCOP algorithms in order to solve StochDCOPs, when the evaluation function can be different; we show how to do this in Section 4.4.

Virtual Adversarial Agents

Matsui et al. proposed several modified versions of ADOPT [Modi et al., 2005] to solve QDCOPs. They are based on the intuition that random variables can be seen as virtual adversaries, whose goal is to *maximize* the overall cost, instead of minimizing it as the normal agents do. Intuitively, they showed that QDCOP algorithms can be obtained from existing DCOP algorithms simply by assigning random variables to virtual agents that perform the “opposite" DCOP algorithm. While this approach is simple and elegant, it suffers from a number of drawbacks and limitations.

First, virtual adversarial agents must be simulated by the real, physical agents, which has important privacy implications: to simulate a random variable $r \in \mathcal{R}$, an agent a needs to know all the constraints involving r, including the constraints that do not involve any variable owned by a. This violates the traditional assumption that a constraint is only known to the agents involved in it, and is a breach of constraint privacy.

The second drawback is that the pseudo-tree used in the algorithm must be *consistent with the partial order* $\mathcal{Q}$. This restriction on the allowed pseudo-trees has implications both on performance and privacy. In terms of performance, restricting the allowed pseudo-trees can only degrade the performance of the algorithm; for instance, ADOPT’s performance is exponential in the depth of the pseudo-tree, and DPOP’s is exponential in the width of the pseudo-tree.

In terms of agent privacy, it is desirable that two agents that do not share any constraint do not need to know each other’s identities (or even existence), and in particular, do not need to communicate directly. This means that, since two parent-child variables in the chosen pseudo-tree need to exchange messages, they should always be owned by agents that share constraints. This can be incompatible with respecting the order in $\mathcal{Q}$. For instance, there does not exist any consistent pseudo-tree for the constraint graph in Figure 4.1 that does not violate the requirement that all parent-child relationships are between neighboring variables. This is due to the fact that the consistency constraint imposes that all three variables x_1, x_2 and x_3 should be in the same branch of the

pseudo-tree, because they are all constrained to the random variable r_1.

To construct a consistent pseudo-tree, Matsui et al. therefore proposed to add *null links* between parents and children, following a distributed algorithm first proposed by Silaghi and Yokoo [2007], which proceeds as follows. The algorithm initially makes three assumptions. The first, explicit assumption is that the agents know a *total* ordering of the variables that is consistent with the partial order $\mathcal{Q}$. Such a total order can be obtained for instance by breaking ties in the partial order using lexicographic order of variable names. The second, implicit assumption is that any agent can communicate directly with any other agent; this is necessary because null links may be added between any pair of variables. The third, also implicit assumption is that each virtual agent has already been assigned to a physical agent that simulates it, so that virtual agents can participate in the construction of the consistent pseudo-tree. These two last assumptions have important consequences on privacy, as already discussed. The algorithm then proceeds in a bottom-up fashion, with each variable incrementally computing the list of its ancestors in the consistent pseudo-tree, and broadcasting this list to its ancestors.

4.4 Complete, Centralized Reasoning: Comp-$\mathbb{E}$[DPOP]

This section proposes an adaptation to StochDCOP of the general approach based on virtual adversarial agents, previously applied by Matsui et al. [2010] to ADOPT to solve QDCOPs (Section 4.3.3). Sections 4.4.2 and 4.4.3 also propose new distributed algorithms to produce respectively pseudo-trees and *consistent* pseudo-trees. Section 4.4.4 then discusses the properties of the resulting StochDCOP algorithm, which we call *Comp-*$\mathbb{E}$*[DPOP]*, because the use of consistent pseudo-trees guarantees completeness.

4.4.1 General Approach Based on Virtual Agents

In Comp-$\mathbb{E}$[DPOP], random variables are assigned to virtual agents that behave exactly like the real agents, except that they project random variables using the StochDPOP's evaluation function, instead of using the *min* like for the projection of decision variables. In the special case of QDCOPs, the evaluation function is the *max*, and therefore the virtual agents are qualified as *adversarial* because their objective is to maximize the total cost, while the real agents want to minimize it.

We generalize this approach to all *commensurable* evaluation functions (Definition 15). For instance, if the evaluation function is the *expectation*, then the virtual agent for random variable r will project it out of a cost $c(r,\cdot)$ by reporting $\mathbb{E}_r\left[c(r,\cdot)|\cdot\right]$. Notice that the evaluation function e may or may not be *supported* (Definition 16), which means that, contrary to decision variables, projecting a random variable may or may

not correspond to choosing a specific value for the variable in the support of e. For instance, the (supported) *worst-case* evaluation function has virtual agents choose counter-optimal values for their random variables, and report the corresponding maximal cost. On the other hand, if a virtual agent uses the (unsupported) *expectation* evaluation function, then it reports the expected cost, which is not necessarily achieved for any allowed value of its random variable.

Like in [Matsui et al., 2010], Comp-$\mathbb{E}$[DPOP] uses a *consistent* pseudo-tree ordering of the variables. In the StochDCOP context, this means that a random variable r must be lower in the pseudo-tree than any decision variable it is constrained with; in other words, it must be a descendent of all its neighboring decision variables. An important consequence of this is that the random variable r and all its neighboring decision variables must all belong to the same branch of the pseudo-tree, since cross-edges between different branches of the pseudo-tree are by definition not allowed.

Generating such a consistent pseudo-tree is not a trivial task, especially since Figure 4.1 showed an example of a constraint graph for which no such consistent pseudo-tree exists without violating the assumption that all parent-child relationships must be between neighboring variables. The following section proposes a new, distributed protocol to generate a consistent pseudo-tree for StochDCOPs. An example of such a consistent pseudo-tree for Figure 4.1 is illustrated in Figure 4.7, along with the UTIL messages exchanged by Rob-Comp-$\mathbb{E}$[DPOP] (represented in intensional form rather than in table form for conciseness). Notice that r_2 is the parent of x_2, even though they are not neighbors.

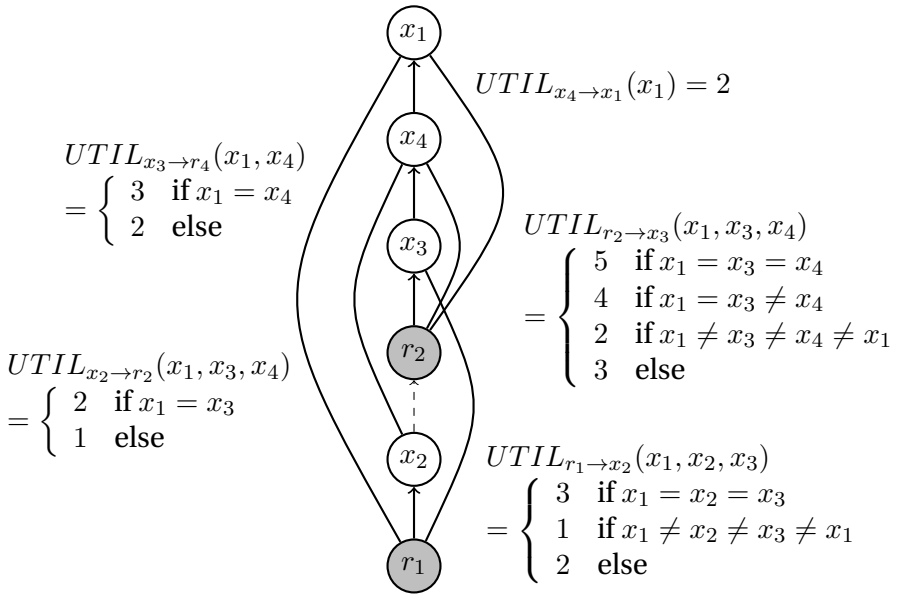


Figure 4.7: A *consistent* pseudo-tree for the constraint graph in Figure 4.1.

4.4.2 Distributed Pseudo-tree Generation

Previous work on pseudo-tree-based algorithms (Section 2.4.2) decomposed the pseudo-tree generation phase into two sub-phases: 1) distributed election of the root of the pseudo-tree, and 2) distributed depth-first traversal of the constraint graph. This approach has two disadvantages:

1. To guarantee correctness and termination, the distributed root election procedure (Algorithm 5) requires that all agents know an upper bound on the diameter of the constraint graph, which is not necessarily the case in reality. Furthermore, loose upper bounds hinder performance, as the number of messages exchanged increases linearly with the bound.
2. The distributed depth-first traversal of the constraint graph cannot start before the root variable has been chosen, which is inefficient.

Parallel Constraint Graph Traversal

We here propose a new distributed algorithm for pseudo-tree generation (Algorithm 24), which does not require any knowledge about the diameter of the constraint graph, and which performs both root election and depth-first traversal in parallel. The algorithm is generally applicable to all DisCSPs or DCOPs[2] (not just to StochDCOPs), and proceeds as follows. First, each variable x is assigned a *score* s_x (line 2) based on a specified heuristic, such as the traditional *most connected* heuristic. Section 4.6.4 studies the impact of the use of other scoring heuristics.

Each variable x then initiates its own distributed, depth-first traversal of the constraint graph, with itself as the candidate root. All messages belonging to its traversal are indexed by the score s_x so as not to be mistaken for messages belonging to other traversals initiated by other variables. For each traversal with candidate score s, variable x stores as $marks[s][y]$ the status of each neighbor y, which can be either a *parent*, a *child*, a *pseudo-parent*, a *pseudo-child*, or *open* if variable y remains to be traversed (line 4). Variable x initiates its traversal by heuristically choosing a neighbor y_0 as its first child (line 5) and sending it a message (CHILD, s_x) (line 6).

When variable x receives a message from a neighbor y_i (line 8), it only reacts if the message's score s is equal to or greater than its own score s_x (line 9). Otherwise, the message is discarded, such that each traversal is interrupted as soon as a variable discovers that the traversal's candidate root does not have the maximum score. If it is the first message with score s received by variable x, the sender y_i is identified as

[2] In particular, this pseudo-tree generation algorithm could, in theory, also be adapted for use in P*-DPOP; however, the potential new privacy leaks resulting from running multiple traversals in parallel would have to be investigated.

Algorithm 24 Pseudo-tree generation algorithm for variable x

```
1: if x has at least one neighbor then
2:     Choose a heuristic score s_x for x
3:     if ∀ neighbors y of x, s_x > s_y then
4:         marks[s_x][y] ← open for all neighbors y of x
5:         marks[s_x][y_0] ← child for a heuristically chosen neighbor y_0 of x
6:         Send the message (CHILD, s_x) to y_0
7:     loop
8:         Wait for an incoming message (type, s) from a neighbor y_i
9:         if s ≥ s_x then
10:            if marks[s] = null then
11:                marks[s][y] ← open for all neighbors y of x
12:                marks[s][y_i] ← parent
13:            else if type = CHILD and marks[s][y_i] = open then
14:                marks[s][y_i] ← pseudo-child
15:                Send message (PSEUDO, s) to y_i
16:                next
17:            else if type = PSEUDO then
18:                marks[s][y_i] ← pseudo-parent
19:            else if type = ROOT then
20:                s_root ← s
21:                break
22:            Heuristically choose y_j such that marks[s][y_j] = open
23:            if there exists such a y_j then
24:                marks[s][y_j] ← child
25:                Send the message (CHILD, s) to y_j
26:            else
27:                Look up y_j such that marks[s][y_j] = parent
28:                if there exists such a y_j then
29:                    Send message (CHILD, s) to y_j
30:                else
31:                    s_root ← s
32:                    break
33:    Record marks[s_root] as the chosen pseudo-tree
34:    Send message (ROOT, s_root) to all children of x in the chosen pseudo-tree
```

the parent of x in this traversal (lines 10 to 12). Else, if the message is a CHILD token received from an *open* neighbor, then the sender y_i is identified as a pseudo-child, which variable x reveals to y_i by responding with a PSEUDO token (lines 13 to 18). The CHILD token is then recursively passed from *open* neighbor to *open* neighbor, until none remains (lines 22 to 26).

When variable x has no more open neighbors, it initiates a traversal backtrack by returning the CHILD token to its parent (lines 27 to 29). If the variable has no parent, it identifies itself as the chosen root (lines 31 to 33); this can only happen for the single

variable with highest score, as all other traversals have necessarily been prematurely interrupted (line 9). The chosen root then notifies all variables of its score by propagating down a ROOT token (lines 34 and 19 to 21).

Complexity Analysis

The worst-case complexity of the pseudo-tree generation phase, in terms of number of messages, can be analyzed as follows. Let us assume for simplification that the variables are indexed by increasing scores: $s_{x_1} < \ldots < s_{x_n}$; this assumption can be made without loss of generality. Each variable x_i initiates a traversal of the constraint graph rooted at x_i, which is interrupted as soon as the first variable $x_{j>i}$ is visited; in the worst case, this happens after all variables $x_{j\leq i}$ have already been visited. This represents the exchange of at most 2 messages per edge in the subgraph containing only the $x_{j\leq i}$ (1 CHILD down and 1 CHILD up for tree edges, 1 CHILD up and 1 PSEUDO down for back-edges), summing up to at most $2\frac{d\cdot i}{2} = d\cdot i$ messages, where d is the degree of the constraint graph. Summing up across all variables x_i and adding the n ROOT messages yields a worst number of messages of $n+\sum_{i=1\ldots n} d\cdot i = n+d\cdot n\frac{n+1}{2} \in O(dn^2)$. All messages have constant size, containing only a score and a destination variable.

4.4.3 Generating a *Consistent* Pseudo-tree

We now describe a simple method to generate *consistent* pseudo-trees, in a distributed fashion. First recall that the consistency criterion imposes that any given random variable must be in the same pseudo-tree branch as all its neighboring decision variables, and that it must be lower than these decision variables in the branch. To achieve this, we add *null links* in a manner similar to Matsui et al. [2010], except that our algorithm does not make any of the restrictive assumptions made by Matsui et al.: it does not assume that the agents know a total ordering of the variables, nor does it assume any-to-any communication between the agents, nor that virtual agents first need to be assigned to physical agents.

We do however make the necessary assumption that, for any given random variable r, all agents a_i owning a decision variable constrained with r must know each other, know that they each control such a decision variable, and must be able to communicate with each other. This assumption is necessary to implement the *virtual agents* approach, because once a consistent pseudo-tree has been created, the agents a_i will have to agree among themselves as to who should simulate the virtual agent for r, and they will then have to send to the chosen agent a^r all their constraints involving r, so that a^r can simulate the virtual agent.[3] Let us now introduce a new definition with respect

[3] In practice, it is sufficient to have at least *one* agent for each random variable r that knows all other agents concerned with r.

to a graphical representation of a StochDCOP.

Definition 23 (Decisional constraint graph)**.** *The* decisional constraint graph *of a given StochDCOP is obtained from the primal constraint graph, in which variables are nodes and edges correspond to sharing a constraint, by removing all nodes corresponding to random variables, as well as all edges linked to these nodes. The resulting graph contains only nodes corresponding to decision variables.*

Notice that if two decision variables in the StochDCOP share only a single n-ary constraint that also involves a random variable, then the two decision variables remain neighbors in the decisional constraint graph, because only the edges *involving random variables are removed, not the full constraints.*

The protocol then proceeds in three steps:

1. **Modification of the constraint graph:** The protocol starts with the *decisional constraint graph*, excluding the random variables. This is necessary because virtual agents have not yet been assigned to real agents, and therefore they cannot be simulated in order to participate in the next step. Agents then create *null links* between any pair of decision variables that share constraints with a common random variable in the StochDCOP, unless they are already neighbors in the decisional constraint graph. More formally: for each random variable r, for each pair of decision variables (x_i, x_j) that are neighbors of r but not neighbors of each other, add a trivial binary constraint between x_i and x_j that assigns the cost 0 to all combinations of assignments to the two variables. This ensures that all decision variables constrained with r will belong to the same branch, and does not change the set of solutions to the problem.

2. **Depth-first traversal:** Run a depth-first traversal algorithm on the modified decisional constraint graph, such as the one in Algorithm 24. This results in a pseudo-tree involving only decision variables.

3. **Assignment of virtual agents:** For each random variable r, the neighboring decision variable x^r that is the lowest in the pseudo-tree is chosen to simulate the virtual agent for r. All other agents send to x^r their constraints involving r. The random variable r is added to the pseudo-tree as a child of x^r, and all constraints involving r are added to the pseudo-tree as well. The result is guaranteed to remain a valid pseudo-tree, as all decision variables constrained with r are ancestors of x^r, and it is also guaranteed to be a *consistent* pseudo-tree, since r is below all its neighboring decision variables.

   Optionally, all null links can now be removed from the pseudo-tree, except if removing a null link would disconnect a child from its parent.

Finally, notice that this algorithm has better privacy properties than the one proposed by Matsui et al. [2010], because it only creates null links between decision variables that are linked to a common random variable, instead of between possibly totally unrelated variables. A parallel with earlier work by Matsui et al. [2008] can also be made (see Section 4.6.3).

4.4.4 Algorithm Properties

This section analyzes some of the properties of Comp-$\mathbb{E}$[DPOP], in terms of complexity, completeness, and privacy.

Complexity

In terms of number of messages and amount of information exchanged, let us first consider the added complexity of generating a *consistent* pseudo-tree, as opposed to just any pseudo-tree like in DPOP (Section 2.4.4). The modification of the constraint graph (Step 1) does not necessitate the exchange of any message, since it only requires each agent to modify its local view of the problem, and can be done with the agents' initial respective knowledge without exchanging information about the problem. The depth-first traversal (Step 2) is $O(d'n^2)$, where d' is the degree of the *modified decisional constraint graph*, and n is the number of *decision variables*. The assignment of virtual agents (Step 3) finally requires sending $O(d_r q)$ messages containing constraints, where d_r is the highest degree across random variables only, and q is the number of random variables.

Overall, the complexity of generating the consistent pseudo-tree still remains negligible compared to that of the UTIL propagation phase, which remains exponential in the *induced width* w_i of the pseudo-tree, since Comp-$\mathbb{E}$[DPOP] only changes the operation performed to project random variables, but does not change the number or sizes of UTIL messages. However, because Comp-$\mathbb{E}$[DPOP] uses only *consistent* pseudo-trees, the induced width w_i is necessarily at least as large as if all pseudo-trees were allowed. This potential increase in the induced width is discussed in more detail in Section 4.5.

Completeness

Section 2.4.4 already gave an intuition behind the DPOP algorithm on DCOPs, which is that, when computing $\min_{x_1,\ldots,x_n}\{c_1(\cdot)+\ldots+c_m(\cdot)\}$, it is possible to appropriately distribute each $\min_{x_i}$ operator over the sum, and push it as deep as possible into the sum so as to reformulate the problem into an equivalent one that is more distributed in nature. The completeness of DPOP follows from the use of a pseudo-tree: the property that two variables appearing in a common constraint must be in the same branch

enforces that each $\min_{x_i}$ operator is not pushed *too* deep into the sum as to violate the non-commutativity of the $\min$ and the sum.

The same intuition holds for Comp-$\mathbb{E}$[DPOP] on StochDCOPs, except that the evaluation function is inserted between the $\min$ and the sum: the goal is now to compute $\min_{x_1,\ldots,x_n} \{e_{c_1+\ldots+c_m}(x_1,\ldots,n_n)\}$. Comp-$\mathbb{E}$[DPOP] makes the least restrictive assumption about the evaluation function e, assuming that it may or may not commute with the sum (i.e. be linear), or with the $\min$. The use of a pseudo-tree guarantees that the potential non-linearity of e is not violated. The use of a *consistent* pseudo-tree also guarantees that potential non-commutativity with the $\min$ is also respected, because random variables are placed lower than their neighboring decision variables, which means that the e operator is applied before the $\min$ operator. Furthermore, commuting the e with the $\min$ would not make sense, because the resulting optimal values for the decision variables could then depend on the values of the random variables (since they would be projected *after* the decision variables). This would be incompatible with the definition of a solution to the StochDCOP (Definition 13), which requires that the solution be independent of the random variables.

Privacy

As already mentioned in Section 4.3.3, the approach based on virtual agents suffers from a number of drawbacks in terms of agent privacy, topology privacy, and constraint privacy.

Agent privacy is clearly violated by the fact that, for each random variable r, all agents constrained with r must communicate to agree on who will simulate the virtual agent for r, even though they might not necessarily know about each other initially.

Topology and constraint privacy are also clearly violated, because all constraints involving a given random variable r must be revealed to the agent x^r responsible for simulating the virtual agent for r. This is a privacy breach because some of these constraints may not necessarily involve x^r, and therefore x^r initially does not know about them.

In practice, the approach based on virtual agents effectively centralizes all the knowledge and reasoning about each given random variable into the hands of a single agent (not necessarily the same for all random variables). In some problem domains, this may correspond to centralizing a large part of the problem knowledge and of the computation at a very small number of agents. For instance, in Figure 4.7, assuming that r_1 is simulated by x_2 and r_2 by x_3 (which makes most sense in terms of message

complexity), then x_2 and x_3 each have to enforce 4 constraints. Therefore, together they concentrate $\frac{8}{9} \approx 89\%$ of the constraints.

The following sections propose other approaches that do not suffer from the same privacy leaks, because they do not use consistent pseudo-trees, and they do not concentrate all reasoning about a given random variable into the hands of a single agent. Instead, agents collaborate to reason about uncertainty, in a distributed fashion.

4.5 Incomplete, *Local* Reasoning: Local-$\mathbb{E}$[DPOP]

Recall from Section 4.4.4 that Comp-$\mathbb{E}$[DPOP] makes the least restrictive assumptions about the evaluation function e, assuming that it may or may not be linear, and may or may not commute with the $\min$. To ensure correctness when reformulating the original problem $\min_{x_1,\dots,x_n} \{e_{c_1+\dots+c_m}(x_1,\dots x_n)\}$ by pushing the $\min$ and the e into the sum, it therefore had to use a consistent pseudo-tree. This section proposes a different approach that adds the assumption that the evaluation function is linear. As a result, it is able to distribute the e all the way down the sum (as it is assumed to commute with the sum), reformulating the problem into $\min_{x_1,\dots,x_n} \{e_{c_1}(\cdot) + \dots + e_{c_m}(\cdot)\}$. The resulting problem is then solvable using any traditional DCOP algorithm on the new problem $\min_{x_1,\dots,x_n} \{\overline{c_1}(\cdot) + \dots + \overline{c_m}(\cdot)\}$, where $\overline{c_i}(\cdot) = e_{c_i}(\cdot)$. Notice that this approach only makes sense if $\overline{c_i}(\cdot)$ remains a cost, i.e. if *the evaluation function e is commensurable*.

In other words, the agents run the traditional DPOP algorithm on their *evaluated* local sub-problems, from which all random variables are *locally* projected using the evaluation function (Algorithm 25, line 5); the resulting algorithm is therefore called *Local-$\mathbb{E}$[DPOP]*, and is illustrated in Figure 4.8. This approach has three important differences with respect to Comp-$\mathbb{E}$[DPOP], when it comes to complexity, privacy, and completeness.

Algorithm 25 Local-$\mathbb{E}$[DPOP]'s UTIL propagation for variable x, with parent y

1: $UTIL_{x\to y} : (x, y) \to 0$
2: **for all** constraints $c_i(x, \cdot)$ involving x **do**
3: **if** all other decision variables in c_i's scope are higher than x in the pseudo-tree **then**
4: $UTIL_{x\to y}(x, y, \cdot) \leftarrow UTIL_{x\to y}(x, y, \cdot) + c_i(x, \cdot)$
5: $UTIL_{x\to y}(x, y, \cdot) \leftarrow e_{UTIL_{x\to y}}(x, y, \cdot)$
6: **for all** messages $UTIL_{x_j\to x}(x, \cdot)$ received from all children x_j **do**
7: $UTIL_{x\to y}(x, y, \cdot) \leftarrow UTIL_{x\to y}(x, y, \cdot) + UTIL_{x_j\to x}(x, \cdot)$
8: $x^*(y, \cdot) \leftarrow \arg\min_x \{UTIL_{x\to y}(x, y, \cdot)\}$
9: $UTIL_{x\to y}(y, \cdot) \leftarrow \min_x \{UTIL_{x\to y}(x, y, \cdot)\}$
10: Send message $UTIL_{x\to y}(y, \cdot)$ to parent variable y

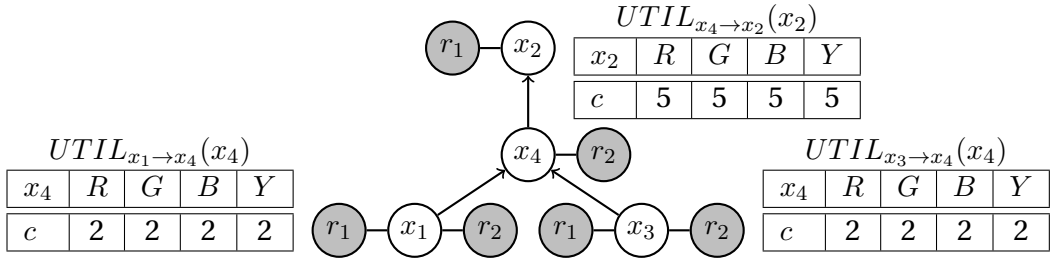
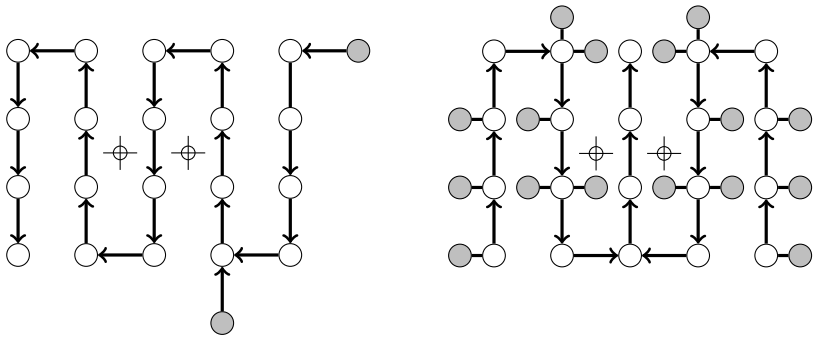


Figure 4.8: Rob-Local-$\mathbb{E}$[DPOP]'s UTIL messages for the problem in Figure 4.1.

Complexity

In terms of information exchange, Local-$\mathbb{E}$[DPOP] has exactly the same complexity as DPOP running on the *decisional* part of the StochDCOP, and therefore is $O(nD^{w_i})$, just like Comp-$\mathbb{E}$[DPOP], where D is the size of the largest decision variable domain. However, the pseudo-tree no longer needs to be consistent: any pseudo-tree will suffice, and therefore it is possible to achieve lower values of the induced width w_i. For instance, in Figure 4.8, the pseudo-tree has an induced width of 1 (all UTIL messages contain a single variable), while Comp-$\mathbb{E}$[DPOP]'s pseudo-tree in Figure 4.7 has an induced width of 3.

Such large differences in induced widths can also be observed on sensor network problems, as illustrated in Figure 4.9 on a 5-by-4-sensor, 2-target problem instance. In Figure 4.9(a), the *consistency* constraint imposes that, for each target, all 16 sensors around it must belong to the same branch of the pseudo-tree; in this example, this constraint is met by putting all 20 variables in a single, common branch. No other (non-homomorphic) consistent pseudo-tree exists if we enforce that tree edges must always be between neighboring sensors. This pseudo-tree has an induced width



(a) consistent (b) inconsistent

Figure 4.9: Possible pseudo-trees for a sensor network problem.

of $w_i = 15$, reached at each virtual agent, each having 15 ancestors (back-edges are not represented for readability). In contrast, Figure 4.9(b) shows an inconsistent pseudo-tree (generated by the *most branching* heuristic presented in Section 4.6.4)) that can be used by Local-$\mathbb{E}$[DPOP], and whose induced width is only of $w_i = 5$.

Privacy

The privacy properties of Local-$\mathbb{E}$[DPOP] are exactly the same as DPOP. In particular, the algorithm does not leak privacy upfront like Comp-$\mathbb{E}$[DPOP] does, because it does not require the use of a consistent pseudo-tree (which leaks agent privacy), and it does not centralize at a single agent all constraints involving each random variable (which violates topology privacy and constraint privacy).

Completeness

The performance improvement in Local-$\mathbb{E}$[DPOP] in terms of information exchange compared to Comp-$\mathbb{E}$[DPOP] comes at a cost in terms of completeness, because it adds the assumption of linearity of the evaluation function. This means that if the evaluation function e is indeed linear, then Local-$\mathbb{E}$[DPOP] solves (optimally) a problem that is a valid reformulation of the original StochDCOP; but if e is non-linear, then Local-$\mathbb{E}$[DPOP] solves a slightly different problem, and is therefore no longer guaranteed to produce optimal solutions (or even feasible solutions) to the original StochDCOP. This is summarized in the following theorem.

Theorem 16. *Local-$\mathbb{E}$[DPOP] is complete if any of the two following criteria is satisfied:*

1. *The evaluation function e is linear, or*
2. *The pseudo-tree is chosen such that, for each random variable r, for all constraints c_i^r involving r, there exists a decision variable x^r also involved in all the c_i^r's that is lower in the pseudo-tree than any other decision variable in the c_i^r's.*

Proof. The proof for the first part of the theorem is straightforward and has already been given earlier. The proof for the second part is the following. First consider the simpler case in which each random variable is only involved in a single constraint. Then Local-$\mathbb{E}$[DPOP] is clearly complete even if e is non-linear, because for each random variable r, the operator e^r can be distributed all the way down the sum of constraints $c_1(\cdot) + \ldots + c_m(\cdot)$ since only one of the c_i's actually depends on r. This observation in itself is worth a corollary (Corollary 1).

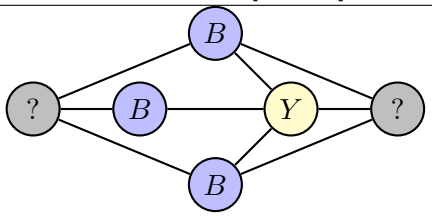
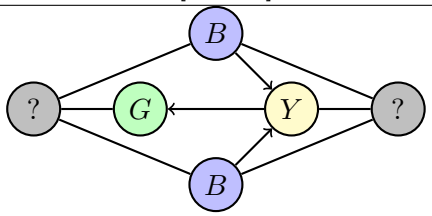
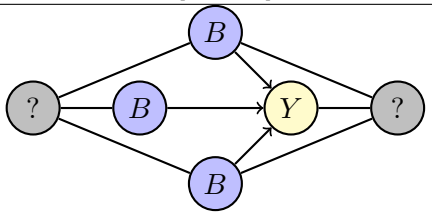
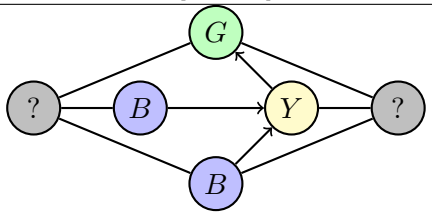
In the more general case, completeness is still guaranteed if the responsibilities for enforcing all the constraints involving r are all assigned to the same agent x^r, because

x^r will then be able to locally *first* compute the sum of all constraints involving r, and *then* apply e to project r. As explained in Section 2.4.4, DPOP assigns the responsibility for each constraint to its lowest variable in the pseudo-tree. Therefore, if the pseudo-tree is such that all constraints involving r have the same lowest decision variable x^r, then completeness holds. □

Corollary 1. *If each random variable is only involved in a single constraint, then Local-*$\mathbb{E}$*[DPOP] is complete.*

Table 4.2 (top left) illustrates the degradation of solution quality in Local-$\mathbb{E}$[DPOP] when none of the two criteria in Theorem 16 is satisfied. Each agent reasons about uncertainty locally, and concludes that its local worst case is always when its neighboring uncertain node(s) choose(s) the same color as itself, regardless of the choice of this color (which explains the constant UTIL messages in Figure 4.8). Therefore, x_1, x_2 and x_3 may choose the same color (other than that of x_4), resulting in a worst-case cost of 5, when the optimum is 2.

Table 4.2: Colorings found for the problem in Figure 4.1.

	Rob-Local-$\mathbb{E}$[DPOP]	Rob-Global-$\mathbb{E}$[DPOP] rooted at x_2
		
max c	5	4
	Rob-Global-$\mathbb{E}$[DPOP] rooted at x_4	Rob-Global-$\mathbb{E}$[DPOP] rooted at x_1
		
max c	5	3

The same phenomenon can also be observed on the single-target, 4-by-4 sensor network problem instance in Figure 4.4, using the *robustness* evaluation function. Figure 4.4(d) shows the optimal solution to the StochDCOP, which is an *on/off* sensor configuration with minimal number of *on* sensors, but in which the target will always remain in sight of 3 sensors, wherever it goes. This corresponds to an optimal worst-case utility of 12, decomposable as: 12 for observing the target in its initial position

with 3 sensors (Eq. (2.2)), plus 12 for observing the target in its next position (whichever it is) with 3 sensors (Eq. (4.4)), minus 12 for turning on 12 sensors (Eq. (2.1)).

In contrast, Local-$\mathbb{E}$[DPOP] is only able to produce the robust 1-configuration in Figure 4.4(b), in which the target will only remain in view of 1 sensor in its next position, which ever it is. This corresponds to a worst-case utility of $12+1-4=9<12$. The intuitive reason why Local-$\mathbb{E}$[DPOP] is unable to produce better configurations is because each sensor independently optimizes its *local* worst-case utility. Furthermore, for each sensor on the boundary, there is always a possible move of the target that will make it become out of sight, and therefore no sensor on the boundary ever takes the risk of turning itself on. More formally, projecting the random variable t_k out of the constraint in Eq (4.4) yields a utility of $\min_{t_k}\left\{u_k^P(x_{i_P}^{j_P}, x_{i_P}^{j_P+1}, x_{i_P+1}^{j_P}, x_{i_P+1}^{j_P+1}, t_k)\right\}=0$, regardless of the values of the decision variables, so the sensors on the boundary choose to be *off* to minimize their local cost (Eq. (2.1)).

Notice that Property 2 in Theorem 16 is equivalent to assuming that the pseudo-tree is consistent. Intuitively, if this is not the case, then a given random variable may be projected by more than one agent (as in Figure 4.8), which is the reason for the incompleteness of the algorithm, because each of these agents projects the random variable *locally* without coordinating with the other agents. The two algorithms proposed in the following section partially address this issue by having the agents better coordinate, through the exchange of information about the dependencies of their respective local sub-problems on the random variables. Intuitively, the goal is to give the agents a broader view of the effect of uncertainty on the overall problem, such that, for instance, the sensors are able to discover that the target actually cannot escape.

4.6 Incomplete, *Global* Reasoning: Global-$\mathbb{E}$[DPOP]

As explained in the previous section, Local-$\mathbb{E}$[DPOP]'s improvement in complexity compared to Comp-$\mathbb{E}$[DPOP] can come at a loss in solution quality when the evaluation function is non-linear. This loss is due to the fact that each agent *locally* projects the random variables out of the constraints it is responsible for enforcing, independently from other agents whose constraints might depend on the same random variables. Because each agent has only a limited view of the overall problem, it only applies the evaluation function to a limited subset of the constraints.

This section proposes a simple modification of the approach in Local-$\mathbb{E}$[DPOP] to partly address this issue, by having agents locally project out random variables like in Local-$\mathbb{E}$[DPOP], but then report to their parent their corresponding *effective costs as functions of the random variables*.

4.6.1 Sharing Information about Dependencies on Random Variables

Having each agent report to its parent in the pseudo-tree its optimal cost as a function of the random variables has the effect of centralizing at the root agent the knowledge about how the overall problem depends on the values of the random variables, given the decisions already taken by agents below. For this reason, we call this new algorithm *Central-*$\mathbb{E}$*[DPOP]*.

Algorithm 26 presents the UTIL propagation phase for Central-$\mathbb{E}$[DPOP]. There are three noticeable differences with respect to Local-$\mathbb{E}$[DPOP] (Algorithm 25). First, Central-$\mathbb{E}$[DPOP] projects out all random variables using the evaluation function before projecting each decision variable like in Local-$\mathbb{E}$[DPOP], but it does so *after* collecting and joining the UTIL messages from its children in the pseudo-tree (line 7). This is because the UTIL messages received can now depend on random variables, and therefore its is better to *first* compute the join and *then* apply the evaluation function e, in order not to further violate e's potential non-linearity. In Local-$\mathbb{E}$[DPOP], the order does not matter since all UTIL messages received are independent on all random variables, and therefore the projection is performed first for performance reasons.

Algorithm 26 Central-$\mathbb{E}$[DPOP]'s UTIL propagation for variable x, with parent y

1: $UTIL_{x \to y} : (x, y) \to 0$
2: **for all** constraints $c_i(x, \cdot)$ involving x **do**
3: **if** all other variables in the scope of c_i are higher than x in the pseudo-tree **then**
4: $UTIL_{x \to y}(x, y, \cdot) \leftarrow UTIL_{x \to y}(x, y, \cdot) + c_i(x, \cdot)$
5: **for all** messages $UTIL_{x_j \to x}(x, \cdot)$ received from all children x_j **do**
6: $UTIL_{x \to y}(x, y, \cdot) \leftarrow UTIL_{x \to y}(x, y, \cdot) + UTIL_{x_j \to x}(x, \cdot)$
7: $EVAL_{x \to y}(x, y, \cdot) \leftarrow e_{UTIL_{x \to y}}(x, y, \cdot)$
8: $x^*(y, \cdot) \leftarrow \arg\min_x \{EVAL_{x \to y}(x, y, \cdot)\}$
9: $UTIL_{x \to y}(y, \cdot) \leftarrow UTIL_{x \to y}(x^*(y, \cdot), y, \cdot)$
10: Send message $UTIL_{x \to y}(y, \cdot)$ to parent variable y

The second important difference is in computing the UTIL message that each agent reports to its parent (line 9). In Central-$\mathbb{E}$[DPOP], this is done by substituting the current variable by its optimal value in the join of all received UTIL messages. As a consequence, the result can now depend on one or more random variables. Notice however that the way the optimal value for the current variable is computed has conceptually not changed (line 8); in particular, this optimal value must still be independent of all random variables, as required by the definition of a StochDCOP (Definition 13).

The third difference is that, contrary to Comp- and Local-$\mathbb{E}$[DPOP], Central-$\mathbb{E}$[DPOP] *is also applicable when the evaluation function* e *is non-commensurable*. This is a direct consequence of the way the outgoing UTIL message is computed. In Comp-$\mathbb{E}$[DPOP] it is a direct output of e, and in Local-$\mathbb{E}$[DPOP], it is the $\min$ of an output of e, and

therefore, in both cases, the output of e needs to remain a cost. On the contrary, in Central-$\mathbb{E}$[DPOP], the outgoing UTIL message is obtained by a variable substitution in the join of all received UTIL messages, and therefore necessarily remains a cost. The evaluation function is only used to compute the optimal value of the current decision variable.

While this approach allows the agents to share information about cross-dependencies on random variables in an effort to improve the quality of the solution found, this comes at an increase in complexity, since UTIL messages can now depend on random variables in addition to decision variables. The following section proposes a way to limit the propagation of information about dependencies on random variables, in order to cut down this increase in complexity.

4.6.2 Limiting the Propagation of Information

As presented in the previous section, Central-$\mathbb{E}$[DPOP] propagates up the pseudo-tree the dependencies on all random variables, hereby centralizing this information at the root agent. Intuitively, given that each random variable might have only a limited, local influence on the overall problem, centralizing at the root the dependency information about all random variables does not seem efficient.

We now propose a modification of Central-$\mathbb{E}$[DPOP] that stops the propagation of information about the dependency on any particular random variable r as soon as no more constraints depend on r. More formally, the propagation is stopped as soon as it reaches the *lowest common ancestor* (lca) in the pseudo-tree of all decision variables that are responsible for enforcing constraints involving r. This is achieved by projecting r once and for all using the evaluation function (Algorithm 27, line 10). Notice that this only makes sense if *the evaluation function e is commensurable*. The resulting algorithm is called *Global-$\mathbb{E}$[DPOP]* because it still enables the agents to acquire a *global* view of the influences of random variables on the overall problem, without centralizing all information at the root.

This is illustrated in Figure 4.10: agent $a(x_4)$ receives UTIL messages from x_1 and x_3 that now indicate the costs of their respective sub-problems, as functions of x_4 as well as of the two random variables r_1 and r_2. The UTIL message received by x_2 also depends on r_1, but not on r_2, because $lca(r_2) = x_4$ is below x_2 in the pseudo-tree. The UTIL messages are represented in partially intensional form rather than in pure table form for conciseness.

Algorithm 28 presents the distributed protocol used in Global-$\mathbb{E}$[DPOP] to compute the lcas for all random variables. It is a two-phase algorithm that operates on the pseudo-tree. During the first, bottom-up phase, lists of random variables are propagated up the pseudo-tree, such that each variable x discovers the list of random variables its

Algorithm 27 Global-$\mathbb{E}$[DPOP]'s UTIL propagation for variable x, with parent y

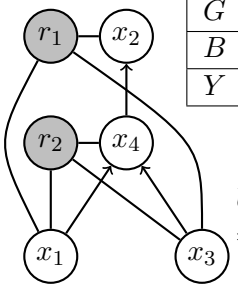
1: $UTIL_{x\to y} : (x, y) \to 0$
2: **for all** constraints $c_i(x, \cdot)$ involving x **do**
3: **if** all other variables in the scope of c_i are higher than x in the pseudo-tree **then**
4: $UTIL_{x\to y}(x, y, \cdot) \leftarrow UTIL_{x\to y}(x, y, \cdot) + c_i(x, \cdot)$
5: **for all** messages $UTIL_{x_j\to x}(x, \cdot)$ received from all children x_j **do**
6: $UTIL_{x\to y}(x, y, \cdot) \leftarrow UTIL_{x\to y}(x, y, \cdot) + UTIL_{x_j\to x}(x, \cdot)$
7: $EVAL_{x\to y}(x, y, \cdot) \leftarrow e_{UTIL_{x\to y}}(x, y, \cdot)$
8: $x^*(y, \cdot) \leftarrow \arg\min_x \{EVAL_{x\to y}(x, y, \cdot)\}$
9: $UTIL_{x\to y}(y, \cdot) \leftarrow UTIL_{x\to y}(x^*(y, \cdot), y, \cdot)$
10: **for all** random variables r for which $x = lca(r)$ **do**
11: $UTIL_{x\to y}(y, \cdot) \leftarrow e^r_{UTIL_{x\to y}}(y, \cdot)$
12: Send message $UTIL_{x\to y}(y, \cdot)$ to parent variable y

$UTIL_{x_1\to x_4}(x_4, r_1, r_2)$

x_4	c
B	$\begin{cases} 0 & \text{if } r_1 \neq Y \neq r_2 \\ 2 & \text{if } r_1 = Y = r_2 \\ 1 & \text{else} \end{cases}$
R, G, Y	$\begin{cases} 0 & \text{if } r_1 \neq B \neq r_2 \\ 2 & \text{if } r_1 = B = r_2 \\ 1 & \text{else} \end{cases}$

$UTIL_{x_4\to x_2}(x_2, r_1)$

$x_2 \backslash r_1$	R	G	B	Y
R	2	2	4	2
G	2	2	4	2
B	2	2	4	2
Y	2	2	2	4



$UTIL_{x_3\to x_4}(x_4, r_1, r_2)$
$= UTIL_{x_1\to x_4}(x_4, r_1, r_2)$

Figure 4.10: Rob-Global-$\mathbb{E}$[DPOP]'s UTIL messages for the problem in Figure 4.1.

subtree depends on. The second, top-down phase proceeds as follows. Each decision variable x computes the list R_x (line 9) of random variables r such that at least two children of x have subtrees that depend on r (i.e. $r \in \cup_{1\leq i<j}(R_i \cap R_j)$), or such that x is responsible for enforcing a constraint involving r (i.e. $r \in R_0$). It then applies a filter to this list R_x by only keeping the random variables whose lcas have not already been found to be higher in the pseudo-tree (line 12). Variable x assigns itself as the lca of the remaining random variables (line 14). Finally, it reports to each of its children the relevant random variables whose lcas remain to be found (line 16).

To compute all lcas, Algorithm 28 exchanges exactly $2 \times (n-1)$ messages, where n is the number of decision variables (one message up and one message down each tree-edge in the pseudo-tree). Each message contains $O(q)$ random variables, where q is the

Algorithm 28 Distributed computation of the lcas, for each decision variable x

1: // **Bottom-up phase**
2: $R_0 \leftarrow \{r \in \mathcal{R} \mid x$ is responsible for enforcing a constraint involving $r\}$
3: **wait** for all $\{R_i\}_{i=1\ldots t}$ from all children
4: $R_x \leftarrow \cup_{i \geq 0} R_i$
5: **send** R_x to parent (if any)
6: **if** $R_x = \emptyset$ **then**
7: **execute** UTIL propagation

8: // **Top-down phase**
9: $R_x \leftarrow \cup_{1 \leq i < j} (R_i \cap R_j) \cup R_0$
10: **if** x has a parent p **then**
11: **wait** for R_p from parent
12: $R_x \leftarrow R_x \cap R_p$
13: **for all** $r \in R_x$ **do**
14: $lca(r) \leftarrow x$
15: **for all** children $i = 1 \ldots t$ such that $R_i \neq \emptyset$ **do**
16: **send** $(R_i \cap R_p) - R_x$ to child i
17: **execute** UTIL propagation

total number of random variables in the StochDCOP. The complexity of computing all lcas is therefore $O(nq)$ in terms of information exchange, which is largely negligible compared to the complexity of the UTIL propagation phase (Section 4.6.6).

4.6.3 Related Work on Resource-Constrained DCOP

An interesting parallel can be made between this work and previous work on *Resource-Constrained DCOPs (RCDCOPs)* by Matsui et al. [2008]. RCDCOPs are DCOPs in which some of the constraints are given the special status of *resource constraints*, which are simply n-ary, additive capacity constraints. The authors propose to handle these constraints in a special manner, by introducing *virtual variables* that are responsible for enforcing these constraints. The parallel with StochDCOPs can be made by comparing a given resource constraint in the RCDCOP framework with the common dependency on a given random variable in the StochDCOP framework.

Their first approach to solve RCDCOPs, which they call *serialization of resource-constrained variables*, consists in generating a pseudo-tree using an algorithm that treats resource constraints like normal n-ary constraints. As a result, all decision variables participating in the same resource constraint are positioned in the same branch of the pseudo-tree. A *virtual variable* enforcing the resource constraint is then added as a child of the lowest of these variables. This approach is very much comparable to their own later work on virtual agents in QDCOPs (Section 4.3.3), which we have adapted to produce Comp-$\mathbb{E}$[DPOP]. Furthermore, the approach they use to gener-

ate a (consistent) pseudo-tree is conceptually the same as the one we propose for StochDCOPs in Section 4.4.3.

In a second approach, Matsui et al. propose to decompose each n-ary resource constraint into lower-arity constraints, in order to allow decision variables participating in a given resource constraint to appear in different branches of the pseudo-tree. Virtual variables then need to be added not only once per resource constraint, but at multiple places in the pseudo-tree, including at lcas. This methodology is comparable to our StochDCOP methodology that consists in allowing the use of inconsistent pseudo-trees, and sharing information about the dependencies on random variables like in Global-$\mathbb{E}$[DPOP].

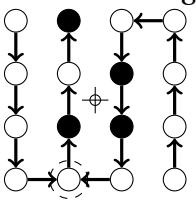
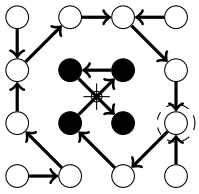
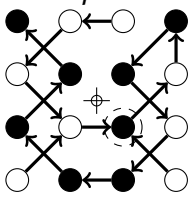
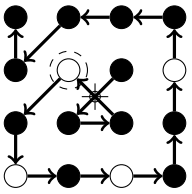
4.6.4 Influence of the Choice of the Pseudo-tree

An important property of Global-$\mathbb{E}$[DPOP] is that, contrary to all other $\mathbb{E}$[DPOP] algorithms presented so far, if the evaluation function is not linear, then *the solutions it outputs depend on the choice of the pseudo-tree*. This is because each pseudo-tree defines a different, incomparable reformulation of the original StochDCOP, by distributing the evaluation function e over the sum of the constraints in a different manner.

This phenomenon is illustrated in Table 4.2, and can be explained intuitively as follows. Given a random variable r, all variables that are below $lca(r)$ in the pseudo-tree make misinformed decisions when they project themselves out, because they do not have a global view of the influence of r on the overall problem. For instance, when the pseudo-tree is rooted at x_4 (Table 4.2, bottom left), x_1, x_2 and x_3 are all below $lca(r_1) = lca(r_2) = x_4$, only see the influences of r_1 and r_2 on their respective local sub-problems, and may end up choosing the same color blue, resulting in the same, low quality coloring as Rob-Local-$\mathbb{E}$[DPOP] (top left). On the other hand, if the pseudo-tree is rooted at x_2 (top right), then x_2 becomes $lca(r_1)$ and is able to make better informed decisions, resulting in a better worst-case cost of 4. Rooting the pseudo-tree at x_1 instead (bottom right) yields an even better worst-case cost of 3, which is however still higher than the true optimal worst-case cost of 2 found by Rob-Comp-$\mathbb{E}$[DPOP].

Table 4.3 illustrates the same phenomenon on a 4-by-4, single-target sensor network problem, using the *robust* evaluation function (back-edges and random variables are omitted for readability). The idea is that various pseudo-trees can result in the lca (dash-circled) for the random variable being positioned at various depths. Below the lca, the sensors have not yet gained a global view of the influence of the random variable on the overall problem: they believe that, in the worst-case, the target can always escape, and therefore they always choose to be *off* (except if they can observe the target in its initial, known position). Only the sensors above the lca realize that, wherever it goes, the target will in fact remain in sight of 4 sensors. The utility reported

Table 4.3: Pseudo-trees for a sensor network problem, varying the heuristic.

heuristic	*most branching*	*most connected*	*most exploration*	*least connected*
				
utility	9	9	10	12
max UTIL	$2^5 \times 8^1 = 2^8$	$2^6 \times 8^1 = 2^9$	$2^9 \times 8^1 = 2^{12}$	$2^{12} \times 8^1 = 2^{15}$

is the worst-case utility, and *max UTIL* is the number of entries in the largest UTIL message (Section 4.6.6).

The quality of the solution found is therefore tightly correlated with the depth of the lca in the pseudo-tree: the higher the lca, the higher the number of sensors that will blindly choose to be *off*, and the lower the chance of having enough *on* sensors to keep track of the target in its next position. Consider for instance the pseudo-tree obtained using the traditional *most connected* heuristic (Table 4.3). The lca is placed relatively high in the pseudo-tree (at depth 6, assuming the root has depth 1), and almost all outer sensors are positioned below the lca, and are therefore turned off (since they cannot observe the target in its initial position). With the only 6 sensors remaining, the best achievable solution is a robust 1-configuration, of worst-case utility 9, which is computed as a utility of 12 for observing the target in its initial position with at least 3 sensors, plus 1 for observing the target in its next position with, in the worst case, only 1 sensor, minus 4 for turning on 4 sensors.

In contrast, consider the pseudo-tree obtained with the reverse, *least connected* heuristic. The lca is positioned very low in the pseudo-tree (at depth 13), with the additional property that the only sensors below the lca are all the inner sensors, which can observe the target in its initial position, and therefore three of them will logically be turned on. As a result, all outer sensors are aware of the fact that the target cannot escape, and Global-$\mathbb{E}$[DPOP] is able to discover an optimal, robust 3-configuration, of worst-case utility 12 ($= 12 + 12 - 12$).

Table 4.3 also illustrates the pseudo-trees obtained with two problem-specific heuristics. The *most branching* heuristic performs two long, vertical traverses of the constraint graph, hereby generating a pseudo-tree that contains of two long branches, and has an induced width lower than with the *most connected* heuristic. The lca is however positioned even higher (at depth 4), and Global-$\mathbb{E}$[DPOP] can also only find robust 1-configurations. The second, *most exploration* heuristic explores the constraint

graph using an efficient, zig-zagging pattern that results in the sensors gathering a global picture of uncertainty relatively early in the bottom-up UTIL propagation (at depth 10). Interestingly, this enables Global-$\mathbb{E}$[DPOP] to discover the existence of robust 2-configurations, of worst-case utility 10 ($= 12 + 6 - 8$).

4.6.5 Hybrid $\mathbb{E}$[DPOP] for Non-commensurable Evaluation Functions

As mentioned in Section 4.6.2, Global-$\mathbb{E}$[DPOP] is only applicable when the evaluation function e is commensurable (Definition 15), because it is used to project random variables at the lcas, and the output of the projection is then sent up the pseudo-tree, and therefore must remain a commensurable to a cost. As a counter-example, Global-$\mathbb{E}$[DPOP] cannot be used to solve StochDCOPs in which the evaluation function is the *consensus* function (Eq. (4.2)), because the output of the consensus operator $\mathbb{P}$ is a *probability*, not a cost. In this case, only Central-$\mathbb{E}$[DPOP] can be used, which performs worse in terms of amount of information exchanged (Section 4.6.2).

In order to overcome this limitation, we now propose a hybrid version of Global-$\mathbb{E}$[DPOP] that is applicable to StochDCOPs with non-commensurable evaluation functions. It is based on the observation that Global-$\mathbb{E}$[DPOP] uses the evaluation function e in two different places, for two different purposes (Algorithm 27). First, it evaluates the join of all UTIL messages received, in order to compute an optimal value to each decision variable that is independent of any random variable (lines 7 and 8). This first use of the evaluation function is also present in Central-$\mathbb{E}$[DPOP], and does not require commensurability, because the output of e is only used as an input to an $\arg\min$. Second, it uses e^r to finally project out each random variable r at $lca(r)$, in order to limit the propagation of information about the dependency of the cost on r (lines 10 and 11). This second use of the evaluation function *does* require commensurability, because the output of e^r must be a cost that is then sent to the parent in the pseudo-tree.

When e^r is non-commensurable, we propose to replace it on line 11 with *another*, *commensurable* evaluation function $\overline{e}^r$. This results in *hybrid* versions of Global-$\mathbb{E}$[DPOP] that use *two* evaluation functions: the original one defined in the StochDCOP on line 7, and an auxiliary, commensurable one on line 11. For instance, using the notations introduced in Section 4.1.2, we call *Cons/Exp-Global-*$\mathbb{E}$*[DPOP]* the version of Global-$\mathbb{E}$[DPOP] that uses the *consensus* evaluation function to compute the optimal values for decision variables, and then the *expectation* function to finally project random variables. Notice that a similar approach can be used to hybridize Local-$\mathbb{E}$[DPOP], by having each variable behave like an lca in Global-$\mathbb{E}$[DPOP].

4.6.6 Complexity Analysis

As argued in Section 4.6.2, computing the lcas has a complexity of $O(nq)$ in terms of information exchange, where n is the number of decision variables, and q is the number of random variables. The UTIL propagation phase is therefore still the main bottleneck of the overall algorithm. During this phase, Global-$\mathbb{E}$[DPOP] exchanges UTIL messages that are conceptually the same as in DPOP, except that they may now also depend on random variables, and not only on decision variables. The worst case is when the root of the pseudo-tree as well as all leaves must enforce constraints involving all random variables; in that case, each UTIL message is a function of all random variables. The worst-case complexity in terms of information exchange in Global-$\mathbb{E}$[DPOP] is therefore $O(nD_{\max}^{w_i}\Delta_{\max}^{q})$, where $D_{\max}$ and $\Delta_{\max}$ are the respectively maximum domain sizes for decision and random variables, and w_i is the induced width of the pseudo-tree.

To illustrate the added complexity of the *global* approach with respect to the *local* approach, consider the messages exchanged by Rob-Global-$\mathbb{E}$[DPOP] on our graph coloring problem (Figure 4.10), compared to those exchanged by Rob-Local-$\mathbb{E}$[DPOP] (Figure 4.8). The largest UTIL message sent by the former contains 3 variables, while the latter only exchanges single-variable messages. A similar observation can be made on the sensor network problem in Table 4.3, except that, in this problem domain, not all variables have the same domain size, and therefore it is more relevant to count the number of entries in the largest UTIL message, rather than just the number of variables. More precisely, all decision variables are binary (sensors can be either *on* or *off*), therefore $D_{\max} = 2$, and the single random variable has domain size $\Delta_{\max} = 8$ (corresponding to the 8 directions of motion for the target in Figure 4.2). One can observe an interesting tradeoff between solution quality and complexity in terms of information exchange: the heuristics that produce low-complexity pseudo-trees only generate poor solution qualities, while higher solution qualities (and even optimality) can be achieved by heuristics that produce higher-complexity pseudo-trees.

4.7 Equivalence Theorems

We now prove that, when the evaluation function e is linear, Global-$\mathbb{E}$[DPOP] produces the same solutions as Local-$\mathbb{E}$[DPOP]. We then show that Central-$\mathbb{E}$[DPOP] and Global-$\mathbb{E}$[DPOP] produce the same solutions, even if the evaluation function e is not linear.

4.7.1 Global-$\mathbb{E}$[DPOP] $\equiv$ Local-$\mathbb{E}$[DPOP]

Theorem 17. *If the evaluation function is linear, then Local-$\mathbb{E}$[DPOP] and Global-$\mathbb{E}$[DPOP] produce the same solutions to the StochDCOP.*

Proof. Starting from Global-$\mathbb{E}$[DPOP], we provide a procedure that transforms this algorithm into the Local-$\mathbb{E}$[DPOP] algorithm, without changing the solution chosen to the StochDCOP. The procedure is iterative on the random variables, and given a random variable r, recursive on the depth in the DFS where r is finally projected out.

Let r be a random variable. Let x be a variable in the DFS that is responsible for projecting out r (in the base case of Global-$\mathbb{E}$[DPOP], $x = lca(r)$). We now show how the projection of r can be pushed down into x's child variables $y_1, \dots y_n$, without changing x's optimal value $\overline{x}^*$ nor the cost it reports to its parent. Let $c_x(x, r, \cdot)$ be x's local cost, which actually does not necessarily depend on r, and let $\overline{c}_i^*(x, r, \cdot)$ be the cost reported by y_i in its UTIL message to x (also possibly independent of r). Variable x reports the following optimal aggregated cost:

$$\overline{c}_x^*(\cdot) = \min_x \left\{ e^r_{c_x + \sum_i \overline{c}_i^*}(x, \cdot) \right\} = \min_x \left\{ e^r_{c_x}(x, \cdot) + \sum_i e^r_{\overline{c}_i^*}(x, \cdot) \right\} \tag{4.5}$$

where (4.5) follows from the linearity of e^r. Furthermore:

$$e^r_{\overline{c}_i^*}(x, \cdot) = e^r_{f:(x,r,\cdot) \to c_i(x, \overline{y}_i^*(x,\cdot), r, \cdot)}(x, \cdot) = \min_{y_i} \left\{ e^r_{c_i}(x, y_i, \cdot) \right\} \tag{4.6}$$

where (4.6) follows from the definition of $\overline{y}_i^*(x, \cdot)$. We can therefore modify the current algorithm into one that has each y_i project out r and report:

$$\overline{\kappa}_i^*(x, \cdot) = \min_{y_i} \left\{ e^r_{c_i}(x, y_i, \cdot) \right\}.$$

In this modified algorithm, variable x reports $\overline{c}_x^*(\cdot) = \min_x \left\{ e^r_{c_x}(x, \cdot) + \sum_i \overline{\kappa}_i^*(x, \cdot) \right\}$, which is the same cost as in (4.5), and is the one that Local-$\mathbb{E}$[DPOP] would report. It follows from the same derivation that x also makes the same decision:

$$\begin{aligned} \overline{x}^*(\cdot) &= \arg\min_x \left\{ e^r_{c_x + \sum_i \overline{c}_i^*}(x, \cdot) \right\} \\ &= \arg\min_x \left\{ e^r_{c_x}(x, \cdot) + \sum_i \overline{\kappa}_i^*(x, \cdot) \right\}, \end{aligned}$$

which is the decision that Local-$\mathbb{E}$[DPOP] would make. Therefore, without changing the solution, we have transformed the initial algorithm in which variable x performs as Global-$\mathbb{E}$[DPOP], into one in which it performs as Local-$\mathbb{E}$[DPOP], and the projection of r has been pushed down into its children in the DFS. We can now recursively apply the same procedure to each of the children. □

One important corollary of this equivalence theorem is that, when the evaluation function is linear, Global-$\mathbb{E}$[DPOP] is complete, since it produces the same solutions as Local-$\mathbb{E}$[DPOP], which is complete as per Theorem 16. However, it is always better to use Local-$\mathbb{E}$[DPOP], because its complexity is lower than that of Global-$\mathbb{E}$[DPOP].

Another interesting theorem can be derived from Theorem 17, which applies to the special case of Rob-$\mathbb{E}$[DPOP], when the StochDCOP has a special, *uniform worst case* property.

Theorem 18. *Rob-Local-$\mathbb{E}$[DPOP] and Rob-Global-$\mathbb{E}$[DPOP] produce the same solutions on* uniform worst case *problems, in which, for any given assignment to the decision variables, the worst-case assignment to the random variables is the same for all agents. More formally, in the single-decision variable, single-random variable case:*

$$\forall x \exists r_0 \forall c_i(x,r) \quad r_0 = \arg\max_r \{c_i(x,r)\}$$

where r_0 may be a function of x.

Proof. The proof is the same as for Theorem 17, except that Equation (4.5), which is the only place in the proof where the assumption of linearity of the evaluation function is used, must be replaced by the following (in reversed order):

$$\begin{aligned}
\min_x \left\{ e^r_{c_x}(x,\cdot) + \sum_i e^r_{\overline{c}^*_i}(x,\cdot) \right\} &= \min_x \left\{ \max_r \{c_x(x,r,\cdot)\} + \sum_i \max_r \{\overline{c}^*_i(x,r,\cdot)\} \right\} && (4.7)\\
&= \min_x \left\{ c_x(x, r_0(x,\cdot),\cdot) + \sum_i \overline{c}^*_i(x, r_0(x,\cdot),\cdot) \right\} && (4.8)\\
&= \min_x \left\{ \max_r \left\{ c_x(x,r,\cdot) + \sum_i \overline{c}^*_i(x,r,\cdot) \right\} \right\} && (4.9)\\
&= \min_x \left\{ e^r_{c_x + \sum_i \overline{c}^*_i}(x,\cdot) \right\} && (4.10)\\
&= \overline{c}^*_x(\cdot) && (4.11)
\end{aligned}$$

where c_x and $\overline{c}^*_i$ may actually not even depend on r, and Equations (4.7) and (4.8) follow from the *uniform worst case* property. □

One example of problem class that exhibits this *uniform worst case* property is the Kidney Exchange Problem (StochDKEP) form Section 4.2.4: given a random variable representing the probability of survival of a given patient, given a choice of exchanges that the agents decide to carry out, the worst-case scenario is the same for all agents, and corresponds to the scenario in which the patient dies. In fact, the StochDKEP class has an even stronger property, since the worst-case scenario does not even depend on the choice of exchanges (r_0 is independent of x).

4.7.2 Central-$\mathbb{E}$[DPOP] $\equiv$ Global-$\mathbb{E}$[DPOP]

Theorem 19. *Global-$\mathbb{E}$[DPOP] and Central-$\mathbb{E}$[DPOP] produce the same solutions to the StochDCOP, even if the evaluation function is not fully linear, but only satisfies the*

following weaker property:

$$\forall c_1 \textit{ independent of any random variable}, \forall c_2\ \forall x \quad e_{c_1+c_2}(x) = c_1(x) + e_{c_2}(x)\ .$$

Intuitively, this property means that, given a random variable r and a constant K, we have that $e^r_{K+c(r)} = K + e^r_{c(r)}$. In particular, this is the case of the *robust* evaluation function, for which $\sup_r\{K + c(r)\} = K + sup_r\{c(r)\}$.

Proof. The exact same mechanism as in the proof of Theorem 17 can be applied to transform Central-$\mathbb{E}$[DPOP] into Global-$\mathbb{E}$[DPOP] without changing the solution produced. The linearity assumption was only used in Equation (4.5), but this assumption is now no longer necessary for the equation to hold; this is because variable x is above the lca of all variables responsible for enforcing a constraint over r, and therefore only one of the terms in the sum $c_x + \sum_i \overline{c}^*_i$ can depend on r. □

A consequence of this theorem is that Exp-Global-$\mathbb{E}$[DPOP] produces the same solutions as Exp-Central-$\mathbb{E}$[DPOP], which is then complete, and Rob-Global-$\mathbb{E}$[DPOP] produces the same solutions as Rob-Central-$\mathbb{E}$[DPOP]. For these two evaluation functions, Central-$\mathbb{E}$[DPOP] is therefore dominated by Global-$\mathbb{E}$[DPOP], since the latter produces the same solutions, but has a lower complexity.

4.8 Approximations by Sampling Scenarios

In some problem domains with rich stochastic uncertainty, the domains of the random variables may be too large to handle, or even infinite, as suggested in Definition 13. The approach commonly used to address this issue is then to *sample* the probability distributions of the random variables, and replace their exact domains with smaller, finite, sampled domains. Intuitively, when random variables represent uncertain, future events, *sampling* corresponds to drawing possible scenarios of future events that are representative of the probability distributions of those events. This section presents how distributed sampling can be performed in the context of StochDCOPs.

4.8.1 Motivation and Challenges

All $\mathbb{E}$[DPOP] algorithms presented in the previous section rely on applying the evaluation function e to project random variables. Among the three examples of evaluation functions proposed in Section 4.1.2, only the *worst-case* evaluation function does not depend on the probability distribution π_r of each random variable r. On the other hand, both the *expectation* and the *consensus* evaluation functions involve computing integrals over π_r, which may or may not be possible to perform in closed form,

depending on the nature of π_r, especially if the domain of r is infinite. Furthermore, Global-$\mathbb{E}$[DPOP] exchanges UTIL messages that are functions of random variables, and therefore assumes that these functions can be represented in some convenient form.

To address this issue, we propose to use the traditional method that consists in *sampling* the probability distribution π_r, to produce a finite list S_r of representative values for the random variable r. Sampling has a certain number of advantages; first, it makes it easy to compute approximations of the evaluation functions, as already described in Section 4.1.2. By discretizing the (possibly infinite) domains of the random variables, it also makes it possible to represent Global-$\mathbb{E}$[DPOP]'s UTIL messages in closed form. Finally, even when the random variable domains are finite, but large, sampling is able to cut down the complexity by only considering a small number of representative values for the random variables, instead of exhaustively enumerating all alternatives. This approach results in *approximate* algorithms, whose solution qualities increase with the number of samples chosen for each random variable.

While sampling is a commonly used method in *centralized* stochastic optimization, its application in the distributed StochDCOP setting is not necessarily straightforward, and raises some challenges. In the *virtual agents* approach to StochDCOPs, sampling is not an issue, because all the reasoning about any given random variable is centralized at a single virtual agent, which is the only agent that needs to apply the evaluation function; therefore sampling is actually centralized (Table 4.4). In all other approaches previously presented however, multiple agents may have to apply the evaluation function, and therefore may need to sample probability distributions. This raises the question of whether and how agents should coordinate to perform the sampling.

Table 4.4: The $\mathbb{E}$[DPOP] family of algorithms.

		final projection of random variables			
		at virtual agents	at leaves	at lcas	at roots
choosing samples	at virtual agents	Comp-$\mathbb{E}$[DPOP]			
	at leaves		indep. sampling + Local-$\mathbb{E}$[DPOP]		
	at lcas		Local-$\mathbb{E}$[DPOP]	Global-$\mathbb{E}$[DPOP]	
	at roots				Central-$\mathbb{E}$[DPOP]

Section 4.8.2 first looks at the possibility of agents *independently* sampling probability distributions, without coordinating to choose the same sets of samples. Section 4.8.3 then proposes a method to perform *collaborative sampling*. Having all agents agree on the sample set of values for a given random variable r is not a trivial task, since

the agents whose costs depend on r initially do not necessarily even know each other. However, experimental results will later show that agreeing on samples can yield significant improvements in solution quality.

4.8.2 Independent Sampling

The simplest approach to sampling in the distributed setting is for each agent to sample the probability distributions for the random variables *independently* of all the other agents. This can be applied to Local-$\mathbb{E}$[DPOP]: each agent produces a sample list S_r of values for each random variable r involved in a constraint it is responsible for enforcing, and then projects r using the corresponding approximate evaluation function (Section 4.1.2). This results in choosing samples at the leaves of the pseudo-tree, as summarized in Table 4.4.

Independent sampling however has the inconvenient drawback of agents reasoning on potentially *different* sample lists S_r for the same random variable r. A possible reason for this is that most sampling methods require drawing numbers from a uniform distribution, using a pseudo-random number generator (PRNG). If all agents do not use the same seed for the PRNG, then independent sampling of the probability distribution π_r will result in different sample lists S_r for r. Even if the algorithm imposes a seed, independent sampling may still produce different sample lists if all agents do not share the exact same knowledge of the probability distribution π_r. In many real-life, multi-agent problems under uncertainty, the probability distributions that describe the sources of uncertainty in the problem are not necessarily known *a priori* to all agents; instead, each agent *learns* distributions based on observed past events. Agents may therefore have varying models for the distribution π_r of a common random variable r, which leads to varying sample lists S_r.

Having each agent use a different sampled domain S_r for r makes it impossible to use Global-$\mathbb{E}$[DPOP] and Central-$\mathbb{E}$[DPOP], because these algorithms exchange UTIL messages that are functions of r, and two UTIL messages using inconsistent sample domains S_r would be impossible to join. More generally, it is necessary for all variables that are below the position in the pseudo-tree where r is finally projected to use the *same* sample domain S_r; this is the reason for the grayed-out top triangle in Table 4.4.

Another important consequence of using inconsistent sampled domains S_r in Local-$\mathbb{E}$[DPOP] is that this can yield lower solution qualities than when agents agree on S_r, as demonstrated by the experimental results in Section 4.9.3. The following section proposes a distributed algorithm to make the agents come to an agreement on S_r for each random variable r.

4.8.3 Collaborative Sampling

Local-$\mathbb{E}$[DPOP] (Section 4.5) has each agent *locally* project all random variables out of its local subproblem, and then report to its parent the resulting cost that is therefore independent of all random variables. As argued in the previous section, this makes it possible to use *independent sampling*. On the other hand, Global- and Central-$\mathbb{E}$[DPOP] (Section 4.6) also perform the same local projection of each random variable r, but then they report the resulting effective costs *as functions of* r. The information on the dependency of costs on r propagates up the pseudo-tree until r is finally projected out, which happens at the root in Central-$\mathbb{E}$[DPOP], and at $lca(r)$ in Global-$\mathbb{E}$[DPOP]. All variables below the position where r is finally projected out must necessarily use the same sample domain S_r for r. We propose to enforce this by assigning to the agent that finally projects out r the additional responsibility of initially choosing S_r. However, because that agent initially does not necessarily know anything about r, we have each agent x_i concerned with r propose a sample list S_r^i; the S_r^i's then only need to be merged and down-sampled to produce a common sample list S_r of the desired size.

Algorithm 29 Collaborative sampling procedure, for each decision variable x

// **Bottom-up phase**
$R_0 \leftarrow \{r \in \mathcal{R} \mid x$ is responsible for enforcing a constraint involving $r\}$
$\mathcal{S}_0 \leftarrow \{sample(\pi_r) \mid r \in R_0\}$
wait for all $\{R_i, \mathcal{S}_i\}_{i=1\ldots t}$ from all children
$R_x \leftarrow \cup_{i \geq 0} R_i$
$\mathcal{S}_x \leftarrow \cup_{i \geq 0} \mathcal{S}_i$
send $\{R_x, \mathcal{S}_x\}$ to parent (if any)
if $R_x = \emptyset$ **then**
    execute UTIL propagation

// **Top-down phase**
$\mathcal{S} \leftarrow \emptyset$ // all known sample sets
$R_x \leftarrow \cup_{1 \leq i < j} (R_i \cap R_j) \cup R_0$
if x has a parent p **then**
    wait for $\{R_p, \mathcal{S}_p\}$ from parent
    $\mathcal{S} \leftarrow \mathcal{S}_p$
    $R_x \leftarrow R_x \cap R_p$
for all $r \in R_x$ **do**
    $lca(r) \leftarrow x$
    $S_r \leftarrow sample(\cup_{S_r \in \mathcal{S}_x} S_r)$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{S_r\}$
for all children $i = 1 \ldots t$ such that $R_i \neq \emptyset$ **do**
    send $\{(R_i \cap R_p) - R_x, \{S_r \in \mathcal{S} \mid r \in R_i\}\}$ to child i
execute UTIL propagation

Algorithm 29 is a *collaborative sampling* procedure for Global-$\mathbb{E}$[DPOP], in which the choice of S_r happens at $lca(r)$. This algorithm is an extension of Algorithm 28 to compute the lcas. Another possible approach would be to first compute the lcas, and only then compute S_r; this would possibly reduce the total amount of information exchanged (as candidate sample lists would only propagate up to the lcas instead of up to the root), but these improvements would remain negligible compared to the overall complexity of Global-$\mathbb{E}$[DPOP], as argued below. In Algorithm 29, messages containing random variables are augmented with corresponding sample domains for the random variables. During the first, bottom-up phase, proposed sample domains are merged and propagated all the way to the root. These merged sample domains are then sent back down until the lca for a specific random variable r is discovered, and the corresponding merged sample domain is down-sampled (line 19) to produce the final common sample domain S_r for r. The chosen S_r is then propagated downwards to all agents whose subtrees depend on r (line 22). This algorithm can be easily adapted for use with Central-$\mathbb{E}$[DPOP], by down-sampling and choosing S_r at the root rather than at the lca.

The complexity of this algorithm depends on the desired size $\Delta_{\max}$ of the sample domains for the random variables, which is taken as a parameter of $\mathbb{E}$[DPOP]. The first, bottom-up phase exchanges exactly $(n-1)$ messages (where n is the number of decision variables), each message containing at most q random variables (where q is the total number of random variables in the StochDCOP), and one merged sample domain of size at most $n\Delta_{\max}$ for each random variable, which sums up to an information exchange that is $O(n^2 q \Delta_{\max})$. The second, top-down phase has the same worst-case complexity. Overall, this is still largely negligible compared to the complexity of the UTIL propagation phase in Global-$\mathbb{E}$[DPOP], which is $O(n D_{\max}^{w_i} \Delta_{\max}^{q})$, where $D_{\max}$ is the maximum domain size of decision variables, and w_i is the induced width of the pseudo-tree.

4.8.4 Consensus vs. Expectation

In the centralized setting, for stochastic optimization problems in which the goal is to minimize the *expected* cost (i.e. the evaluation function is $e_c(\cdot) = \mathbb{E}[c(\cdot)]$), Bent and van Hentenryck [2004] have shown that it can be more efficient to make decisions that maximize the probability of optimality, instead of decisions that have lowest expected cost. In other words, they have shown that it can be better to use the *consensus* operator to project random variables, instead of the *expectation* operator (Section 4.1.2). The intuition behind this result is that the *expectation* approach can have a higher complexity than *consensus*, and therefore, when sampling is used, *consensus* can process more samples than *expectation* in the same amount of time.

Indeed, the *expectation* approach computes the optimal value for a decision variable x

as follows:

$$x^* = \arg\min_x \{\mathbb{E}_r [c(x,r)|x]\} \approx \arg\min_x \left\{ \sum_{r \in S_r} c(x,r) \right\}$$

which generally requires evaluating the cost $c(x,r)$ (i.e. *checking* the constraint c) against each possible assignment to x, in each possible scenario (i.e. for each value of $r \in S_r$). In many real-life problems, computing $c(x,r)$ for given values of x and r can require solving an expensive sub-problem, and doing it for all pairs $(x,r) \in D_x \times S_r$ can become unreasonably expensive. In contrast, the *consensus* approach computes the following:

$$x^* = \arg\min_x \{\mathbb{P}_r[c(x,r)|x]\} \approx \arg\min_x \left\{ -\left| \left\{ r \in S_r \mid x = \arg\min_{x'} c(x',r) \right\} \right| \right\}$$

which only requires finding the optimal value for x for each value of r, and can often be done without having to explicitly compute the cost $c(x,r)$ of each possible assignment to both x and r.

Algorithm 30 Computational steps for the Exp-Local- approach.

```
1: x* ← null; c* ← ∞
2: for each decision x ∈ D_x do
3:     c ← 0
4:     for each scenario r ∈ S_r do
5:         c ← c + (1/|S_r|) c(x,r) // 1 constraint check
6:         if c > c* then continue to next decision
7:     if c ≤ c* then
8:         x* ← x; c* ← c
9: E_r [c(x*,r)|x*] ← c*
```

Algorithms 30 and 31 present in more details the computational steps performed by the expectation and consensus approaches respectively, in the particular case of non-negative costs (which allows for performance improvements). In the worst case, the *expectation* approach requires $|D_x| \cdot |S_r|$ constraint checks (CCs) to compute $x^* = \arg\min_{x \in D_x} \{\mathbb{E}_r [c(x,r)|x]\}$ and report $\mathbb{E}_r [c(x^*,r)|x^*]$ using in Eq. (4.1). On the other hand, in order to compute $x^* = \arg\max_{x \in D_x} \{\mathbb{P}_r[c(x,r)|x]\}$ using Eq. (4.2), and report $\mathbb{E}_r [c(x^*,r)|x^*]$ as in Eq. (4.1), for each scenario (Algorithm 31, line 4), *consensus* computes an optimal decision x'^*, which requires 1 CC per $x' \in D_x$ (line 7), but it might not be necessary to consider all values of r (all scenarios) to find the decision x^* with the highest probability of optimality $\pi[x^*]$. The exploration can be interrupted as soon as the remaining scenarios are too improbable for the second-best solution (of probability of optimality π_2) to catch up with the first best (line 15). In the best case, when the first $\frac{1}{2}|S_r|$ scenarios have the same optimal decision, *consensus* only performs $\frac{1}{2}|S_r|(|D_x|+2)$ CCs (where the $+2$ comes from line 16). However, in the worst case it requires $|S_r|(|D_x|+1)$ CCs, which is $|S_r|$ more than *expectation*.

Algorithm 31 Computational steps for the Cons/Exp-Local- approach.

1: $\forall x \in D_x \ \pi[x] \leftarrow 0$ // initial (non-normalized) probabilities of optimality
2: $x^* \leftarrow$ **null**
3: $\pi_2 \leftarrow 0$ // second-best (non-normalized) probability of optimality
4: **for** each scenario $r_i \in S_r = \{r_1, \ldots, r_k\}$ **do**
5: $\quad x'^* \leftarrow$ **null**; $c'^* \leftarrow \infty$
6: $\quad$ **for** each decision $x' \in D_x$ **do**
7: $\quad\quad c' \leftarrow c(x', r_i)$ // 1 constraint check
8: $\quad\quad$ **if** $c' < c'^*$ **then**
9: $\quad\quad\quad x'^* \leftarrow x'$; $c'^* \leftarrow c'$
10: $\quad \pi[x'^*] \leftarrow \pi[x'^*] + 1$
11: $\quad$ **if** $\pi[x'^*] \geq \pi[x^*]$ **then**
12: $\quad\quad$ **if** $x'^* \neq x^*$ **then** $\pi_2 \leftarrow \pi[x^*]$
13: $\quad\quad x^* \leftarrow x'^*$
14: $\quad$ **else if** $\pi[x'^*] > \pi_2$ **then** $\pi_2 \leftarrow \pi[x'^*]$
15: $\quad$ **if** $k - i \leq \pi[x^*] - \pi_2$ **then break**
16: $\mathbb{E}_r\left[c(x^*, r) | x^*\right] \leftarrow \frac{1}{|S_r|} \sum_{r_i \in S_r} c(x^*, r_i)$ // $|S_r|$ constraint checks

Therefore, while *consensus* inherently produces solutions that may have a lower expected quality than *expectation* (since it maximizes the probability of optimality instead of minimizing the expected cost), it is often capable of compensating this loss by processing more samples than *expectation*, hereby improving the approximations of the probability distributions.

While this result has been observed in many *centralized* stochastic optimization problems, it does not straightforwardly translate to the distributed setting of StochDCOPs. The reason is that the *expectation* evaluation function is linear and commensurable, but the *consensus* evaluation function is neither. As a consequence, the *expectation* approach can be implemented efficiently using Exp-Local-$\mathbb{E}$[DPOP], which is then complete (modulo sampling). On the contrary, the *consensus* approach can only be implemented using Cons-Central-$\mathbb{E}$[DPOP], or a Cons/Exp- hybrid of Global- or Local-$\mathbb{E}$[DPOP] (Section 4.6.5), which are all incomplete algorithms (even without sampling), and whose complexity is higher than that of Exp-Local-$\mathbb{E}$[DPOP] (except for Cons/Exp-Local-$\mathbb{E}$[DPOP]). As a result, the intrinsic gap in solution quality that *consensus* has to compensate for by processing more samples is much larger, and the complexity advantage of *consensus* is smaller.

4.9 Experimental Results

This section presents empirical performance comparisons between the StochDCOP algorithms described in this chapter, which were all implemented inside the FRODO platform (Chapter 5). The experiments were run on a 2.2 GHz computer, with Java 6

and 2 GB of heap space. To evaluate the runtime of our distributed algorithms, we used two metrics that were proposed by the DCOP community: the *simulated time* metric [Sultanik et al., 2007], and the number of *Non-Concurrent Constraint Checks (NCCCs)* [Gershman et al., 2008]. When it comes to measuring performance in terms of communication, we report the total amount of information exchanged by the agents, in bytes. We could have also reported the total numbers of messages exchanged, however all algorithms in this paper tend to exchange roughly the same numbers of messages, so we do not report these results for the sake of conciseness. Finally, to compare the qualities of the solutions output by the algorithms, we report the worst-case cost for the algorithms using the *robust* evaluation function, and the expected cost for those that use the *expectation* and *consensus* evaluation functions.

We report these results on three classes of problem benchmarks: graph coloring problems, Vehicle Routing Problems, and Kidney Exchange Problems . The first problem class and its StochDCOP formulation were already presented in Section 4.2.1. We generated random graphs of fixed density 0.4, varying the total number of nodes, while maintaining the fraction of stochastic nodes to a constant value of 25%. The number of colors was set to 3. Since Section 4.6.4 showed that the choice of the pseudo-tree could have an influence on the solution quality of some algorithms, we investigated the use of three pseudo-tree generation heuristics: the *most connected* heuristic (denoted by the letter *m*), which tends to put high-degree variables higher in the pseudo-tree; the opposite *least connected* heuristic (denoted by the letter *l*), and the random heuristic (letter *r*). Each datapoint is the median over 101 random problem instances, with 95% confidence intervals.

The second problem class is the new StochDKEP benchmark introduced in Section 4.2.4. We produced random Kidney Exchange Problem instances of various sizes using the real statistics of the U.S. population given by Saidman et al. [2006], except for the vital prognosis: for each patient we used a probability of survival drawn from the uniform distribution in $[0, 1]$. Before applying our StochDCOP algorithms, we enforced arc consistency to simplify the problem.

The third class of benchmark problems is the stochastic vehicle routing class described in Section 4.2.3. The benchmark problem instances we used for our experiments were adapted from the Cordeau MDVRP instances from [Dorronsoro, 2007], adding uncertainty in the positions of the customers that can be served by more than one depot, using 4 uncertain positions with the probability distribution $[\frac{1}{10}, \frac{2}{10}, \frac{3}{10}, \frac{4}{10}]$. Due to the larger runtimes (each VRP constraint check involves solving an NP-hard problem), we only report the median over 11 runs.

4.9.1 Solution Quality/Complexity Tradeoff

This section reports the experimental results for the algorithms in terms of average or worst-case solution cost, and in terms of runtime and communication complexity.

Graph Coloring Results

To illustrate the experimental results on graph coloring problems, we used the following conventions. The line color corresponds to the evaluation function used by the algorithm: *Exp*, *Rob*, *Cons*, or the *Exp*/*Cons* hybrid, following the naming conventions introduced in Sections 4.1.2 and 4.6.5. The datapoint symbol varies with the reasoning approach used (*Comp*, *Local*, *Central* or *Global*), and the line stroke stands for the pseudo-tree generation heuristic (*m*, *l* or *r*).

Figure 4.11 shows the runtime performance for algorithms using the *Exp*, *Cons* and *Exp*/*Cons* evaluation functions (lower is better); the winner is Exp-Local-$\mathbb{E}$[DPOP] with the *most connected* heuristic. Among the remaining algorithms, the parameter with the highest impact on runtime is the choice of the heuristic: the *most connected* heuristic always outperforms the random heuristic, which in turn always outperforms the *least connected* heuristic, except in simulated time for very small problem instances. For a given heuristic, *Local* outperforms *Comp*, which outperforms *Global* and *Central*, the two latter performing the same. Figure 4.12 shows similar results for algorithms

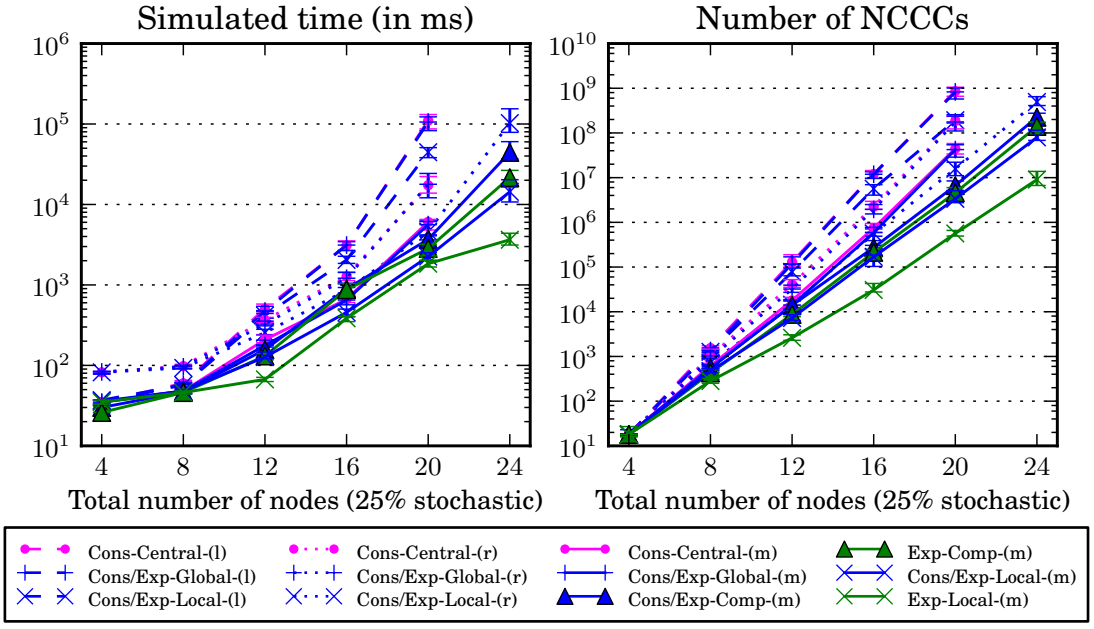


Figure 4.11: Runtime on random graph coloring problems.

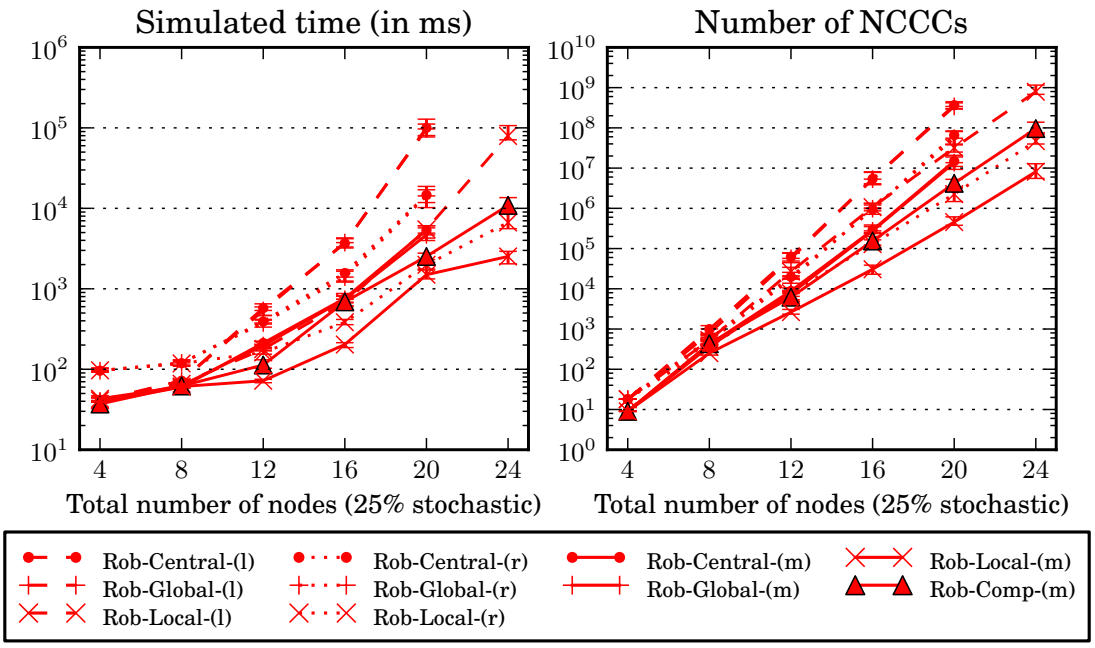


Figure 4.12: Runtime on random graph coloring problems.

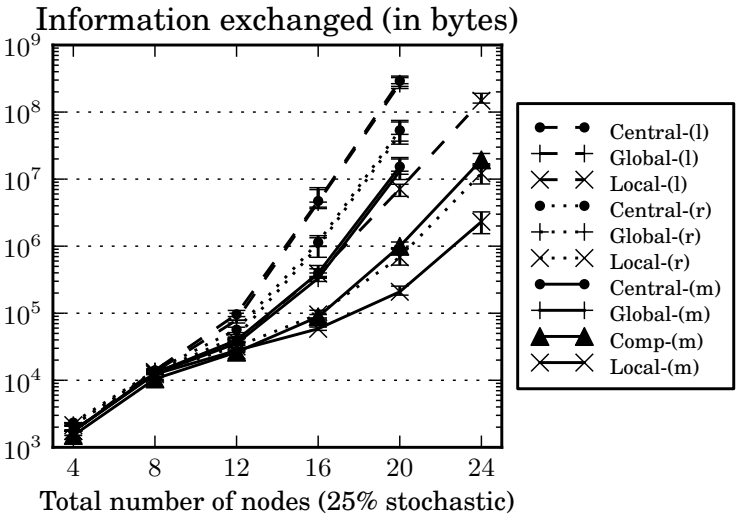


Figure 4.13: Amount of information exchanged on graph coloring problems.

using the *Rob* evaluation function; the difference with Figure 4.11 is that the choice of the reasoning approach seems to have an equally important influence as the heuristic. The fastest algorithm remains the one that reasons *locally*, using the *most connected* heuristic. The performance results in terms of information exchanged (which does not depend on the evaluation function) in Figure 4.13 are consistent.

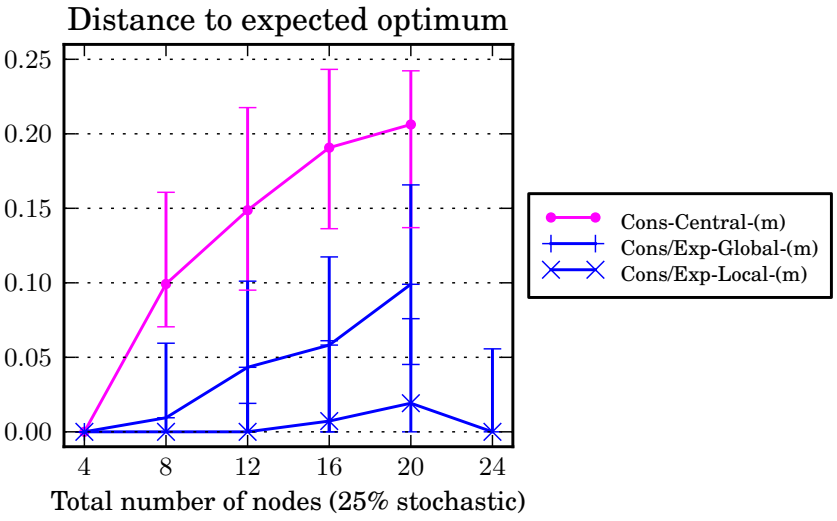


Figure 4.14: Optimality gap for Cons and Cons/Exp on graph coloring problems.

When it comes to solution quality, Figure 4.14 reports the distance from the optimal solution (computed by Exp-Local-$\mathbb{E}$[DPOP]) for various algorithms using the Cons and Cons/Exp evaluation functions, as a function of the reasoning approach used (Central, Global or Local), for a given pseudo-tree generation heuristic (the *most connected* heuristic). Surprisingly, one can see that the solution quality degrades as the agents exchange more information about their respective dependencies on random variables: Local outperforms Global, which outperforms Central. While this may seem counter-intuitive, this phenomenon can be explained as follows. It is probably the case that exchanging more information about uncertainty enables the agents to make better-informed decisions; however, the resulting potential increase in solution quality does not compensate a second, adversarial factor, which is the accumulation of approximation errors. Every time an agent applies the *consensus* evaluation function in order to choose a value for its variable, it performs an approximation, since it chooses the value that has the highest probability of optimality, instead of the one that has the lowest expected cost. The higher in the pseudo-tree random variables are finally projected out, the more agents need to apply the *consensus* evaluation function, and the more the solution quality degrades. In the Local approach, for a given random variable r, only the agents responsible for enforcing a constraint involving r apply the operator $\mathbb{P}_r$. At the other extreme, in the Central approach, the root variable must apply the *consensus* evaluation function for *all* random variables in the problem. This can explain why Cons/Exp-Local-$\mathbb{E}$[DPOP] produces better solutions than Cons/Exp-Global-$\mathbb{E}$[DPOP], which itself produces better solutions than Cons-Central-$\mathbb{E}$[DPOP]. The sudden decrease in the optimality gap for the latter algorithm on problems of size 20 can be explained by the fact that the algorithm started to often time out, and when it did, we were not able to compute its solution quality, so we did not

consider the timeout instances in the computation of the median, which ends up being biased downwards. We have also measured the differences in solution quality between the various pseudo-tree generation heuristics, but they turned out to be statistically insignificant.

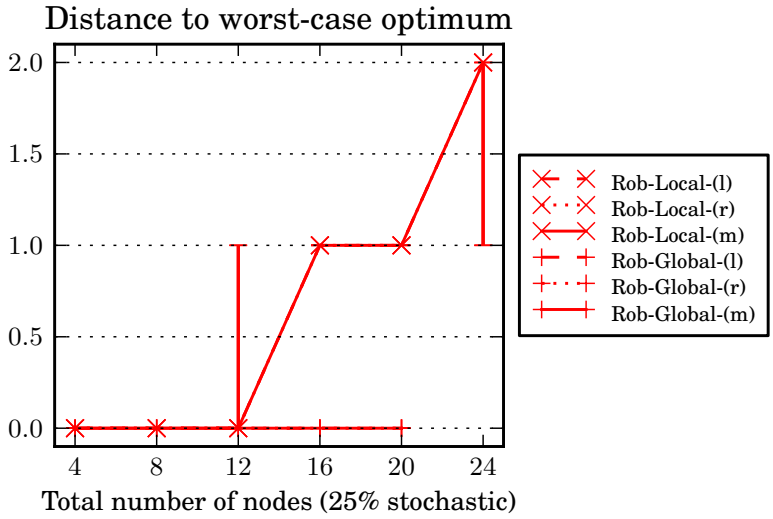


Figure 4.15: Optimality gap for Rob-$\mathbb{E}$[DPOP] on graph coloring problems.

Figure 4.15 shows that, when using the *robust* evaluation function, the tradeoff between the loss in solution quality due to approximations and the gain due to coordination is reversed. While, for small problems, all versions of Rob-$\mathbb{E}$[DPOP] perform equally well (their distance from Rob-Comp-$\mathbb{E}$[DPOP]'s optimal solution is zero), for larger problems, the performance of Rob-Local-$\mathbb{E}$[DPOP] starts to degrade, while Rob-Global-$\mathbb{E}$[DPOP] is still able to produce optimal solutions more than 50% of the time (until it starts timing out most of the time above 20 nodes). This shows that, in this case, exchanging more information about uncertainty *does* yield higher-quality solutions. Notice also that the choice of the pseudo-tree generation heuristic does not seem to have any influence on solution quality.

Distributed Kidney Exchange

Recall from Section 4.7.1 that Theorem 18 applies to the StochDKEP problem class, and from Theorem 19 it further follows that all Rob-$\mathbb{E}$[DPOP] algorithms will produce the same solutions to the StochDKEP, and it does not make sense to compare them in terms of solution quality. In fact, all possible solutions to the StochDKEP always have the same worst-case cost of 0 (when all patients die).

We only report in Figure 4.16 the median (over 151 runs) of the average-case optimality gap for Cons- and Cons/Exp-$\mathbb{E}$[DPOP] with respect to the optimal solution computed

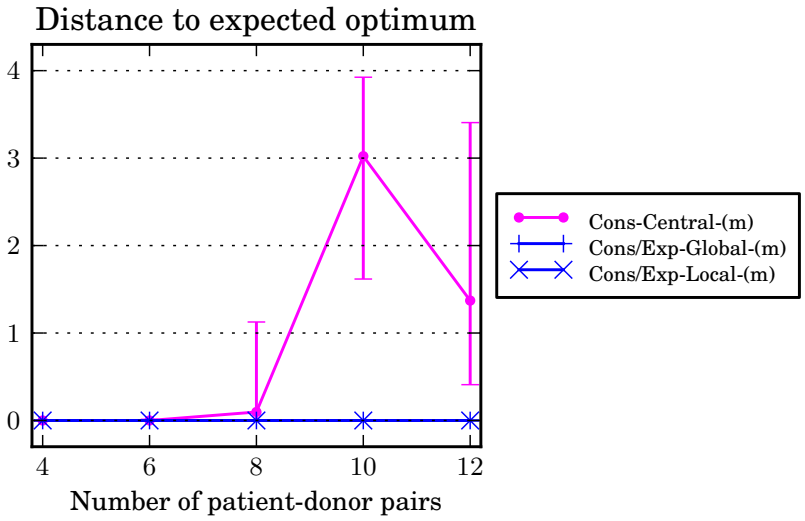


Figure 4.16: Optimality gap for Cons and Cons/Exp on kidney exchange problems.

by Exp-$\mathbb{E}$[DPOP]. It appears that, in this problem class, like for graph coloring, propagating the information about the dependencies on all random variables all the way to the root of the pseudo-tree (as in Cons-Central-$\mathbb{E}$[DPOP]) degrades the solution quality. However, unlike for graph coloring, Cons/Exp-Global- and Cons/Exp-Local-$\mathbb{E}$[DPOP] have a median optimality gap of 0, which means that most of the time there are able to find the optimal solution.

VRP Results

Table 4.5 reports the solution quality and the performance of various algorithms on stochastic VRP benchmarks. H is the visibility horizon, and $Q_{\max}$ is the vehicle capacity; the runtime is in seconds, the NCCCs in thousands, the information exchanged in kB, and the probability of optimality $\mathbb{P}[c]$ is in %. It is worth mentioning that, when counting constraint checks towards the computation of the NCCC metric, only the VRP constraint checks were counted, since checking the other *sum* constraints involves a negligible amount of computation in comparison. When the *robust* evaluation function is used, the *Local-* approach is always the fastest (both in terms of runtime and NCCCs), and it is also the one that exchanges the least amount of information. Furthermore, even though the *Local-* approach is not guaranteed to find the optimal solution, empirically it almost always does.

When it comes to algorithms using the *expectation* or the *consensus/expectation* evaluation functions (Table 4.5, bottom), two remarkable observations can be made. First, when using the *Cons/Exp* hybrid, while it is still the case that the *Comp-* approach has the highest complexity, it is no longer the case that the *Local-* approach systemati-

Table 4.5: Solution quality and performance on VRP benchmarks.

#	inst.	H	$Q_{\max}$	Rob-Local-$\mathbb{E}$[DPOP]				Rob-Global-$\mathbb{E}$[DPOP]				Rob-Comp-$\mathbb{E}$[DPOP]			
				$\min c$	time	NCCC	info	$\min c$	time	NCCC	info	$\min c$	time	NCCC	info
1	p04	18.1	61	465.8	**57.3**	**74.0**	**5.9**	465.8	59.1	78.6	38.0	465.8	82.5	139.3	37.0
2			62	*476.9*	**57.2**	**74.0**	**5.9**	474.8	59.0	78.6	38.0	474.8	82.8	139.3	37.0
3			63	474.9	**57.3**	**74.0**	**5.9**	474.9	58.7	78.6	38.0	474.9	82.6	139.3	37.0
4			64	455.3	**57.4**	**74.0**	**5.9**	455.3	59.2	78.6	38.0	455.3	82.7	139.3	37.0
5			65	446.3	**57.7**	**74.0**	**5.9**	446.3	59.3	78.6	38.0	446.3	83.3	139.3	37.0
6			66	453.3	**57.8**	**74.0**	**5.9**	453.3	59.6	78.6	38.0	453.3	83.3	139.3	37.0
7	p08	55	490	1815.2	**123.3**	**33.6**	**6.1**	1815.2	133.9	38.7	41.5	1815.2	177.9	57.6	16.9
8			494	1815.2	**121.8**	**33.6**	**6.1**	1815.2	133.6	38.7	41.5	1815.2	179.6	57.6	16.9
9			498	1740.0	**121.4**	**33.6**	**6.1**	1740.0	132.5	38.7	41.5	1740.0	180.7	57.6	16.9
10	p11	26	140	*466.3*	**58.7**	**495.2**	**13.5**	460.9	63.6	515.6	154.8	460.9	158.5	956.4	243.8
11			144	445.2	**59.2**	**499.2**	**13.5**	445.2	63.9	519.6	154.8	445.2	158.7	960.0	243.8

#	Cons/Exp-Local-$\mathbb{E}$[DPOP]				Cons/Exp-Global-$\mathbb{E}$[DPOP]				Cons/Exp-Comp-$\mathbb{E}$[DPOP]				Exp-Local-$\mathbb{E}$[DPOP]			
	$\mathbb{E}[c]$	$\mathbb{P}[c]$	time	NCCC	$\mathbb{E}[c]$	$\mathbb{P}[c]$	time	NCCC	$\mathbb{E}[c]$	$\mathbb{P}[c]$	time	NCCC	$\mathbb{E}[c]$	$\mathbb{P}[c]$	time	NCCC
1	465.15	40	**48.9**	**64.7**	465.15	40	50.1	66.3	465.15	40	135.1	248.1	465.15	40	57.5	74.0
2	472.49	0	50.2	66.4	472.49	0	**49.1**	**64.7**	472.49	0	138.2	248.1	472.49	0	58.1	74.0
3	471.25	0	51.9	68.6	471.25	0	**50.8**	**67.2**	471.25	0	135.1	248.1	471.25	0	57.7	74.0
4	453.63	50	49.5	**65.2**	453.63	50	**49.3**	65.3	453.63	50	136.3	248.1	453.63	50	57.7	74.0
5	445.28	80	49.0	64.4	445.28	80	**48.6**	**64.1**	445.28	80	136.4	248.1	445.28	80	57.6	74.0
6	*452.17*	0	**49.6**	**64.9**	*452.17*	0	50.9	66.9	452.1	0	135.3	248.1	452.1	0	58.1	74.0
7	1789.93	0	62.0	19.0	1789.93	0	**59.3**	**18.4**	1789.93	0	310.7	102.6	1789.93	0	121.7	33.6
8	*1790.43*	**20**	64.9	19.9	*1790.43*	**20**	**60.8**	**18.5**	1789.93	*0*	309.4	102.6	1789.93	*0*	120.8	33.6
9	1735.95	10	**111.4**	**32.2**	1735.95	10	111.8	32.3	1735.95	10	310.4	102.6	1735.95	10	120.9	33.6
10	454.95	17	54.4	442.8	454.95	17	**53.8**	**439.6**	454.95	17	277.8	1703.7	454.95	17	58.9	495.1
11	436.67	38	57.2	471.8	436.67	38	**56.8**	**464.2**	436.67	38	281.0	1710.0	436.67	38	59.7	499.2

cally outruns the *Global-* approach: the two approaches appear to have roughly the same complexity. This phenomenon can be explained as follows: the only difference between Cons/Exp-Local-$\mathbb{E}$[DPOP] and Cons/Exp-Global-$\mathbb{E}$[DPOP] is that the former has agents report their *expected* costs to their parents (independently of the random variables), while in the latter agents report their *effective* costs (as functions of the random variables). The two approaches have roughly the same runtime complexity because computing the expected costs requires computing the effective costs, and only involves additional computations (weighted sums) that are negligible compared to the computational bottleneck of checking the VRP constraints. This phenomenon is *not* present when using the *robust* evaluation function (Table 4.5, top), because computing the worst-case costs (independently of the random variables) comes for free as a side-effect of computing the optimal values to the variables, and does not require computing the effective costs as functions of the random variables.

The second remarkable observation is that we were able to reproduce the following result that was known in the centralized setting [Bent and van Hentenryck, 2004]: when constraint checks are expensive, and the goal is to minimize the expected cost of the solution, it can make sense to maximize the probability of optimality instead, because it requires fewer constraint checks. In other words, on our stochastic VRP benchmarks, the Cons/Exp-Local-$\mathbb{E}$[DPOP] and Cons/Exp-Global-$\mathbb{E}$[DPOP] algorithms always outrun Exp-Local-$\mathbb{E}$[DPOP], while still most often producing the same optimal solution with minimal expected cost.

4.9.2 Partial Centralization and Privacy Loss

As already discussed in Sections 4.3.3 and 4.4.4, the *Comp-* approach based on virtual agents has the disadvantage of centralizing all the information about each given random variable into one virtual agent, which has to be simulated by one of the real agents. In this section, we experimentally quantify this phenomenon, by introducing a new metric for information centralization in DCOPs and StochDCOPs.

We first define the amount of information in a given constraint as its number of tuples, i.e. the product of the domains of the variables in its scope. We then compute the *baseline knowledge* of each agent as the total amount of information in all constraints involving its variables. Our measure of the *level of centralization* corresponds to the total amount of information that must be known to the most knowledgeable agent, re-scaled such that 100% corresponds to the case when the most knowledgeable agent must know all constraints, and 0% corresponds to the baseline in which each agent only needs to know the constraints over its variables. The *Local-* and *Global-* approach have a level of centralization of 0%. One the contrary, the *Comp-* approach may have a non-zero level of centralization, because the agent that simulates the virtual agent for a given random variable r must know all constraints involving r, even if they do not

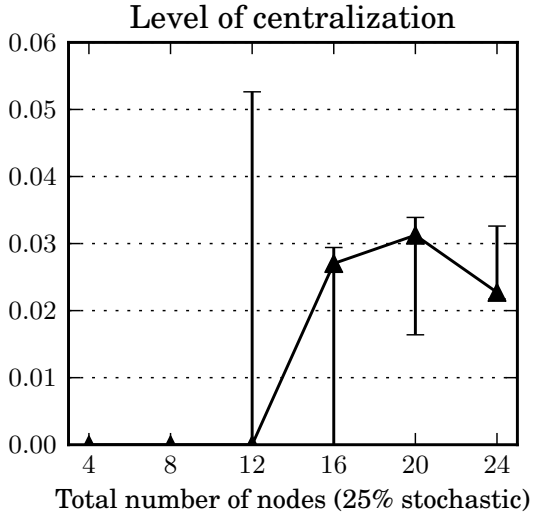


Figure 4.17: Level of centralization of Comp-$\mathbb{E}$[DPOP] on graph coloring problems.

involve any of its decision variables.

For our graph coloring benchmarks, it turns out that the centralization performed by the *Comp-* approach remains minimal, as can be seen in Figure 4.17: the median level of centralization always remains below 6%, with 95% confidence. On the other hand, the results for the VRP benchmarks were completely different, with the *Comp-* approach systematically centralizing 100% of the problem knowledge into one single agent. This can be explained by the fact that all VRP instances we have used involve only two depots, and therefore all agents' subproblems depend on all random variables.

4.9.3 Effects of Sampling

To study the effects of sampling on the solution quality for graph coloring problems, we fixed the total number of nodes to 16, we increased the number of colors to 4, and we varied the number k of samples per random variable. Figure 4.18 reports the distance from the optimal solution (lower is better) computed by Exp-$\mathbb{E}$[DPOP] (without sampling), when using the *Exp-* and *Cons/Exp-* approaches. As a baseline, we also report the optimality gap of the traditional DPOP algorithm, ignoring all random variables and all constraints involving random variables. Due to the increased variance introduced by the sampling mechanism, for these results we have computed the median over a larger number of runs (200 instead of 100 previously).

Three observations can be made. First, $\mathbb{E}$[DPOP] always performs at least as good as DPOP, and significantly better for $k > 1$. This is a reassuring indication that reasoning about uncertainty in any way always produces higher solution qualities than ignoring

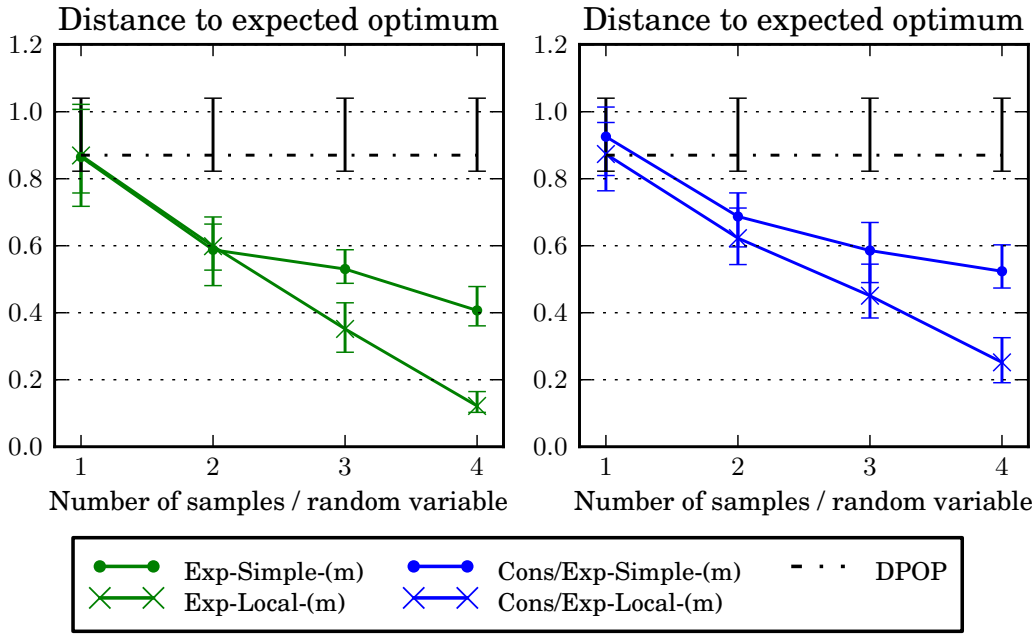


Figure 4.18: Solution quality convergence with the number of samples.

uncertainty. Second, one can see that the solution quality also improves with the number of samples; this is expected, since using more samples allows the agents to consider a larger part of the uncertainty space.

The third, and most important observation relates to the relative performances of independent sampling (referred to as *Simple*-$\mathbb{E}$[*DPOP*] in this graph) and of collaborative sampling. While, for low numbers of samples, the difference is not statistically significant, for higher numbers of samples, collaborative sampling clearly outperforms independent sampling. This means that, when the agents want to minimize the expected cost, they can produce higher-quality solutions by reasoning on the same, consistent lists of samples for the random variables.

The results in Figure 4.19 are totally different, when performing robust optimization instead of average-case optimization. A first, surprising observation is that DPOP (the flat line at cost $= 1.0$) always performs at least as good as Rob-Local-$\mathbb{E}$[DPOP]. This is, in fact, an artifact of the problem class, and can be explained intuitively as follows. By ignoring all uncontrollable nodes, DPOP only makes sure that neighboring agents take different colors, regardless of their neighboring uncontrollable nodes. On the contrary, due to sampling, $\mathbb{E}$[DPOP] may incorrectly think that it can be more beneficial for two neighboring agents to take the same color, if this color appears risk-free because it has not been sampled as part of the possible colors for the neighboring uncontrollable nodes. To illustrate this, consider a simple problem instance involving two neighboring

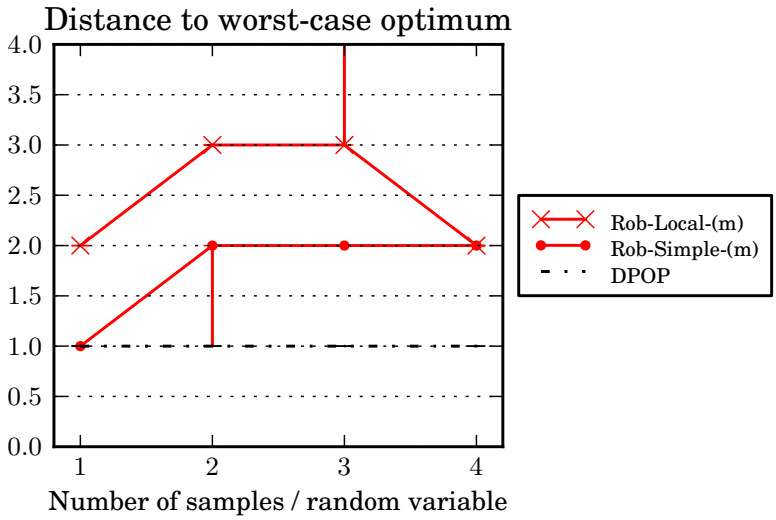


Figure 4.19: Solution quality convergence with the number of samples.

agents x and y, which are both neighbors of an uncontrollable node r. DPOP will make sure that $x \neq y$, and therefore the worst case is when r takes the same color as one of the two agents, in which case only one constraint is violated. On the contrary, in Rob-$\mathbb{E}$[DPOP], since the agents are (under-)sampling the possible colors for the uncontrollable node, it is possible that they both assume that it cannot take the color blue. Therefore, Rob-$\mathbb{E}$[DPOP] may choose the solution $x = y = B$, because it thinks that the corresponding worst-case cost is 1, while in fact it is 3 ($x = y = B = r$). Notice however that it is still more beneficial to reason about uncertainty than to totally ignore it, since DPOP's solution remains sub-optimal: its worst-case cost is greater by 1 than the optimal worst-case cost computed by Rob-Comp-$\mathbb{E}$[DPOP] (without sampling).

The second, very interesting observation is that independent sampling (denoted Simple-$\mathbb{E}$[DPOP]) now outperforms collaborative sampling when $k < 4$, i.e. the number of samples per random variable is too low to cover the entire domain of the random variable. A possible intuition behind this result is that, when performing robust optimization, it is crucial for the agents to have the most complete view as possible of the uncertainty space, otherwise they can take sub-optimal decisions based on the incorrect assumption that one particular scenario cannot occur (as discussed in the previous paragraph). As a result, it is more beneficial for the agents to independently sample the uncertainty space, since together they will be able to explore more of it than if they coordinate to only consider consistent samples. This phenomenon is not observed in the case of average-case optimization, probably because, in that case, it is actually more crucial for the agents to accurately and consistently estimate the most probable scenarios, than to explore more improbable ones.

4.10 Summary

In this chapter, we have introduced the new formalism of Distributed Constraint Optimization under Stochastic Uncertainty (StochDCOP), which is an extension of the classical DCOP framework in which random variables are used to model sources of uncertainty with known probability distributions. A solution to the StochDCOP is an assignment to the decision variables only; the random variables are uncontrollable, and correspond to future uncertain events that the agents must anticipate. Therefore, the effective quality of a solution to the StochDCOP actually depends on the values of these random variables; in order to summarize the quality of a solution into a single criterion, we have also introduced the notion of *evaluation function* that corresponds to the agents' method of anticipation of the uncertainty. The *expectation* evaluation function corresponds to maximizing the expected solution quality, the *robust* evaluation function to maximizing the worst-case solution quality, and we have also considered the *consensus* evaluation function, which maximizes the probability of optimality. We have illustrated this new StochDCOP framework on various application domains, and compared it with previous work on uncertainty in DCOP.

In order to address StochDCOPs, we have proposed three main techniques, which, applied to the DPOP algorithm, give rise to a family of algorithms that we call $\mathbb{E}$[DPOP]. In the *complete* approach (Comp-$\mathbb{E}$[DPOP]), which we derived from related work on QDCOPs, the responsibility for reasoning about each source of uncertainty is centralized at one specific, *virtual* agent. We have also investigated the possibility for agents to reason *locally* about uncertainty, but still exchange to various extents information about their respective dependencies on random variables. In the *local* approach (Local-$\mathbb{E}$[DPOP]), this exchange is inexistent, and the agents just blindly resolve the uncertainty locally, ignoring the fact that other agents may depend on the same sources of uncertainty. In the *global* approach (Global-$\mathbb{E}$[DPOP], the agents propagate up the pseudo-tree the information about their dependencies on each random variable, until the lowest common ancestor of all decision variables that depend on this random variable. We have also investigated the *sampling* techniques that can be used to scale to larger problems in which the uncertainty space is too large to be exhaustively explored.

In the introduction to this thesis, we have raised two research questions, which we have addressed in this chapter. The first question was based on the observation that each source of uncertainty often only has a limited, local influence on the agents, and asked whether this property could be exploited to produce more efficient algorithms. Exploiting the locality of uncertainty is precisely what is done in Local- and Global-$\mathbb{E}$[DPOP], compared to the Comp-$\mathbb{E}$[DPOP] approach that centralizes the reasoning about uncertainty. Our experimental results on various benchmark problem classes have shown that the two former approaches indeed provide significant performance improvements compared to the latter. However, in some circumstances, these perfor-

mance improvements may come to a price in terms of solution quality, as Local- and Global-$\mathbb{E}$[DPOP] are not always complete. We have analyzed theoretically the cases in which completeness is still guaranteed, which is the case in many settings, such as when using the *expectation* evaluation function, and, in certain special conditions, also when using the *robust* evaluation function. We have also observed empirically that, even in the cases when they are not guaranteed to find the optimal solution, these two algorithms most often do. Additionally, they do not suffer from the loss of privacy that is inherent to Comp-$\mathbb{E}$[DPOP]'s approach of centralizing the reasoning about uncertainty. Furthermore, when the goal is to maximize the expected solution quality, we have shown that it can be beneficial to use the *consensus* evaluation function instead, i.e. to maximize the probability of optimality, because this technique can be computationally less expensive and can empirically provide good solution qualities.

The second question was inspired by the fact that multiple agents may depend in different ways on common random variables, and asked whether discovering and exchanging information about such co-dependencies could help the agents make better-informed decisions, yielding solutions of higher quality. In this chapter, we have explored two ways in which agents could exchange such information: by propagating it up the pseudo-tree as previously mentioned, and by coordinating on the choice of consistent samples for the random variables, when sampling is used to address large uncertainty spaces. Our experimental results demonstrated that the pros and cons of such information exchange depend on the evaluation function used. We have shown that, when using the *expectation* evaluation function, propagating more information up the pseudo-tree could provably not improve solution quality, and could only lead to a decrease in runtime and communication performance. When using the *consensus* evaluation function, if this information exchange is propagated too high up the pseudo-tree, it can even additionally lead to a decrease in solution quality. However, when using the *robust* evaluation function, exchanging more information about the cross-dependencies on random variables empirically does help the agents make better-informed decisions. When it comes to coordinating on the choice of samples, we have shown that *collaborative sampling* is indeed beneficial in the context of average-case optimization; however, when performing robust optimization, higher solution qualities can be achieved by simply performing independent sampling.

5 The FRODO 2 Platform

All the algorithms described in this thesis were implemented, tested, and experimented with in the *FRODO 2* platform[1] for DisCSP, DCOP and StochDCOP. This piece of Java software is a complete re-design of the original FRODO platform developed by Petcu [2006], of which it only retained the name. The new design was performed in collaboration with Brammert Ottens and Radoslaw Szymanek, and it was implemented by myself, Brammert Ottens, and multiple students whose semester projects the two of us supervised; a partial list of contributors is provided in Section 5.3.

Some of the motivations for this re-design effort were the following. First, we wanted a platform that was easily maintainable and extendable, as well as clearly documented. To address this, FRODO 2 was designed from the ground up with modularity in mind, and based on high standards in terms of documentation, forcing ourselves to write *Doxygen* documentation for every single class, method, and attribute, be they public or private.

Second, we wanted a platform that was highly reliable. This was also achieved by imposing high requirements upon ourselves, through the extensive and systematic use of *JUnit* tests to check each new piece of code as it was being developed. The `RepeatedTest` feature of JUnit was used to run each type of test multiple times on randomized inputs. At the time of this writing, FRODO includes:

- 8'500+ randomized tests for the communications layer;
- 100'000+ randomized tests for the solution spaces layer;
- 450'000+ randomized tests for the algorithms layer.

Third, we wanted to use a rich, structured file format to express DisCSPs and DCOPs. At the time, very few decent file formats had been proposed for (centralized) Constraint

[1] FRODO stands for FRamework for Open and Distributed Optimization

Satisfaction Problems (CSPs), let alone for distributed CSPs and distributed optimization problems. The most natural choice was the XCSP file format [2008], which we decided to extend and adopt for use in FRODO 2. Being an XML-based format, JDOM [2009] was chosen as a simple-to-use parsing library, and a full-fledge XML Schema was developed for XCSP, which did not exist before. More recently, with the release of the DisCHOCO 2 platform [Wahbi et al., 2011], which also uses a (then slightly different) extension of the XCSP format, we adapted FRODO's format to make it compatible with the one used by DisCHOCO 2. This made it possible to use DisCHOCO 2's problem generators to produce benchmark instances for FRODO.

FRODO currently represents about 100'000 lines of code. In order to allow the entire research community to benefit from this significant implementation effort, FRODO's source code has been released to the public under the Affero GPL license. Our hope was that we would get constructive feedback in return from external users; at the time of this writing, we have even received code contributions to FRODO from two different external developers.

5.1 Architecture

This section describes the multi-layer, modular architecture chosen for FRODO. The three layers are illustrated in Figure 5.1; we describe each layer in some more detail in the following subsections.

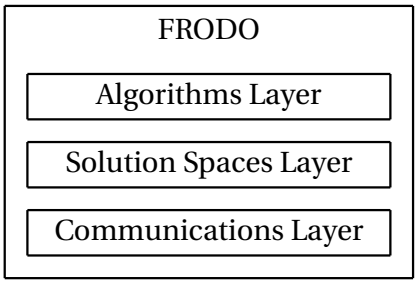


Figure 5.1: General FRODO software architecture.

5.1.1 Communications Layer

The communications layer is responsible for passing messages between agents. At the core of this layer is the `Queue` class, which is an elaborate implementation of a message queue. Queues can exchange messages with each other (via shared memory if the queues run in the same JVM, or through TCP otherwise), in the form of `Message` objects. Classes implementing `IncomingMsgPolicyInterface` can register to a queue in order to be notified whenever messages of specific types are received. Such classes

can be called *policies* because they decide what to do upon reception of certain types of messages.

Typically, in FRODO each agent owns one queue, which it uses to receive and send messages. Each queue has its own thread, which makes FRODO a multi-threaded framework. Special care has been put into avoiding threads busy waiting for messages, in order to limit the performance implications of having one thread per agent, in experiments where a large number of agents run in the same JVM.

5.1.2 Solution Spaces Layer

FRODO is a platform designed to solve distributed combinatorial optimization problems; the *solution spaces* layer provides classes that can be used to model such problems. Given a space of possible assignments to some variables, a *solution space* is a representation of assignments of special interest, such as assignments that correspond to solutions of a given problem. Intuitively, one can think of a solution space as a constraint or a set of constraints that describes a subspace of solutions to a problem.

In the context of optimization problems, *utility solution spaces* are used in order to describe solutions spaces in which each solution is associated with a *utility*. Alternatively, the utility can be seen as a *cost*, if the objective of the problem is to minimize cost rather than maximize utility.

In order to reason on (utility) solution spaces, FRODO implements operations on these spaces. Examples of operations are the following:

- *join* merges two or more solutions spaces into one, which contains all the solutions in the input spaces;

- *project* operates on a utility solution space, and removes one or more variables from the space by optimizing over their values in order to maximize utility or minimize cost;

FRODO provides several implementations of utility solution spaces; the implementation used to solve a given DCOP depends on the problem parser that is passed to FRODO as an argument. The `XCSPparser` converts the DCOP problem file into *hypercubes*; the special `XCSPparserVRP` is additionally able to output *VRP spaces*. Finally, it is also possible to use the `XCSPparserJaCoP`, which outputs *JaCoP spaces* that serve as bridges between FRODO and the centralized solver JaCoP [2011].

Hypercubes

The simplest implementation of a solution space is the *hypercube*. A *hypercube* is an extensional representation of a space in which each combination of assignments to variables is associated with a given utility (or cost). Infeasible assignments can be represented using a special infinite utility/cost. In many cases, operations on hypercubes can be implemented so as to output an *intensional* space rather than explicitly compute the output space in extension. For example, the *join* method returns a `JoinOutputHypercube` object that is simply a wrapper around the inputs to the *join*, to save memory. The solution space is only made extensional (i.e. converted into a hypercube) if it is used by agents to exchange information about their subproblems, such as in the UTIL messages sent by DPOP.

VRP Spaces

To address distributed Vehicle Routing Problems (VRPs), FRODO also provides an implementation of a specialized intensional space representing a (single-depot) VRP constraint (Section 2.2.4). The variables in the space's scope correspond to whether or not the depot serves a given customer (in the non-split delivery variant), or to how much of the customer's demand is assigned to the depot (in the split delivery variant). To any possible assignment to its variable, the space associates the cost (i.e. the length) of the optimal delivery route to serve the customers. Computing the optimal route involves solving a VRP instance, which FRODO performs by calling the third-party library OR-Objects [2010].

JaCoP Spaces

JaCoP spaces are special spaces whose constraints are instantiated inside an instance of the JaCoP solver. Whenever operations need to be performed on such as space, it internally calls JaCoP to efficiently perform them. This makes it possible to handle structured, intensional constraints more efficiently than if they were expressed extensionally as hypercubes, and also makes it possible to leverage the powerful propagation and reasoning algorithms provided by JaCoP, especially when it comes to *global constraints*. At the moment of this writing, FRODO version 2.9 supports the following constraints thanks to JaCoP, in addition to the traditional, extensional soft constraints:

- extensional hard constraints,
- predicate-based hard constraints,
- weighted sum constraints,
- all-different constraints.

5.1.3 Algorithms Layer

The algorithms layer builds upon the solution spaces layer and the communication layer in order to provide distributed algorithms to solve DCOPs. In FRODO, an algorithm is implemented as one or more *modules*, which are simply policies that describe what should be done upon the reception of such or such message by an agent's queue, and what messages should be sent to other agents, or to another of the agent's modules. This modular design makes algorithms highly and easily customizable, and facilitates code reuse, simplicity, and maintenance.

FRODO currently supports the following algorithms:

DCOP algorithms:

- Complete algorithms:
  - SynchBB [Hirayama and Yokoo, 1997],
  - ADOPT [Modi et al., 2005],
  - DPOP [Petcu and Faltings, 2005b], and the following variants:
    - S-DPOP [Petcu and Faltings, 2005d],
    - O-DPOP [Petcu and Faltings, 2006],
    - ASO-DPOP [Ottens and Faltings, 2008],
    - Our own privacy-aware algorithms:
      - P-DPOP [Faltings et al., 2008],
      - P^2-DPOP [Léauté and Faltings, 2009b],
      - $P^{3/2}$-DPOP [Léauté and Faltings, 2011a],
  - MPC-DisWCSP4 [Silaghi and Mitra, 2004; Silaghi, 2005a], which we used as a baseline for comparison with our P*-DPOP algorithms,
  - AFB [Gershman et al., 2006],
- Incomplete algorithms:
  - MGM and MGM-2 [Maheswaran et al., 2004a],
  - DSA [Zhang et al., 2005],

StochDCOP algorithms:

- Our own family of $\mathbb{E}$[DPOP] algorithms [Léauté and Faltings, 2011b],
- Our own (unpublished) Param-DPOP, which computes a optimal solution as a function of the random variables,

Pure DisCSP algorithms:

- MPC-DisCSP4 [Silaghi, 2005a], which we used as a baseline for comparison with our P*-DPOP algorithms.

To illustrate FRODO's modular philosophy, let us consider the implementation of the DPOP algorithm. A DPOP agent uses a single queue, and is based on the generic, algorithm-independent `SingleQueueAgent` class. The behavior of this generic agent is specialized by plugging in four modules, which correspond to DPOP's four phases. Alternatively, the first two phases can be replaced by the new, integrated pseudo-tree generation algorithm described in Section 4.4.2.

1. The *Variable Election* module describes how the agent should exchange messages with its neighboring agents in order to collectively elect one variable of the overall problem. This corresponds to the naive variable election algorithm in Section 3.3.2.
2. The *DFS Generation* module has the agents exchange tokens so as to order all variables in the DCOP along a pseudo-tree, rooted at the variable elected by the Variable Election module, and following the algorithm in Section 2.4.2.
3. The *UTIL Propagation* module implements DPOP's traditional *UTIL propagation* phase (Section 2.4.4), during which hypercubes describing solutions to increasingly large subproblems are aggregated and propagated along the DFS tree in a bottom-up fashion, starting at the leaves of the tree.
4. The *VALUE Propagation* module corresponds to DPOP's *VALUE propagation* phase (Section 2.4.4), which is a top-down propagation of messages containing optimal assignments to variables.

This modular algorithm design makes it easy to implement various versions of DPOP, either by parametrizing one or more modules to make them behave slightly differently, or by completely replacing one or more modules by new modules to implement various behaviors of the agents.

5.2 Features

This section describes the remaining features of FRODO not already mentioned in the previous sections, and in whose implementation I was personally involved. For a full list of FRODO's features (including, for instance, the ability to run the platform on multiple machines) and their descriptions, please refer to the user manual.

5.2.1 Performance Metrics

FRODO supports the following performance metrics:

Number of Messages and Amount of Information Sent If told to do so, FRODO can compute and report the total number of messages sent, sorted by message type, as well as the total sum of the sizes of all messages sent, also sorted by message type. Note that this can be computationally expensive, as measuring message sizes involves serialization.

Simulated Time By default, FRODO uses a centralized mailman to ensure that messages are delivered sequentially one at a time, in increasing order of their time stamps; this makes it possible to compute the runtime of the algorithm in *simulated time* [Sultanik et al., 2007]. If this feature is disabled, then each agent gets its own thread, and messages get delivered completely asynchronously; timing measurements can then only be valid if the computer on which FRODO is running is such that each agent gets a dedicated processor/core.

Non-Concurrent Constraint Checks (NCCCs) If an implementation-independent measurement of runtime is preferred to the simulated time metric, FRODO also supports the logical-step-based NCCC metric [Gershman et al., 2008], which counts the length (in number of constraint checks, which are assumed atomic) of the longest, non-concurrent execution path of the algorithm.

Other Statistics Several algorithmic modules can also report other statistical information about the problem. For instance, in the case of DPOP and ADOPT, the DFS Generation module can report the DFS tree that was generated, in DOT format [Graphviz, 2011], while the VALUE Propagation module can report the optimal assignments to the DCOP variables and the corresponding total utility.

5.2.2 Graphical User Interface

FRODO comes with a limited graphical user interface; a screen capture of the main window is illustrated in Figure 5.2. This window allows the user to input a problem file in XCSP format, to choose an algorithm from the drop-down menu of supported algorithms (or to input a custom algorithm configuration file), and to specify a timeout. Of most use is probably the *Render* button, which brings up a new window displaying the constraint graph of the input problem, as shown in Figure 5.3 (left). This functionality is provided through the third-party utility Graphviz [2011]. In a similar manner, FRODO can also display the pseudo-tree used by DPOP or ADOPT (Figure 5.3, right).

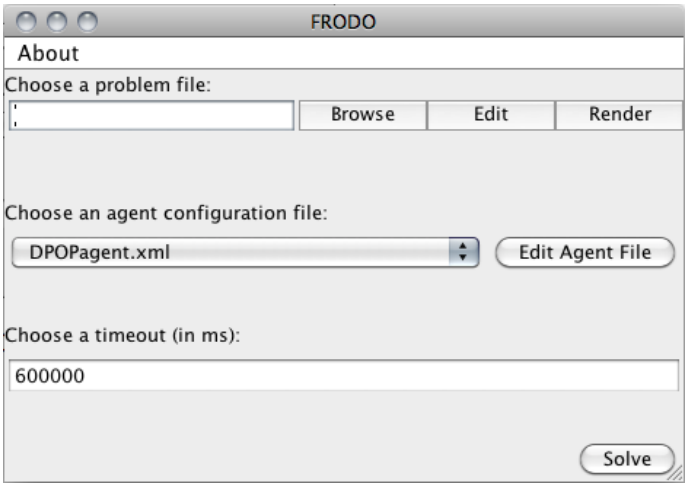


Figure 5.2: FRODO's main GUI window.

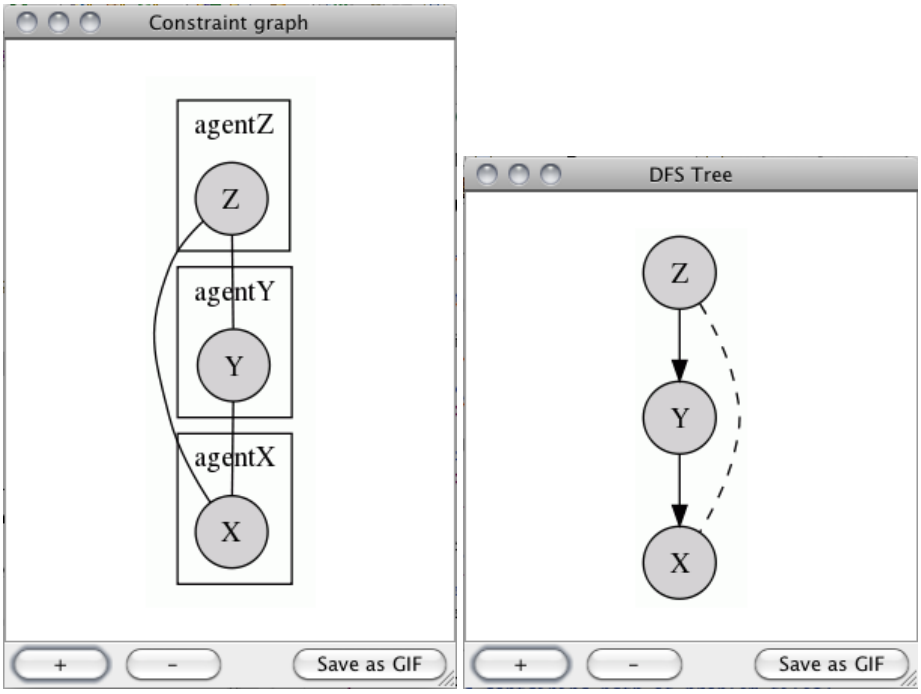


Figure 5.3: FRODO's auxiliary GUI windows.

5.2.3 Websites

FRODO's official website is currently hosted at the LIA, and is accessible at the following address: http://liawww.epfl.ch/frodo/. This website was developed using the Content Management System Joomla! [2011], and provides announcements of new releases, download links, as well as an HTML version of the user manual (also available in PDF),

and a thorough and systematic documentation of the API, generated using Doxygen [2011].

A SourceForge page has also been setup: https://sourceforge.net/projects/frodo2/. SourceForge provides collaboration tools such as a mailing list to which release announcements are sent, as well as a bug tracker and a feature request tracker, which are used to communicate about the ongoing and planned developments concerning the platform.

5.3 Contributors

The implementation in FRODO of the research described in this thesis was partly performed by the following EPFL students (in alphabetical order), whom I would like to personally thank for their respective contributions. The full list of contributors (beyond the work in this thesis) can be found on the FRODO website.

- Jonas Helfer implemented the Kidney Exchange problem generator;
- Arnaud Jutzeler implemented the valuable interface with JaCoP;
- Nacereddine Ouaret and Stéphane Rabie were involved in the preliminary implementation of the solution spaces layer;
- Andreas Schaedeli implemented the GUI and the Combinatorial Auctions problem generator;
- Achraf Tangui implemented a preliminary version of the meeting scheduling problem generator;
- Éric Zbinden did a remarkable job at implementing P-DPOP and P^2-DPOP.

6 Conclusion

In this thesis, we have addressed two orthogonal issues that arise in many real-life multi-agent decision-making problems: 1) how to make coordination possible while still protecting the privacy of the agents, and 2) how to best commit to decisions when all costs and rewards are not necessarily known exactly beforehand, but rather depend on future, uncontrollable events. To model such multi-agent decision problems, we have adopted — and extended — the traditional framework of Distributed Constraint Satisfaction (DisCSP), and its generalization to optimization problems, Distributed Constraint Optimization (DCOP).

We have illustrated how these two frameworks can be used to formalize many distributed problems, using the following examples of applications. In a distributed graph coloring problem, each agent must make a decision (choose a color) that is different from the decisions made by its neighbors; this class of problems includes for instance the problem of the allocation of time schedules in a Time Division Multiple Access (TDMA) scheme for an ad-hoc wireless network. Another very common, real-life problem class is that of multi-agent meeting scheduling, in which participants want to schedule multiple meetings among themselves, while taking into account each attendee's availabilities and preferences. We have also introduced a new class of sensor network problems, with omni-directional sensors that must coordinate their decisions to turn themselves on or off so as to observe a number of targets; we have used this problem class to illustrate some of the properties of the algorithms we have developed and presented in this thesis. Another class of real-life-inspired problems we have considered is the class of resource allocation problems, in which agents need access to various resources of limited capacity, and therefore need to decide who should reserve access to which resource. We have also considered an even more concrete class of optimization problems, in the form of multi-agent vehicle routing: depots control fleets of delivery vehicles, and they must plan how they should serve a set of customer requests at minimal total route length. Finally, we have described the more theoretical problem of computing equilibria in one-shot, strategic games. After describing how all

these examples of problems can be formalized as DisCSPs and/or DCOPs, we have also reviewed the literature on DisCSP/DCOP algorithms; in particular, we have described in more detail the DPOP algorithm, upon which we have based all the algorithms we have developed.

One important feature of all these problems is that there is no omniscient entity that knows the whole problem; instead, the knowledge of the problem is inherently distributed among the agents, each agent being initially only aware of the part of the problem that directly affects the decisions it has to make. In the DisCSP/DCOP framework, it is traditionally assumed that the agents are cooperative, and are willing to exchange information if this exchange can help them eventually reach an agreement that maximizes the social welfare. However, in many real-life problems, this cooperation is hindered by the agents' desires to hide from other agents some of their information about the problem and/or about the solution eventually chosen to the problem. For instance, in meeting scheduling problems, participants may want to hide their respective availability schedules and preferences, whom they are scheduling meetings with, and what the times eventually chosen for their meetings are.

Novel Contributions on Privacy

In this thesis, we have proposed to define four types of information about the problem and its solution that agents might want to protect. We have called *agent privacy* the notion of hiding the presence and identities of participants from other unrelated agents with which they do not directly share constraints. *Topology privacy* involves protecting the knowledge of the inter-dependencies between various decisions, introduced by the coordination constraints that must be enforced between agents, and by the constraints and preferences that the agents express over these decisions. *Constraint privacy* covers the knowledge of the nature of these constraints and preferences themselves: which combinations of decisions are allowed or disallowed, and how much cost or utility an agent assigns to various joint decisions. Finally, *decision privacy* is about the knowledge of the solution that is chosen to the problem. After introducing these definitions, we have described how they relate to previous work on privacy in DisCSP and DCOP, and how various algorithms have been proposed to address subsets of these four types of privacy, with various degrees of success. We have shown that these algorithms can be seen as belonging to two different classes: 1) improvements over known DisCSP/DCOP algorithms to reduce their privacy leaks, without ever really removing these leaks entirely, and 2) algorithms inspired by work in cryptography and Secure Multi-party Computation, which usually are able to provide some relatively strong privacy guarantees in terms of constraint privacy and decision privacy, but generally violate agent privacy and topology privacy.

Besides identifying these four types of privacy, the first main contribution of this thesis

is in the form of several new, privacy-preserving algorithms for DisCSP and DCOP, which borrow the best of both worlds: they are based on the well-known DPOP algorithm for DCOP, but they also make use of various cryptographic techniques in order to provide strong privacy guarantees on constraint and decision privacy, without compromising on agent and topology privacy. The first of these algorithms, called *P-DPOP*, involves three privacy-preserving techniques: 1) a novel leader election algorithm to construct a hierarchy on the decision variables; 2) the introduction of codenames to hide agent names, variable names, and variable values; and 3) a cost/utility obfuscation scheme based on the addition of secret, large, random numbers in such a way that the agents are able to perform operations on obfuscated costs/utilities without having to decrypt them. Under the assumption that agents are *honest but curious*, we have given formal proofs of the levels of privacy that P-DPOP guarantees: full agent privacy, and limited topology, constraint, and decision privacy. In terms of the last three privacy types, we have analyzed in detail what information P-DPOP may and may not leak about the problem. We have also investigated a possible tradeoff between topology privacy and performance, through the introduction of the P^--DPOP variant, which makes minor topology privacy concessions in order to reduce the amount of information exchanged by the agents.

The second algorithm we have proposed, called $P^{3/2}$-DPOP, is an improvement over P-DPOP that is able to guarantee full decision privacy. To achieve this level of privacy, we have replaced the *VALUE propagation* phase in P-DPOP, during which the highest agent in the hierarchy sends its optimal decisions to its descendants, with a novel procedure that shuffles the hierarchy in order to put each other agent at the root of the hierarchy in turn; we then iteratively restart the algorithm. In order to prevent the agents from making inferences and attempting to game the protocol, our rerooting algorithm makes sure that no agent knows which decision variable is at the root of the hierarchy at each iteration (other than its owner agent), and that no agent can predict the iterations at which its own decision variables will be roots. This is done by exchanging messages encrypted using *ElGamal homomorphic encryption*, which are passed around among the agents in a round-robin fashion. We also show how such a round-robin communication structure can be set up without violating agent privacy, through a routing protocol in which agents only exchange messages with their parents and children in the hierarchy. We have formally proven that $P^{3/2}$-DPOP guarantees full agent and decision privacy, at a minor loss of topology privacy compared to P-DPOP, and that it provides the same (partial) constraint privacy guarantees as P-DPOP.

Finally, the third algorithm we have proposed, which we call P^2-DPOP, is an improvement over $P^{3/2}$-DPOP, in order to achieve full constraint privacy. To this effect, we have replaced the obfuscation scheme in $P^{3/2}$-DPOP and P-DPOP with a cryptographically more secure scheme, based on ElGamal homomorphic encryption. We have formally proven that P^2-DPOP guarantees full agent, constraint and decision privacy, and partially protects topology privacy. We have analyzed what may and may not be leaked in

terms of topology privacy.

In order to evaluate the performances of these three algorithms against the state of the art in privacy-preserving DisCSP and DCOP algorithms, we have carried out extensive experiments on a large number of benchmark problem classes. All our algorithms have been implemented inside the new FRODO 2 platform, which we have described in the last chapter of this thesis. We have also implemented the MPC-DisCSP4 and MPC-DisWCSP4 algorithms, which we consider the previous state of the art in algorithms that provide strong privacy guarantees for DisCSP and DCOP respectively. In terms of privacy, our algorithms are generally better than these two previous algorithms, because MPC-Dis(W)CSP4 violates agent and topology privacy, and because it is subject to collusions by the agents. In addition to providing better privacy guarantees, our experiments have shown that, on most of the benchmark problem classes we have considered, our algorithms also scale better. We have observed that the use of strong cryptographic primitives such as ElGamal encryption in our $P^{3/2}$-DPOP and P^2-DPOP algorithms comes at a high computational price compared to our simpler, but less secure obfuscation scheme, with these two algorithms performing often several orders of magnitude worse than P-DPOP. The latter algorithm seems to be able to scale to much larger problems than the previous state of the art, which was often impractical for realistically-sized problems.

Novel Contributions on Uncertainty

Besides the desire to protect their respective knowledge of the problem, another common issue that can prevent agents from being able to make optimal decisions is the presence of uncertainty in the problem. It is often the case that the agents need to commit to decisions by a certain deadline, but the actual reward they will receive from their decisions depends on uncontrollable events that may occur after the deadline. An illustrative example in the meeting scheduling problem is the presence of uncertainty in the time needed by an attendee to reach the location of a meeting, which has an impact on whether or not she will be able to arrive on time. To help them anticipate uncontrollable, future events, the agents can typically make use of models of uncertainty, which predict the probabilities with which each event considered may occur. For instance, the attendees of a meeting may be able to predict with some level of accuracy the probabilities of heavy road traffic at the various meeting locations. In this thesis, we have proposed an extension of the traditional DCOP framework, which we call *DCOP under Stochastic Uncertainty (StochDCOP)*, in which the agents use random variables with known probability distributions to model the various sources of uncertainty in the problem. We have illustrated how this new formalism can be used to model stochastic versions of some of the aforementioned problem classes: stochastic graph coloring, sensor networks with moving targets, and stochastic vehicle routing. We have also proposed a new class of benchmarks to the DCOP community,

the Distributed Kidney Exchange Problem.

In a StochDCOP, the effective cost or utility of a solution, which is defined as an assignment to the decision variables only, is now a function of the values of the uncontrollable, random variables, instead of a single value like in the traditional DCOP framework. Therefore, it is necessary to clarify the notion of optimality in this new setting; to this purpose, our StochDCOP definition includes the concept of an *evaluation function*, which summarizes in one single criterion the quality of a solution, independently of the values of the random variables. We have investigated three examples of evaluation functions, corresponding to three different ways of dealing with uncertainty: minimizing the *expected* cost (i.e. average-case optimization), minimizing the *worst-case* cost (i.e. robust optimization), and maximizing the probability of optimality (i.e. the so-called *consensus* approach).

In many StochDCOP classes, the uncertainty in the problem possesses two remarkable properties: 1) each source of uncertainty only has a limited, local influence on some of the agents, and 2) several agents can depend differently on the same source of uncertainty. In this thesis, we have proposed algorithms that attempt to discover and exploit these two properties: 1) by *localizing* the reasoning about uncertainty, and 2) by having the agents exchange information about their respective dependencies on common random variables. We have proposed three general approaches that explore the tradeoff between local reasoning and information exchange. In the *complete* approach, which we derive from previous work on Quantified DCOP, the information exchange is maximal, with all information and reasoning about each source of uncertainty being centralized at a specific agent. This approach has evident drawbacks in terms of privacy, but has the advantage to be guaranteed to find the optimal solution. At the other end of the spectrum, we have also proposed a *local* approach, in which each agent resolves its uncertainty locally, without exchanging any information with other agents about how its local subproblem depends on various sources of uncertainty. Finally, we have also introduced an intermediate, *global* approach, in which the extent to which the information about the dependencies on each particular source of uncertainty is limited. We have called $\mathbb{E}$[DPOP] the family of variants of DPOP that we have introduced and that make use of these approaches to solve StochDCOPs.

We have compared the performances of our algorithms, both theoretically through formal equivalence theorems, and empirically through experiments on stochastic graph coloring, stochastic vehicle routing, and stochastic distributed kidney exchange benchmarks. When the uncertainty space is too large for our algorithms to scale, we have also proposed the use of *sampling* to reduce the uncertainty space by only considering a few, representative *scenarios* of possible future events. We have introduced an algorithm, called *collaborative sampling*, which allows the agents to cooperate in order to choose consistent samples for their common sources of uncertainty. We were able to draw a number of conclusions from our experimental results.

First, our performance results in terms of computational and communication complexity have shown that the choice of which approach to use (between the *complete*, *local* and *global* approaches) depends on the class of problems. On stochastic graph coloring problems, the *local* approach is clearly the most efficient, both in terms of runtime and of amount of information exchanged between the agents. Between the two remaining approaches, the *global* one turns out to be the more expensive. On the contrary, for stochastic vehicle routing problems, in which the computational bottleneck is in solving each agent's local sub-problem, the *global* approach runs faster than the *complete* approach, and even sometimes than the *local* approach when using *consensus*.

Second, we have shown that, as was known to be the case in the centralized setting, it can be more beneficial to maximize the probability of optimality (*consensus*) instead of minimizing the expected cost, even if the end goal is to perform average-case optimization. This results holds for problem classes in which each agent has to solve a computationally expensive sub-problem, such as the class of stochastic vehicle routing problems. In such problem classes, the *consensus* approach can turn out to be less expensive than the *expectation* approach, and still produce solutions of comparable qualities.

Third, we have discovered that the benefits of sharing more information about uncertainty over blindly resolving it locally depend highly on the evaluation function used. When using the *consensus* approach in order to perform average-case optimization, it turns out that the *global* approach produces solutions of lower quality than the *local* approach, because it accumulates approximation errors; in this case, exchanging more information is counter-productive, unless all information about each source of uncertainty is fully shared to one agent, as it is done in the *complete* approach. On the contrary, in the context of robust optimization, exchanging more information about uncertainty allows the agents to make better-informed decisions, and produces higher-quality solutions compared to the *local* approach.

Finally, we have investigated the possible benefits of performing collaborative sampling instead of independent sampling, and here again, we have discovered that these benefits depend on the evaluation function. When performing average-case optimization (be it with the *expectation* or the *consensus* approach), collaborative sampling yields higher solution qualities, because reasoning on consistent samples allows the agents to better evaluate the most probable scenarios, and make decisions based on these in order to minimize the expected cost. On the contrary, in robust optimization, it is better when the agents independently produce inconsistent samples for the sources of uncertainty, because they are then able to more thoroughly explore the uncertainty space.

Possible Future Avenues of Research

Our work on privacy in DisCSP and DCOP has been based on the assumption that the agents are *honest but curious*; in other words, the agents faithfully execute the protocol, and we have only considered them as *passive adversaries* that attempt to infer as much as possible about the other agents' knowledge of the problem from the messages they receive from them. In some application domains, this restriction might not be realistic, as agents might be tempted to actively manipulate the protocol, for instance by reporting false constraints and preferences or by executing a different algorithm, in order to steer the solution towards one that better suits their own selfish interests, and/or in order to learn more about the other agents' constraints and preferences.

One way to address such selfish behavior from the agents could be to enforce incentives towards cooperation, for instance through side payments. This thread of research has already been partly explored, in the form of the M-DPOP algorithm [Petcu et al., 2008], in which the agents have to pay *VCG taxes* that effectively align their selfish utility functions with the goal of maximizing the social welfare. It is unclear how such a mechanism could be applied in a context when privacy is a critical issue; this could be the subject of further investigation. Another way to make sure that agents correctly execute the protocol could be to introduce verification schemes (preferably in the form of *zero-knowledge proofs*), possibly combined with a reputation mechanism that would punish agents that actively try to manipulate the algorithm.

On the topic of uncertainty in DCOP, in this thesis we have restricted ourselves to single-lookahead, anticipatory algorithms. One possibility for generalization is to investigate algorithms that reason over longer horizons, in which case the deadlines to commit to some decisions may be *after* the times at which the values of some of the random variables become observable. This would correspond to a generalization to *multi-stage StochDCOP*, for which some applicable work already exists in the context of Quantified DCOP. One might also want to compare our anticipatory algorithms with *reactive* algorithms, which apply to problems in which some of the decisions may be revokable after the uncertainty has been resolved, and the solution may be adapted online (possibly at some cost) to run-time observations. Examples of such reactive approaches include the S-DPOP [Petcu and Faltings, 2005d] and RS-DPOP [Petcu and Faltings, 2005c] algorithms.

Bibliography

Atlas, J. and Decker, K. (2007). A complete distributed constraint optimization method for non-traditional pseudotree arrangements. In Durfee et al. [2007], pages 1–8.

Atlas, J. and Decker, K. (2010). Coordination for uncertain outcomes using distributed neighbor exchange. In Luck and Sen [2010], pages 1047–1054.

Bednarz, A., Bean, N., and Roughan, M. (2009). Hiccups on the road to privacy-preserving linear programming. In Al-Shaer, E. and Paraboschi, S., editors, *Proceedings of the 2009 ACM Workshop on Privacy in the Electronic Society (WPES'09)*, pages 117–120, Chicago, Illinois, USA. ACM.

Béjar, R., Domshlak, C., Fernández, C., Gomes, C., Krishnamachari, B., Selman, B., and Valls, M. (2005). Sensor networks and distributed CSP: communication, computation and complexity. *Artificial Intelligence*, 161(1–2):117–147.

Ben-Or, M., Goldwasser, S., and Wigderson, A. (1988). Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC'88)*, pages 1–10, Chicago, Illinois, USA. ACM.

Bent, R. and van Hentenryck, P. (2004). The value of consensus in online stochastic scheduling. In Zilberstein, S., Koehler, J., and Koenig, S., editors, *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS'04)*, pages 219–226, Whistler, British Columbia, Canada. AAAI Press.

Bessiere, C., Faltings, B., Silaghi, M.-C., Yokoo, M., and Zhang, W., editors (2003). *Proceedings of the Fourth International Workshop on Distributed Constraint Reasoning (DCR'03)*, Acapulco, Mexico.

Bessiere, C., Maestre, A., Brito, I., and Meseguer, P. (2005). Asynchronous backtracking without adding links: a new member in the ABT family. *Artificial Intelligence*, 161:7–24.

Bessiere, C., Maestre, A., and Meseguer, P. (2001). Distributed dynamic backtracking. In *Proceedings of the IJCAI'01 Second International Workshop on Distributed Constraint Reasoning (DCR'01)*, Seattle, WA, U.S.A.

Bilogrevic, I., Jadliwala, M., Hubaux, J.-P., Aad, I., and Niemi, V. (2011). Privacy-preserving activity scheduling on mobile devices. In Sandhu, R. S. and Bertino, E., editors, *Proceedings of the First ACM COnference on Data and Application Security and PrivacY (CODASPY'11)*, pages 261–272, San Antonio, Texas, USA. ACM.

Brito, I. and Meseguer, P. (2003). Distributed forward checking. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP'03)*, volume 2833, pages 801–806, Kinsale, Ireland. Springer Berlin / Heidelberg.

Brito, I. and Meseguer, P. (2007). Distributed forward checking may lie for privacy. In Pearce [2007].

Brito, I. and Meseguer, P. (2010). Cluster tree elimination for distributed constraint optimization with quality guarantees. *Fundamenta Informaticae*, 102:263—286.

Chandy, K. M. and Lamport, L. (1985). Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 3(1):63–75.

Chechetka, A. and Sycara, K. (2006). No-commitment branch and bound search for distributed constraint optimization. In Nakashima et al. [2006], pages 1427—1429.

Cheung, T.-Y. (1983). Graph traversal techniques and the maximum flow problem in distributed computation. *IEEE Transactions on Software Engineering*, 9(4):504–512.

Chevaleyre, Y., Dunne, P. E., Endriss, U., Lang, J., Lemaître, M., Maudet, N., Padget, J., Phelps, S., Rodríguez-Aguilar, J. A., and Sousa, P. (2006). Issues in multiagent resource allocation. *Informatica*, 30(1):3–31.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581.

Cohn, A., editor (2006). *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI'06)*, Boston, Massachusetts, U.S.A. AAAI Press.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

Dorronsoro, B. (2007). The VRP Web. http://neo.lcc.uma.es/radi-aeb/WebVRP/.

Doshi, P., Matsui, T., Silaghi, M.-C., Yokoo, M., and Zanker, M. (2008). Distributed private constraint optimization. In Zhang and Cercone [2008], pages 277–281.

Doxygen (2011). http://www.doxygen.org.

Du, W. and Atallah, M. J. (2001). Privacy-preserving cooperative scientific computations. In *Proceedings of the Fourteenth IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 273–282, Cape Breton, Nova Scotia, Canada. IEEE Computer Society.

Durfee, E. H., Yokoo, M., Huhns, M. N., and Shehory, O., editors (2007). *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'07)*, Honolulu, Hawaii. ACM.

Elgamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472.

Ezzahir, R., Bessiere, C., Wahbi, M., Benelallam, I., and Bouyakhf, E. H. (2009). Asynchronous inter-level forward-checking for DisCSPs. In Gent, I. P., editor, *Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP'09)*, volume 5732, pages 304–318, Lisbon, Portugal. Springer.

Faltings, B., Léauté, T., and Petcu, A. (2008). Privacy Guarantees through Distributed Constraint Satisfaction. In Zhang and Cercone [2008], pages 350–358.

Farinelli, A., Rogers, A., Petcu, A., and Jennings, N. R. (2008). Decentralised coordination of low-power embedded devices using the max-sum algorithm. In Padgham et al. [2008], pages 639–646.

Fitzpatrick, S. and Meertens, L. G. L. T. (2001). An experimental assessment of a stochastic, anytime, decentralized, soft colourer for sparse graphs. In Steinhöfel, K., editor, *Proceedings of the International Symposium on Stochastic Algorithms: Foundations and Applications (SAGA'01)*, volume 2264, pages 49–64, Berlin, Germany. Springer.

Franzin, M. S., Freuder, E. C., Rossi, F., and Wallace, R. J. (2004). Multi-agent constraint systems with preferences: Efficiency, solution quality, and privacy loss. *Computational Intelligence*, 20(2):264—286.

Gendreau, M., Potvin, J.-Y., Bräysy, O., Hasle, G., and Løkketangen, A. (2007). Metaheuristics for the vehicle routing problem and its extensions : A categorized bibliography. Technical Report 2007-27, CIRRELT.

Gershman, A., Meisels, A., and Zivan, R. (2006). Asynchronous forward-bounding for distributed constraints optimization. In Brewka, G., Coradeschi, S., Perini, A., and Traverso, P., editors, *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI'06)*, pages 103–107, Riva del Garda, Italy. IOS Press.

Gershman, A., Meisels, A., and Zivan, R. (2009). Asynchronous forward bounding for distributed COPs. *Journal of Artificial Intelligence Research*, 34:61–88.

Gershman, A., Zivan, R., Grinshpoun, T., Grubshtein, A., and Meisels, A. (2008). Measuring distributed constraint optimization algorithms. In *Proceedings of the AAMAS'08 Distributed Constraint Reasoning Workshop (DCR'08)*, pages 17–24, Estoril, Portugal.

Gillett, B. E. and Miller, L. R. (1974). A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22(2):340–349.

Goldreich, O. (2009). *Foundations of Cryptography*, volume 2, Basic Applications. Cambridge University Press.

Goldreich, O., Micali, S. M., and Wigderson, A. (1987). How to play any mental game. In Aho, A. V., editor, *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC'87)*, pages 218–229, New York, NY, USA. ACM.

Graphviz (2011). Graphviz – Graph Visualization Software. http://www.graphviz.org/.

Greenstadt, R., Grosz, B., and Smith, M. D. (2007). SSDPOP: Using secret sharing to improve the privacy of DCOP. In Pearce [2007].

Greenstadt, R., Pearce, J. P., and Tambe, M. (2006). Analysis of privacy loss in distributed constraint optimization. In Cohn [2006], pages 647–653.

Grinshpoun, T. and Meisels, A. (2008). Completeness and performance of the APO algorithm. *Journal of Artificial Intelligence Research (JAIR)*, 33:223–258.

Grubshtein, A., Grinshpoun, T., Meisels, A., and Zivan, R. (2009). Asymmetric distributed constraint optimization. In Hirayama et al. [2009], pages 60–74.

Herlea, T., Claessens, J., Preneel, B., Neven, G., Piessens, F., and Decker, B. D. (2001). On securely scheduling a meeting. In *Proceedings of the Sixteenth International Conference on Information Security – Trusted information: the new decade challenge (SEC'01)*, International Federation For Information Processing (IFIP) Series, pages 183–198, Disneyland Paris, France. Kluwer.

Hirayama, K., Yeoh, W., and Zivan, R., editors (2009). *Proceedings of the IJCAI'09 Distributed Constraint Reasoning Workshop (DCR'09)*, Pasadena, California, USA.

Hirayama, K. and Yokoo, M. (1997). Distributed partial constraint satisfaction problem. In Smolka, G., editor, *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP'97)*, volume 1330, pages 222–236, Linz, Austria. Springer.

Hirayama, K. and Yokoo, M. (2005). The distributed breakout algorithms. *Artificial Intelligence*, 161(1–2 (Special issue: Distributed constraint satisfaction)):89–115.

JaCoP (2011). JaCoP java constraint programming solver. http://jacop.osolpro.com/.

Jain, M., Taylor, M. E., Tambe, M., and Yokoo, M. (2009). DCOPs meet the real world: Exploring unknown reward matrices with applications to mobile sensor networks. In Boutilier, C., editor, *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 181–186, Pasadena, California, USA.

JDOM (2009). The JDOM XML toolbox for Java. http://www.jdom.org/.

Jennings, N. R., Sierra, C., Sonenberg, L., and Tambe, M., editors (2004). *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, Columbia University, New York City, U.S.A. ACM Special Interest Group on Artificial Intelligence (SIGART), IEEE Computer Society.

Joomla! (2011). http://www.joomla.org.

Kaelbling, L. P. and Saffiotti, A., editors (2005). *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, Edinburgh, Scotland. Professional Book Center, Denver, USA.

Kearns, M. J., Littman, M. L., and Singh, S. P. (2001). Graphical models for game theory. In Breese, J. S. and Koller, D., editors, *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI'01)*, pages 253–260, Seattle, WA, U.S.A. Morgan Kaufmann.

Kerschbaum, F. (2011). Computation on randomized data. In *Proceedings of the 2011 Workshop on Cryptography and Security in Clouds (CSC'11)*, Zurich, Switzerland.

Kiekintveld, C., Yin, Z., Kumar, A., and Tambe, M. (2010). Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In Luck and Sen [2010], pages 133–140.

Léauté, T. and Faltings, B. (2009a). $\mathbb{E}$[DPOP]: Distributed constraint optimization under stochastic uncertainty using collaborative sampling. In Hirayama et al. [2009], pages 87–101.

Léauté, T. and Faltings, B. (2009b). Privacy-Preserving Multi-agent Constraint Satisfaction. In *Proceedings of the 2009 IEEE International Conference on PrivAcy, Security, riSk And Trust (PASSAT'09)*, pages 17–25, Vancouver, British Columbia. IEEE Computer Society Press.

Léauté, T. and Faltings, B. (2011a). Coordinating Logistics Operations with Privacy Guarantees. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 2482–2487, Barcelona, Spain. AAAI Press.

Léauté, T. and Faltings, B. (2011b). Distributed Constraint Optimization under Stochastic Uncertainty. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence (AAAI'11)*, pages 68–73, San Francisco, USA.

Léauté, T., Ottens, B., and Faltings, B. (2010). Ensuring privacy through distributed computation in multiple-depot vehicle routing problems. In *Proceedings of the ECAI'10 Workshop on Artificial Intelligence and Logistics (AILog'10)*, pages 25–30, Lisbon, Portugal.

Leyton-Brown, K., Pearson, M., and Shoham, Y. (2000). Towards a universal test suite for combinatorial auction algorithms. In Jhingran, A., Mason, J. M., and Tygar, D., editors,

Proceedings of the Second ACM Conference on Electronic commerce (EC'00), pages 66–76, Minneapolis, Minnesota, USA. ACM Special Interest Group on Electronic Commerce (SIGEcom), ACM.

Li, W., Li, H., and Deng, C. (2011). Privacy-preserving horizontally partitioned linear programs with inequality constraints. *Optimization Letters*.

Luck, M. and Sen, S., editors (2010). *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)*, Toronto, Canada.

Maheswaran, R. T., Pearce, J. P., Bowring, E., Varakantham, P., and Tambe, M. (2006). Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. *Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 13(1):27–60.

Maheswaran, R. T., Pearce, J. P., and Tambe, M. (2004a). Distributed algorithms for DCOP: A graphical-game-based approach. In Bader, D. A. and Khokhar, A. A., editors, *Proceedings of the ISCA Seventeenth International Conference on Parallel and Distributed Computing Systems (ISCA PDCS'04)*, pages 432–439, Francisco, California, USA. ISCA.

Maheswaran, R. T., Tambe, M., Bowring, E., Pearce, J. P., and Varakantham, P. (2004b). Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In Jennings et al. [2004], pages 310–317.

Mailler, R. and Lesser, V. R. (2003). A mediation based protocol for distributed constraint satisfaction. In Bessiere et al. [2003].

Mailler, R. and Lesser, V. R. (2004). Solving distributed constraint optimization problems using cooperative mediation. In Jennings et al. [2004], pages 438–445.

Mangasarian, O. L. (2010a). Privacy-preserving horizontally partitioned linear programs. *Optimization Letters*.

Mangasarian, O. L. (2010b). Privacy-preserving linear programming. *Optimization Letters*, 5(1):165–172.

Matsui, T., Matsuo, H., Silaghi, M.-C., Hirayama, K., and Yokoo, M. (2008). Resource constrained distributed constraint optimization with virtual variables. In *Proceedings of the Twenty-Third Conference on Artificial Intelligence (AAAI'08)*, pages 120–125, Chicago, Illinois, USA. AAAI Press.

Matsui, T., Matsuo, H., Silaghi, M.-C., Hirayama, K., Yokoo, M., and Baba, S. (2010). A quantified distributed constraint optimization problem. In Luck and Sen [2010], pages 1023–1030.

Meisels, A. (2008). *Distributed Search by Constrained Agents – Algorithms, Performance, Communication*. Advanced Information and Knowledge Processing. Springer.

Meisels, A. and Zivan, R. (2003). Asynchronous forward-checking on DisCSPs. In Bessiere et al. [2003].

Meisels, A. and Zivan, R. (2007). Asynchronous forward-checking for DisCSPs. *Constraints*, 12:131–150.

Modi, P. J., Jung, H., Tambe, M., Shen, W.-M., and Kulkarni, S. (2001). A dynamic distributed constraint satisfaction approach to resource allocation. In Walsh, T., editor, *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP'01)*, number 2239 in Lecture Notes In Computer Science, pages 685–700, Paphos, Cyprus. Springer.

Modi, P. J., Shen, W.-M., Tambe, M., and Yokoo, M. (2005). ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180.

Monier, P., Piechowiak, S., and Mandiau, R. (2009). A complete algorithm for DisCSP: Distributed backtracking with sessions (DBS). In *Proceedings of the Second International Workshop on Optimisation in Multi-Agent Systems (OPTMAS'09)*, Budapest, Hungary.

Mura, P. L. (2000). Game networks. In Boutilier, C. and Goldszmidt, M., editors, *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI'00)*, pages 335–342, Stanford, California, USA. Morgan Kaufmann.

Nakashima, H., Wellman, M. P., Weiss, G., and Stone, P., editors (2006). *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'06)*, Hakodate, Japan. ACM Press.

Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(1):48–49.

Netzer, A., Meisels, A., and Grubshtein, A. (2010). Concurrent forward bounding for DCOPs. In van der Hoek, W., Kaminka, G. A., Lespérance, Y., Luck, M., and Sen, S., editors, *Proceedings of the Twelfth International Workshop on Distributed Constraint Reasoning (DCR'10)*, pages 65–79, Toronto, Canada.

Olteanu, A., Léauté, T., and Faltings, B. (2011). Asynchronous forward bounding (AFB): Implementation and performance experiments. Semester project report, EPFL Artificial Intelligence Lab (LIA). http://liawww.epfl.ch/cgi-bin/Pubs/single_entry?bibtex_key=Olteanu2011.

OR-Objects (2010). Operations Research – Java Objects. http://or-objects.org/.

Ottens, B. and Faltings, B. (2008). Coordinating agent plans through distributed constraint optimization. In *Proceedings of the ICAPS'08 Multiagent Planning Workshop (MASPLAN'08)*, Sydney, Australia.

Padgham, L., Parkes, D. C., Müller, J. P., and Parsons, S., editors (2008). *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*, Estoril, Portugal.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In Stern, J., editor, *Advances in Cryptology – EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Proceedings*, number 1592 in Lecture Notes In Computer Science, pages 223–238, Prague, Czech Republic. Springer.

Parsons, S., Rodríguez-Aguilar, J. A., and Klein, M. (2011). Auctions and bidding: A guide for computer scientists. *ACM Computing Surveys (CSUR)*, 43(2).

Pearce, J. P., editor (2007). *Proceedings of the Ninth International Workshop on Distributed Constraint Reasoning (CP-DCR'07)*, Providence, RI, USA.

Pearce, J. P. and Tambe, M. (2007). Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In Veloso [2007], pages 1446–1451.

Pedersen, T. B. and Savaş, E. (2009). Impossibility of unconditionally secure scalar products. *Data and Knowledge Engineering*, 68(10):1059–1070.

Pedersen, T. P. (1991). A threshold cryptosystem without a trusted party (extended abstract). In Davies, D. W., editor, *Advances in Cryptology – EUROCRYPT '91, Workshop on the Theory and Application of Cryptographic Techniques, Proceedings*, volume 547, pages 522–526, Brighton, U.K. Springer.

Petcu, A. (2006). FRODO: A FRamework for Open/Distributed constraint Optimization. Technical Report 2006/001, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland).

Petcu, A. (2007). *A Class of Algorithms for Distributed Constraint Optimization*. PhD thesis no. 3942, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland).

Petcu, A. and Faltings, B. (2005a). A-DPOP: Approximations in distributed optimization. In van Beek, P., editor, *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP'05)*, volume 3709, pages 802–806, Sitges, Spain. Springer.

Petcu, A. and Faltings, B. (2005b). DPOP: A Scalable Method for Multiagent Constraint Optimization. In Kaelbling and Saffiotti [2005], pages 266–271.

Petcu, A. and Faltings, B. (2005c). RS-DPOP: Optimal solution stability in continuous-time optimization. In *Proceedings of the Sixth International Workshop on Distributed Constraint Reasoning (DCR'05)*, Edinburgh, Scotland.

Petcu, A. and Faltings, B. (2005d). S-DPOP: Superstabilizing, fault-containing multiagent combinatorial optimization. In Veloso, M. M. and Kambhampati, S., editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI'05)*, pages 449–454, Pittsburgh, Pennsylvania, U.S.A. AAAI Press / The MIT Press.

Petcu, A. and Faltings, B. (2006). O-DPOP: An algorithm for open/distributed constraint optimization. In Cohn [2006], pages 703–708.

Petcu, A. and Faltings, B. (2007). MB-DPOP: A new memory-bounded algorithm for distributed optimization. In Veloso [2007], pages 1452–1457.

Petcu, A., Faltings, B., and Parkes, D. C. (2008). M-DPOP: Faithful distributed implementation of efficient social choice problems. *Journal of Artificial Intelligence Research (JAIR)*, 32:705–755.

Saidman, S. L., Roth, A. E., Sönmez, T., Ünver, M. U., and Delmonico, F. L. (2006). Increasing the opportunity of live kidney donation by matching for two- and three-way exchanges. *Transplantation*, 81(5):773–782.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

Silaghi, M.-C. (2005a). Hiding absence of solution for a distributed constraint satisfaction problem (poster). In *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS'05)*, pages 854–855, Clearwater Beach, FL, USA. AAAI Press.

Silaghi, M.-C. (2005b). Using secure DisCSP solvers for generalized Vickrey auctions — complete and stochastic techniques. In Kaelbling and Saffiotti [2005].

Silaghi, M.-C., Faltings, B., and Petcu, A. (2006). Secure combinatorial optimization simulating DFS tree-based variable elimination. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida. Springer.

Silaghi, M.-C. and Mitra, D. (2004). Distributed constraint satisfaction and optimization with privacy enforcement. In *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04)*, pages 531–535, Beijing, China. IEEE Computer Society Press.

Silaghi, M.-C., Sam-Haroud, D., and Faltings, B. (2000). Asynchronous search with aggregations. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI'00)*, pages 917–922, Austin, TX, U.S.A. AAAI Press/The MIT Press.

Silaghi, M.-C., Sam-Haroud, D., and Faltings, B. (2001). ABT with asynchronous reordering. In *Proceedings of the Second Asia-Pacific Conference on Intelligent Agent Technology (IAT'01)*, pages 54–63, Maebashi City, Japan. World Scientific Publishing Company Pte Ltd.

Silaghi, M.-C. and Yokoo, M. (2006). Nogood-based asynchronous distributed optimization (ADOPT-ng). In Nakashima et al. [2006], pages 1389–1396.

Silaghi, M.-C. and Yokoo, M. (2007). Dynamic DFS tree in ADOPT-ing. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI'07)*, pages 763–769, Vancouver, British Columbia, Canada. AAAI Press.

Singh, S., Soni, V., and Wellman, M. P. (2004). Computing approximate Bayes-Nash equilibria in tree-games of incomplete information. In Breese, J. S., Feigenbaum, J., and Seltzer, M. I., editors, *Proceedings of the Fifth ACM Conference on Electronic Commerce (EC'04)*, pages 81–90, New York, NY, USA. ACM.

Soni, V., Singh, S., and Wellman, M. P. (2007). Constraint satisfaction algorithms for graphical games. In Durfee et al. [2007], pages 67–74.

Stranders, R., Fave, F. M. D., Rogers, A., and Jennings, N. R. (2011). U-GDL: A decentralised algorithm for DCOPs with uncertainty. In *Proceedings of the Fourth International Workshop on Optimisation in Multi-Agent Systems (OptMAS'11)*, Taipei, Taiwan.

Sultanik, E. A., Lass, R. N., and Regli, W. C. (2007). DCOPolis: A framework for simulating and deploying distributed constraint optimization algorithms. In Pearce [2007].

Taylor, M. E., Jain, M., Jin, Y., Yokoo, M., and Tambe, M. (2010). When should there be a "me" in "team"? Distributed multi-agent optimization under uncertainty. In Luck and Sen [2010].

Toth, P. and Vigo, D., editors (2001). *The Vehicle Routing Problem*. SIAM.

Tsiounis, Y. and Yung, M. (1998). On the security of Elgamal-based encryption. In Imai, H. and Zheng, Y., editors, *Proceedings of the First International Workshop on Practice and Theory in Public Key Cryptography (PKC'98)*, volume 1431, pages 117–134, Pacifico Yokohama, Japan. Springer.

Vaidya, J. (2009). Privacy-preserving linear programming. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Applied Computing (SAC'09)*, pages 2002–2007, Honolulu, Hawaii, USA. ACM.

Veloso, M. M., editor (2007). *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, Hyderabad, India.

Vickrey, D. and Koller, D. (2002). Multi-agent algorithms for solving graphical games. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, pages 345–351, Edmonton, Alberta, Canada. AAAI Press.

Vinyals, M., Pujol-Gonzalez, M., Rodríguez-Aguilar, J. A., and Cerquides, J. (2010a). Divide-and-coordinate: DCOPs by agreement. In Luck and Sen [2010], pages 149–156.

Vinyals, M., Rodríguez-Aguilar, J. A., and Cerquide, J. (2011a). A survey on sensor networks from a multiagent perspective. *The Computer Journal*, 54(3):455–470.

Vinyals, M., Rodríguez-Aguilar, J. A., and Cerquides, J. (2010b). Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 22(3):439–464.

Vinyals, M., Shieh, E., Cerquides, J., Rodríguez-Aguilar, J. A., Yin, Z., Tambe, M., and Bowring, E. (2011b). Quality guarantees for region optimal DCOP algorithms. In Tumer, K., Yolum, P., Sonenberg, L., and Stone, P., editors, *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS'11)*, pages 133–140, Taipei, Taiwan. IFAAMAS.

Wahbi, M., Ezzahir, R., Bessiere, C., and Bouyakhf, E. H. (2011). DisChoco 2: A platform for distributed constraint reasoning. In *Proceedings of the Thirteenth International Workshop on Distributed Constraint Reasoning (DCR'11)*, pages 112–121, Barcelona, Spain. http://www.lirmm.fr/coconut/dischoco/.

Wallace, R. J. and Freuder, E. C. (2005). Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. *Artificial Intelligence*, 161(1–2):209–227.

Wang, C., Ren, K., and Wang, J. (2011). Secure and practical outsourcing of linear programming in cloud computing. In *Proceedings of the Thirtieth IEEE International Conference on Computer Communications (IEEE INFOCOM'11)*, pages 820–828, Shanghai, China.

XCSP (2008). *XML Representation of Constraint Networks — Format XCSP 2.1*. Organising Committee of the Third International Competition of CSP Solvers.

Yao, A. C.-C. (1986). How to generate and exchange secrets. In *Proceedings of the Twenty-Seventh Annual Symposium on Foundations of Computer Science (SFSC'86)*, pages 162–167, Toronto, Ontario, Canada. IEEE Computer Society.

Yeoh, W., Felner, A., and Koenig, S. (2008). BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm. In Padgham et al. [2008], pages 591–598.

Yokoo, M. (1995). Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In Montanari, U. and Rossi, F., editors, *Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP'95)*, number 976 in Lecture Notes In Computer Science, pages 88–102, Cassis, France. Springer.

Yokoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K. (1992). Distributed constraint satisfaction for formalizing distributed problem solving. In *Proceedings of the Twelfth International Conference on Distributed Computing Systems (ICDCS'92)*, pages 614–621, Yokohama, Japan. IEEE Computer Society Press.

Yokoo, M. and Hirayama, K. (1996). Distributed breakout algorithm for solving distributed constraint satisfaction problems. In Tokoro, M., editor, *Proceedings of the Second International Conference on Multiagent Systems (ICMAS'96)*, pages 401–408, Kyoto, Japan. AAAI Press.

Yokoo, M. and Suzuki, K. (2002). Secure multi-agent dynamic programming based on homomorphic encryption and its application to combinatorial auctions. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'02)*, pages 112–119, Bologna, Italy. ACM Press.

Yokoo, M., Suzuki, K., and Hirayama, K. (2002). Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In van Hentenryck, P., editor, *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'02)*, volume 2470, pages 387–401, Cornell University, Ithaca, NY, U.S.A. Springer-Verlag.

Yokoo, M., Suzuki, K., and Hirayama, K. (2005). Secure distributed constraint satisfaction: Reaching agreement without revealing private information. *Artificial Intelligence*, 161(1–2, Distributed Constraint Satisfaction):229–245.

Zhang, C. and Cercone, N., editors (2008). *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'08)*, Sydney, Australia. IEEE Computer Society Press.

Zhang, W., Wang, G., Xing, Z., and Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1–2):55–87.

Zivan, R., Glinton, R., and Sycara, K. P. (2009). Distributed constraint optimization for large teams of mobile sensing agents. In *Proceedings of the 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'09)*, pages 347–354, Milan, Italy. IEEE Computer Society Press.

Zivan, R. and Meisels, A. (2004). Concurrent dynamic backtracking for distributed CSPs. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP'04)*, volume 3258/2004, pages 782–787, Toronto, Canada. Springer Berlin / Heidelberg.

Zivan, R. and Meisels, A. (2006). Concurrent search for distributed CSPs. *Artificial Intelligence*, 170(4):440–461.

Index

AAS, 21
ABT, 19
  ABT_{not}, 20, 44
  ABTR, 21
ADOPT, 25, 114, 163
  ADOPT-ng, 29
  BnB-ADOPT, 29, 33
AFB, 32, 163
  AFB-BJ, 32
AFC, 23
  AFC-ng, 23
agent, 7, 8, 100, 111, 113
  honest but curious, 69, 175
  knowledge, 44, 69, 119, 122
  neighbor, 40, 61, 69, 71
  virtual, 114, 115, 119, 120, 122, 139
AILFC, 23
APO, 23
auction
  combinatorial, 14, 91, 167
AWC, 20

back-edge, **26**, 77

codename, **53**, 56, 58, 70, 72, 76, 77, 80, 84
  collision, 57
collusion, 49, 60, 89, 98
CompAPO, *see* APO
CompOptAPO, *see* OptAPO
ConcDB, 24
ConcFB, 33
consensus, **102**, 103, 105, 134, 138, 142, 145, 148–150
constraint
  asymmetric, 44
  check, *see* NCCC
  global, 162
  graph, 85, 166
    decisional, **120**
    diameter, **27**, 73, 78, 117
  hard, 7
  knowledge, 44, 69, 85, 86, 92, 94, 114, 122
  probabilistic, 111
  soft, 9, 55, 100, 113
cost, 9
  distribution, 111, 112
cycle, 40, 53, 73, 74, 76, 78

DaCSA, 36
DALO, 37
DBA, 33
DBS, 20
DCEE, 111
DCOP, 8
  algorithms, 163
    complete, 24
    incomplete, 33
  Quantified (QDCOP), 112, 115, 131
  Resource-Constrained (RCDCOP), 131
  stochastic uncertainty, *see* StochDCOP
DCPOP, 32
DCTE, 32
decryption, 60, 64, 65, **67**, 68, 72, 80, 84
DisCHOCO, 160
DisCSP, 7
  algorithms, 19, 164
DisDB, 20
DisFC, 20, 45

DisSDMDVRP, *see* Vehicle Routing Problem (VRP)
DisWCSP, **9**, 65, 96
DKEP, *see* kidney exchange
domain, 7, 8, 100, 111, 113, 138, 142
  knowledge, 44, 54, 69, 101
DPOP, 29, 153, 154, 163
  A-DPOP, 35
  $\mathbb{E}$[DPOP], 163
    Central-, 128
    Comp-, 115
    equivalence, 135
    Global-, 129
    hybrid, 134
    Local-, 123
  M-DPOP, 175
  MB-DPOP, 32
  O-DPOP, 163
    ASO-DPOP, 163
  P-DPOP, **50**, 70, 77, 82, 84, 163, 167
    P-DPOP$^-$, **58**, 70, 73, 82, 84
    P^2-DPOP, **65**, 72, 81, 84, 85, 163, 167
    P$^{3/2}$-DPOP, **58**, 71, 78, 83, 85, 163
  Param-DPOP, 163
  S-DPOP, 163, 175
    RS-DPOP, 175
DSA, 34, 163
dynamic programming, 29, 45, 57, 66

encryption
  ElGamal, **41**, 60, 64, 65, 67, 68, 84, 86
evaluation function, **100**, 101, 112
  commensurability, **103**, 115, 123, 128, 129, 134, 144
  consensus, *see* consensus
  expectation, *see* expectation
  linearity, **103**, 122, 123, 125, 127, 128, 135–137, 144
  robust, *see* robustness
  support, **103**, 115
expectation, 101, **102**, 103–105, 107, 115, 134, 138, 142, 145, 150, 153

FRODO, 86, 96, 144, **159**

game
  graphical, 16, 93
  party, 18, 95
GDL
  Action-, 32
graph coloring, 10, 86
  stochastic, **104**, 126, 135, 145, 146, 153

heuristic, 27, 132, 135
  least connected, 133, 145
  most branching, 125, 133
  most connected, 27, 133, 145
  most exploration, 133
  random, 52, 145
hypercube, 162

IDB, 33
induced width, *see* width

JaCoP, 161, 162, 167
join, 30, 31, 128, 134, **161**

kidney exchange, **109**, 137, 145, 149, 167

lowest common ancestor (lca), 129, 132–134, 138, 142

Max-DisCSP, **9**, 34, 56, 82–84
Max-Sum, 35
meeting scheduling, 10, 88, 167
MGM, MGM-2, 35, 163
MPC, *see* SMC
MPC-Dis(W)CSP4, 24, 85, 86, 163, 164

NCBB, 33
NCCC, 32, 143, 145, 152, 165
null link, 44, 115, 119, 120

obfuscation, **54**, 56, 82, 83, 85
OptAPO, 29
ordering of variables
  circular, 60, 71, 79
  linear, 19, 65, 66, 81

pseudo-tree, *see* pseudo-tree

permutation, 54, 60, 63, 65, 85
priority, 19
privacy
agent, **40**, 44, 49, 53, 56, 60, 69, 114, 122, 125
constraint, **41**, 45, 46, 55, 65, 81, 114, 122, 125, 152
decision, **41**, 44, 46, 56, 58, 83, 84, 86, 95
guarantees, 69
topology, **40**, 45, 49, 53, 54, 56, 58, 60, 73, 82, 86, 89, 92, 95, 122, 125
probability distribution, **100**, 138, 145
knowledge, 101
projection, 31, 68, 115, 123, 127–129, 134, 136, 139, 141, **161**
pseudo-
child, **26**, 76, 77
parent, **26**, 75, 85
tree, 23, **26**, 29, 33, 57, 60, 70, 75, 117, 164, 166
consistency, **114**, 115, 116, 119, 121–124, 127, 132

reformulation, 31, 121, 123, 125, 132
rerooting, 59, **62**, 64, 72, 80, 85, 86
resource allocation, 13, 91
robustness, **102**, 103–105, 107, 114–116, 124, 126, 130, 132, 135, 137, 138, 145, 149, 150, 154, 155
root election, 27, **51**, 57, 70, 73, 117, 164
routing, **60**, 68, 71, 79

sampling, 102, 103, **138**, 153, 154
collaborative, 141, 154, 155
independent, 140, 154, 155
SBB, *see* SynchBB
scenario, 137, **138**, 143, 155
secret sharing, 45
semi-private information, **40**, 69, 82, 84, 89
sensor network, 11
stochastic, **105**, 124, 126, 132, 133, 135
separator, **31**, 56–58, 65, 68
simulated time, 85, 145, 165
SMC, 46
solution
space, 161, 167
to a DCOP, 9
to a DisCSP, 8
to a QDCOP, 113
to a StochDCOP, **100**, 122, 128
StochDCOP, 99
algorithms, 163
StochDisSDMDVRP, *see* Vehicle Routing Problem (VRP)
StochDKEP, *see* kidney exchange
subtree, 31, 62, 130
SynchBB, 24, 163

third party, 46
threshold scheme, *see* collusion
treewidth, *see* width

utility, 9
distribution, 111

variable
decision, 7, 8, 100, 111, 113
knowledge, 44, 69, 92, 101
random, 100, 113
Vehicle Routing Problem (VRP), 14, 96, 162
stochastic, **107**, 145, 150

width
induced, 32, 58, 121, 124, 135, 142
treewidth, 32, 58
worst case, *see* robustness

XCSP, 160

Thomas Léauté
EPFL LIA, Station 14
CH-1015 Lausanne, Switzerland
(+41) (0)765 210 200
thomas.leaute@alum.mit.edu

- I am looking forward to **contributing to software projects, possibly involving A.I.**
- I like **multi-cultural teamwork** as well as **contact with customers**.

Graduate Studies and Diplomas

2006 — 2011 **École Polytechnique Fédérale de Lausanne**, *Switzerland*
PhD in Computer Science
*Specialization in **Privacy and Uncertainty in Distributed Artificial Intelligence***

2003 — 2005 **Massachusetts Institute of Technology**, *USA*
Master of Science in Aeronautics and Astronautics
*Specialization in **Applications of Artificial Intelligence to Autonomous Robots***

2001 — 2003 **École Centrale des Arts et Manufactures de Paris**, *France*
Diplôme d'Ingénieur — "Engineering Diploma"

Professional Experience

2005 — 2006 **European Space Agency — European Space Operations Centre**, *Germany*
Young Graduate Trainee
Advanced Mission Concepts and Technology Office

I delivered a **graphical decision support tool** for the scheduling of data downloads from the Mars Express spacecraft, which is subject to very tight memory, bandwidth and timing constraints. The tool was developed in Java using **rapid prototyping**, and has since been adopted for the day-to-day operation of the spacecraft.

Key Skills and Projects

Open-Source Development in Java — Distributed Artificial Intelligence

Project manager and core developer of the FRODO 2 distributed platform for **multi-agent decision-making**, which has raised **interest from academics and industrials in 13 different countries** since its release in 2009, and has lead to industrial projects with Nokia, Armasuisse, and Novea (express delivery service).

My PhD thesis addresses **privacy** and **stochastic uncertainty** in Distributed Constraint Satisfaction (DisCSP) and Distributed Constraint Optimization (DCOP), using techniques such as **Secure Multi-Party Computation** (MPC), ElGamal and Paillier **homomorphic encryption**, Shamir **secret sharing**, and collaborative **sampling**.

Hardware-in-the-loop Development in C/C++ using CPLEX — A.I. and Robotics

My Master's thesis tackles the problem of **high-level guidance of hybrid discrete-continuous systems**, and was applied to the coordination of Unmanned Aerial Vehicles (UAVs) under a contract with Boeing, and of autonomous space exploration rovers. The techniques involved include **model-based programming**, **Mixed-Integer Linear Programming** (MILP), **Hierarchical Temporal Networks** (HTNs), and **TCP/IP sockets**.

Team Management — Student associations and community service

Since 2007, I am an active member of the **Rotaract Club** of Lausanne, a community service association for students and young professionals; I served as Secretary in 2008-2009, and as Webmaster ever since.

In 2001-2002 I was Secretary General of Forum Centrale Entreprises, a **€400,000-turnover student association** in charge of organizing an annual job fair at École Centrale Paris that brings together one thousand students and **more than one hundred firms and universities** of European and international fame.

Language Proficiency

French:	**native**	German:	intermediate (B1-B2)	Spanish:	beginner (A2-B1)
English:	**fluent (C2)**	Dutch:	intermediate (B1-B2)	Italian:	beginner (A2-B1)

Personal Activities

Ambassadorial Scholar of the Rotary Foundation — *Cambridge, USA, 2003-2004*

The purpose of this scholarship is to **further international understanding**. Scholars serve as ambassadors of goodwill to the people of their host country, and give presentations to Rotary clubs and other groups.

First Aid Volunteer — *EPFL campus, since 2007*

The mission of first aid volunteers is to respond to every-day medical emergencies. First aid volunteers are trained to deliver cardiopulmonary resuscitation (CPR) and to use automated external defibrillators (AED).

Personal Information — 29, single, French citizenship

List of Personal Publications

Faltings, B., Léauté, T., and Petcu, A. (2008). Privacy Guarantees through Distributed Constraint Satisfaction. In Zhang, C. and Cercone, N., editors, *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'08)*, pages 350–358, Sydney, Australia. IEEE Computer Society Press.

Hirayama, K., Yeoh, W., and Zivan, R., editors (2009). *Proceedings of the IJCAI'09 Distributed Constraint Reasoning Workshop (DCR'09)*, Pasadena, California, USA.

Léauté, T. and Faltings, B. (2009a). $\mathbb{E}$[DPOP]: Distributed constraint optimization under stochastic uncertainty using collaborative sampling. In Hirayama et al. [2009], pages 87–101.

Léauté, T. and Faltings, B. (2009b). Privacy-Preserving Multi-agent Constraint Satisfaction. In *Proceedings of the 2009 IEEE International Conference on PrivAcy, Security, riSk And Trust (PASSAT'09)*, pages 17–25, Vancouver, British Columbia. IEEE Computer Society Press.

Léauté, T. and Faltings, B. (2011a). Coordinating Logistics Operations with Privacy Guarantees. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 2482–2487, Barcelona, Spain. AAAI Press.

Léauté, T. and Faltings, B. (2011b). Distributed Constraint Optimization under Stochastic Uncertainty. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence (AAAI'11)*, pages 68–73, San Francisco, USA.

Léauté, T., Ottens, B., and Faltings, B. (2010). Ensuring privacy through distributed computation in multiple-depot vehicle routing problems. In *Proceedings of the ECAI'10 Workshop on Artificial Intelligence and Logistics (AILog'10)*, pages 25–30, Lisbon, Portugal.

Léauté, T., Ottens, B., and Szymanek, R. (2009). FRODO 2.0: An Open-Source Framework for Distributed Constraint Optimization. In Hirayama et al. [2009], pages 160–164. http://liawww.epfl.ch/frodo/.