Scaling DCOP algorithms for cooperative multi-agent coordination

by

Marc Pujol-Gonzalez

A dissertation presented in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy in Computer Science

Tutor:
Dr. Jordi González Sabaté

Advisors:
Dr. Jesús Cerquides
Dr. Pedro Meseguer
Dr. Juan A. Rodríguez-Aguilar

September 24, 2014

Universitat Autònoma de Barcelona
Departament de Ciències de la Computació

In memory of

Marta Majó i Ayma

and

Juan González Coll

you passed away,
but your legacy remains.

Abstract

This dissertation focuses on scaling up distributed constraint optimization problem (DCOP) solving algorithms to cope with larger-scale and dynamic applications. The DCOP framework represents a coordination problem as a distributed function to optimize by a group of agents. Although it has been employed to tackle numerous application domains (*e.g.*, meeting scheduling, sensor networks), DCOPs are NP-hard, and thus solving them poses challenging scalability issues.

Here we identify the Generalized Distributive Law (GDL) as a promising foundation to build complete and approximate DCOP algorithms. On optimal solving, GDL-based algorithms offer scale-up opportunities thanks to their exponential complexity on the treewidth instead of on the number of agents. Firstly, we scale up the GDL with function filtering algorithm, a state-of-the-art algorithm that employs upper and lower bounds to prune (filter) suboptimal solutions. We present novel techniques to improve the quality of these bounds, hence reducing the algorithm's computation and communication costs. Our experiments show that agents using our techniques can solve larger problems than the state-of-the-art. Thereafter, we introduce a novel scheme that allows agents to trade off computation and communication costs. Using this scheme, we propose novel, optimal GDL algorithms, including the best-in-class for communication-constrained and for computation-constrained applications.

Secondly, we turn our attention to larger-scale, dynamic applications for which optimality is not an option. We introduce the Limited-range Online Routing Problem (LORP) as a benchmark for such applications, along with the MAS-Planes toolkit to facilitate research on it. Then we present LORP solutions based on Max-Sum, the approximate version of GDL. Using Tractable Higher-Order Potentials (THOPs), we show that it is possible to reduce Max-Sum's complexity from exponential to polynomial. Empirically, our novel approach for the LORP achieves better results than current state-of-the-art methods. However, THOP models are harder to design than standard DCOP models. Therefore, we also introduce a methodology for developing such models for complex applications involving heterogeneous teams of agents. Finally, we validate our methodology by implementing an inter-team coordination model for the RoboCup Rescue simulation that empirically yields notable performance benefits.

In summary, this thesis widens the range of applications that optimal and approximate GDL-based DCOP algorithms can successfully tackle.

Acknowledgements

First and foremost I want to thank my advisors. This work would never have been possible without your unconditional guidance, support and dedication. My most sincere gratitude to Jesus Cerquides, for your always open door, your sharpness, and your open mind. We have not changed the world —yet?— but you definitely have changed me for the better. To Pedro Meseguer, for betting on me, for your relentless encouragement and your attention to detail. To Juan A. Rodgriguez-Aguilar, for your unbreakable optimism, your tenacity, your perpetual willingness to help and our countless conversations. Chavales, you are my heroes.

To Prof. Milind Tambe and the entire TEAMCORE crew for receiving me with open arms. Your friday's lunches, dinners and everyday discussions in the lab inspired the entire second part of this thesis. To Dr. Alessandro Farinelli, for a wonderful and productive stay in the charming and colorful Verona. For the neverending discussions at coffee time and the lovely italian lunches. Those days were an oasis of calm in-between storms.

To Meritxell Vinyals, because you laid the foundation stone of my entire career, and I will never be able to thank you enough for that. Since those first year classes until today, you have always been an inspiration. Yes we did!

Most of this dissertation has been developed at a very special place, the Artificial Intelligence Research Institute (IIIA-CSIC). The institute has become my second house, and all of you my second family. Thanks to all my fellow colleagues in this adventure. To Norman, Andrew, and Marc, wherever you are, because those were the times. To Isaac, Adrián, and Manu, because you showed me the way forward. To Pere, for your contagious calmness. To Mari, for so many little steps we took together. To Pablo, the life and soul of the party. To Toni, my brother-in-thesis, whose suffering made mine seem petty. To Jesús, because one joke a day is always cheering (even terrible ones). To Xavi, for being the nicest guy I've ever met. And to all others whom I failed to mention, sorry mates.

Thanks to the IIIA's direction team for striving to maintain such great work environment. To Montse and Ana for saving me from the madness of spanish bureaucracy. To Tito for keeping everything working against all odds. To Dani and Ángel because a smile before work is priceless, and to Isa and Anna for never complaining about my messiness.

I also want to thank all those who have supported me throughout this long journey. To my mother Conxi, for everything. You taught me to work hard and never give up. It took a while, but I got the message. To my father Xevi, because whatever the space and time between us, I know I can always count on you. To Ignasi, the voice of reason in our crazy family. To my grandma Aurora, for your wholehearted proudness on everything I do. A la meva germana Marina, perquè t'estimo molt guapa! To Anna, for all the fights we have left behind. To Helena, because you make me believe in a better world. To Pol, who showed me that no matter how lost, you will find your way if you search hard enough. To my Calella family Antònia, Carlota and Helios for your undemanding encouragement. You make me feel great even in the worst times.

To Oriol, for all the whining you have put up with during our morning teas. For counting on me and backing me up. You are my very definition of friend. To my Mataró gang Sònia, Clara, Xoán, Pere, Maria, Sílvia, David, and the kids! For your unreserved support and all the moments we shared together. I would not be me without you. To Josep and Iván, because you made home that special place where you retreat to renew your energies, and the place you leave to burn the night away. To Ricky, Iván "Asta", and all the university friends, because you made these years the best of my life.

Above all, thanks to my soulmate Eva. Thanks for picking me up when I am down. For putting up with my worst facets without complaints, and bringing out the best in me. I would be lost without you.

–Marc Pujol-Gonzalez

This work has been partially funded by the Spanish Ministry of Science and Innovation "Formación de Personal Investigador" (FPI) grant BES-2010-030466 within the RECEDIT project (TIN2009-13591-C02-02). It has also received support from the AT (CSD2007-0022), COR (TIN2012-38876-C02-01), MECER (201250E053), and DAMAS (TIN2013-45732-C4-4-P) projects.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Coordination is the process by which independent actors (agents) align their decisions to achieve greater outcomes than when acting separately. From insect colonies to human associations, coordination has naturally emerged everywhere around us in many different forms. However, the advent of autonomic computing devices with communication abilities presents an unprecedented challenge: to identify and develop mechanisms that allow agents to coordinate their actions without human supervision. Moreover, scalability is a key feature of such mechanisms in many emerging applications, including sensor networks, smart grids and emergency response [Estrin et al., 1999].

There exist multiple frameworks to handle Multi-Agent coordination in the literature, each with its own strengths and weaknesses [Tambe et al., 2005]. For instance, the Distributed POMDPs framework allows for cooperative decentralized coordination when the outcome of the agents' actions is uncertain. However, handling uncertainty explicitly leads to very complex (NEXP-Complete) problems [Bernstein et al., 2002]. Similarly, the Belief-Desire-Intentions (BDI) framework empowers agents with cognitive abilities, but requires agents with symbolic reasoning capabilities [Rao et al., 1995]. In this dissertation we center on the Distributed Constraint Optimization Problem (DCOP) framework [Modi et al., 2005]. Despite being a simpler framework than the others above, our hypothesis is that DCOPs are powerful enough to handle many application domains, and that this simplicity provides tangible efficiency benefits. In essence, the DCOP framework captures the value of local interactions between agents to subsequently maximize the social value.

A DCOP is basically a distributed Markov Random Field [Rogers et al., 2011]. Variables represent the decisions that agents must make, and edges between variables indicate that joint choices for those decisions yield different values for the system. This model captures the distributed nature of a problem because an agent only needs to know how its decisions interact with those of its direct neighbors in the graph. Then it is the task of a DCOP algorithm to find the combination of choices that maximize the value obtained from all these inter-related interactions.

This approach has two main advantages. Firstly, it can represent a vast number of coordination situations [Pecora and Cesta, 2007; Junges and Bazzan, 2008; Kim et al., 2010]. Secondly, it supports the development of generic algorithms to solve them (*e.g.*, ADOPT [Modi et al., 2005], DPOP [Petcu and Faltings, 2005b], AFB [Gershman et al., 2009], DSA [Zhang et al., 2005], MGM [Maheswaran et al., 2004a], Max-Sum [Farinelli et al., 2008]). Moreover, although DCOP algorithms may pursue the optimal solution, they may also trade-off solution quality for execution speed, providing the flexibility required to reach our goals. Hence, in the following we first introduce the reasons that lead to this work, and then present the research questions that settle these goals in Section 1.2.

1.1 Motivation

The DCOP model enables agents to make coordinated decisions in a decentralized manner. However, some may argue that the burden of decentralized solving is unnecessary: agents may send their information to a central authority who decides the actions each agent should execute and communicates those actions back. This is a sensible strategy in some situations, but has some very undesirable properties for certain applications.

Consider a number of personal assistant software agents whose task is to schedule meetings between the people they represent. These people have their own agendas and preferences, and they would rather not reveal them to each other. However, they want their software agents to decide where and when each meeting should be held. This is a canonical example application of the DCOP framework [Maheswaran et al., 2004b], and showcases one of the motivations for decentralized solving: to enable coordination without making agents reveal their private information completely. A DCOP model for this problem is to introduce one variable for each agent and meeting a user has to attend. The possible values for that variable are the possible start times of the meeting. Then we capture the relationships between these choices by introducing constraints between the variables. These constraints yield some cost when either two agents choose a different start time for the same meeting, or an agent chooses the same start time for two different meetings. A DCOP algorithm will try to find the combination of starting times that minimizes the sum of the costs among all constraints. If the cost is zero, then the algorithm has found a conflict-free schedule.

This hints the first and most studied dimension of scalability issues that DCOPs must handle: the scalability in execution time. Notice that, to find a conflict-free schedule, the DCOP algorithm may have to try all possible combinations of starting times (variable values). Nonetheless, the number of possible combinations grows exponentially on the number of meetings and the number of possible starting times. This follows from the fact that, despite being one of the least complex multi-agent coordination models, DCOP problems are shown to be NP-Hard [Modi et al., 2005].

Table 1.1 summarizes the scalability levels of some DCOP algorithms in

Execution scalability (approx. num. of agents)	Low (tens)	Mid (hundreds)	High (thousands)
Quality guarantee	Optimal	Tight bound	Loose or no bound
DCOP Algorithms	ADOPT DPOP Action-GDL	Bounded Max-Sum DaCSA p-optimality	DSA MGM Max-Sum

Table 1.1: Example DCOP algorithms and their execution scalability.

terms of solving time. Despite the intractability of DCOPs, researchers have successfully developed several complete algorithms for DCOP solving, such as ADOPT [Modi et al., 2005], DPOP [Petcu and Faltings, 2005b] and its generalization Action-GDL [Aji and McEliece, 2000; Vinyals et al., 2010b]. Optimal algorithms guarantee the maximum possible solution quality, but they can only cope with small problems of a few tens of agents at most. Another approach is to drop optimality in exchange for faster execution times, yet still provide a tight bound on the quality of the solutions. Examples of such algorithms include the Bounded Max-Sum algorithm [Rogers et al., 2011], DaCSA [Vinyals et al., 2010a], and the p-optimality framework [Okimoto et al., 2011]. The caveat of these algorithms is that, to guarantee such tight bounds, they must maintain some global information of the problem. For instance, Bounded Max-Sum includes a pre-processing step that identifies the constraints with maximum costs. In a decentralized setting, these kinds of global computations take a long time if the number of agents is large. Finally, there are also algorithms that provide approximate solutions with loose or no bounds on their quality, such as DSA [Fitzpatrick and Meertens, 2003], MGM [Maheswaran et al., 2004a] and Max-Sum [Farinelli et al., 2008]. These algorithms exclusively operate with local states, and thus can handle applications with large numbers of agents.

Although execution time scalability is one of the major concerns of DCOP algorithms, it is not the only one. To illustrate this, consider another application: the coordination of a network of autonomous sensors [Zhang et al., 2005]. In this scenario, an institution deploys a number of small, battery-operated autonomous sensors to monitor a hazardous or hard-to-reach location. The sensors should coordinate their sensing schedules to cover as much area as possible while minimizing their battery usage to extend the network's lifetime. In this case, privacy (between the sensors) is simply not an issue. However, it is still desirable to allow the sensors to coordinate in a decentralized manner to improve the robustness of the network and eliminate any single point of failure. Moreover, communication in this scenario is particularly expensive, because the sensors' radios consume more battery than their CPUs and sensing units. Therefore, a coordination mechanism for such wireless sensor network should strive to employ as little communication as possible, even at the expense of increasing computational cost.

The above example highlights that an important characteristic that determines whether a DCOP algorithm is appropriate for some application domain

Resource scalability (approx. num. of agents)	Low (tens)	Mid (hundreds)	High (thousands)
DCOP Algorithms	DPOP Action-GDL p-optimality	Bounded Max-Sum Max-Sum DaCSA	ADOPT DSA MGM

Table 1.2: Example DCOP algorithms and their resource scalability.

or not is the amount of resources that agents require to run it. Table 1.2 shows the same DCOP algorithms considered in Table 1.1, now classified regarding resource scalability. For instance, DPOP scales badly on this front because it uses a linear (in the number of variables) amount of steps to solve a problem, but each step requires an exponential amount of memory and computation. In contrast, Max-Sum typically requires more steps, but the resources required at each step are exponential only on the number of neighbors of any variable, instead of on all the problem's variables. At the further end of the scale we find algorithms that do need a constant amount of memory and computation at each step. Notice that this table classifies the algorithms regarding the amount of resources employed at each step, disregarding the number of steps they require. Hence, ADOPT (which takes an exponential number of steps) belongs to the same category than DSA and MGM (which typically need much fewer iterations), even though the cumulative amount of resources they use is widely different.

Execution time and resource requirements tell us whether an algorithm can or cannot operate on a particular application domain. Hence, we now have a some rough guidelines to pick a DCOP solving algorithm depending on the number of agents or decisions involved in our problem. Nonetheless, there is another, more subtle issue we should consider: the modeling scalability. In the two previous examples, all agents were of the same kind, and made similar decisions. Even in this relatively simple cases, a non-trivial amount of research has been devoted to the modeling of different problems [Maheswaran et al., 2004b; Pecora and Cesta, 2007; Farinelli et al., 2008; Junges and Bazzan, 2008; Kim et al., 2010]. Moreover, many situations in the actual world require heterogeneous agents to cooperate between them. Consider an emergency response scenario: a city has just suffered a major natural disaster (*e.g.*, an earthquake). As a result, some buildings collapsed, trapping civilians inside and partially blocking the streets, other buildings caught fire, and the city traffic is chaotic. In this situation, medical teams must cooperate with firefighter brigades and police patrols to rescue as many civilians as possible while keeping fires at bay. Consequently, to coordinate these heterogenous teams becomes a critical issue in such scenarios.

As explained before, one of the major advantages of the DCOP model is that it can represent a vast number of coordination situations. Hence, it is certainly possible to model a complex scenario such as the rescue one using DCOPs, as shown by Ramchurn et al. [2010a]. However, this is a non-trivial task: there are multiple ways to model a problem, and different models work better with some algorithms than others. Therefore, it is important to develop languages and methodologies that ease the development of DCOP models to increase their potential applicability. Furthermore, such languages and methodologies should allow the models to capture as much information about the problem as the designer has, and the algorithms should employ this information. Unfortunately, to the best of our knowledge, the design of methodologies and languages for DCOP modeling has received very little attention so far. Although a few works exist on this topic [Sultanik et al., 2007; Delle Fave et al., 2012a], we argue that this area requires further research.

1.2 Challenges

There are many challenges that may be tackled to improve the scalability (and thus applicability) of the DCOP framework. In this dissertation we begin by focusing on those problems where it is feasible to compute optimal solutions. Looking at Table 1.1 it may seem that all complete algorithms have the same time costs. However, each algorithm employs different techniques and heuristics, and hence their performance varies on different types of problems. As a consequence, a sensible question to ask is

Question 1. Can we identify application characteristics that provide cues as to which solving algorithm is better for that application?

For instance, based on the analysis in Section 1.1, DPOP and Action-GDL do not seem to be good algorithms, because they scale poorly both in terms of execution time and resource requirements. However, it has been shown that they actually outperform other complete algorithms on many applications [Petcu and Faltings, 2005b; Junges and Bazzan, 2008], provided the agents can cope with the resource requirements of the algorithm. Thus, we can formulate a follow-up question:

Question 2. Can we improve the resource scalability of DCOP algorithms for which this scalability is a limitation?

By answering Question 1 we can identify problems where a certain DCOP algorithm is preferable over others, up to the point where it may be the only choice to optimally solve those scenarios within their required execution time. Then, if that algorithm has certain resource limitations and we can successfully answer Question 2, we can effectively extend the range of optimally solvable problems using state-of-the-art techniques.

In the second part of the dissertation we focus on larger-scale, dynamic applications where optimal solving is not an option anymore. A typical assumption on the current DCOP literature is that problems are static. That is, that the value of the interactions between agents does not change and no decisions are introduced or removed while the agents are running the algorithm. This is a reasonable assumption on some application domains, such as the meeting scheduling described above. However, large scale coordination problems usually involve agents that can move and operate within a dynamic environment that is constantly changing. A clear example of such application is the rescue scenario. There, rescue teams have to move to accomplish their goals, and meanwhile the city's conditions keep constantly evolving. As a result, an important question to answer before dealing with scalability issues in such scenarios is:

Question 3. Can the DCOP framework handle dynamic problems? And if so, how?

Intuitively, it is reasonable to think that such dynamic settings require agents to make decisions quickly. Moreover, quality guarantees in this setting are still desirable, but much less important because the environments' evolution quickly invalidates those guarantees. Hence, the best candidates here are algorithms that scale well in terms of execution time. Ideally, we should be able to answer Question 1 and Question 2 for this kind of applications, too. There is a significant difference though: where complete algorithms compete for the best possible execution times, approximate algorithms usually take a similar amount of time, and compete on the quality of the obtained solutions instead.

This distinction is important because the quality of the solutions obtained by approximate DCOP algorithms heavily depends on how the problem is modeled. Unfortunately, it is currently hard to identify which model works best with each algorithm without actually testing them. An appealing possibility is thus to approach the situation from another angle, and asking:

Question 4. Can we develop modeling techniques that benefit certain DCOP algorithms?

This question rises awareness of the fact that modeling is a fundamental issue to consider to improve the practical ability of the DCOP framework. In addition, notice that potential DCOP applications are rapidly growing not just in the number of involved agents, but also in the heterogeneity of the decisions these agents must make. As a consequence, it is desirable to study modeling approaches that explicitly consider and exploit such heterogeneity to facilitate the designer's task. Therefore, the last major challenge considered in this dissertation is

Question 5. Can we ease the modeling of complex scenarios as DCOPs?

1.3 Contributions

In this dissertation we contribute to the challenges posed by improving DCOP algorithms to scale better, and applying them to actual-world problems. In particular, we study techniques to improve the scalability and applicability of DCOP algorithms exploiting the Generalized Distributive Law (GDL) idea [Aji and McEliece, 2000]. The GDL is a generic theoretical framework that captures the core ideas behind several DCOP algorithms [Vinyals et al., 2010b], and also from many algorithms in other research areas [Aji and McEliece, 2000].[1] The main reasons to choose the GDL framework are the following: (i) it is flexible enough to encompass both complete and approximate algorithms; (ii) it has nice theoretical properties, such as a guaranteed better worst case time than any other complete algorithm; and (iii) being such a general framework, it fosters opportunities for cross-pollination between research areas.

Due to the significantly different issues and possible applications between optimal and approximate DCOP algorithms, we divided this dissertation in two parts: a first part focused on optimal solving, and a second one concerned with approximate solving.

1.3.1 On optimal solving

In the first part of the thesis we exploit the aforementioned theoretical properties of the GDL framework to answer our Question 1 above. Specifically, most complete DCOP algorithms have a worst time complexity exponential on the number of variables of the problem. In contrast, the complexity of GDL-based algorithms is bounded by the treewidth of the problem, which is always lower than or equal to the number of variables [Dechter et al., 2001].

However, GDL-based algorithms bear large computation and communication costs [Vinyals et al., 2009]. As a result, in this part we focus on answering Question 2 in Section 1.1, regarding GDL-based algorithms on applications where the treewidth is typically low with respect to the number of variables. Against this background, we make the following contributions:

Pure algorithmic improvements

The direct approach to lessen the resource requirements of an algorithm is to devise techniques that make it more efficient. Hence, we begin by building on top of the state-of-the-art GDL with function filtering algorithm [Brito and Meseguer, 2010a]. Function filtering is a technique introduced by Sánchez et al. [2005] that can reduce the resource requirements of the classic GDL algorithm by pruning (filtering) parts of the solution space that are known to be suboptimal.

The amount of filtering (and hence the efficiency increase) depends on two measures computed during the execution of the algorithm: lower bounds on the exchanged constraints, and the best known solution so far which is used as a global upper bound. Therefore, in Chapter 3 we make improvements on both fronts. First, we introduce the so-called two-sided filtering technique, which improves the tightness of the computed lower bounds. Then, we show how better solutions can be found during the algorithm's execution by exploring multiple solutions in parallel.

[1]We note that the GDL framework was introduced after several of the algorithms we call GDL-based were already developed. Hence, we use the GDL-based qualifier to mean algorithms that can be explained through the GDL framework, disregarding their chronological development.

Related publications:

- Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., and Rodriguez-Aguilar, J. A. (2011c). Two-sided function filtering. *11th Workshop on Preferences and Soft Constraints*, pages 104–112
- Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., and Rodriguez-Aguilar, J. A. (2011b). Improving function filtering for computationally demanding dcops. *Workshop on Distributed Constraint Reasoning at IJCAI 2011*, pages 99–111

Source code available at:

- Pujol-Gonzalez, M. (2010–2014a). GDLFiltering: a GDL with function filtering DCOP solver. `http://github.com/kilburn/GDLFiltering`

Adapting the algorithm to agents' resources

A more subtle approach to increase the scalability of a distributed algorithm is to better adapt its functioning to the agents' capabilities. Hence, in Chapter 4 we focus on techniques allowing agents to tradeoff between computation and communication costs to achieve such adaptation.

With this aim, we first introduce the so-called top-down message computation approach, which is specifically designed to reduce the communication requirements of the algorithm, disregarding the computational costs. As a result, top-down approximations are good for application domains where communication is vastly more expensive than computation. For instance, the sensors of a sensor network possess low-bandwidth, high-latency radios to communicate. In contrast, their computation abilities are much better.

However, top-down approximations by themselves are an all-or-nothing approach, and do not provide us with the required flexibility. Therefore, we then present a scheme that generalizes several variants of GDL-based algorithms. This general scheme sheds some light on the differences between these methods, and which conditions fit each of them best. Moreover, it allows the system designer to tradeoff communication for computation resources, adapting it to the specific characteristics of the application domain.

Related publications:

- Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., and Rodriguez-Aguilar, J. A. (2011a). Communication-constrained dcops: Message approximation in gdl with function filtering. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 379–386. International Foundation for Autonomous Agents and Multiagent Systems

Source code available at:

- Pujol-Gonzalez, M. (2010–2014a). GDLFiltering: a GDL with function filtering DCOP solver. `http://github.com/kilburn/GDLFiltering`

1.3.2 On approximate solving

In the second part of this dissertation we focus on realistic, larger scale problems where optimal solving is not possible anymore. Unfortunately, there is a lack of realistic datasets, benchmarks and testbeds to explore DCOP-based solutions for such applications. A notable exception is the RoboCup Rescue Simulation Platform [Skinner and Ramchurn, 2010], where several teams of rescue agents must coordinate to mitigate damages after an earthquake has taken place on a large city. However, the RoboCup simulation platform represents a fairly complex problem involving heterogeneous agents with uncertain information. Considering that, we start undertaking our Question 3 on a novel, simpler yet realistic problem, and only after turning our attention to more involved applications such as the RoboCup.

Tackling dynamic problems

In Chapter 5 we first introduce the Limited-range Online Routing Problem, where a team of UAVs with limited communication range must coordinate to service tasks as requested by the human operators. Then we identify state-of-the-art approaches to deal with this problem. Furthermore, we present a full simulation toolkit to develop new solutions for the LORP, and to compare them with the readily-implemented sate-of-the-art approaches mentioned above.

Thereafter we show how such a highly-dynamic problem can be approached as a series of distributedly-built DCOPs, and then solved using a fast, approximate DCOP algorithm. Additionally, we introduce two solutions using the approximate version of GDL, commonly known as the Max-Sum algorithm. We selected Max-Sum because of its theoretical guarantees and good experimental results [Farinelli et al., 2008]. Nonetheless, the standard version of Max-Sum also suffers from some exponentiality issues. Despite that, we show that by using a clever encoding of the problem and the Tractable Higher-Order Potentials (THOPs) introduced in Tarlow et al. [2010] it is sometimes possible to overcome that limitation. Namely, we show that by designing using binary DCOPs and THOP constraints, it is possible to reduce Max-Sum's complexity from exponential to polynomial, thus answering our Question 4 above.

Related publications:

- Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., Rodríguez-Aguilar, J. A., and Tambe, M. (2013b). Engineering the decentralized coordination of UAVs with limited communication range. In *Advances in Artificial Intelligence - 15th Conference of the Spanish Association for Artificial Intelligence, CAEPIA*, volume 8109 of *Lecture Notes in Computer Science*, pages 199–208. Springer

- Pujol-Gonzalez, M., Cerquides, J., and Meseguer, P. (2014b). MASPlanes: A multi-agent simulation environment to investigate decentralised coordination for teams of UAVs (demonstration). In *The 13th International*

Conference on Autonomous Agents and Multiagent Systems, pages 1695–1696. International Foundation for Autonomous Agents and Multiagent Systems

Source code available at:

- Pujol-Gonzalez, M. (2013–2014b). MASPlanes simulator for the development of distributed coordination algorithms. `https://github.com/MASPlanes/MASPlanes`
- Pujol-Gonzalez, M. and Penya-Alba, T. (2013–2014). Binary max-sum java library. `http://binarymaxsum.github.io/`

Tackling design complexity

The above work highlights the impact modeling can have when dealing with highly dynamic environments. As a consequence, in Chapter 6 we focus on answering Question 5 above. To this end, we present a new methodology to model problems where heterogeneous teams of agents must cooperate in non-trivial ways to achieve a common objective. Our methodology exploits the additive nature of DCOPs to break the design task into smaller and simpler composable sub-tasks. Essentially, each sub-task models a different functional area of the problem (*e.g.,* a team in the RoboCup example).

Using this methodology we introduce an inter-team coordination model for the RoboCup Rescue Simulation scenario. Furthermore, we show that the resulting model can also be encoded as a binary DCOP using only THOP constraints. As a consequence, we are able to employ Max-Sum on the full inter-team model to allow agents to reach joint decisions within the constraints of this demanding domain.

Related publications:

- Pujol-Gonzalez, M., Cerquides, J., Farinelli, A., Meseguer, P., and Rodriguez-Aguilar, J. A. (2014a). Binary max-sum for multi-team task allocation in robocup rescue. *International Joint Workshop on Optimisation in Multi-Agent Systems and Distributed Constraint Reasoning (OptMAS-DCR)*
- Pujol-Gonzalez, M., Cerquides, J., Escalada-Imaz, G., Meseguer, P., and Rodriguez-Aguilar, J. A. (2013a). On binary max-sum and tractable hops. *11th European Workshop on Multi-agent Systems (EUMAS 2013)*, 1113

Source code available at:

- Pujol-Gonzalez, M., Kleiner, A., Farinelli, A., Ramchurn, S., Shi, B., Maffioletti, F., and Reffato, R. (2012–2014c). RMASBench: Multi-agent coordination benchmark. `https://github.com/MASPlanes/MASPlanes`

In addition, we have submitted two journal papers from the material presented in Chapters 4 and 5 respectively, which are currently undergoing the review process.

1.4 Guide to the thesis

The remainder of this dissertation is organized as follows.

Chapter 2. We first formalize the DCOP model and introduce some definitions and notation we use throughout the thesis. Then we present an example application, we show how it can be modeled as a DCOP, and we present a few different DCOP visualizations used elsewhere in the dissertation. Thereafter, we contextualize our work within the state-of-the-art literature, and discuss different existing approaches to optimal and approximate DCOP solving. In this chapter we also introduce the General Distributive Law and explain how it can be employed to develop both optimal and approximate DCOP algorithms.

Chapter 3. We introduce the GDL with function filtering algorithm for optimal DCOP solving, and show how two-sided filtering improves the computation of lower bounds during the algorithm's execution. Then we focus on the computation of upper bounds within the algorithm, introducing a scheme to compute multiple upper bounds simultaneously. Finally, we present a couple of techniques to exploit such scheme and empirically evaluate all these improvements.

Chapter 4. We describe the top-down approximations technique to compute GDL messages in communication-constrained scenarios. After that, we present our general framework for message computation, that allow us to trade-off computation and communication requirements of the algorithm.

Chapter 5. We present the LORP and the MASPlanes simulation environment for the development and testing of LORP solution techniques. Thereafter we show how to model the LORP problem as a sequence of DCOP algorithms, and introduce two LORP solutions using the Max-Sum algorithm. Additionally, we introduce the THOP constraints and show how these can help at scaling the Max-Sum algorithm.

Chapter 6. We explain our methodology to model complex problems with heterogeneous agent teams, and provide an example by modeling the RoboCup Rescue Simulation challenge problem.

Chapter 7. We conclude this dissertation by providing a summary of the presented contributions, drawing the most notable conclusions derived from this work, and outlining possible directions for future research.

Chapter 2

Background and related work

In this chapter we provide the necessary context to position this dissertation within the extensive literature about DCOPs. With this aim, we first explain the DCOP model and the GDL framework, as well as some definitions required to understand it. Thereafter we examine the most prominent DCOP solving approaches and algorithms, with special emphasis on their scalability characteristics. Ultimately, this survey allows us to highlight the necessity and relevance of the techniques developed throughout this thesis.

2.1 Background

In Section 2.1 we introduce the Distributed Constraint Optimization Problem (DCOP) framework for multi-agent coordination. We begin by formally defining what a DCOP is and introducing some notation that we will use across the dissertation. Thereafter, we present an example application, show how it can be modeled as a DCOP, and introduce a number of different representations commonly used to represent DCOP models. Finally, we introduce the Generalized Distributive Law framework and explain why it is our foundation of choice to build DCOP algorithms upon.

2.1.1 The DCOP model

A Distributed Constraint Optimization Problem models a number of agents that must make some decisions. These agents have a finite number of possible choices for each decision. Then, the synergies between decisions are captured using constraints: functions that specify a different value for each combination of choices of a subset of the problem's decisions. An assumption of the DCOP framework is that the global utility of the system is simply the sum of the utilities obtained from each constraint. Hence, the task of a DCOP algorithm is to find the choice for each decision that maximizes this global utility.

In the following we introduce a formal definition of a DCOP, as well as some further definitions that we require to formalize the objective of a DCOP algorithm.

Definition 2.1. A DCOP problem is a tuple $\Omega = \langle A, X, D, C, \mathfrak{m} \rangle$, where:

- A is the set of agents involved in this problem.
- X is a set of discrete variables that represent all choices to make. Each variable $x \in X$ has a finite domain D_x, which is the set of possible choices for that decision.
- $D = \{D_x \mid x \in X\}$ is the set of all domains of all variables. That is, the set that contains all possible choices for each decision in this problem.
- C is the set of constraints. Each constraint $f \in C$ is a function defined over a subset $S \subseteq X$ of the problem's variables, and specifies a utility for each combination of choices for these variables. Namely, $f : \prod_{x \in S} D_x \rightarrow \mathbb{R} \cup \{-\infty\}$, where $\prod$ is the set cartesian product. The set S is also known as the scope of the constraint, namely $S = sc(f)$.
- $\mathfrak{m} : X \rightarrow A$ is a function mapping each variable (decision) to an agent (that must make that decision).

We require some further definitions prior to the introduction of the notion of solution of a DCOP.

Definition 2.2. Given a set of variables $S \subseteq X$, the *assignment space* of S, noted as $\mathbf{S}$, is the set of all possible combinations of values for the variables in S. That is, $\mathbf{S} = \prod_{x \in S} D_x$.

Assignment spaces capture all possible combinations of choices that the agents may make for a subset of the problem's decisions.

Definition 2.3. An *assignment* of a set of variables $S \subseteq X$, noted as $\mathbf{s} \in \mathbf{S}$, maps each variable $y \in S$ to some value in its domain.

For instance, the assignment $\mathbf{s} = \langle x = a, y = b \rangle$ maps (assigns) value a to variable x and value b to variable y. In other words, an assignment $\mathbf{s}$ is a tuple of specific choices for the decisions in S. When an assignment specifies a choice for all of the decisions of the problem, we say that it is a complete assignment. Namely,

Definition 2.4. A *complete assignment* of a DCOP is an assignment $\mathbf{x}$ that assigns a value to each variable in X.

In some situations it is useful to consider only a subset of the choices represented in an assignment. Hence, we define the projection of an assignment as an operation that defines a new assignment, but containing only the subset of choices related to specific decisions:

Definition 2.5. The *projection of an assignment* $\mathbf{s}$ *over* a set of variables $T \subseteq S$, written $\mathbf{s}[T]$, is a new assignment $\mathbf{t}$ formed by the values that $\mathbf{s}$ assigns to the variables in T.[1]

With these definitions in place, we can readily introduce the DCOP model as an optimization problem whose objective is to maximize the utility obtained from the sum of all constraints. Formally,

Definition 2.6. The *objective function* of a DCOP $\Omega = \langle A, X, D, C, \mathfrak{m} \rangle$ is the function that captures the aggregated utility of a complete assignment $\mathbf{x}$, namely

$$\sum_{f \in C} f(\mathbf{x}[sc(f)]). \tag{2.1}$$

Moreover, DCOP algorithms must function in a distributed manner because the problem's information is assumed to be spread between the agents. Additionally, we make the following common assumption:

Assumption 2.1. An agent knows about all constraints involving its own decisions. Namely, an agent a only knows about the set of constraints C_a, where

$$C_a = \{f \in C \mid \exists x \in sc(f) \text{ such that } \mathfrak{m}(x) = a\}.$$

Furthermore, all agents involved in C_a (except a itself) form the neighbors of a. In other words,

Definition 2.7. The *neighbors* of an agent $a \in A$, written $nb(a)$, is the set of agents with which a shares some constraint. Formally,

$$nb(a) = \{a' \in A \mid \exists f \text{ such that } f \in C_a \text{ and } f \in C_{a'}\}.$$

Also, we make the following assumption regarding the communication abilities of agents in a DCOP:

Assumption 2.2. An agent a can only communicate directly with its neighbors $nb(a)$.

Consequently, a DCOP algorithm is a distributed algorithm that aims to find a complete assignment that maximizes the DCOP's objective function in Equation (2.1). While this is the goal of any DCOP algorithm, there are a few discrepancies regarding what constitutes a DCOP solution. Research on complete DCOP algorithms considers that a solution is a complete assignment whose utility is not $-\infty$, meaning that no hard constraint is violated. In contrast, research on approximate DCOP algorithms considers any complete assignment to be a solution, because approximate algorithms generally cannot guarantee that no hard constraint is violated. In any case,

[1] In an abuse of notation, we write $\mathbf{s}[x]$ to mean the projection of assignment $\mathbf{s}$ to a set containing the single variable x, namely $\mathbf{s}[\{x\}]$.

Definition 2.8. An *optimal solution* $\mathbf{x}^*$ is any complete assignment that maximizes the DCOP's objective function in Equation (2.1).

This concludes the basic definitions employed by the DCOP framework, but it may still be a bit unclear how to actually model situations using it. Hence, the following Section introduces a few different applications and shows how they can be modeled as DCOPs.

2.1.2 Example application: meeting scheduling

The DCOP model has already been used to represent a wide range of application domains. Examples include meeting scheduling [Maheswaran et al., 2004b], sensor networks [Ali et al., 2005], traffic light control [Junges and Bazzan, 2008], and radar coordination [Kim et al., 2010] among others. In this Section we first introduce the meeting scheduling problem and then show how it can be tackled using the DCOP framework.

Arranging meetings with other people is a naturally distributed, very time consuming task that we constantly perform. In essence, the process involves matching our personal agendas with those of the people we need or want to meet. In an actual-world scenario it would be interesting to consider additional features (e.g., restrictions about the meeting places), but we leave those out of our example to keep it simple. The main problems with manual meeting arrangements are that: (i) the communication latency is huge (receiving a response from another person can take from seconds to days); (ii) our agendas change often; and (iii) the whole process is very time-consuming for everyone involved.

As a result, we have decided to automate this task using software assistants. Each person on the system has her own software assistant that knows the user's preferences and manages her agenda. Now, when we want to schedule a new meeting, we just need to notify our assistant. Then, our assistant will arrange the meeting with the assistants of other attendants, possibly rescheduling other meetings in the process. Hence, we require some model that allows these software assistants to represent their users' preferences and to compute meeting schedules. Hereafter we present an example problem, and define a DCOP model that encodes the situation as an optimization problem.

Example 2.1. In our example scenario, three users u_1, u_2 and u_3 need to arrange two meetings μ_1 and μ_2. Figure 2.1 shows that only u_2 and u_3 should attend μ_1, whereas u_3 and u_1 should attend μ_2. Moreover, each meeting can be scheduled at either 8 am or 9 am, and emoticons represent each user's preferences for each time slot. The goal is to decide the time at which each meeting should be scheduled to maximize users' preferences.

Definition 2.9. The scenario in Example 2.1 can be formalized as a DCOP $\Omega = \langle A, X, D, C, \mathfrak{m} \rangle$, where:

- $A = \{u_1, u_2, u_3\}$ is the set of agents in the scenario.

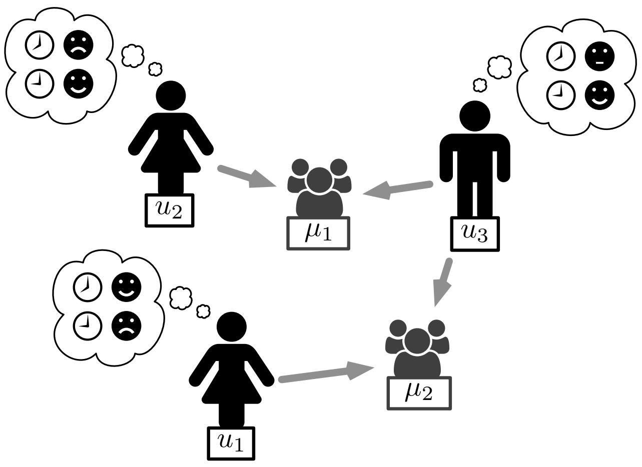


Figure 2.1: Example of a meeting scheduling scenario.

- There is one variable $x_{am} \in X$ for each user $u_a \in A$ and each of the meetings μ_m that user has to attend. That is, each variable x_{am} represents the time at which user u_a decides to attend to meeting μ_m.
- The domain of each variable $D_{x_{am}} \in D$ is the set of available time slots for the meetings, i.e., $\{8\,\text{am}, 9\,\text{am}\}$.
- There are three types of constraints in C:
  1. First, constraints specifying the user's preferences. Notice that our constraints must yield real values, so we assign the utilities ☻=1, 😐=0, and ☹=−1 to the preferences in Figure 2.1. Now we can introduce on preference constraint for each user and meeting she has to attend. In this case the constraints are

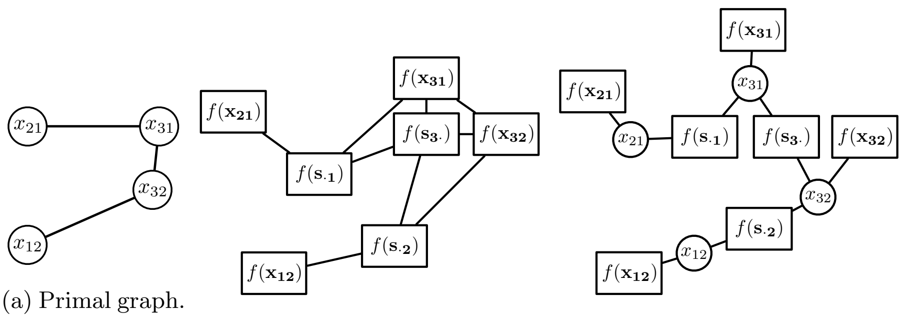
$$f(\mathbf{x_{21}}) = \begin{cases} -1 & \text{if } \mathbf{x_{21}}{=}8\,\text{am} \\ 1 & \text{if } \mathbf{x_{21}}{=}9\,\text{am} \end{cases}, \qquad f(\mathbf{x_{31}}) = \begin{cases} 0 & \text{if } \mathbf{x_{31}}{=}8\,\text{am} \\ 1 & \text{if } \mathbf{x_{31}}{=}9\,\text{am} \end{cases},$$

$$f(\mathbf{x_{12}}) = \begin{cases} 1 & \text{if } \mathbf{x_{12}}{=}8\,\text{am} \\ -1 & \text{if } \mathbf{x_{12}}{=}9\,\text{am} \end{cases}, and\ f(\mathbf{x_{32}}) = \begin{cases} 0 & \text{if } \mathbf{x_{32}}{=}8\,\text{am} \\ 1 & \text{if } \mathbf{x_{32}}{=}9\,\text{am} \end{cases}.$$

  2. Second, constraints enforcing that all the participants of a meeting must schedule it at the same time. Hence, there is one constraint f with scope $S_{\cdot m}$ for each meeting, defined as

$$f(\mathbf{s_{\cdot m}}) = \begin{cases} 0 & \text{if all } \mathbf{x_{am}} \in \mathbf{s_{\cdot m}} \text{ are equal} \\ -\infty & \text{otherwise} \end{cases},$$

     where $S_{\cdot m}$ is the set of variables of the meeting's participants, namely $S_{\cdot m} = \{x_{am'} \in X \mid m' = m\}$.



(a) Primal graph.

(b) Dual graph.

(c) Factor graph.

Figure 2.2: Graphical representations of the meeting scheduling DCOP.

3. Third, constraints ensuring that an agent cannot attend multiple meetings at the same time. Thus, there is one constraint of this type for each agent a, with scope $S_{a\cdot}$, defined as

$$f(\mathbf{s_{a\cdot}}) = \begin{cases} 0 & \text{if all } \mathbf{x_{am}} \in \mathbf{s_{a\cdot}} \text{ are different} \\ -\infty & \text{otherwise} \end{cases},$$

where $S_{a\cdot} = \{x_{a'm} \in X \mid a' = a\}$ is the set of all variables of the same agent.

- Finally, $\mathfrak{m}$ simply maps each variable x_{am} to agent a.

With this definition the DCOP objective is aligned with the goal in our example application. Hence, the optimal solution

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \sum_{f \in C} f(\mathbf{x}[sc(f)]).$$

with value $\mathbf{x}^* = \langle x_{12}{=}8\,\text{am}, x_{21}{=}9\,\text{am}, x_{31}{=}9\,\text{am}, x_{32}{=}8\,\text{am}\rangle$ corresponds to the best possible schedule, which is to conduct meeting μ_1 at 9 am and meeting μ_2 at 8 am, with a global utility of 3.

2.1.3 DCOP representations

In the previous section we presented a mathematical description of the DCOP model for our example meeting scheduling problem. However, some may argue that it is easier to grasp the agent relationships in the graphical representation of Figure 2.1 than in our mathematical DCOP model. Because such interactions form the basis of the DCOP model, multiple alternative visualizations have been proposed in the literature. In this section we quickly review these graphical representations, encoding our example problem in each one to highlight their major advantages and caveats.

Primal graph

One of the most common graphical representations of DCOP models is the primal graph, which focuses on providing a clean representation of the dependencies between decisions (variables). Formally,

Definition 2.10. The *primal graph* of a DCOP $\Omega = \langle A, X, D, C, \mathfrak{m} \rangle$ is a graph $G_p = \langle V_p, E_p \rangle$ that contains one vertex $v_i \in V_p$ for each variable $x_i \in X$ and one edge $e_{ij} \in E_p$ between every two nodes v_i and v_j whose variables appear together in a constraint. That is,

$$e_{ij} \in E_p \text{ if and only if } \exists f \in C \text{ such that } x_i \in f \text{ and } x_j \in f.$$

Figure 2.2a shows the primal graph of our meeting scheduling DCOP model. As expected, the result is a very simple graph where dependencies between meeting attendance times are extremely easy to visualize. For instance, we immediately see that x_{21} (the time at which user u_2 decides to attend meeting μ_1) depends on x_{31} (the time at which user u_3 decides to attend meeting μ_1). Moreover, because $\mathfrak{m}$ maps each variable to some agent, this graph also represents the direct communication links between agents.

Nonetheless, this simplistic visualization cannot represent certain information about the nature of these interactions. In this simple example this is not a problem. However, say that the graph contained an additional edge between the x_{21} and x_{32} nodes. In such case, we would not be able to tell whether the problem has a single interaction between x_{21}, x_{32} and x_{31}, or three separate pairwise interactions between these decisions. This difference is a minor issue when trying to understand a model, but can make an important difference for solving algorithms.

Dual graph

For some solving algorithms, the most important part of the problem are the constraints and how they depend on each other. Hence, a better visualization for these algorithms is the dual graph, where the nodes are interactions (constraints), and links between nodes indicate that these interactions depend on at least one shared decision (variable). Formally,

Definition 2.11. The *dual graph* of a DCOP $\Omega = \langle A, X, D, C, \mathfrak{m} \rangle$ is a graph $G_d = \langle V_d, E_d \rangle$ that contains one vertex $v_i \in V_d$ for each constraint $f_i \in C$ and one edge $e_{ij} \in E_d$ between every two nodes v_i and v_j whose constraint contains the same variable. That is,

$$e_{ij} \in E_d \text{ if and only if } \exists x \in sc(f_i) \text{ such that } x \in sc(f_j).$$

The dual graph of our meeting scheduling example is represented in Figure 2.2b. This representation is less intuitive than the primal graph, but is better at characterizing the objective function of our model. Hence, it is primarily used to asses the complexity of computing solutions for a problem, or as a foundation to distribute the computation load between agents.

Factor graph

Another common visualization is the factor graph, which seeks to provide all the information of both the primal and the dual graphs in a single view. With this aim, it represents the problem as a bipartite graph with two types of nodes: variable nodes, usually represented as circles; and constraint nodes, represented as rectangles. Then, an edge is added between every variable node and constraint node such that the constraint depends on that variable.

Definition 2.12. The *factor graph* of a DCOP $\Omega = \langle A, X, D, C, \mathfrak{m} \rangle$ is a bipartite graph $G_f = \langle (X, F), E_f \rangle$ that contains one variable node for each variable $x_i \in X$ and one constraint node for each constraint $f_i \in C$. The graph contains an edge $e_{ij} \in E_f$ for each pair of variable and constraint nodes such that the constraint depends on that variable. Formally,

$$e_{ij} \in E_f \text{ if and only if } x_i \in sc(f_j).$$

The caveat of this visualization is that, as shown in Figure 2.2c, the resulting graph is larger than when using other representations. However, the fact that it accurately represents all interactions and dependencies of the model makes it a very compelling choice to develop and analyze DCOP models for actual-world applications.

2.1.4 The Generalized Distributive Law

This Section introduces the Generalized Distributive Law (GDL) as described by [Aji and McEliece, 2000], which is the foundation upon which most of the algorithms developed in this dissertation build upon. In the following we only describe briefly the main idea exploited by the large family of GDL-based algorithms. The particular details of different realizations of this basic idea will be studied in depth in the corresponding chapters.

In mathematics, the Distributive Law is a widely known law relating the operations of multiplication and addition. In its simplest form, the law is typically stated as

$$ab + ac = a(b + c) \ . \tag{2.2}$$

A first insight on why the Distributive Law is important for computer science in general is that it provides the means for saving computation. That is, the results of the operations on both sides of the equality are exactly the same. However, we require three operations to compute the left-hand side (two multiplications and one addition), whereas the right-hand side requires only two operations (one addition and one multiplication).

The second key insight is that the gains can also be obtained when operating functions defined over discrete variables, such as the constraints of a DCOP. To illustrate this, consider a DCOP with constraints $f_1(\mathbf{x_1})$ and $f_2(\mathbf{x_1}, \mathbf{x_2})$, whose variables all have d possible values. Thus, the domain of the constraint $f_1(\mathbf{x_1})$, defined over a single variable, has size $|D_{x_1}| = d$. Analogously, the domain size of constraint $f_2(\mathbf{x_1}, \mathbf{x_2})$ defined over two variables, is $|D_{x_1} \times D_{x_2}| = d \times d = d^2$. The objective function of such DCOP is

$$\max_{x_1, x_2} f_1(\mathbf{x_1}) + f_2(\mathbf{x_1}, \mathbf{x_2}) \ .$$

To make our point clearer, this objective function can be rewritten as

$$\max_{x_1} \max_{x_2} \left[f_1(\mathbf{x_1}) + f_2(\mathbf{x_1}, \mathbf{x_2}) \right] \ . \tag{2.3}$$

Now, because the maximization (max) is distributive over the sum (+), we can apply the generalized distributive law to obtain

$$\max_{x_1} \left[\max_{x_2} f_1(\mathbf{x_1}) + \max_{x_2} f_2(\mathbf{x_1}, \mathbf{x_2}) \right] \ ,$$

and since f_1 does not depend on x_2, this is equivalent to

$$\max_{x_1} \left[f_1(\mathbf{x_1}) + \max_{x_2} f_2(\mathbf{x_1}, \mathbf{x_2}) \right] \ . \tag{2.4}$$

To proceed, we require the following facts, which are explained in detail in Chapter 3:

- We can *sum* two constraints f_i and f_j, obtaining a new constraint f_k whose scope is the union of the scopes of f_i and f_j. The cost of this computation is the product of the domain sizes in the resulting constraint's scope. For instance, computing $f_1(\mathbf{x_1}) + f_1(\mathbf{x_1})$ yields a new constraint $f_3(\mathbf{x_1})$ with cost d, whereas computing $f_1(\mathbf{x_1}) + f_2(\mathbf{x_1}, \mathbf{x_2})$ yields a new constraint $f_4(\mathbf{x_1}, \mathbf{x_2})$ with cost d^2.

- We can *maximize* a constraint f_i over a subset $S \subseteq sc(f_i)$ of its variables, obtaining a new constraint defined over the remaining variables. This operation's cost is equal to f_i's domain size. For example, we can compute $\max_{x_1} f_2(\mathbf{x_1}, \mathbf{x_2})$ to obtain $f_4(\mathbf{x_2})$ with cost d^2. When we maximize a constraint over all its variables, the result is simply the maximum value of the constraint.

Using these facts, we can now assess that the time it takes to compute our objective function using Equation (2.3) is $2d^2$ (d^2 for the sum plus d^2 for the max). In contrast, if we compute using Equation (2.4) the cost is $d^2 + 2d$ (d^2 for the right-hand max, plus d for the sum, plus d for the final max), which is a lower cost for any domain size $d > 2$. The savings in this example are modest because the problem is very simple. However, these savings can become dramatic on more involved problems.

Aji and McEliece [2000] also introduce the so-called GDL algorithm, a message passing algorithm to compute arbitrary functions defined over any semiring. In its purest form, the algorithm expects a tree as an input, where nodes represent combinations of functions, and edges represent projections. The theoretical properties of the algorithm only guarantee a correct solution if the input is a (special kind of) tree. However, there exists extensive literature showing that, in practice, the algorithm provides good approximate solutions when repeatedly iterated on arbitrary input graphs [Kschischang et al., 2001]. Hence, the GDL algorithm can be used as a basis for both complete and approximate algorithms.

One of the most appealing points of the GDL framework is that it is not defined for some particular model such as DCOPs. Instead, it is defined over semi-rings, which are abstract mathematical structures defined by a set of possible values and two operations. For instance, here we employed a semi-ring whose possible values are the real numbers plus $-\infty$, and whose operations are the *sum* and *max*. By using other semi-rings, the same techniques can be employed to solve a wide range of problems. In fact, the GDL algorithm has been shown to generalize algorithms from many research areas [Aji and McEliece, 2000], including signal processing (e.g., Fast Fourier Transform on finite Abelian groups), information theory (e.g., the Viterbi and BCJR decoding algorithms for convolutional codes), and artificial intelligence (e.g. discrete-state Kalman filtering, belief propagation) among others. As a consequence, algorithms and techniques based on the GDL create a huge opportunity for cross-pollination between these areas.

At this point it should be clear that the GDL framework is an appealing foundation to build DCOP algorithms upon. Nonetheless, in DCOP solving there are certain issues that are not addressed by the GDL algorithm, and hence must be addressed specifically. First, notice that the GDL algorithm works over a tree or graph which is assumed to be known before the algorithm's execution. Thus, GDL-based DCOP algorithms must provide techniques to build such graph from the DCOP definition. Second, DCOP algorithms must operate in a decentralized manner. Being a message passing algorithm, the GDL algorithm is relatively easy to distribute. However, DCOPs define a communication topology that must be respected, and hence the allocation of GDL nodes to agents is also an important concern to address. Third, the GDL algorithm allows us to compute the *value* of a discrete function, whereas the DCOP objective is to compute the *assignment* that yields this value. Therefore, GDL-based DCOP algorithms must also devise techniques to obtain assignments instead of just values.

The core of this dissertation can be seen as a thorough exploration of different approaches to deal with these issues. Particularly, we first study how the current approaches affect the scalability of the resulting algorithms. Thereafter, we propose improvements and new techniques that improve the scalability of both optimal (Chapters 3 and 4) and approximate (Chapters 5 and 6) GDL-based DCOP solving.

		Type		
		Complete	Approximate	
			Global state	Local state
Solving approach	Search	SyncBB AFB ADOPT BnB-ADOPT ADOPT(k) DJAO		DSA MGM C-DALO
	Inference	(*-)DPOP Action-GDL Filtered DPOP	DaC Bounded Max-Sum p-optimality	Max-Sum
	Sampling		DUCT D-Gibbs	

Table 2.1: Classification of DCOP algorithms.

2.2 Related work

In the background section of this chapter we have introduced the DCOP and GDL frameworks, and explained how they can help us achieve multi-agent coordination with a simple example. However, we have not yet discussed the extensive literature about DCOP solving, and how this dissertation relates to all this body of work.

To address this issue, Table 2.1 presents most existing DCOP algorithms classified by type (columns) and solving approach (rows). Traditionally, these algorithms are considered as either complete or approximate: complete algorithms are guaranteed to find an optimal solution whereas approximate algorithms are not. In this dissertation we go one step further and divide the approximate algorithms into two additional categories: global-state and local-state algorithms. Global state algorithms require some kind of global computation. For instance, the algorithm may need to compute the total utility value of a configuration, which can only be achieved by involving all the problem's agents. In contrast, local-state algorithms completely avoid such computations, and operate exclusively using a local view of the problem.

In the remainder of this section we review the algorithms introduced in Table 2.1 by going through the different solving approaches, briefly explaining how they operate, and identifying their strengths and weaknesses.

2.2.1 Search-based algorithms

As the name implies, the basic idea of search-based DCOP algorithms is to perform a search over the assignment space to identify an assignment with optimum utility. Moreover, this search must be carried in a distributed manner, because this a basic requirement of the DCOP framework. Also, recall that the assignment space of a DCOP is exponential on its number of variables, so brute-force search methods do not scale beyond very small numbers of agents. This shortcoming can be addressed either by using exhaustive search with clever heuristics and pruning techniques (leading to complete and global-state algorithms), or by performing non-exhaustive local search (leading to local-state algorithms).

Complete and global-state search algorithms

The first class of search algorithms are those that employ exhaustive search techniques, meaning that they orderly check the assignment space and can thus explore (or discard) it entirely. Notice that most algorithms in this class can be seen as both complete and global-state approximate algorithms: complete because they are guaranteed to find an optimal solution given enough time; and global-state approximate because they can be easily modified to maintain the best solution found so far during the search. Then we could potentially stop them at any time and use that solution.

We can differentiate algorithms of this type by looking at four main characteristics: (i) the method used to build the search tree; (ii) the operation mode (synchronous or asynchronous); (iii) the search strategy; and (iv) the heuristics employed to prune the search space.

One of the first presented DCOP algorithms is the Synchronous Branch and Bound (SBB) algorithm [Hirayama and Yokoo, 1997]. As its name implies, SBB is a synchronous algorithm. The search tree is actually a search chain defined by an ordering submitted to the algorithm. Then, SBB performs a depth-first search strategy which backtracks whenever a partial solution is found to have larger cost than that of a known complete assignment. That is, the algorithm keeps accumulating the known cost of the current partial assignment, and backtracks whenever that cost is already larger than the cost of a known solution.

The most famous among search-based DCOP algorithms is the Asynchronous Distributed OPTimization (ADOPT) algorithm [Modi et al., 2005]. In its original form, ADOPT is essentially a distributed and asynchronous implementation of the classic A* algorithm. First, the DCOP's variables are arranged in a Depth-First Search (DFS) tree, also known as *pseudotree*, which defines parent-child relationships between them. Next, ADOPT performs a best first search along this tree, changing the exploration path whenever a possibly better alternative is detected or a path is identified as sub-optimal. Later on, several improvements of the original ADOPT algorithm have been proposed. Some works focused on techniques to compute better DFS trees (e.g., ADOPT-ng [Silaghi and Yokoo, 2006] and ADOPT-ing [Silaghi and Yokoo, 2007]), while others focused on constraint preprocessing techniques [Ali et al., 2005]. One of the most notable ADOPT spin-offs is the Branch and Bound ADOPT (BnB-ADOPT) algorithm [Yeoh et al., 2008], which uses the same infrastructure than ADOPT but switches the search strategy from best-first to depth-first search. BnB-ADOPT has also seen a number of improvements over time. For instance, Yeoh et al. [2009] introduced several caching techniques to speed-up the algorithm, and Gutierrez and

Meseguer [2010]; Gutierrez et al. [2013] combined it with soft arc consistency to achieve large reductions in the number of exchanged messages. Later on, these two algorithms have been generalized in the ADOPT(k) algorithm [Gutierrez et al., 2011], whose k parameter serves as a gauge between best-first and depth-first search strategies. Finally, the very recent DJAO algorithm [Kim and Lesser, 2014] is essentially the ADOPT(k) algorithm, but adapted to operate over a different kind of trees known as AND-OR trees. This enables DJAO to employ heuristics computed by an approximate global state inference algorithm, significantly increasing its pruning capabilities.

Another significant algorithm is the Asynchronous Forward Bounding (AFB) algorithm [Gershman et al., 2009]. Despite its name, the major insight of this algorithm is that partial synchronization can be beneficial. That is, whereas BnB-ADOPT operates in a fully asynchronous manner, AFB uses a mixed strategy: it explores assignments synchronously, but computes their bounds asynchronously.

Local-state search algorithms

All the search algorithms mentioned above employ exhaustive search techniques, which have the advantage of providing quality guarantees but must operate within a global view of the problem. In this subsection we present some non-exhaustive search algorithms for DCOPs, which can operate on entirely local views of the problem and can find good assignments quickly, but cannot make strong guarantees about the quality of their solutions. Essentially, algorithms of this family are iterative algorithms where, at each iteration, the agents have the opportunity to update their decisions depending on what their neighboring agents choices are.

An excellent framework for the understanding of local-state search algorithms is introduced by Chapman et al. [2011]. In that work, the authors identify three issues that these algorithms must handle, as well as the different strategies available to tackle them. In the following we briefly describe these issues and how different algorithms solve them.

First, a local-state search algorithm must define its *target function*, which is the local function each agent uses to assess its individual utility given the neighbors' choices. In the DCOP context, these algorithms usually employ one of the following functions:

- The *immediate payoff* function, which is simply the sum of utilities obtained from all constraints that involve the agent.
- The *fictious play* function, that maintains (and updates) an estimation of the utility of each choice throughout the algorithm's iterations.
- The *joint strategy fictious play*, which is similar to the fictious play function, but computes the estimations conditioned on the neighbors' choices.

Second, given a target function, the agents must decide which of their choices to pick next. This is defined by a *decision rule*, which is typically one of the following ones:

	Target function	Decision rule	Adjustment schedule
DSA	Immediate payoff	argmax	Parallel random (p)
MGM	Immediate payoff	argmax	Maximum gain
BR-I	Immediate payoff	Gain	Parallel random (p)
SAP	Immediate payoff	Logistic	Sequential random
FP	Fictious play	argmax	Flood
smFP	Fictious play	Logistic	Flood
JSFP-I	Joint strategy FP	argmax	Parallel random (p)
WRM-I	Immediate payoff	Logistic	Parallel random (p)

Table 2.2: Overview of local search DCOP algorithms.

- The *argmax* rule, where each agent simply picks the choice that maximizes its target function.
- The *gain* rule, where each agent selects any of the choices that improves its value with equal probability.
- The *logistic* rule, where each agent picks a choice randomly, but the probability of picking each choice is weighted by the value it provides.

Third, an algorithm must define which agents are allowed to change their choices at each iteration. This is known as the *adjustment schedule*, and admits different strategies such as:

- The *flood* strategy, that allows all agents to update their choices at each iteration.
- The *parallel random (p)* strategy, that allows each agent to change with probability p at each iteration.
- The *sequential random* strategy, that only allows a single (randomly chosen) agent to update its choices at each iteration.
- The *maximum gain* strategy, that operates in two rounds: first, each agent computes the gain (increase in utility) it would obtain if it is allowed to change, and communicates this information to its neighbors; next, an agent is only allowed to pick that choice if its gain is larger than the gains of all its neighbors.

Using these characteristics, Table 2.2 presents an overview of the landscape of local-search algorithms in the literature. Despite their differences, most of these algorithms perform similarly, and no particular algorithm has been identified as better than the others, even for particular classes of problems [Chapman et al., 2011].

Nonetheless, an algorithm that deserves a special mention is the Distributed Stochastic Algorithm (DSA) [Fitzpatrick and Meertens, 2003; Zhang et al., 2005]. This was the very first local-search DCOP algorithm presented, yet it has been shown to quickly attain notably good solutions [Maheswaran et al., 2004a; Katagishi and Pearce, 2007; Chapman et al., 2011], and is guaranteed to converge to a Pareto optimal solution. Intuitively, the DSA algorithm implements a parallel version of a simple hill-climbing algorithm. That is, at each iteration the agents simply pick the choices that maximize the utility they obtain, given the choices made by the other agents in the previous iteration. However, allowing all agents to change at every iteration leads to a very chaotic and unstable algorithm. DSA addresses this issue by using the parallel random (p) adjustment schedule. First, the system designer sets a *probability of changing* parameter p. Then, at each iteration, each agent generates a random value and only tries to improve if that value is lower than p. This technique allows the system designer to gauge the amount of parallelism that best fits the application at hand.

Later on, Maheswaran et al. [2004a] introduced the Maximum Gain Message (MGM) algorithm, that uses the same target function and decision rule than DSA but a different adjustment schedule. Instead of using a stochastic parameter, MGM restricts the parallelism by using the maximum gain strategy presented above. Although MGM does not generally improve on DSA's results [Maheswaran et al., 2004a; Vinyals et al., 2010a], it is significant because it guarantees that the global utility increases monotonically during the algorithm's operation.

Furthermore, the idea of maintaining a monotonically increasing utility led to the development of a new class of algorithms, which are not entirely local yet not global either. These algorithms operate essentially like local search algorithms, but enforce (usually small) groups agents to operate jointly. For instance, MGM-2 [Maheswaran et al., 2004a] is like MGM but tracks the possible gains for joint choices of pairs of agents instead of individually. Likewise, KOPT [Katagishi and Pearce, 2007] is a synchronous algorithm that can track joint movements of an arbitrary number of agents k, and k-DALO [Kiekintveld et al., 2010] implements the same idea in an asynchronous manner, obtaining significant speed gains. Along the same line, t-DALO [Kiekintveld et al., 2010] tracks joint choices of agents within t communication hops, and c-DALO [Vinyals et al., 2011] generalizes the algorithm to operate on agent neighborhoods of arbitrary shapes/sizes. Nonetheless, notice that the cost per iteration of running these algorithms is exponential on the parameters k, t or c.

One advantage of algorithms using an *immediate payoff* target with the *maximize* decision rule (from DSA to c-DALO) is that they provide quality guarantees [Pearce and Tambe, 2007; Bowring et al., 2008; Kiekintveld et al., 2010; Vinyals et al., 2011]. Unfortunately, the guaranteed quality drops quickly with the number of involved variables, to the point where it becomes irrelevant for mid-to-large problems. For instance, in a problem with 35 variables and binary constraints, DALO($k = 10$) (which is very costly to run) guarantees a solution within 15% of the optimum. In contrast, the typical quality of the actual solutions obtained by these algorithms is much larger, even with the simplest ones such as DSA.

2.2.2 Inference based algorithms

In the previous section we presented search-based algorithms, whose approach to finding good assignments is to explore the DCOP assignment space. Next we present a family of algorithms whose solving approach is radically different: inference based algorithms. Instead of exploring the search space, inference based algorithms collect information about the problem and try to infer good assignments based on this knowledge.

Complete inference algorithms

The first introduced algorithm of this class is the Dynamic Programming Optimization Protocol (DPOP) algorithm [Petcu and Faltings, 2005b]. As its name implies, DPOP is a dynamic programming algorithm, meaning that it tries to solve the problem by breaking it into a series of smaller, easier to solve subproblems. In the context of DCOPs, our problem is to maximize the objective function, and we start with that problem already divided in subproblems: the DCOP's constraints. Unfortunately, these constraints cannot be maximized (solved) independently because they share some variables. Hence, a sensible approach is to build larger constraints in such a way that some variables are not shared by multiple constraints anymore. This can be achieved by exploiting the *sum* operation introduced in Section 2.1.4. For instance, assume a DCOP with three constraints $f_1(x, y)$, $f_2(y, z)$ and $f_3(x, z)$. None of these constraints can be maximized independently, because all their variables are involved in some other constraint. However, we can build a larger constraint (subproblem) by summing two of the initial constraints

$$f_4(x, y, z) = f_1(x, y) + f_3(x, z) \ .$$

As a result, we now have our DCOP represented as just two constraints $f_2(y, z)$ and $f_4(x, y, z)$. Moreover, realize that now variable x is involved in only one of these constraints. Hence, we can now partially solve this constraint using the *maximize* operation to remove variable x from f_4 to obtain

$$f_5(y, z) = \max_x f_4(x, y, z) \ ,$$

and our problem is still represented as two constraints $f_2(y, z)$ and $f_5(y, z)$, but with one less variable. Hence, the dynamic programming approach to solve DCOPs involves repeatedly summing and partially maximizing constraints until no variables remain (meaning that the we have found the optimal utility of our DCOP).

Notice that this approach is analogous to that of the Generalized Distributive Law. Hence, the basic issues that DPOP must overcome are the same ones we presented at the end of Section 2.1.4, namely: (i) how to distributedly determine the order of summations and maximizations to perform; (ii) which agent should perform which operations; and (iii) how to obtain the optimal assignment from the optimal DCOP utility. DPOP resolves these issues by arranging the agents into a DFS tree (pseudotree), just like most exhaustive search methods presented above. The algorithm works in two phases over this pseudotree. First, agents compute the optimal utility by performing summations and maximizations flowing from the leafs to the root of the tree. Then, the optimal assignment is obtained using a so-called *value propagation* procedure down the tree. This approach has two main advantages:

1. It requires a **linear number of messages**. Because the algorithm operates on just two passes over the tree, and each agent only sends one message per pass, the total number of messages is guaranteed to be twice the number of agents in the problem.

2. The complexity of the algorithm is **exponential on the treewidth of the pseudotree, not on the number of agents**. This is a result of the first phase of the algorithm, during which agents are performing summations and maximizations. Due to how the pseudotree is constructed, it can be proven that the largest constraint computed during this phase involves exactly as many variables as the treewidth of the pseudotree. [2] Hence, the costlier operation performed by the algorithm is to compute this function, whose assignment space is exponential on the number of variables in its domain.

However, these benefits comes at a stiff price: each computed constraint needs to fit within the memory of the agent computing it. Thus, some agent requires an amount of memory exponential on the treewidth.

Along time, many techniques have been introduced to alleviate this problem. For instance, H-DPOP [Kumar et al., 2007] tries to reduce memory requirements by using a compact constraint representation for constraints involving many forbidden combinations. Instead, MB-DPOP(k) [Petcu and Faltings, 2007a] operates like DPOP but switches to a search method when the constraints become too large to hold in memory. In contrast, agents in O-DPOP [Petcu and Faltings, 2006] generate the constraints as streams of assignments with decreasing utility, and try to prove that the solution has been found before computing them entirely. Finally, BT-IDPOPf [Brito and Meseguer, 2010b] introduced the idea of function filtering, which iteratively builds better approximations of the constraints to compute, while saving space by filtering out (eliminating) suboptimal assignments from the computed constraints.

The introduction of the Action-GDL [Vinyals et al., 2009] algorithm explicitly connected this body of research with the GDL framework. Action-GDL itself operates exactly like DPOP, but over a *junction tree* instead of a pseudotree. This difference is not a major one, but it has been shown that junction trees allow for better representations (and hence reduced solving complexity) on some problems [Atlas and Decker, 2007; Vinyals et al., 2010b]. More importantly,

[2]The treewidth is a widely studied measure of a graph. Intuitively, it measures the interconnectedness density of the graph: a treewidth of one means that the graph is a tree, whereas a treewidth equal to the number of nodes means that it is fully connected. For further information we refer the reader to [Dechter, 2003].

the connection with the GDL clarified the relationship between the seemingly unrelated DPOP and Max-Sum algorithms (as explained below), and created a huge opportunity for cross-pollination between research areas. For instance, this connection has already been exploited to develop a number GDL-based algorithms for more complex models such as: (i) DCOPs whose constraints yield uncertain utilities [Stranders et al., 2011, 2012], (ii) DCOPs with multiple objectives [Delle Fave et al., 2011]; and (iii) resource-aware DCOPs [Stefanovitch et al., 2011] among others.

Global-state inference algorithms

In the previous section we introduced a number of inference algorithms that are guaranteed to compute the optimal solution. Contrary to complete search algorithms, most of the these algorithms are not *anytime* because they do not compute any solution until they finish. Nonetheless, two of the variants introduced to alleviate the exponential memory requirements of these algorithms turn out to be of this anytime class, and can thus be considered global-state approximate algorithms too. First, MB-DPOP(k) can be considered anytime because, when computing the full constraint becomes too expensive, it turns into a search algorithm (and hence inherits the search's anytime characteristic). Second, BT-IDPOPf has some anytime flavor, because it first computes an approximate solution, and then keeps refining it until the optimal solution is found.

Another breed of global-state inference algorithms are those that follow the same basic recipe than DPOP/Action-GDL, but mitigate the exponential memory requirement by dropping optimality.

On the one hand, optimality can be dropped during the algorithm's operation. This is the case of both A-DPOP [Petcu and Faltings, 2005a] and DMCTE(r) [Brito and Meseguer, 2010a]. In these algorithms, when an agent must calculate a constraint too large to be exactly computed, it assesses an approximate of the constraint instead. As a result, these algorithms end up finding an approximate solution to the problem, but can still provide a bound on the quality of that solution.

On the other hand, optimality can be dropped before even running the algorithm by simply discarding some of the problem's constraints. This approach was first used in the Bounded Max-Sum algorithm [Rogers et al., 2011], where agents heuristically discard problem constraints (tree edges) until the treewidth of the pruned tree is one. At this point, the algorithm can efficiently obtain the optimal solution of the relaxed problem, and provide a bound by assessing the maximum utility lost because of the discarded constraints. Extending this idea, the p-optimality framework [Okimoto et al., 2011] proposes different heuristics to discard constraints until the tree is guaranteed to have treewidth lower than or equal to p (a parameter chosen by the system designer).

Finally, there is also a completely different approach to perform global-state approximate inference in DCOPs, as first introduced by the DaCSA [Vinyals et al., 2010a] algorithm. Recall that the above algorithms basically divide the problem into a series of partially-independent subproblems. Then, these problems are partially solved one after the other, leading to the algorithm's solution for the DCOP. In contrast, the Divide-and-Coordinate (DaC) approach breaks the problem into one trivially solvable subproblem per agent. These subproblems may be dependent between them, yet DaC agents solve them as if they were independent. As a result, two subproblems oftentimes pick different choices for the same decision. When this happens, DaC algorithms adapt the subproblems by exchanging utilities between them according to some update rules. Thus, DaC agents keep re-solving and adapting the subproblems until either all agents obtain the same choices for all of the problem's decisions or they run out of time. The only differences between DaC algorithms are thus (i) how they divide the original problem into subproblems; and (ii) the update rules used to adapt the subproblems when the agents disagree about the best choice for some decision. Being the first of this class, DaCSA is the simplest algorithm: each initial subproblem is simply the sum of all constraints involving the agent, and updates are based on the specific choices made by each agent. EU-DaC [Vinyals et al., 2010c,d] improved on the update rules by considering the choice's utilities too, and DeQED [Hatano and Hirayama, 2013] employs larger (though efficiently solvable) subproblems.

Local inference algorithms

As briefly mentioned at the end of Section 2.1.4, it is also possible to perform purely local inference to assess good solutions to DCOP problems. The canonical algorithm for this task is known as Max-Sum [Farinelli et al., 2008]. Max-Sum was introduced as an adaptation of Pearl's Loopy Belief Propagation algorithm [Pearl, 1988] to solve DCOPs. Nonetheless, it has been shown that Max-Sum can be regarded as a particular instance of the GDL algorithm [Aji and McEliece, 2000]. The actual difference between Max-Sum and DPOP or Action-GDL lays on how they approach the basic issues that any GDL algorithm must handle.

Where Action-GDL uses a junction tree, the summations and maximizations to perform in Max-Sum are dictated by the factor graph. Additionally, because the factor graph is not necessarily a tree, a sequential execution of the algorithm could never terminate. Hence, Max-Sum does not terminate after a fixed number of steps. Instead, it terminates either because the messages do not change anymore, or because some designer-imposed limit has been reached (e.g.: maximum time, maximum number of iterations). Nonetheless, this also enables Max-Sum's nodes to operate in a fully parallel schedule, where each node of the factor graph computes and sends messages at the same time. Finally, the actual solution is not recovered through a global-state phase such as in DPOP. On the contrary, Max-Sum typically lets each agent independently pick whatever choices it deems best according to the latest information received from its neighbors.

2.2.3 Sampling-based algorithms

A last approach to DCOP solving, introduced very recently, is that of sampling-based algorithms. Algorithms of this class operate by letting agents sample their decision space, and make decisions based on the outcome of these samples. At the time of this writing, there are only two algorithms of this kind in the literature, both being approximate and global-state.

The first algorithm of this class is the Distributed Upper-bound Confidence on Trees (DUCT) algorithm [Ottens et al., 2012]. In DUCT, agents are first organized in a pseudotree. From there on, the algorithm operates in rounds. At each round, each agent randomly picks a choice for its variables, given the choices of its parents in the tree. Then, the agents compute the global utility of the complete assignment formed by their individual random choices. Finally, each agent keeps track of the combination of choices picked by itself and all its parents, as well as the associated outcome obtained. After a number of rounds it is provable that each agent knows its best choice with a certain confidence (the larger the number of rounds, the higher the confidence). The key element that makes this algorithm work well is that the random choices made by agents are not uniform. Instead, they are guided by the outcomes obtained in the previous rounds of the algorithm. Unfortunately, notice that leaf nodes have to keep track of a large number of combinations of choices, which grows exponentially with the height of the tree. Hence, DUCT requires exponential memory to operate, which is highly undesirable, especially for an approximate algorithm.

The very recent D-Gibbs algorithm [Nguyen et al., 2013] solves this issue by tracking outcomes differently, and using another approach to weigh the random choice picks. Essentially, agents in D-Gibbs only need to track the outcome for their own choices, disregarding their parent's ones. Hence, the algorithm does not require exponential memory anymore, and turns into a competitive global-state approximate algorithm. Furthermore, it has been shown that the algorithm can be parallelized and executed on GPUs [Fioretto et al., 2014] to achieve significant speed boosts.

2.2.4 Scalability analysis of current DCOP algorithms

After introducing the most significant DCOP algorithms, we now analyze their scalability characteristics. The classification presented in Table 2.1 helps us at this task, because algorithms of the same type (complete, global state approximate or local-state approximate) have very similar scalability characteristics. Hence, we now review these characteristics by type, and identify the most promising algorithms (from a scalability standpoint) of each class.

Recall that, as introduced in Section 1.1, DCOPs are NP-Hard. Because complete algorithms are guaranteed to find the optimal solution, their scalability in the general case is doomed by the problem's complexity. This means that, without any additional information, we cannot risk using an optimal DCOP algorithm in a multi-agent system because failing to obtain a solution would lead to a complete lack of coordination. Nonetheless, we can study which optimal solving algorithms work better on which problem classes, and how these algorithms scale. As a result, we may be able to identify application domains where it is actually feasible to perform optimal solving (due to the problems' characteristics and the application scale).

Unfortunately, no matter what the problem's characteristics are, there are no results in the literature that guarantee a better runtime for any complete search algorithm. In contrast, inference based algorithms do provide such a guarantee. Namely, the guarantee of a runtime exponential on the treewidth instead of on the total number of agents. Even though computing the pseudotree or junction tree of a problem with minimal treewidth is an NP-Hard problem itself, good approximations can be found quickly. Furthermore, these trees can be computed ahead of time, because they depend on the structure of the constraints but not on their utilities. As a result, we argue that inference based algorithms are very appealing candidates for practical DCOP applications at a scale where optimal solving is still feasible.

On this front, the GDL framework provides a solid foundation that encompasses most works developed to this time. Nonetheless, there is a significant open issue regarding the usage of GDL-based algorithms: agents in the actual world have limited computation and communication resources, and the GDL framework simply does not consider those characteristics at all. Thus, on the first part of this work we focus on optimal solving within the GDL framework umbrella, with special attention to minimizing and satisfying the resource requirements of actual-world agents.

Once optimal solving becomes unfeasible, there are a wide range of global-state approximate algorithms that provide varying degrees of solution qualities and quality guarantees. However, all these algorithms employ techniques where global computations must be made. Moreover, notice that performing such computations is inherently frail: a single agent failing stalls the whole computation, and hence the algorithm stops functioning. This is not a huge problem in small-scale applications that can be solved optimally, but quickly becomes a liability when the applications scale up. Furthermore, even in an ideal setting without such failures, larger problems require more communication hops to perform a global computation. Thus, the solving time necessarily grows larger when the problems get larger.

In contrast, local-state approximate algorithms are inherently well equipped to deal with such nuisances. A non-functioning agent may slightly decrease the performance of others, but does not prevent the algorithm from working. Likewise, agents running these algorithms can always provide solutions very quickly, even on problems vast amounts of agents. Additionally, although local-state algorithms cannot provide good quality guarantees, their actual solutions are on the same range (quality-wise) than those of global-state approximate algorithms. Therefore, in the second part of this work we focus on local-state algorithms for actual-world, moderate-to-large scale problems.

2.3 Summary

In this chapter we provided the necessary context to position this work within the extensive literature about DCOPs. With this aim, we first formally introduced the DCOP model. Then we have shown an example of the kind of practical applications it can deal with, as well as some useful representations that help us understand the problem's structure. Next we presented the GDL and the abstract framework built around it to support the development of efficient solving algorithms for a wide range of problems. Moreover, we have shown that DCOPs are one of the problem types that fit such a framework.

Then we explored the large literature about DCOP solving. We analyzed the most prominent DCOP algorithms, classifying them by solving approach and algorithm type. Such classification provided us an overview of the most promising methods depending on the practical applications' characteristics. As a result, this survey allowed us to highlight the necessity and relevance of the techniques developed throughout this thesis.

Namely, we identified two areas that are worth exploring and improving. First, GDL-based complete algorithms appear as a very promising approach for relatively small-scale applications (i.e.: tens of decisions). At this scale, robustness is not a major issue, whereas algorithms with a cost exponential on the number of agents are already non-applicable. Thus, GDL-based algorithms are a good fit because they guarantee a cost exponential on the treewidth (which is typically much lower than the number of decisions). However, this guarantee comes at the price of increased resource requirements on the agents. Hence, in the first part of this work we explore techniques to lower and/or trade-off the different kinds of resources (e.g., network usage and computation time) possessed by the agents. In this manner we effectively widen the applicability of GDL-based complete solving algorithms.

Second, robustness becomes an important issue on larger scale, practical applications. Here, the Max-Sum GDL-based local algorithm rises as a more suitable and promising technique. On the one hand, because it provide empirically good solutions while tolerating agent failures and communication losses. On the other hand, because it is one of the few local-state algorithms that does not behave in a greedy manner. That is, agents in Max-Sum do not simply try to maximize their own utility. On the contrary, they truly strive to optimize the global objective instead. Nonetheless, Max-Sum has an important caveat: its cost is exponential on the number of neighbors of each agent, whereas other local algorithms take only linear costs. As a result, the second part of this work focuses on developing actual-world, larger-scale dynamic DCOP applications and lowering the Max-Sum's resource requirements to solve them.

Part I

Optimal solving

Chapter 3

Scaling by better filtering

3.1 Introduction

In Section 2.3 we identified GDL-based algorithms as a promising approach for optimal DCOP solving. The grounds of this argument is that GDL-based algorithms are the only complete algorithms for which we can identify easy to solve problem classes. Namely, the worst case complexity bound on the problem's treewidth (instead of on the total number of decisions to make) directly identifies such problem classes, thus answering our Question 1 in the introductory chapter.

As a result, we now focus our efforts on answering Question 2. That is, in this chapter we strive to improve the applicability of GDL-based algorithms by considering their computation and communication requirements. The straightforward approach to achieve such goal is to devise techniques that lower the amount of resources required by the algorithm. Therefore, in this chapter we aim at developing purely algorithmic improvements that lower the required amount of resources.

As introduced in Section 2.2.2, the most advanced state-of-the-art GDL-based algorithms is known as GDL with function filtering. In this variant, the algorithm does not find an optimal solution in a single pass through the junction tree. Instead, it performs a number of passes through that tree. Initial passes are cheap on resources and produce loose upper and lower bounds on the optimal utility. Subsequent passes are increasingly more expensive but produce tighter bounds. After a fixed number of passes the algorithm is guaranteed to find an optimal solution. The key insight is that bounds found in previous iterations allow the algorithm to prune (filter) the search space in latter iterations, thus alleviating their computation and communication costs.

As a consequence, the tighter the bounds computed at each iteration, the lower the overall cost of the algorithm. In this chapter we present two different techniques to improve such bounds. First, we introduce the so-called two-sided function filtering technique. The basic idea behind two-sided filtering is the same as in classical filtering (which we call one-sided filtering from now on).

However, agents employing two-sided filtering make a better use of the information they have. As a result, they are able to compute tighter lower bounds and thus prune larger parts of the search space, reducing the algorithm's costs. Second, we present a distributed solution exploration technique that enables agents to explore multiple solutions. In contrast, previous approaches to GDL with function filtering computed a single solution each pass over the tree. The cost of that solution is then used as the upper bound employed by the filtering technique. Therefore, by exploring multiple solutions, the agents can find better upper bounds and further improve the filtering's results.

Combining our two novel techniques we reduce the amount of resources required to optimally solve DCOPs. Empirically, we estimate a reduction of up to about 70% on communication costs and up to around 30% on computational costs with respect to the state-of-the-art one-sided filtering. Furthermore, we also obtain a significant memory reduction. This is a crucial improvement because, if an agent runs out of memory, the problem cannot be solved by GDL. As a result, our agents can solve up to 75% more problem instances than before given the same amount of memory. To summarize, we manage to increase the range of problems that can be solved optimally by algorithms employing function filtering.

In the remainder of this chapter we first introduce some further background that is specifically useful to solve DCOPs optimally. Next we outline GDL with function filtering as a DCOP solving approach. Thereafter in Section 3.4 we present the so-called two-sided filtering technique, focused on improving lower bounds. Next in Section 3.5 we turn our attention to improving upper bounds by computing multiple candidate solutions in parallel. Finally we draw some conclusions from the observed results, which lead to the further work presented in the next chapter.

3.2 Background for optimal DCOP solving

In Section 2.1.1 we introduced a general DCOP model and the associated objective function that DCOP algorithms try to optimize. That definition can capture any kind of DCOP with any kind of utilities. However, certain algorithms require a more stringent definition, enabling them to employ techniques that could not be used otherwise. In this section we first present an alternative DCOP definition that is particularly suitable for complete DCOP algorithms. Next we present the general operations employed by all GDL-based algorithms, and introduce the idea of bound computation that enables function filtering to prune the solution space.

3.2.1 Minimizing DCOPs

There is a specific characteristic in Definition 2.1 that makes optimal DCOP solving particularly hard: utilities of the constraints can be any real value or $-\infty$. This prevents us from using partial evaluations of a solution as a bound of its global utility. To clarify, consider a DCOP with two constraints $f_1(x)$ and $f_2(x, y)$, where both variables are binary. Say that $f_2(0, 0) = 10$. Unfortunately, this provides us with no immediate information about the global utility of the complete assignment $\langle x = 0, y = 0\rangle$ because $f_1(0)$ could be any value.

Now consider an alternative definition of DCOPs where the constraint utilities can only be negative values or 0. Formally, the constraints in this new definition are of the form $f : \times_{x\in S} D_x \rightarrow \mathbb{R}^- \cup \{0, -\infty\}$. Under this new definition, a partial evaluation of a complete assignment immediately tells us that this assignment is going to provide at most that amount of utility. That is, using the same example above, say that $f_2(0, 0) = -10$. Then, we can guarantee that the global utility of the complete assignment $\langle x = 0, y = 0\rangle$ is lower than or equal to -10. Such information can be very valuable for DCOP algorithms, because it enables them to reason in terms of bounds.

Nonetheless, thinking in terms of negative utilities is fairly counter-intuitive for us humans. Thus, we can further adjust the DCOP definition and its objetive to ease our reasoning. Instead of using negative utilities, we can now think in terms of costs, and define the constraints as yielding positive costs instead ($f : \times_{x\in S} D_x \rightarrow \mathbb{R}^+ \cup \{0, \infty\}$). Moreover, since the constraints' values are now costs instead of utilities, the objective of DCOP algorithms under this new formulation is to minimize (instead of maximizing) the objective function in Equation (2.1).

Notice that any DCOP represented using Definition 2.1 can be transformed into an equivalent DCOP using this new formulation. Formally, given a DCOP $\Omega = \langle A, X, D, C, \mathfrak{m}\rangle$ whose utility we want to maximize, we can formulate another DCOP $\Omega = \langle A, X, D, C^+, \mathfrak{m}\rangle$ that uses only positive (or ∞) costs and should be minimized instead of maximized. Furthermore, C^+ can be easily constructed from C by applying a simple transformation to each constraint. Namely, $C^+ = \{f_1^+, \cdots, f_n^+\}$, where

$$f_i^+(\mathbf{x}) = v_i^* - f_i(\mathbf{x})$$

and v_i^* is the maximum value within the f_i constraint:

$$v_i^* = \max_{\mathbf{x}} f_i(\mathbf{x}).$$

Given the aforementioned advantages, in the remainder of this part (Chapters 3 and 4) we employ this formulation instead of the formulation in Section 2.1.1.

3.2.2 Operations between constraints

The objective of a complete (optimal) DCOP algorithm is to find a complete assignment that is an optimal solution of a DCOP (and possibly the corresponding optimum cost). In what follows, we provide an alternative definition to that in Equation (2.1). This alternative definition will help us introduce the GDL with function filtering framework and explain the improvements presented in this chapter.

First, we introduce an operator that allows us to assess the cost of an assignment for two constraints.

Definition 3.1. The *combination of two constraints* f_1 with scope S *and* f_2 with scope T, noted as $f_1 \bowtie f_2$, is a new cost function defined over the union of their scopes $U = S \cup T$, which we define as:

$$(f_1 \bowtie f_2)(\mathbf{u}) = f_1(\mathbf{u}[S]) + f_2(\mathbf{u}[T]),$$

where $\mathbf{u}$ stands for all possible assignments to the variables in U.

Notice that the combination operator is both associative and commutative. It is straightforward to generalize the combination operator over a set of functions as follows.

Definition 3.2. Given a set of functions $F = \{f_1, \ldots, f_m\}$, *the combination of* F, noted as $\bowtie F$, is the function resulting from the joint combination of all the functions in F:

$$\bowtie F = f_1 \bowtie \ldots \bowtie f_m.$$

At this point, we can readily offer an alternative definition to solving a DCOP. Thus, finding the optimal solution of a DCOP $\Omega = \langle A, X, D, C^+, \mathfrak{m} \rangle$ amounts to finding a complete assignment $\mathbf{x}$ that minimizes the following expression over all complete assignments:

$$(\bowtie C)(\mathbf{x}). \tag{3.1}$$

3.2.3 Computing lower bounds

As mentioned above, function filtering [Brito and Meseguer, 2010a] is a technique that has been employed to solve DCOPs. The basis of function filtering is to filter out suboptimal solutions from the assignment space. With this aim, function filtering proposes to employ two different types of bounds: (i) an upper bound on the global cost, given by the best solution found so far; and (ii) a lower bound on the assignments of the constraint to be filtered. Those partial assignments whose lower bound is larger than the global upper bound represent suboptimal solutions, and hence they can be safely filtered out. Next we formally define the notion of lower bound on assignments, constraints and sets of constraints.

First, the notion of lower bound of an assignment requires to define the notion of extension of an assignment.

Definition 3.3. Let $V \subseteq X$ be a subset of variables and $\mathbf{v}$ an assignment of values to each of the variables in V. An *extension of an assignment* $\mathbf{v}$ to X is a new assignment with scope X that keeps the values of $\mathbf{v}$ and sets new values to the variables in $X \setminus V$.

Therefore, we can easily generate an extension of a tuple $\mathbf{v}$ to produce a an assignment $\mathbf{x}$ with scope X by simply adding values for the variables in $X \setminus V$ to $\mathbf{v}$.

Definition 3.4. Given an assignment $\mathbf{v}$ with scope V, where $V \subseteq X$, if the cost of every possible extension of $\mathbf{v}$ to X is larger than or equal to some value $LB \in \mathbb{R}^+$, we say that LB is a *lower bound on the cost of the assginment* $\mathbf{v}$.

Next, we generalize the definition above to encompass constraints. Before that, we introduce the notion of projection of a constraint over a set of variables (as a generalization of Definition 2.5), on which our definition of lower bound on a cost function relies.

Definition 3.5. The *projection of a constraint* f with scope S *over* a set of variables $T \subseteq S$, noted as $f[T]$, is a new constraint with scope T that assigns to each tuple $\mathbf{t}$ the minimum cost that can be obtained by extending $\mathbf{t}$ with all possible values for the variables in $S \setminus T$. Formally,

$$f[T](\mathbf{t}) = \min_{\substack{\mathbf{s} \text{ extension of } \mathbf{t} \\ \text{to } S}} f(\mathbf{s})$$

Definition 3.6. A constraint f_1 with scope T is a *lower bound on a constraint* f_2 with scope S, noted as $f_1 \leq f_2$, iff:

- $T \subseteq S$, and
- $f_1(\mathbf{s}[T]) \leq f_2(\mathbf{s})$ for all possible assignments $\mathbf{s}$.

Finally, based on the definition of lower bound on a constraint, we introduce the notion of lower bound on a set of constraints.

Definition 3.7. A constraint f is a *lower bound on a set of constraints* F if it is a lower bound on the constraint $\bowtie F$ resulting from the combination of all constraints in F.

From Equation (3.1), observe that a constraint f with scope S is a lower bound on a DCOP $\langle A, X, D, C^+, \mathfrak{m}\rangle$ if and only if it is a lower bound on the combination of the set of constraints C^+. Namely, if for each assignment $\mathbf{s}$, $f(\mathbf{s})$ is a lower bound of the cost of the best extension of $\mathbf{s}$ to X (standing for a possible solution to the DCOP).

Now consider the matter of obtaining tight bounds for both a constraint or a set of constraints. On the one hand, regarding a single constraint, consider a constraint f with scope S and a set of variables T such that $T \subseteq S$. We can readily obtain the tightest bound on function f as its projection on T, namely as $f[T]$.

Definition 3.8. Given a set of constraints $F = \{f_1, \ldots, f_m\}$, the *combination of a set of constraints over a set of variables* $T \subseteq \cup_{i=1}^{m} sc(f_i)$, noted as $(\bowtie F)[T]$ is a single constraint that stands as the tightest possible lower bound of F.

Observe that the time to compute the tightest bound of F is bounded by $O(d_T^{|T|})$, where d_T is the largest domain among the variables in T. Depending on the features of the problem, such computation can become overly demanding. Interestingly, there is a less costly way of computing a tight lower bound on a set of functions. This way requires the prior introduction of a further operation, namely the projection on a set of cost functions:

Definition 3.9. Given a set of constraints $F = \{f_1, \ldots, f_m\}$, the *projection of a set of constraints under* a set of variables $T \subseteq \cup_{i=1}^m sc(f_i)$, noted as $F[T]$, is a new set composed of the projections of each constraint in F over T. Formally,

$$F[T] = \{f_1[T], \ldots, f_m[T]\}$$

Based on Definition 3.9, we can compute a lower bound of F as the combination of all projections in $F[T]$, namely as $\bowtie(F[T])$. We call this lower bound *the combination of F under T*. This bound can be computed in $O(\max[\max_{i=1}^m d_{V_i}^{|V_i|}, d_T^{|T|}])$ time, where $V_i = sc(f_i)$. Notice that this can be largely smaller than $O(d_V^{|V|})$ above, hence reducing computational time.

Example

To illustrate the above concepts on operations of constraints, we consider the following toy example: a DCOP with 3 agents each holding a variable x, y, and z, whose domains are all $\{a, b\}$, and two constraints $f_1(x, y)$ and $f_2(y, z)$. Figure 3.1 shows the combination of f_1 and f_2 over $\{x, z\}$, and the combination of f_1 and f_2 under $\{x, z\}$. Observe that the latter is a lower bound of the former.

3.3 Solving DCOPs using the Generalized Distributive Law

In this Section we review the algorithms in the literature that have extended the generalized distributive law (GDL) algorithm, originally introduced in [Aji and McEliece, 2000], to solve DCOPs. In Section 3.3.1 we provide an overview on the GDL algorithm. Thereafter, we analyze the contributions in the literature that have focused on specializing GDL to solve DCOPs. First, we analyze both complete and approximate GDL-based algorithms in Sections 3.3.2 and 3.3.3 respectively. Second, Section 3.3.4 reviews GDL-based algorithms that exploit function filtering [Sánchez et al., 2005] to prune the space of solutions of a DCOP.

3.3.1 The GDL algorithm

As introduced in Section 2.1.4, the GDL algorithm [Aji and McEliece, 2000] is a general message-passing algorithm that exploits the way an objective function factors into a combination of functions (constraints), to compute it in an efficient

$$f_1 = \begin{array}{cc|c} x & y & \\ \hline a & a & 1 \\ a & b & 2 \\ b & a & 3 \\ b & b & 4 \end{array} \qquad f_2 = \begin{array}{cc|c} y & z & \\ \hline a & a & 8 \\ a & b & 7 \\ b & a & 6 \\ b & b & 5 \end{array}$$

$$f_1 \bowtie f_2 = \begin{array}{ccc|c} x & y & z & \\ \hline a & a & a & 1{+}8 \\ a & a & b & 1{+}7 \\ a & b & a & 2{+}6 \\ a & b & b & 2{+}5 \\ b & a & a & 3{+}8 \\ b & a & b & 3{+}7 \\ b & b & a & 4{+}6 \\ b & b & b & 4{+}5 \end{array} \qquad (f_1 \bowtie f_2)[\{x, z\}] = \begin{array}{cc|c} x & z & \\ \hline a & a & 8 \\ a & b & 7 \\ b & a & 10 \\ b & b & 9 \end{array}$$

$$f_1[\{x, z\}] \bowtie f_2[\{x, z\}] = \begin{array}{c|c} x & \\ \hline a & 1 \\ b & 3 \end{array} \bowtie \begin{array}{c|c} z & \\ \hline a & 6 \\ b & 5 \end{array} = \begin{array}{cc|c} x & z & \\ \hline a & a & 7 \\ a & b & 6 \\ b & a & 9 \\ b & b & 8 \end{array}$$

Figure 3.1: Example of operations on cost functions.

manner. In particular, notice that this is the case of a DCOP, since the definition of its objective function Equation (3.1) results from the combination of cost functions.

GDL is defined over two binary operations that form a semi-ring. Considering that the objective function of a DCOP is to minimize Equation (3.1), such operations are minimization and addition. To ensure optimality and convergence, GDL operates on a special structure named junction tree (JT) [Jensen and Jensen, 1994] (also known as joint tree, cluster tree or tree decomposition), which exploits the way an objective function factors into a combination of functions.

A JT is a tree structure $\langle N, E\rangle$, where N is a set of nodes and E is a set of edges. For a DCOP $\Omega = \langle A, X, D, C^+, \mathfrak{m}\rangle$, each node $i \in N$ in the JT contains a subset of variables $V_i \subseteq X$ and a subset of constraints $C_i \subseteq C$ satisfying the following conditions:

- For each cconstraint f in node i of the JT (i.e., $f \in C_i$), the variables in its scope are contained in V_i, namely $sc(f) \subseteq V_i$.
- For each constraint $f \in C$ of the DCOP, there is exactly one node i of the JT such that $f \in C_i$.
- For each variable $x \in X$, the set of nodes of the JT where x is present, namely $\{i \in N$ such that $x \in V_i\}$, forms a connected subtree.[1]

[1]This is the so-called running intersection property.

If two nodes i and j are connected, it means that they share some variable. Shared variables between nodes i and j, $S_{ij} = V_i \cap V_j$, are known as the *separator* between i and j.

In general, there are several JTs that can represent an objective function for GDL. Hence, there is the issue of finding the *best* JT. This is not an easy endeavor because, although it is known that the complexity of GDL is exponential in the size (in terms of number of variables) of its largest node, finding the JT whose largest node has minimum size is NP-complete [Jensen and Jensen, 1994]. Therefore, the construction of a JT is typically performed by means of heuristics, which cannot assure optimality but often provide good enough JTs.

As mentioned above, GDL is a message-passing algorithm on a JT. The purpose of the algorithm is that nodes distributedly compute some objective function that is factored among them. Aji and McEliece [2000] distinguish two cases regarding the application of GDL: the *single-vertex* problem, when the goal is to compute the objective function at a single node, and the *all-vertices* problem, when the goal is to compute the objective function at all nodes. In what follows we focus on the all-vertices problem because of its relationship with DCOPs (as we discuss further ahead in Section 3.3.2) and we also consider that the objective function is the one in Equation (3.1). To solve this problem, the fully-serial version of GDL (also known as CTE [Dechter et al., 2001]) operates in two passes through the JT. The first pass flows up the tree (from leaves to root), where each node sends a message to its parent node after receiving a message from each of its children. The second pass flows down the tree (from root to leaves), where each node sends a message to each of its children after receiving a message from its parent. In both passes, the messages exchanged contain a single constraint. The basic operation to compute these constraints is simple. Namely, to compute the message for some recipient, the sending node combines its own constraints (those in C_i) with the constraints received from its neighboring nodes except from the recipient. This produces a constraint that is then projected over the variables of the separator linking both nodes. The result is yet another constraint, which is the message that is actually sent. Considering the objective function in Equation (3.1), the combination of cost functions can be performed using the operation in Definition 3.2, while a projection can be assessed using Definition 3.5.

To understand the operation of GDL, it is fundamental to understand the semantics of the messages that nodes in a JT send and receive. Consider the JT depicted in Figure 3.2. Say that node j contains variables x_1, x_2 and node i contains variables x_2, x_4. Hence, their separator contains variable x_2. Then, node j would send to node i a constraint that contains the best cost that x_2 could obtain for each value of its scope after considering all the possible values of x_1. Therefore, the message from node j to node i *summarizes* the cost of the assignments to x_2 when only considering the values for the variables in node j. In general, if node j has further children below the JT, as shown in Figure 3.2, then the message from node j to node i would summarize the cost of the assignments to the variables in the separator between both nodes when only considering the

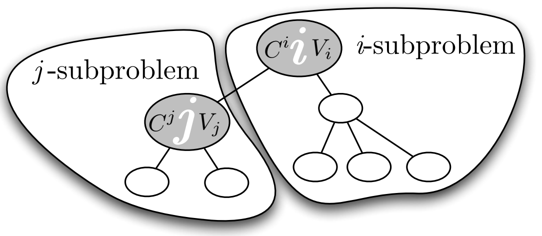


Figure 3.2: Subproblems in a junction tree.

values for the variables of the subtree whose root is j.

After the nodes in the JT finish exchanging messages, each node contains the costs of the objective function for its variables. Then, each node can locally assess the values of its variables that optimize the objective function.

Finally, we must be aware of the complexity of GDL in terms of computation and communication. As to computation, notice that the cost of combining constraints, performed by each node, is exponential on the arity of the resulting constraint. Hence, the size of the constraints to send from node to node may be prohibitively large when GDL operates in communication-constrained scenarios.

3.3.2 Complete GDL-based algorithms

GDL can be adapted to solve DCOPs in a distributed manner, as required by this type of problems. The first step is to build a JT in a distributed setting, as described in [Paskin et al., 2005]. In this setting, each node of the resulting JT represents an agent operating over a subset of variables and constraints of a DCOP. Each edge represents a communication link used by two agents to exchange messages containing constraints defined over their shared variables (the variables in their separator). Figure 3.2 shows agent i and j linked by an edge in a JT. Observe that removing the edge connecting i and j splits the JT into two different connected components, each one standing for a subproblem of the very same DCOP. Thus, we say that the i-subproblem involves every cost function in the component containing i after the edge is removed. Subproblems i and j are coupled by a set of variables they share, namely their *separator* ($S_{ij} = V_i \cap V_j$). By exchanging messages, subproblems i and j aim at agreeing on the value of their shared variables.

Recall from Section 3.3.1 that the basic GDL operation only involves the exchange of constraints. After the execution of the two passes required by the fully-serial all-vertices GDL (henceforth referred as *cost propagation* phase), each node has complete knowledge of the global objective function for the variables in the node. Thus, each node can locally compute the assignment of variables that minimizes the objective function. Nonetheless, since it might be the case that two optimal solutions $\mathbf{x}_1$ and $\mathbf{x}_2$ (with the same cost) exist, some agents might choose $\mathbf{x}_1$ while others choose $\mathbf{x}_2$. This would lead to assigning different values to the same variable in different nodes. To prevent such inconsistent assignments, GDL can be extended with a *solution propagation* phase aimed at having the nodes agree on the assignments of shared variables. Solution propagation requires that the JT root decides the optimal variable assignment and informs its children in the JT, which in turn inform their own children and so on. The result of this phase is a coherently optimal solution. This algorithm was named DCTE in [Brito and Meseguer, 2010a].

Action-GDL [Vinyals et al., 2010b] is a specialized application of GDL to solve DCOPs. Thus, it optimally solves the problem by performing two phases. During the first phase, constraints flow from the leaves to the root of a JT. Once the first phase is over, the second phase starts with the root making a decision regarding the assignment of values to its variables. Thereafter, the root informs about its decision to its children, which in turn decide on the assignment of values to their own variables, and subsequently send their assignments to their children. The process continues until reaching the leaves, where the last variable assignments are made.

Likewise DCTE, Action-GDL involves a cost propagation and a solution propagation phase. However, the cost propagation phase of Action-GDL is *one-way* (constraints are sent up the JT from leaves to root), whereas DCTE's is *two-way* (constraints are sent up the tree, and thereafter down the tree). In fact, Action-GDL can be regarded as the application of GDL to solve the single-vertex problem (the root of the JT is the only node that computes the objective function) followed by a solution propagation phase that runs down the JT (from root to leaves) to make the optimal variable assignments.

We can resort to the operations introduced in Section 3.2 to algorithmically describe the operation of Action-GDL. At the beginning of the cost propagation, each leaf node starts by computing the projection of its constraints over the variables the node shares with its parent. This operation is computed following Definition 3.5 and the result is sent to the leaf node's parent. After a node has received messages from all its children, it combines them with its own constraints (using the combination operator in Definition 3.2) and projects the resulting function over the variables shared with its parent. The resulting constraint is sent to the node's parent. Once the root has received cost messages from all its children, it combines them with its own constraints to yield the objective function. Thereafter, the solution propagation phase starts. The root decides the best assignment (the minimum cost partial assignment) for its variables. Then, it broadcasts this assignment to its children, who in turn assess their best assignments and send them down the tree. The process ends after all leaves compute their assignments.

3.3.3 Global approximate GDL-based algorithms

A significant drawback of GDL-based complete algorithms is the exponential size of the constraints exchanged during the cost propagation phase. Therefore, instead of sending a constraint as a whole, we can alternatively opt for sending an *approximation* of the constraint as a lower-arity function [Dechter, 1997], at the expense of losing optimality.

Notice that the quality of a DCOP solution is expected to increase as the arity of these approximations increases. This can serve as the basis to build algorithms that find better solutions by gradually increasing the arity of the approximated messages. This is the idea behind the DMCTE(r) algorithm [Brito and Meseguer, 2010a], that can be seen as a global approximate version of GDL that produces an approximate solution. With this aim, during the cost propagation phase the nodes in the JT send approximations of the messages that they would send when running GDL. Given two nodes in the JT sharing n variables in their separator, a message approximation from i to j takes the shape of a list of r-arity constraints, where $r < n$. This list of constraints is a lower bound of the message (constraint) that GDL would send from i to j.

Unlike GDL, once the cost propagation phase ends up, the nodes in the JT do not have complete information regarding the objective function. Therefore, the solutions after the second phase, the solution propagation phase, are not guaranteed to be optimal. Hence, DMCTE(r) introduces a third phase, the so-called *bound propagation* phase. During this phase, each node computes the costs of both an upper bound (from the decisions made by the node) and a lower bound of the objective function.

3.3.4 GDL-based algorithms with function filtering

DMCTE(r) can be run iteratively by increasing the arity limit at each iteration. However, the cost of such approach is larger than directly running DMCTE(r) considering the highest possible arity. This is true unless there is some way to exploit the information exchanged by the nodes at previous iterations. With this aim, Brito and Meseguer introduced the function filtering technique to solve DCOPs [Brito and Meseguer, 2010a]. Function filtering was originally proposed in [Sánchez et al., 2005] to solve constraint optimization problems in a centralized manner. Function filtering is employed by each node prior to sending cost messages to filter out (prune) assignments that cannot be part of the optimal solution. To filter out assignments, a node can resort to the information received during a previous iteration. Thus, to decide whether an assignment can be filtered out or not, a node uses a global upper bound (obtained by the bound propagation phase during a previous iteration) along with the lower bound of the constraint to send. Thus, a node can safely filter out an assignment whenever its lower bound is larger than the global upper bound.

From the discussion above, there is the issue of computing lower bounds at each node. Henceforth, consider that the DMCTE(r) algorithm is run with an increasing value for r over a DCOP such as the one depicted in Figure 3.2. During the r-th iteration of the algorithm, agent i sends to agent j a message that we note as $M^r_{i\to j}$ and receives from j another message that we note as $M^r_{j\to i}$. According to the description of DMCTE(r) above, each message contains a list of constraints over the variables of the separator shared between agents i and j, namely S_{ij}. Next, during the $(r+1)$-th iteration, the message that i will compute to send to j, $M^{r+1}_{i\to j}$, can exploit the information contained in the message received from j during the last iteration, namely $M^r_{j\to i}$.

Notice that each constraint f_i in the outgoing message $f_i \in M^{r+1}_{i\to j}$ readily specifies a minimum cost for each of its possible assignments. That is, each constraint f_i, with scope S_i, specifies some cost $c(\mathbf{s}_i)$ for each possible assignment $\mathbf{s}_i$ as shown in Equation (3.3). In the previous iteration r, agent i received from j the message $M^r_{j\to i}$, specifying that the minimum cost of $\mathbf{s}_i$ in the j-subproblem is c'. This cost is simply the sum of the costs specified in each of the constraints within the message, as shown in Equation (3.4). These two costs can be added to obtain a lower bound on the cost of $\mathbf{s}_i$, because they represent two costs for disjoint parts of the DCOP (the subproblem containing node i and the subproblem containing node j). If this lower bound on the cost of $\mathbf{s}_i$, namely $lb_i(\mathbf{s}_i)$, exceeds the upper bound on the global cost of the optimal solution (obtained by node i during the bound propagation phase), the assignment $\mathbf{s}_i$ cannot be extended to an optimal solution and can thus be eliminated from f_i. Formally, we compute the lower bound for each assignment $\mathbf{s}_i$ of each function f_i as:

$$lb_i(\mathbf{s}_i) = c(\mathbf{s}_i) + c'(\mathbf{s}_i) \ , \tag{3.2}$$

where

- $c(\mathbf{s}_i)$ is a lower bound on the contribution of the i-subproblem, computed as

$$c(\mathbf{s}_i) = f_i(\mathbf{s}_i) \ . \tag{3.3}$$

- $c'(\mathbf{s}_i)$ is a lower bound on the contribution of the j-subproblem, assessed as

$$c'(\mathbf{s}_i) = \sum_{g \in M^r_{j\to i}} g[S_i](\mathbf{s}_i) \ . \tag{3.4}$$

Therefore, we say that $M^r_{j\to i}$ *filters* each constraint $f_i \in M^{r+1}_{i\to j}$, and name the process *function filtering*.[2]

This approach defines the DIMCTEf algorithm [Brito and Meseguer, 2010a], and has been shown very effective to assess the optimal solution of a DCOP while maintaining a reasonable consumption of resources (especially memory and communication). In some cases, this allows solving DCOPs that DCTE is unable to solve because it runs out of memory.

3.4 Two-sided filtering

In this section we aim at tightening the one-sided lower bound described above. With tighter lower bounds, the algorithm will be able to prune more assign-

[2]Strictly speaking, we refer to this technique (presented in [Sánchez et al., 2005]) as *one-sided* function filtering, to differentiate with the *two-sided* function filtering technique that we describe later on.

ments, hence reducing the communication and computation overhead even further. Moreover, notice that the effects of filtering accumulate exponentially. That is, filtering out an assignment at iteration r means that there are exponentially less assignments to consider at iteration $r+1$. As a result, we expect the algorithm to be able to cope with larger problems that could not be handled before.

To this end, we now describe the so-called two-sided filtering technique, which improves on the one-sided filtering described above. Consider that agent i has already received $M^r_{j\to i}$ from agent j. After that, it intends to send a set of constraints $M^{r+1}_{i\to j}$, summarizing the cost information in the i-subproblem, to agent j. Since no constraint appears in both the i-subproblem and the j-subproblem, we can assess a lower bound for the complete problem by adding a lower bound of each of them. Notice that the one-sided lower bound in Equation (3.2) already assesses the summary of the costs of the j-subproblem from $M^r_{j\to i}$. Likewise, it assesses the cost in the i-subproblem by considering each constraint $f_i \in M^{r+1}_{i\to j}$. However, each constraint f_i is considered separately. As a result, it is missing some of the information that agent i has about the i-subproblem. Namely, the other constraints $f_j \in M^{r+1}_{i\to j}$ may also contain some known costs for the assignments in f_i which are not taken into account by one-sided filtering.

The key idea of two-sided filtering is to also consider such costs when assessing the contribution of the i-subproblem to the lower bound of each assignment. Hence, two-sided filtering computes $c(\mathbf{s}_i)$ in Equation (3.3) by also accumulating the minimum costs specified by the other constraints in $M^{r+1}_{i\to j}$. Formally, two-sided filtering computes the lower bound of each assignment $\mathbf{s}_i$ of each function $f_i \in M^{r+1}_{i\to j}$ as:

$$lb_i(\mathbf{s}_i) = c''(\mathbf{s}_i) + c'(\mathbf{s}_i) \ , \tag{3.5}$$

where

- $c''(\mathbf{s}_i)$ is a lower bound on the contribution of the i-subproblem, computed as

$$c''(\mathbf{s}_i) = \sum_{f \in M^{r+1}_{i\to j}} f[S_i](\mathbf{s}_i) \ . \tag{3.6}$$

- $c'(\mathbf{s}_i)$ is a lower bound on the contribution of the j-subproblem, assessed as

$$c'(\mathbf{s}_i) = \sum_{g \in M^{r}_{j\to i}} g[S_i](\mathbf{s}_i) \ . \tag{3.7}$$

Observe that there is no double counting of costs because no cost function appears in both the i-subproblem and the j-subproblem, and each constraint in the messages is only considered once. Hereafter, we refer to the lower bound in Equation (3.5) as *two-sided lower bound.* The name stems from the symmetrical use of both subproblems.

Given

$x\,y$	$g(x,y)$
$a\,a$	3
$a\,b$	4
$b\,a$	3
$b\,b$	3

,

$x\,y$	$f_1(x,y)$
$a\,a$	5
$a\,b$	2
$b\,a$	8
$b\,b$	6

,

$x\,z$	$f_2(x,z)$
$a\,a$	4
$a\,b$	3
$b\,a$	5
$b\,b$	2

, UB=10

				One-sided	**Two-sided**
$x\,y$	$f_1 \bowtie f_2[\{x,y\}] = c''$			$f_1 + c'$	$c'' + c'$
$a\,a$	5	3	8	$5 + 3$	$8 + 3$ ✗
$a\,b$	2	3	5	$2 + 4$	$5 + 4$
$b\,a$	8	2	10	$8 + 3$ ✗	$10 + 3$ ✗
$b\,b$	6	2	8	$6 + 3$	$8 + 3$ ✗

Figure 3.3: Example of one-sided vs. two-sided filtering. Tuples ticked off (✗) are the ones being filtered out.

To further illustrate the differences between one-sided and two-sided bounds, consider the following example. Say that agent i has received a message $M^r_{j\to i}$ containing a single constraint g as shown in Figure 3.3. Furthermore, agent i knows that the cost of the optimal solution is smaller than or equal to 10 ($UB = 10$). Now agent i wants to send functions $M^{r+1}_{i\to j} = \{f_1, f_2\}$ to agent j (also shown in Figure 3.3). Consider that it starts by sending constraint f_1. At this point, the agent employs some form of filtering to try to remove suboptimal assignments from f_1 before sending it.

When using the one-sided lower bound in Equation (3.2), the computed contribution of the i-subproblem is $c = f_1(\mathbf{s}_i)$. After adding this cost to the contribution of the j-subproblem ($c' = g[\{x,y\}] = g$), the agent realizes that assignment $\langle x = b, y = a\rangle$ is suboptimal and can be filtered out as shown in Figure 3.3.

Alternatively, the agent can compute the two-sided lower bound using Equation (3.5). In this case it assesses the lower bound on the contribution of its own subproblem as $c'' = f_1 \bowtie f_2[\{x,y\}]$. Figure 3.3 shows that, in this example, two-sided filtering realizes that, in fact, assignment $\langle x = a, y = b\rangle$ is the only feasible one.

This example illustrates the potential advantages of two-sided filtering. Nonetheless, such advantages come at the expense of some computational overhead. Namely, assessing the i-subproblem contribution using Equation (3.6) takes time $O(d^{|S_i|})$ (the number of assignments in f_i), where d is the maximum domain size of the functions in S_i. In contrast, computing the i-subproblem contribution for the two-sided lower bound takes time $O(md^{|S_i|})$, where m is the number of functions in the message to send $M^{r+1}_{i\to j}$. Hence, it remains to be seen whether the gains from filtering more assignments surpass the additional computational cost or not. To this end, the following section presents an empirical evaluation where we compare both approaches to determine whether it is worth performing two-sided filtering, or it is generally better to stick to one-sided filtering instead.

3.4.1 Empirical evaluation

In this section we empirically compare the performance of DIMCTEf when using one-sided filtering and two-sided filtering. For each experiment, we track the amount of communication used by the algorithm (i.e., the total number of bytes) along with the total amount of serial computation (i.e., the number of non-concurrent constraint checks). Moreover, we performed signed rank tests [Wilcoxon, 1945] on all results to ensure that differences between methods are statistically significant ($\alpha = 0.01$).

As explained in Section 3.3.1, the JT's treewidth is one of the most important indicators of problem hardness for GDL-based algorithms. Hence, we segmented our experiments according to this parameter, and ensured that all algorithms use the very same JT when solving the same problem instance. These JTs were generated using the Distributed Junction Tree Generator algorithm described in [Vinyals et al., 2010b]. Given a DCOP instance, we generated a JT with as many nodes as variables. The Distributed Junction Tree generator algorithm is initialized assigning one variable to each node. After enforcing the running intersection property, other variables may appear in a node, but no variable may disappear. Then, the JT has as many nodes as variables, fulfilling the common assumption of one agent per variable.

In our first experiment we used the meeting scheduling and sensor networks datasets [Maheswaran et al., 2004b] from the USC DCOP repository [Yin, 2008]. Unfortunately, these problems are of low treewidth, and hence are easy to solve for the GDL with function filtering algorithm. Therefore, we obtained similar results for both one-sided and two-sided filtering, with only marginal gains for two-sided filtering.

As a consequence, we decided to design new datasets harder than those typically used in the DCOP literature. We characterized each scenario by three parameters: number of variables, variables' domain size, and treewidth. For each scenario, we generated 100 problems by: (1) randomly drawing problem structures following an Erdös-Rényi $G(n,p)$ model [Bollobas, 2001]; (2) selecting those structures having the treewidth requested for the scenario; and (3) randomly drawing costs from a $\mathcal{N}(0,1)$ distribution.

First, we ran an experiment to evaluate the savings as the treewidth increases. We generated scenarios with 100 variables of domain 8, and treewidths ranging from 6 to 9. Figure 3.4 shows that two-sided filtering reduces, with respect to one-sided filtering, the amount of communication required by a median of 26% on the easier problems (treewidth 6). Furthermore, it achieves even better results on the harder problems (a median of 52% less communication on the problems of treewidth 9).

Next, we designed an experiment to measure the trend of both filtering styles as the variables' domain sizes increase. Thus, we generated scenarios with 100 variables, treewidth 9 and domain sizes ranging from 2 to 8. Once again, two-sided filtering achieves significant communication savings for all the experiment's problems. Further, as the domain increases, so do the savings with respect to one-sided filtering: starting with a narrow 8% reduction for the binary variables

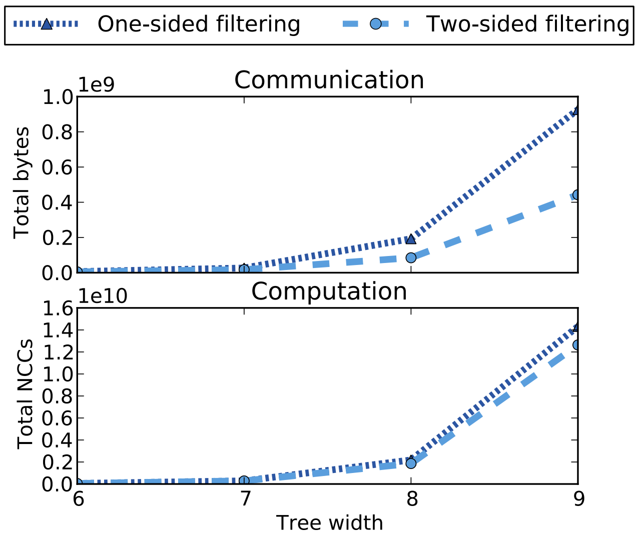


Figure 3.4: One-sided vs two-sided filtering results. Increasing treewidth, constant domain 8 and 100 variables.

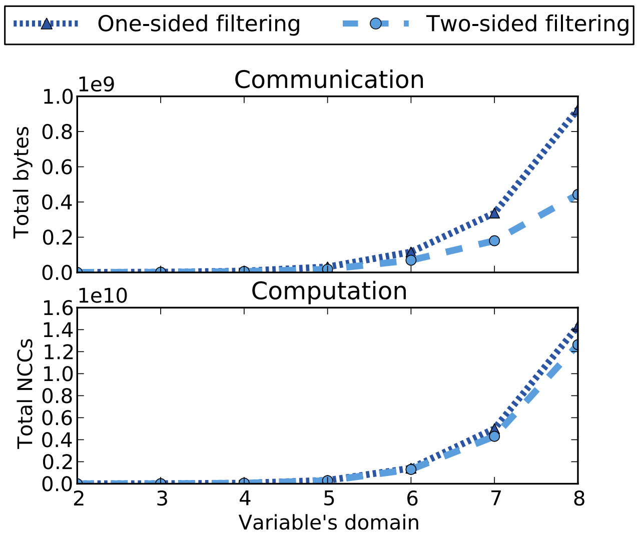


Figure 3.5: One-sided vs two-sided filtering results. Increasing domain size, constant treewidth 9 and 100 variables.

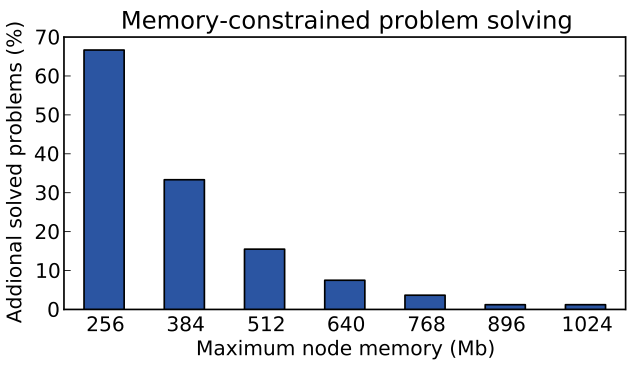


Figure 3.6: Additional problems solved by two-sided filtering w.r.t one-sided filtering when agents have limited memory.

set, and reaching a 52% reduction for the toughest scenario (domain size 8). Furthermore, note that in all but the easiest experiments (the ones with variables' domains 2 to 4), two-sided filtering performs up to 15% less non-concurrent constraint checks. This effectively answers our concerns, indicating that performing two-sided filtering is actually worth it in the vast majority of cases. That is, the savings from filtering more assignments surpass the computational overhead of two-sided filtering with respect to one-sided filtering.

Because the size of the constraints to exchange is the main limiting factor of GDL-based algorithms, this suggests that two-sided filtering can aid to solve problems that are too hard for one-sided filtering. Therefore, we re-ran the hardest set of problems (domain size 8, treewidth 9), but now limiting the maximum amount of memory available for each agent. Figure 3.6 shows that, indeed, two-sided filtering solves as much as 67% more problems than one-sided filtering when the agents have limited memory.

3.5 Improving upper bounds

In the previous section we focused on finding purely algorithmic improvements to compute better lower bounds. However, the global upper bound is also important in the filtering process because it determines the cutoff cost over which assignments may be filtered. Hence, in this section we now focus on trying to improve such upper bound.

Recall that the cost of any complete assignment is an upper bound on the cost of the optimal solution. Therefore, a reasonable approach to find better upper bounds is to explore multiple candidate solutions at the same time instead of a single one. Hence, in the following we first introduce different approaches that allow us to perform such exploration. Furthermore, the proposed approaches are able to compute many solutions in parallel without requiring more messages than to compute a single solution. Afterwards, we experimentally evaluate these novel approaches, showing that the benefits of providing the filtering process with better upper bounds can outweigh the cost of computing them on some occasions.

3.5.1 Centralized exploration

The simplest approach to propagating multiple assignments is to perform the very same procedure as DMCTE(r) does, but with multiple assignments instead. This is, the root node begins by choosing the best m assignments for its variables, and subsequently sends them to its children. Thereafter, each child extends each assignment by choosing its best cost extension (according to its knowledge), and relays them to its own children. The solution propagation phase terminates once each leaf node has received (and extended) its assignments.

Then, agents need to calculate the cost of each solution. With this aim, there is a third phase where: (1) the cost of each assignment is aggregated up the tree; and (2) the best assignment and its cost (the new global UB) are sent down the tree. Firstly, each leaf node i evaluates the cost of each assignment in its problem's stake C_i, and sends the resulting costs to its parent. Subsequently, once a parent node j receives the costs of each assignment from its children, it aggregates them with the costs in its own problem stake C_j. Thereafter, it sends the resulting costs up the tree. After the root has received and aggregated the costs from all its children, it can readily identify the best assignment (the one with lowest cost). Finally, the root sends the best assignment along with its cost down the tree.

The main advantage of this method lays in its simplicity. However, its main drawback is that it offers limited exploration capabilities because: (1) it cannot propagate more than k candidate solutions, where k stands for all possible assignments for the root's variables; and (2) when a node finds several good extensions for a candidate solution, it is enforced to choose only one of them. For instance, say that an agent receives assignment $\langle x = a \rangle$ from its parent, and has to choose a value for variable y. According to its knowledge, extension $\langle x = a, y = a \rangle$ costs 1, and so does extension $\langle x = a, y = b \rangle$. Because centralized exploration forces the agent to extend each received assignment exactly once, extension $\langle x = a, y = b \rangle$ must be discarded. This restriction implies that the root is the only node able to explore new candidate solutions, whereas other nodes simply exploit them.

3.5.2 Distributed exploration

To overcome the limited exploration capabilities of centralized exploration, we need mechanisms allowing any node to explore new assignments.

However, we still need to somehow limit the amount of assignments to explore. Notice that, in the centralized exploration method above, the number of assignments to explore is effectively limited by the m initial assignments chosen by the root agent. Likewise, we now impose the restriction that each agent cannot consider more than m assignments. Furthermore, we enforce nodes to extend each received assignment at least once. However, we allow each agent to extend any partial assignment received from its parent multiple times.

Consequently, after a node receives a set of assignments A, it needs to decide the number of new assignments to explore n_e, which cannot exceed $n_{max} = m - |A|$. With this aim, we propose that an agent employs one of the following strategies:

Greedy exploration. An agent always extends as many assignments as possible, namely $n_e = n_{max}$. This approach is similar to the centralized exploration strategy above, but overcomes its impossibility to explore more than k solutions.

Stochastic exploration. An agent chooses the number of assignments to extend n_e from a binomial distribution $B(n_{max}, p)$, where $p \in (0,1]$ is the ratio of assignments to extend. Intuitively, larger p values favor exploitation, whereas lower p values favor exploration.

Notice that it is possible that the number of extensions requested n_e is larger than the number of possible extensions (specially when using greedy exploration). In that case the agent will communicate every possible extension. The process to calculate the cost of each solution is analogous to the one described for centralized exploration. The difference lies in the aggregation of costs up the tree. Although an agent may extend a parent's assignment multiple times, during the bound propagation phase the agent sends up only the cost of the best extension of each assignment out of the different extensions it has tried. In this manner, the parent never needs to know which assignments have been extended multiple times by its children.

Using these mechanisms, the agents can consider a large number of assignments without a hefty computation and communication overhead. Nonetheless, it remains to be seen whether the overhead is offset by the gains from checking more candidate solutions and thus having possibly better upper bounds. In the following section we strive to answer this question by empirically evaluating the novel multiple solution exploration techniques presented here against the state-of-the-art single-solution upper bound computation.

3.5.3 Empirical evaluation

To assess the performance of GDL with two-sided function filtering and the tighter upper bounds obtained by propagating multiple solutions, we ran a number of experiments on the same scenarios we used in Section 3.4.1. Specifically, we assessed the communication and computation savings obtained by: (1) centralized exploration; (2) greedy distributed exploration; and (3) stochastic distributed exploration. Regarding the stochastic case, we empirically observed that different exploration ratios (different values for p), do not lead to very significant differences when filtering. Here we set p=0.1 because it provided slightly better results.

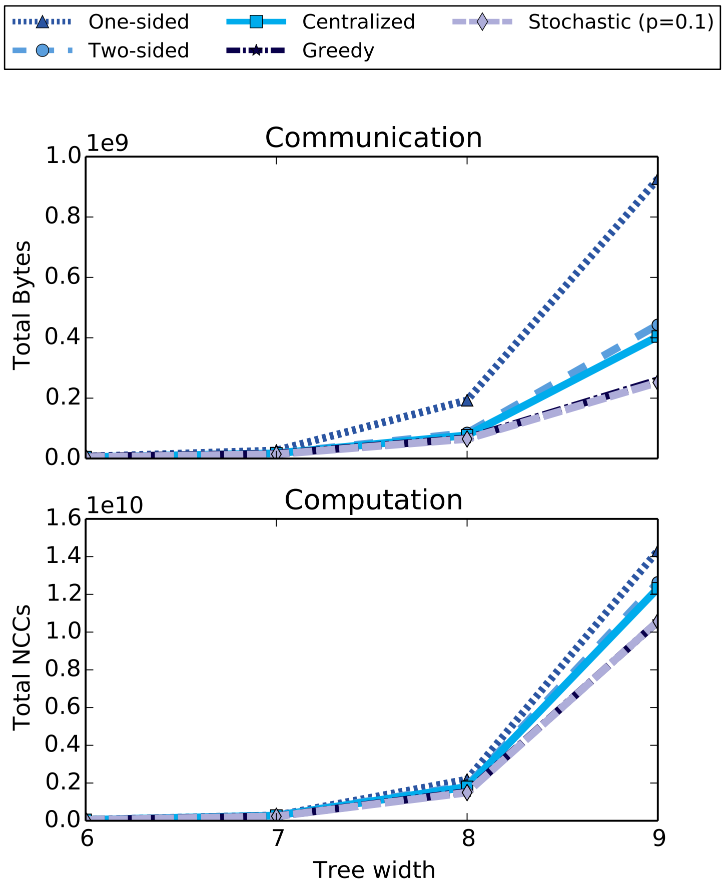


Figure 3.7: Multiple solution exploration results: increasing treewidth, constant domain 8 and 100 variables.

Note that, the harder the problem, the cheaper to propagate multiple solutions with respect to the cost propagation phase. Hence, we ran our experiments with different numbers of propagated solutions, and found that propagating 1024 solutions achieved the best overall results on our problems. This is, propagating less than 1024 solutions slightly decreased the computation and communication used when solving the easier problems, but significantly increased when solving the harder ones. Likewise, propagating more solutions led to no additional savings on harder problems, while increasing costs on easier ones.

Figure 3.7 shows the evolution of the median results as the treewidth of the problems increases. On the one hand, centralized exploration achieves between 1 and 4% extra communication savings on top of two-sided filtering. On the other hand, both greedy and stochastic exploration obtain similar results (with stochastic being very slightly better). Nonetheless, they clearly outperform centralized exploration, consistently saving a median 20% communication cost, for a grand total of up to 72% savings when compared to the state-of-the-art one-sided filtering. Figure 3.8 displays very similar trends as the variables' domain grows. Centralized exploration provides a low reduction in communication with respect to two-sided filtering, whereas greedy and stochastic exploration strategies obtain up to 24% extra savings.

It is important to note that both greedy and stochastic exploration further reduce the number of non-concurrent constraint checks by as much as 24%. Furthermore, the reduction of computational effort goes up to 32% once multiple solutions propagation strategies are combined with two-sided filtering. Figure 3.9a reveals the effect of this reduction on the number of problems that can be solved when nodes have limited memory. Specifically, using two-sided filtering with distributed exploration helps solve up to 75% more problems than one-sided filtering in the most restricted scenarios.

Finally, we also experimented with the meeting scheduling problems from the USC DCOP repository. Unlike what we observed with two-sided filtering, in this case performing multiple solution exploration actually worsened the results by as much as 15% in communication costs and 20% in computation costs. That is, on easier problems, the overhead of performing multiple solution exploration (with any of the above methods) is larger than the gains obtained by having tighter upper bounds. In fact, we even experimented by seeding the function filtering procedure with the optimal cost (the best possible upper bound) at the beginning of the algorithm. With these problems, the results were discouraging: even with the optimal upper bound obtained at no cost, the efficiency of the algorithm did not improve significantly. The explanation for such results is that, on easy problems, the lower bounds computed at the first iterations are not tight enough to perform any filtering. In contrast, the lower bounds computed at later iterations are already good enough to filter most assignments even with a looser upper bound. As a consequence, any effort to tighten the upper bound in this situation is actually futile.

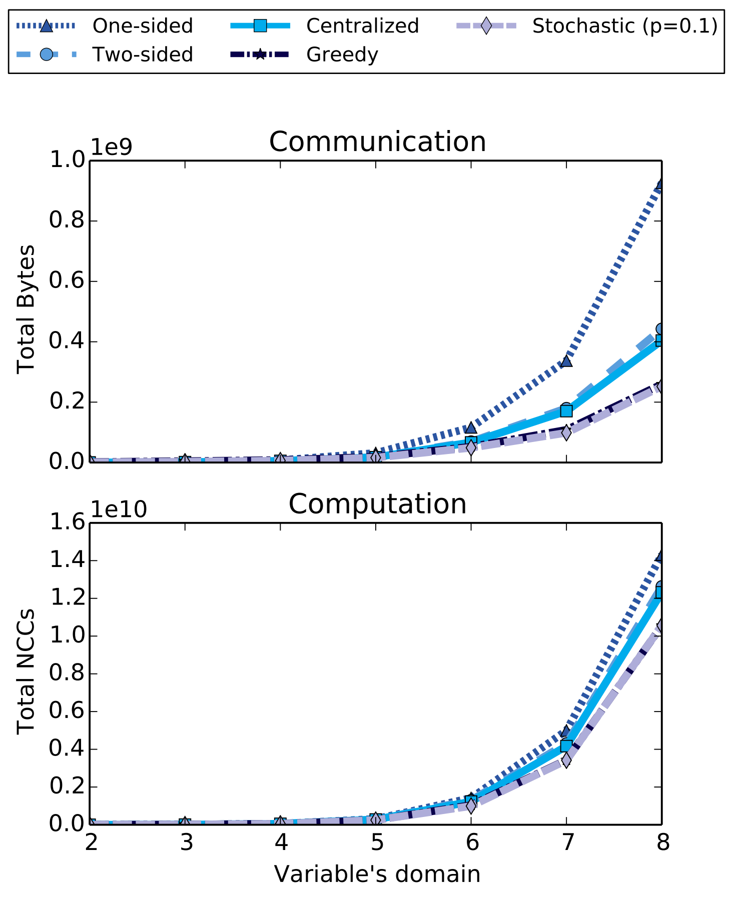
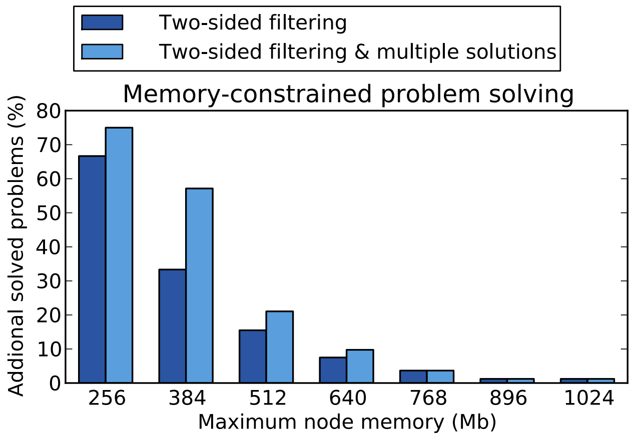


Figure 3.8: Multiple solution exploration results: increasing domain size, constant treewidth 9 and 100 variables.



(a) Additional problems solved when agents have limited memory (w.r.t. one-sided filtering).

Figure 3.9: Experimental results when tightening the upper bound.

3.6 Conclusions

Questions addressed in this chapter:

Q. 1. Can we identify application characteristics that provide cues as to which solving algorithm is better for that application?

Q. 2. Can we improve the resource scalability of DCOP algorithms for which this scalability is a limitation?

The broad analysis performed in Section 2.3 identified the GDL-based family of algorithms as one of the most promising approaches for optimal DCOP solving. The worst case complexity bound on the problem's treewidth (instead of on the total number of decisions to make) readily answers our Question 1, showing that these algorithms are an excellent fit for a number of easily identifiable problems. Hence, in this chapter we focused on GDL with function filtering [Sánchez et al., 2005] because it stands as the most advanced state-of-the-art variant among the family of GDL-based algorithms.

Accordingly, we presented a number of techniques to increase the scalability of GDL with function filtering. Recall that function filtering is a technique that reduces the size of cost functions by filtering out assignments that cannot be extended into an optimal solution. As a consequence, the size of the constraints that agents need to exchange during the algorithm execution can be largely reduced. Such reductions provide two significant benefits. On the one hand, smaller constraints imply that agents have to compute and communicate less information, thus reducing the computational requirements and execution time of the algorithm. On the other hand, the size of the constraints is the major limiting factor of the GDL algorithm. That is, in the worst case, agents running a GDL algorithm have to compute a constraint of d^t assignments, where t is the treewidth of the JT. Now, if function filtering manages to significantly reduce the number of assignments required within that function, then the algorithm will scale to larger problems with larger treewidths. As a result, a potential approach to answer Question 2 for the GDL with function filtering algorithm is to improve the filtering quality.

As explained in Section 3.3.4, the effectiveness of function filtering heavily depends on the quality of the computed lower and upper bounds. Therefore, in this chapter we improved the effectiveness of state-of-the-art function filtering by providing techniques to assess tighter lower and upper bounds.

First we presented a novel lower bound computation method, the so-called two-sided filtering bound. Two-sided filtering improves the quality of the lower bounds computed during the filtering process by taking into account more information than the previous state-of-the-art one-sided filtering method described in [Brito and Meseguer, 2010b]. Our experiments showed that in the worst case, when solving easy problems such as the meeting scheduling problems from the USC DCOP repository, two-sided filtering does not provide significant advantages yet it does not worsen the algorithm's performance either. In contrast, experiments with harder problems showed that two-sided function filtering can lead to significant reductions in the amount of resources required to optimally solve DCOPs. Namely, two-sided filtering achieved reductions of up to 52% less communication and 15% less computation in the toughest problems.

Next we introduced several techniques to compute better upper bounds. Because the cost of each explored solution can be readily used as an upper bound for the filtering process, our techniques are all based on exploring multiple solutions instead of a single one. We empirically evaluated these novel techniques, showing that they can further improve the efficiency of GDL with function filtering. Used in combination with two-sided filtering, distributed stochastic exploration can reduce up to 72% on communication costs and up to 32% on computational costs on the hardest problems. Unfortunately, we have also found out that in easier problems, such as the meeting scheduling ones from the USC DCOP Repository, the overhead of exploring multiple solutions is not compensated by the marginal benefits obtained in the filtering process.

Finally, these algorithmic improvements to the filtering process also obtain a significant memory reduction, allowing agents to solve up to 75% more problem instances given the same resource constraints. As a consequence, we increased the range of problems that can be optimally solved by GDL-based algorithms, positively answering our Question 2.

Chapter 4

Scaling by resource trade-offs

4.1 Introduction

In the previous chapter we introduced a number of techniques to improve the effectiveness of function filtering, and hence increase the scalability of the GDL with function filtering algorithm. All those techniques are purely algorithmic improvements aimed at lowering the amount of resources required by the algorithm. However, there is another, more indirect approach to increase the practical scalability of GDL-based algorithms. In actual-world settings, DCOP algorithms must be executed by the agents in some application domain. Moreover, the agents' computation and communication capabilities may greatly differ between applications. Hence, another way to improve the applicability of a DCOP algorithm is to enable agents to trade-off their computation and communication resources to better match the algorithm to their particular capabilities.

For instance, consider two applications widely used in the DCOP literature: meeting scheduling and sensor networks. In a meeting scheduling scenario, agents represent meeting participants and are actually programs running on their personal computers. Hence, the agents' computation capabilities are those of a typical desktop machine, and their communication capabilities range from those of a company LAN to broadband internet connections. In contrast, agents in a sensor network are the wireless sensor network nodes. These nodes employ embedded processors which are significantly slower than desktops. However, their communication capabilities are even more stringent. On the one hand, they must employ small radios that provide high-latency and low bandwidth connections. On the other hand, these devices typically operate on battery power, and hence must try to reduce energy usage as much as possible. Because using radios is the most expensive operation energy-wise, in such applications it is preferable to perform more computation in exchange for reduced communication.

Therefore, in this chapter we direct our efforts at devising ways to adjust the algorithm's computation and communication usage for particular applications. Notice that, in the previous chapter, we realized that the quality of the lower bounds computed during the algorithm's execution has a much larger impact on its performance than the quality of the upper bounds. Moreover, we studied how to compute the tightest possible lower bounds given the message that an agent is about to send and those it received in the previous iteration. Nonetheless, we did not explore how exactly such outgoing messages are computed. On this matter, Rollon *et* al. [Rollon and Dechter, 2010] introduced a scheme for the definition of message computation strategies, and studied how those strategies affected the approximate and centralized MCTE algorithm. However, because MCTE is a centralized algorithm, such scheme is designed to produce computationally efficient strategies while ignoring potential communication costs. Furthermore, MCTE does not employ filtering because it is a single-iteration approximate algorithm.

Hence, our initial contributions in this chapter are:

- We port the scheme in [Rollon and Dechter, 2010], which we name *bottom-up approximations*, to the DIMCTEf (distributed GDL with function filtering) algorithm.
- We then develop a novel scheme, called *top-down approximations*. This new scheme explicitly aims at the development of communication-efficient message computation strategies instead of computation-efficient ones. As a result, top-down approximation strategies are particularly suited for communication-constrained applications. Additionally, we present the *brute-force* and the *zero-tracking decomposition* strategies as particular realizations of this framework.
- We empirically evaluate the overall computation time and communication costs of all these strategies on several experiments. The results show that DIMCTEf employing top-down approximations achieves much larger communication savings with respect to classical GDL than bottom-up ones. Furthermore, zero-tracking decomposition achieves such savings while still being competitive in computational effort with respect to current state-of-the-art approximation methods [Dechter and Rish, 1997; Rollon and Dechter, 2010].

These contributions are already a significant step towards answering our Question 2 in the introduction, because they significantly increase the scalability of GDL on communication-constrained applications. Nevertheless, all these strategies are basically all-or-nothing approaches, where we can either heavily favor the computational cost or the communication cost, but not seek some balance between them to match the application at hand. Thus, in the remainder of this chapter we aim at finding out how to combine strategies to realize such trade-off between computation and communication costs. With this aim we begin by performing an in-depth analysis of how the different GDL-based approaches introduced in Section 3.3 compute their messages. We observe that each algorithm implements differently some trade-off between computation and communication. However, to the best of our knowledge there has not been any study in the literature that analyzes which strategies are more adequate depending on the resources available in different application domains.

Against this background, we then make the following further contributions:

- Based on a thorough analysis of the GDL-based DCOP algorithms in the literature, we abstract a general cost message computation model that considers limits on the usage of resources. We show that our general scheme effectively captures the aforementioned algorithms as particular instances. Our general scheme is intended to help design new message computation methods that can effectively trade off communication and computation according to a user's requirements.

- Based on our general scheme, we identify three "families" of message computation methods that are expected to yield different resource trade-offs, tailored to cope with the specific resource availabilities on different applications.

- Finally, we provide empirical guidelines to select the most appropriate message computation method depending on an application's available resources. Furthermore, our empirical analysis identifies one of our novel methods as the best available one for computationally constrained scenarios, and another one as the best for heavily communication-constrained scenarios.

The remainder of this chapter is structured as follows. In Section 4.2 we first introduce the problem of computing GDL messages as approximations of the constraint that Action-GDL would compute. Then, we present the so-called *bottom-up approximations* framework to compute messages within a strict communication bound. We also introduce the novel *top-down approximations*, that heavily favor reducing the amount of communication, and empirically compare both approaches. Thereafter, in Section 4.3 we provide a detailed analysis of how current GDL-based algorithms compute their messages, and introduce our novel message computation scheme that generalizes all those algorithms. Afterwards, in Section 4.4 we show how this novel scheme can be used to derive novel GDL-based algorithms that provide different trade-offs between computation and communication costs. Additionally, we present an extensive empirical evaluation that serves as a guideline to highlight which methods work better on different application types, characterized by the communication and computation capabilities of the involved agents. Finally, in Section 4.5 we draw some conclusions and explain how the obtained results help us answering our research questions in the introductory chapter.

4.2 Communication-efficient message approximation strategies

In this section we take a closer look at how GDL-based algorithms compute messages during the cost propagation phase. First we describe how classical GDL algorithms (Action-GDL, DMCTE) compute such messages. Next we review the message approximation scheme introduced in [Rollon and Dechter, 2010], which aims to find computationally-efficient lower-bound approximations to the messages computed by Action-GDL. Afterwards we present a novel scheme designed to produce strategies to compute communication-efficient approximations instead of computation-efficient ones. We also present two novel strategies within such scheme, the *brute-force* and *zero-tracking* decomposition strategies.

During the cost propagation phase, a GDL agent's task is to propagate constraints along the Junction Tree. These constraints represent lower bounds on the costs contained in the subproblem of the sending agent (the slice of the JT rooted at that agent). In classical GDL, an agent i computes and sends the tightest possible lower bound to its parent j by simply combining all the constraints received from its children with its own constraints C_i, and then projecting the result to the variables on the separator with its parent S_{ij}. Formally, the message computed by agent i for agent j is:

$$m_{i \to j} = (m_{\widehat{j} \to i} \bowtie C_i)[S_{ij}] \ , \tag{4.1}$$

where $m_{\widehat{j} \to i}$ is the combination of all messages received by agent i except the one from j, namely:

$$m_{\widehat{j} \to i} = \mathop{\bowtie}_{k \in N(i) \setminus \{j\}} m_{k \to i} \ ,$$

where $N(i)$ is the set of neighbors of node i in the JT.

However, recall from Section 3.2.2 that computing the tightest lower bound of a set of constraints is an exponentially costly operation. Hence, the approximate algorithm DMCTE(r) relaxes this phase by sending a tight lower bound instead of the tightest lower bound, so that less information needs to be computed and sent.

DMCTE(r) [Brito and Meseguer, 2010a] is an extension of the centralized MCTE(r) algorithm. However, notice that MCTE(r) has no communication costs because it is centralized. Therefore, the r bound in MCTE(r) is a pure computation bound. Namely, the algorithm cannot compute constraints of more than r variables. In contrast, being a distributed algorithm, DMCTE(r) does have communication costs. Hence, it uses a different interpretation of the r bound. Specifically, r is a bound on the maximum number of variables involved in the constraints sent between agents, but the algorithm can compute constraints of any arity. That is, agents in DMCTE(r) can compute constraints of any number of variables, but can only communicate constraints of at most r variables.

Now consider an agent operating in DMCTE(r). Eventually, it will receive messages from all its children in the JT. Then, the agent combines this information with its own, projects the result over the variables in the separator, and finally sends the result to its parent. Nevertheless, since the agent is now constrained by the arity limit r, it can not send the tightest lower bound and it has to compute an approximation. Furthermore, notice that the tighter the computed lower bounds, the better the results of DMCTE(r) given a fixed parameter r. Hence, our goal is to find a lower bound as tight as possible while communicating only constraints of at most r variables. With this aim, we first introduce some further definitions.

Definition 4.1. Given a constraint f with scope V, we say that *a set of constraints F is a V-lower bound of f* if and only if the combination of F under V is a lower bound of f.[1] That is, if and only if $\bowtie(F[V]) \leq f$.

This definition is useful because it allows us to speak about a set of constraints as a lower bound of the exact constraint that classical GDL would compute as shown in Equation (4.1).

Observation 4.1. By definition, the projection of F under V is a V-lower bound of the combination of F over V. That is, $F[V]$ is necessarily a V-lower bound of $(\bowtie F)[V]$. Furthermore, the set containing the single constraint $(\bowtie F)[V]$ is the tightest V-lower bound of F.

However, since we also need to consider the r arity communication limit, we introduce a more refined definition:

Definition 4.2. Given a constraint f with scope V, *a set of constraints F is an (r,V)-lower bound of f* if and only if F is a V-lower bound of f, and the arity of each function in $F[V]$ is smaller than or equal to r. That is, if and only if

$$|sc(g)| \leq r \text{ for all } g \in F[V] \; .$$

Given these definitions, we can now precisely describe the message (analogous Equation (4.1)) that an agent must compute during the operation of to DMCTE(r). Namely, the message that agent i computes for agent j in DMCTE(r) is an as tight as possible (r,S_{ij})-lower bound of the constraint set $F = M_{\widehat{j}\to i} \cup C_i$, where

- $M_{\widehat{j}\to i} = \bigcup_{k\in N(i)\setminus\{j\}} M_{j\to i}$ is the set of all constraints contained in all messages i has received from its neighbors except from j.
- C_i is the set of constraints that represents agent i's own stake in the problem.

Finally, this allows us to formalize the idea of a message approximation strategy as a procedure to compute DMCTE(r) messages, namely:

Definition 4.3. A *message approximation strategy* is a procedure that receives as input a set of constraints F, a set of variables V, and an arity limit r. Then its goal is to find an as tight as possible (r,V)-lower bound for the combination of F over V.

[1]Recall from Section 3.2.3 that the combination of F under V is the combination of the independent projections of each function $f \in F$ over V.

4.2.1 Bottom-up approximations

As explained before, some algorithms for this task have already been proposed in the literature [Dechter and Rish, 1997; Rollon and Dechter, 2010]. Hence, we now review them to fit them into a common framework which we call bottom-up approximation methods.

Informally, the fundamental idea behind bottom-up approximations is the following. If we combine any pair of functions from a set that is a V-lower bound, the result is another V-lower bound, most times tighter than the former one. Thus, given an initial (r,V)-lower bound F we can improve it by simply combining pairs of constraints in F so long as the result is also an (r,V)-lower bound.

To realize this idea, we first extend Observation 4.1 to take the r-arity communication limit into account:

- $(\bowtie F)[V]$, the combination of F over V, is the tightest possible V-lower bound. Also, it is an (r',V)-lower bound of F where $r' = |sc(\bowtie F)| \geq |V|$.
- $F[V]$, the projection of F under V, is a looser V-lower bound of F. Moreover, it is also an (r'',V)-lower bound, where r'' is the maximum number of variables of V in the scope of a single constraint $f_i \in V$. That is, r'' is typically smaller than r' and can be assessed as
$$r'' = \max_{f_i \in F} |sc(f_i) \cap V| \leq |V| \ .$$

Next we introduce two further definitions that allow us to compute (r,V)-lower bounds that lie between those two above in terms of bound tightness and r-arity communication limit.

Definition 4.4. For any $f_a, f_b \in F$, *the combination of* f_a *and* f_b *in* F, noted $F_{f_a \bowtie f_b}$, is the set of constraints that results from replacing f_a and f_b by $f_a \bowtie f_b$, namely
$$F_{f_a \bowtie f_b} = (F \setminus \{f_a, f_b\}) \cup \{f_a \bowtie f_b\} \ .$$

Definition 4.5. Two constraints f_a *and* f_b *are (r,V)-combinable* if and only if the combination of $\{f_a, f_b\}$ over V is a constraint of arity smaller than or equal to r. That is, any f_a and f_b such that $|(sc(f_a) \cup sc(f_b)) \cap V| \leq r$ are (r,V)-combinable.

At this point we can accurately describe the bottom-up approximations intuition above. To ease the explanation, notice that for a given message approximation, the set of variables V is fixed to S_{ij}. Thus, we write r-lower bound and tightest bound without explicitly mentioning V.

Algorithm 1 shows the pseudocode for bottom-up approximations. Since by Observation 4.1 we know that F is a lower bound of the combination of F over V, a bottom-up algorithm starts from the original set of constraints F. At each iteration, the algorithm: (1) selects a pair of r-combinable constraints f_a and f_b from the current set of constraints; and (2) updates the set of constraints to the

Algorithm 1 Bottom-up approximation(F, V, r)

```
1: $(found, (f_a, f_b)) \leftarrow$ selectCombinablePair$(F, V, r)$
2: while $found$ do
3:     $F \leftarrow F_{f_a \bowtie f_b}$
4:     $(found, (f_a, f_b)) \leftarrow$ selectCombinablePair$(F, V, r)$
5: end while
6: return $F$
```

$$\begin{aligned} F &= \{f_1(x,t), f_2(y,t), f_3(z,t)\} \\ V &= \{x, y, z\} \\ r &= 2 \end{aligned}$$

$x\,t$	f_1
$a\,a$	2
$a\,b$	1
$b\,a$	3
$b\,b$	2

,

$y\,t$	f_2
$a\,a$	4
$a\,b$	1
$b\,a$	2
$b\,b$	2

,

$z\,t$	f_3
$a\,a$	0
$a\,b$	2
$b\,a$	0
$b\,b$	1

Figure 4.1: Example functions to approximate.

combination of f_a and f_b in F, that is $F_{f_a \bowtie f_b}$. Since $F_{f_a \bowtie f_b}$ is also an r-arity lower bound and it is at least as good as F, the iterations are likely to improve the lower bound. When no more pairs of r-combinable constraints are found, the algorithm returns the approximation represented by the last F.

Hence, the difference between bottom-up approximation strategies lies on how to select which pair of r-combinable constraints to combine next. That is, on how each particular strategy implements the `selectCombinablePair` function in Algorithm 1. In the following we describe the most notable strategies from this scheme.

Scope-based partitioning

Scope-based partitioning (SCP) is the most common bottom-up strategy [Dechter and Rish, 1997; Brito and Meseguer, 2010b]. Basically, it tries to combine as many constraints as possible by choosing the two highest arity functions at each iteration, so long as they are r-combinable.

More in detail, the r-combinable pairs are selected as follows. First, the set of functions F is sorted decreasingly by arity and each function in the list is marked as non-finished. At each iteration, SCP takes the first non-finished element f_1 of F and the element f_i of F closer to the head such that f_1 and f_i are r-combinable. It removes them from F and inserts its combination at the head of the list. When there is no function f_i r-combinable with f_1, it marks f_1 as finished. The algorithm proceeds until all functions are marked as finished.

Figure 4.2a depicts how SCP would compute an approximation for the example in Figure 4.1. Since all functions in F have the same arity (two), they are readily sorted. Hence, SCP would merge the two leftmost ones, and send the third one independently, resulting in the approximation:

$$F' = \{(f_1 \bowtie f_2)[xy], f_3[z]\} = \left\{ \begin{array}{c|c} x\,y & f_1 \bowtie f_2[xy] \\ \hline a\,a & 2 \\ a\,b & 3 \\ b\,a & 3 \\ b\,b & 4 \end{array} \quad , \quad \begin{array}{c|c} z & f_3[z] \\ \hline a & 0 \\ b & 0 \end{array} \right\}$$

The main advantage of SCP is the low computational complexity that results from its simplicity. The algorithm performs a nested scanning through the list of constraints. During this scanning process, the algorithm computes up to $|F| - 1$ constraint combinations. To determine the maximum arity of these functions, consider that J is the joint domain of all constraints in F. Then, the number of variables that do not appear in V is $|J \setminus V|$. Since the arity limit is r, the maximum arity of each merged constraint is $|J \setminus V| + r$. As a result, the complexity of the algorithm is $O(|F| \exp(|J \setminus V| + r))$.

Content-based partitioning

A major advantage of scope-based partitioning is its small computational overhead. Nonetheless, its main drawback is that it does not consider the information within each function. For instance, consider again the previous example. We showed that scope-based partitioning would produce partition F' above. However, notice that there are further approximations that satisfy the r-bound.

Content-based partitioning techniques guide the bottom-up approximation by consulting the functions' contents in addition to their scopes. In general, content-based partitioning tries to assess which pair of r-combinable functions yield the largest improvement.

In [Rollon and Dechter, 2010] Rollon and Dechter present a framework for content-based partitioning that implements the general approach outlined in Algorithm 1. Given an r-lower bound F, content-based partitioning provides the mechanism for selecting the best r-combinable pair of constraints $f_1, f_2 \in F$ such that the approximation represented by $F_{f_1 \bowtie f_2}$ is better than the approximation represented by F. At each iteration, the technique: (1) generates every pair of r-combinable constraints $f_a, f_b \in F$; (2) measures the gain obtained by combining f_a and f_b; and (3) selects the pair that maximizes the gain.

Notice that the advantage of combining two constraints before sending them is that they will be projected together. Hence, the gain can be calculated based on the difference between projecting together or projecting separately, which can be computed as:

$$f = (f_a \bowtie f_b)[V] - (f_a[V] \bowtie f_b[V]).$$

Therefore, the gain function is a metric that takes a constraint f as its input. Rollon and Dechter present two such functions.

Firstly, the local relative error (LRE) metric, which is equivalent to the averaged 1-norm of f, assesses the gain as the sum of costs of all assignments in f divided by the total number of assignments in the constraint.

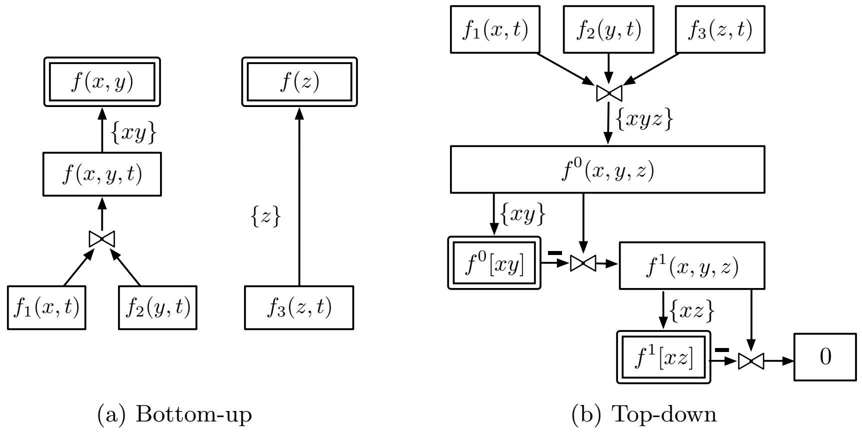


Figure 4.2: Examples of approximation strategies. Constraints in a double-lined box are the ones finally sent.

Secondly, the local maximum relative error (LMRE) metric, which is equivalent to the ∞-norm of f, assesses as gain the maximum cost among all assignments in f.

The downside of content-based decomposition is that, in the worst case, the algorithm performs up to $|F|-1$ selections, computing $|F|-1$ differences for each selection. Hence, the complexity of the algorithm is $O(|F|^2 \exp(|J \setminus V| + r))$, which is significantly larger than the complexity of SCP.

4.2.2 Top-down approximations

Bottom-up approximation methods focus on lowering computational costs, while we primarily focus on reducing communication costs. With this aim, we propose a new approach to generate approximations based on: (1) initially computing the tightest lower bound, and; (2) subsequently decomposing it into lower arity output functions. Figure 4.2b represents the process of building a top-down approximation of the example in Figure 4.1. As a first step, f_1, f_2 and f_3 are combined over $\{x, y, z\}$ to produce f^0, the constraint to approximate. After that, the decomposition process starts. Firstly, consider that we select $\mathbf{S}^0 = \{x, y\}$ out of all possible subsets of $\{x, y, z\}$ with arity two. Secondly, f^0 is projected to $\mathbf{S}^0$ to produce $f^0[xy]$, and this constraint is subtracted from f^0 to obtain f^1 (namely $f^1 = f^0 \bowtie (-f^0[xy])$).[2] Therefore, f^0 is decomposed as $f^0[xy] \bowtie f^1$, where $f^0[xy]$ can be regarded as a constraint ready to be communicated and f^1 as the *remainder* after communicating $f^0[xy]$. The process continues searching for a decomposition for this remainder.

The main advantage of top-down methods is that they can compute approximations that bottom-up methods cannot, and hence should be able to produce

[2]We define the "$-$" operator as a function that negates the costs of a constraint. It is easy to see that $f \bowtie -f$ is the null constraint (a constraint whose costs are all 0).

Algorithm 2 Top-Down Approximation(F, V, r).

```
1: $f \leftarrow (\bowtie F)[V]$
2: $F' \leftarrow \emptyset$
3: $(found, f') \leftarrow$ selectBestLowerBound($f$,$r$)
4: while $found$ do
5:     $F' \leftarrow F' \cup \{f'\}$
6:     $f \leftarrow f \bowtie (-f')$
7:     $(found, f') \leftarrow$ selectBestLowerBound($f$,$r$)
8: end whilereturn $F'$
```

more accurate results. Unlike bottom-up methods, which start from an initial set of input constraints and proceed by deciding which ones to join, top-down methods start from the constraint to approximate and proceed by successively extracting the best lower-bound constraint of the desired arity. While such constraint is already part of the decomposition, the process continues by further decomposing the result of subtracting the extracted constraint from the function to approximate. In general a top-down approximation method is an iterative procedure that at each step i transforms a constraint to approximate f^i into: (i) a lower-bound constraint $f^i[S^i]$ whose arity is smaller than or equal to r; and (ii) a new constraint to approximate f^{i+1}. At each step, the selected lower-bound constraint $f^i[\mathbf{S}^i]$ is added to the set of output constraints, and the new constraint to approximate is computed as follows:

$$f^{i+1} = f^i \bowtie (-f^i[\mathbf{S}^i]). \tag{4.2}$$

When the iterative process terminates, the following set of constraints stands for the resulting decomposition of f:

$$F = \{f^0[\mathbf{S}^0], f^1[\mathbf{S}^1], \ldots, f^n[\mathbf{S}^n]\}.$$

More in detail, a general top-down approximation method works as outlined in Algorithm 2. First, it computes the constraint to approximate (f) by combining the input constraints in F over V. After that, it uses some heuristic to select the best lower-bound constraint f' of arity at most r. Finally, f' is added to the set of output constraints and subtracted from f to produce the new constraint to approximate. This process is repeated until no lower-bound constraint of arity at most r provides additional information. In the ramainder of the section we introduce two top-down approximation methods that implement the general method outlined in Algorithm 2.

Brute force decomposition

In order to determine the most informative lower-bound constraint, a first approach is to consider every possible projection over r variables from V.[3] Then,

[3]Note that discarding functions whose arity is lower than r does not reduce the space of representable functions.

we can readily use the gain functions from content-based partitioning in Section 4.2.1 such as LRE and LMRE to rank the lower-bound constraints and select the most informative one.

This procedure has, however, a high computational cost. At the first iteration, it must compute $\binom{|V|}{r}$ constraints and evaluate them, each requiring $\exp(|V|)$ operations. At each following iteration, the number of constraints to compute decreases by one (the selected lower-bound constraint is never computed again, but all others have to be reevaluated because f^i is different from f^{i-1}). Hence, its worst case time complexity is $O\left(\binom{|V|}{r}^2 \cdot exp(|V|)\right)$.

Zero-tracking decomposition

The main disadvantage of brute force decomposition is its high computational cost. Here we introduce zero-tracking decomposition, a top-down approximation method that aims at dodging this burden to reduce the computational cost.

Zero-tracking decomposition uses the zero norm of the extracted constraints as the heuristic to assess its quality. The zero norm of a function is simply the number of elements in the domain whose image is not zero. Intuitively, if a function is only composed of zeros, it communicates no information whatsoever. The larger the number of non-zero entries in a function, the more informational it is considered to be.

The reduction in computational cost comes from realizing that, at each iteration, there is a way to compute the zero norms of each possible projection directly from the results of the previous iterations. That is, we can assess how informative each projection is without actually computing it. Therefore, we avoid the need to recompute every possible r-arity lower-bound at each iteration, significantly reducing the computational cost of the decomposition process.

In what follows we detail the operation of the zero-tracking method when applied to the example in Figure 4.1. Let $S = \{\mathbf{S} \subseteq V$ such that $|\mathbf{S}| = r\}$ be all subsets of V of r variables. Let $p = |S| = \binom{|V|}{r}$ be the number of possible r-arity lower-bounds and $q = d^r$ the number of assignments for each r-arity lower bound.[4] First, the algorithm allocates a boolean table $Zeroes$ of p rows and q columns. Each entry $[\mathbf{S}, \mathbf{s}]$ in $Zeroes$ encodes whether the value for assignment $\mathbf{s}$ of the projection of f (the lower-bound to approximate) over $\mathbf{S}$ is zero or not. That is, $Zeroes[\mathbf{S}, \mathbf{s}]$ is true whenever $f[\mathbf{S}](\mathbf{s})$ is zero. Hence, all entries are initialized to *false*. Additionally, it allocates a vector $\vec{c}$ of p integers to count the number of non-zero assignments for r-arity lower bound. Since $\vec{c}$ counts non-zero values, it is initialized to the number of assignments in each projection (q above). At the top of Figure 4.3a we show (for iteration i=0) table $Zeroes$ and vector $\vec{c}$ after initialization. Next, the tightest lower bound $(\bowtie F)[V]$ is calculated by combining all the initial constraints and projecting the result over $\{x, y, z\}$. The result is shown in Figure 4.3b as f^0.

[4]For simplicity of exposition we assume that all variables are defined over the same domain d, but the algorithm works fine otherwise.

$i = 0$	aa	ab	ba	bb	$\vec{c}$
xy					4
xz					4
yz					4

$i = 1$	aa	ab	ba	bb	$\vec{c}$
xy	X				3
xz		X			3
yz		X			3

$i = 2$	aa	ab	ba	bb	$\vec{c}$
xy	X	X			2
xz	X	X			2
yz	X	X	X	X	0

$i = 3$	aa	ab	ba	bb	$\vec{c}$
xy	X	X	X	X	0
xz	X	X	X	X	0
yz	X	X	X	X	0

(a) Zeroes tracking table and counter vector

$x\,y\,z$	f^0	f^1	f^2	f^3
$a\,a\,a$	4	2	0	0
$a\,a\,b$	2	0	0	0
$a\,b\,a$	4	2	0	0
$a\,b\,b$	3	1	0	0
$b\,a\,a$	5	3	1	0
$b\,a\,b$	3	1	1	0
$b\,b\,a$	5	3	1	0
$b\,b\,b$	4	2	1	0

(b) Per iteration remainders

$$f^0[\emptyset] = 2$$

$$f^1[yz] = \begin{array}{c|c} y\,z & \\ \hline a\,a & 2 \\ a\,b & 0 \\ b\,a & 2 \\ b\,b & 1 \end{array}$$

$$f^2[xz] = \begin{array}{c|c} x\,z & \\ \hline a\,a & 0 \\ a\,b & 0 \\ b\,a & 1 \\ b\,b & 1 \end{array}$$

(c) Selected lower bounds

Figure 4.3: Zero-tracking decomposition example.

Notice that, at this point, our example f^0 constraint does not have any zero. Hence, to introduce some zeroes, the algorithm first subtracts from f^0 its minimum value (which amounts to the projection of f^0 over the empty set). In the example, this subtraction yields the constraint $f^0[\emptyset]$, shown at the top of Figure 4.3c. Subsequently, it calculates the next remainder f^1 using Equation (4.2).

After calculating the remainder f^1 (the new constraint to approximate), the algorithm proceeds to update the *Zeroes* table along with the $\vec{c}$ counter. Back to our example, notice that f^1 contains a single assignment with zero cost $\mathbf{v} = \langle x = a, y = a, z = b\rangle$. Then, the algorithm calculates the projection of $\mathbf{v}$ to each row $\mathbf{S}$ and sets cell $[\mathbf{S}, \mathbf{v}[\mathbf{S}]]$ to *true* in the *Zeroes* table. In the example, the cell for row xy and column aa is set to true in the *Zeroes* table. Moreover, the counter for row xy decreases to record that there is one less non-zero assignment. Figure 4.3a (i=1) shows the state of both the *Zeroes* table and the counter vector after iteration i=1. In general, for each *new* zero cost assignment $\mathbf{v}'$, the algorithm checks the *Zeroes* table cell at row $\mathbf{S}$ and column $\mathbf{v}'[\mathbf{S}]$. If the cell is *false*, it is set to *true* to indicate that the cost of the r-arity lower bounds for the tuple will be zero from iteration i onwards. Moreover, the value of the counter of non-zero assignments for the extracted constraint, $\vec{c}(\mathbf{S})$, decreases by one.

Once the *Zeroes* table and counters are updated, there are two possible cases:

- If all counters' values are zero, it means that the cost for all assignments of all subsequent r-arity lower bounds will be zero. Therefore, since it is not possible to extract more information using any r-arity constraints, the algorithm terminates and returns the list of extracted constraints, $\{f^0[\emptyset], f^1[\mathbf{S}^1], \ldots, f^m[\mathbf{S}^m]\}$, as the resulting r-lower-bound.
- Otherwise, the r-arity constraint with more non-zero tuples is selected as the best to be extracted, and the algorithm continues.

In the example in Figure 4.3, all candidate r-arity constraints (see the rows in table *Zeroes* at iteration i=1) contain 3 non-zero tuples. Thus, at the next iteration (i=2), the algorithm can randomly choose the projection of f^1 over any pair of variables. Say that the algorithm chooses $\mathbf{S}^1 = \{yz\}$. Therefore, the selected r-arity lower bound is $f^1[yz]$, and hence the new remainder f^2 can be computed. After updating the *Zeroes* table, there are still two counters larger than zero, as shown in Figure 4.3a (i=2). In our case, the algorithm selects $f^2[xz]$ (discarding $f^2[xy]$), calculates the new remainder f^3, and updates the *Zeroes* table to yield the table in Figure 4.3a (i=3). At this point, since all counters are zero, the algorithm terminates to return the following set of selected constraints as the resulting decomposition:

$$F' = \{f^0[\emptyset], f^1[yz], f^2[xz]\}.$$

On the one hand, notice that the whole procedure —shown in Algorithm 3— never calculates a projection unless it is going to be returned as part of the resulting decomposition. Furthermore, since the maximum number of constraints in a decomposition is $\binom{|V|}{r}$, the worst case complexity of calculating the decomposition is $O\left(\binom{|V|}{r} \cdot \exp(|V|)\right)$. On the other hand, the algorithm has to maintain

Algorithm 3 ZeroDecomposition(F, V, r).

```
1: initialize($Zeroes$, $\vec{c}$)
2: $f \leftarrow (\bowtie F)[V]$
3: $F' \leftarrow \emptyset$
4: $(f', gain) \leftarrow (f[\emptyset], 1)$
5: while $gain > 0$ do
6:     $F' \leftarrow F' \cup \{f'\}$
7:     $f \leftarrow f \bowtie (-f')$
8:     $(f', gain) \leftarrow$ selectBestLowerBound($f$,$r$,$Zeroes$,$\vec{c}$)
9: end whilereturn $F'$
10:
11: function selectBestLowerBound($f$,$r$,$Zeroes$,$\vec{c}$)
12:     for all new $\mathbf{v}$ s.t. $f(\mathbf{v}) = 0$ do // new zeroes in $f$
13:         for all $\mathbf{S} \in S$ do // subsets of $r$ variables
14:             if not $Zeroes(\mathbf{S}, \mathbf{v}[\mathbf{S}])$ then
15:                 $\vec{c}(\mathbf{S}) \leftarrow \vec{c}(\mathbf{S}) - 1$
16:                 $Zeroes(\mathbf{S}, \mathbf{v}[\mathbf{S}]) \leftarrow true$
17:             end if
18:         end for
19:     end for
20:     $\mathbf{S}^* \leftarrow \arg\max_{\mathbf{S} \in S} \vec{c}(\mathbf{S})$
21: return $f[\mathbf{S}^*]$ , $\vec{c}(\mathbf{S}^*)$
22: end function
```

the zeroes table, which also has a cost. Note that function selectBestLowerBound only processes the assignments that are zero in the current iteration and were not zero in the previous one.[5] This means that to maintain the table, each assignment will be processed at most once. Since for each assignment we mark each possible r-arity lower-bound, the time complexity of maintaining the table is $O\left(\binom{|V|}{r} exp(|V|)\right)$, and thus does not increase the overall time complexity.

4.2.3 Empirical evaluation

In this section we evaluate the performance of the different approximation strategies on the DIMCTEf algorithm. For each experiment, we present both the communication savings and increase in overall computational cost with respect to standard GDL (DCTE). We choose to track these measures because they are the key ones in constrained environments. For instance, consider a wireless sensor networks setting. Since running out of battery disables a node, battery consumption is probably the most important figure to consider. Therefore, both communication and computation costs are important because they directly determine battery consumption. We estimate the overall computational cost by

[5]New zeros can be detected at no cost while computing the combination in line 7.

adding the processing times incurred by each node, while ignoring communication times. Similarly, the overall communication cost can be easily determined by adding the number of bytes of all sent messages.

Because GDL's communication and computation is mainly determined by the maximum clique size of the computed Junction Tree, experiments are segmented by this parameter. Consequently, both GDL and all DIMCTEf approaches use the same JTs generated by the DJTG algorithm [Vinyals et al., 2010b], like in Section 3.4.1. Additionally, notice that the parallelism degree is roughly the same for all algorithms, because it mainly depends on the computed JT. As a consequence, since DIMCTEf always communicates less information than DCTE, the relative increase in real solving time between GDL and DIMCTEf would be lower than the relative increase in overall computation shown in this paper.

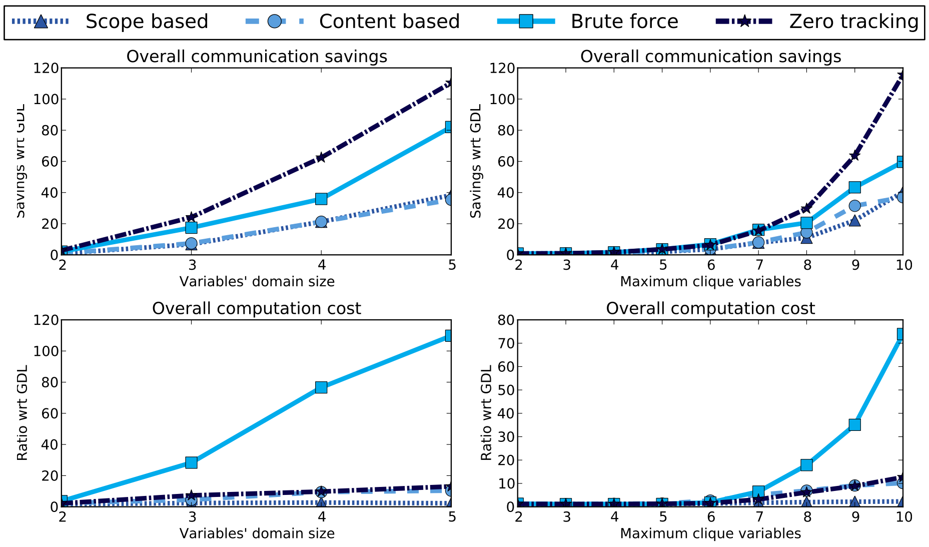
Since DIMCTEf removes assignments, it generates sparse constraints. Sending sparse constraints can lead to communication savings, but only if the implementation uses a special codification to transmit them. However, exploring the codification of sparse constraints is not one of the objectives of this work. Hence, we simply set a special value as cost for the filtered assignments, and compressed the messages. Specifically, we chose an Arithmetic Encoder [Cleary and Witten, 1984] with a Partial Prediction Matching model of 8 bytes. This compression method is known to achieve good compression ratios, so long as its input contains repeated values. The downside is that compressing has a high computational cost, which is considered as part of our overall cost. Regarding DCTE, compression hurts because the overall computation increases by an order of magnitude, while communication savings are practically negligible. Therefore, we report GDL results without compressing (following the idea of presenting results for each algorithm in its best possible condition). Likewise, although we tried both the LRE and LMRE metrics for both content-based partitioning and brute-force decomposition, we only report the best results obtained.

As in Sections 3.4.1 and 3.5.3, we conducted tests with the sensor networks instances from [Maheswaran et al., 2004b], but they were very easy for GDL-based algorithms in general (5 maximum clique variables).[6] As a result, all strategies achieved basically the same results, requiring 3 times less communication while maintaining the same computation cost as standard GDL.

Next, we designed an experiment to measure the methods' trends as the variables' arity increases. The experiment is composed of 35 problems of 20 variables for each domain size,[7] with a random structure of densities ranging from p=0.1 to p=0.3, where p is the probability of appearance for all edges. Constraints' costs are taken from a normal distribution $\mathcal{N}(0,1)$, and then each constraint is made positive by adding its minimum value to each relation. Results in Figure 4.4a show that top-down approximation methods perform significantly better than bottom-up approximations in the communication front, with nearly

[6]The maximum number of clique variables in a JT is equal to its treewidth plus one. Thus, 5 maximum clique variables amounts to a treewidth of 4.

[7]We introduce less variables than in Section 3.4.1 because top-down methods are more expensive computationally, and the experiments would take too much time otherwise.



(a) Increasing domain size, constant maximum clique variables of 10.

(b) Increasing maximum clique variables, constant domain size of 5.

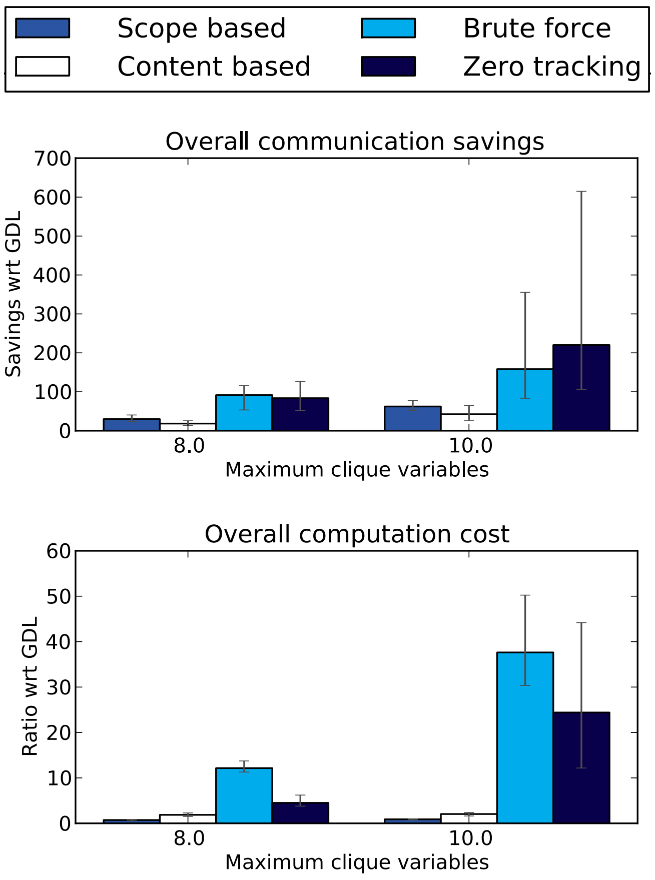
Figure 4.4: Performance evaluation of approximation strategies on random graphs.

constant savings between two and three times better. As expected, the brute force approach is more expensive computationally than other methods (up to 100 times slower than GDL in the worst case). Nevertheless, zero-tracking's overall computation cost is just 24% larger (13 times that of GDL at most) than that of the content-based approach (10.5 times GDL), whereas its savings are 189% larger (110.5 times less bytes sent for zero-tracking against 38.3 times for content-based). Moreover, zero-tracking's savings in communication increase almost 10 times faster than the computational cost.

Then, we conducted a second experiment that measures the trends when the problems' maximum clique variables increases. Consequently, it contains problems of 20 variables of arity 5 for each maximum clique variables, also with random structures between p=0.1 and p=0.3, and normal costs. Figure 4.4b shows that the communication savings increase exponentially for all methods, yet zero-tracking grows at a much faster rate than the others while keeping the computational cost under control.

Finally, the third experiment measures the impact of structure in the problems' constraint graph. Thus, it contains lattice-structured problems of 25 and 36 variables, leading to JTs of 8 and 10 maximum clique variables.[8] Once again, top-down approximation methods achieve the largest communication savings. In particular, zero-tracking decomposition requires up to 612 times less bytes than

[8]It is known that a JT of at most $n + 1$ clique variables can be built for any $n \times n$ lattice graph [Diestel, 2000]. However, the DJTG algorithm was unable to construct such trees.



(a) Maximum clique variables variation in lattice-structured problems.

Figure 4.5: Performance evaluation of approximation strategies on lattice-structured graphs.

GDL in 25% of the clique size 8 problems, while being only 44 times slower.

In summary, top-down approximations result in large communication savings. Additionally, zero-based decomposition remains competitive in computational effort with respect to state-of-the-art approximation methods. Hence, we have effectively improved the scalability of GDL with function filtering for heavily communication constrained applications such as wireless sensor networks.

4.3 A general workflow for computing messages

In Section 4.2 we focused on heavily communication-constrained scenarios, and were able to improve the scalability of GDL with function filtering for those kind of applications. This is already a step towards answering our Question 2 in the introductory chapter, but disregards one fundamental aspect of it: ideally we should be able to tradeoff the different kinds of resources depending on the application. Instead, up to now we have strategies that are tuned for either communication or computation constrained scenarios, but not to find the right balance between them. Hence, our objective in the remainder of this chapter is to coin mechanisms that allow us to perform such tradeoff tuning.

A sensible approach to achieve this goal is to try to combine the aforementioned message approximation strategies to come up with new ones. However, notice that each GDL-based algorithm we reviewed in Section 3.3 implements the cost message computation in its particular way. Therefore, these particularities may also help at designing a more flexible cost message computation approach that can be tuned to the specific characteristics of different application domains. Therefore, we now present an outline for message computation that comprises the manner in which all current GDL-based algorithms compute their messages. Notice that, while very similar to the description of an approximation strategy in Definition 4.3, this scheme is more general because: (i) it does not make any assumptions about the semantics of the r parameter; and (ii) it explicitly incorporates function filtering.

Figure 4.6 shows our general outline, which we detail in what follows. Consider an agent i that is about to send a message to agent j, its parent in the JT. Agent i has received a number of messages from its children in the JT. Additionally, agent i has its own stake on the problem represented as a set of constraints C_i. Then, the agent joins all these sets into a single set F, put together in Figure 4.6 as the input constraints $F = \{f_1, \ldots, f_n\}$. Then, agent i must compute the message for agent j, which is a set of (output) constraints $F' = \{f'_1, \ldots, f'_m\}$. This set of constraints must be a V-lower bound (see Definition 4.1) of the input constraints, where $V = S_{ij}$ are the separator variables between agents i and j in the JT.

Additionally, some of the algorithms in Section 3.3 require an input parameter r that determines the maximum arity of the constraints to be computed by the message computation task. Finally, some algorithms enable the message computation task to exploit the information contained in the message sent by agent j to agent i in the previous iteration. This information can be regarded

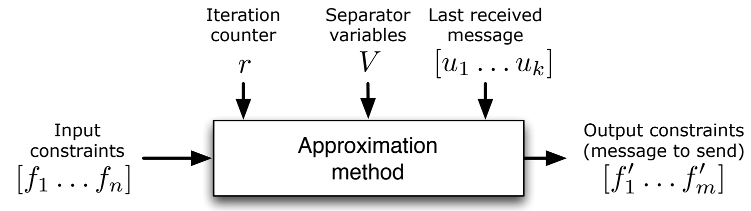


Figure 4.6: Outline of the cost message computation task.

as a further input to the task of computing messages, and thus we represent it as the set of constraints $U = \{u_1, \dots, u_k\}$ in Figure 4.6.

In the following we first analyze how each algorithm described in Section 3.3 implements the cost message computation task. Thereafter, we abstract a general scheme for computing cost messages that comprises all current GDL-based algorithms as particular instances. Based on this novel general scheme, in Section 4.4 we propose new algorithms to solve DCOPs that can effectively trade off communication and computation according to a user's requirements.

4.3.1 Complete GDL-based algorithms

Figure 4.7 shows how the message computation task is implemented by the complete GDL-based algorithms (CTE, DCTE, and Action-GDL) described in Section 3.3.2. The first operation that all these algorithms perform is the combination of all input constraints (using the operation in Definition 3.2) to produce a single function f. Thereafter, function f is projected over the separator's variables (using the operation in Definition 3.5) to yield an output constraint f', the message that will be finally sent. Notice that the output constraint is exactly the combination of the input constraints over the separator's variables, and hence the tightest possible V-lower bound.

On the one hand, recall that the advantage of this message computation approach is that it produces the optimal message [Aji and McEliece, 2000].[9] However, recall also that the complexity of combining all the input constraints is exponential on the number of constraints' variables. Hence, the message computation task can become prohibitively expensive depending on the size of the problem.

4.3.2 Approximate GDL-based algorithms

The method used to compute cost messages by the centralized MCTE(r) is very similar to the one used by the complete GDL-based algorithms analyzed above. Compare the implementation of MCTE(r) in Figure 4.8 with the one in Figure 4.7. The main difference lays on how the input constraints are combined.

[9]This is why these algorithms obtain the optimal solution after a single cost propagation iteration.

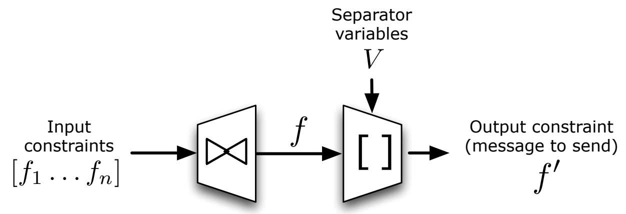


Figure 4.7: CTE/DCTE/Action-GDL's message computation method.

In this case, the combination is computationally bounded by the r parameter, namely the maximum arity of the constraints that the combination operator can compute. Thus, prior to their combination, the set of input constraints is first partitioned into groups of constraints while ensuring that no group has more than r different variables. As a result, applying the combination over each group produces a set of m constraints $\{g_1, \ldots, g_m\}$ whose arity is limited by r. In a second step, each of these functions is independently projected over the separator variables (V), and the resulting set of constraints forms the outgoing message $\{f'_1, \ldots, f'_m\}$. This set of constraints is also a V-lower bound of the combination of the input constraints over the separator's variables. However, it is not necessarily the tightest lower bound anymore because of the independent projections. Therefore, because the messages are not optimal and MCTE(r) runs a single cost propagation iteration, this algorithm obtains an approximate solution.

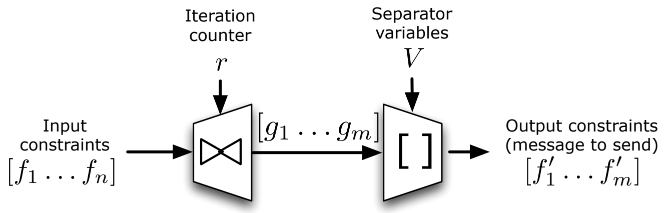


Figure 4.8: MCTE(r)'s message computation method.

4.3.3 GDL with function filtering

Next we analyze the IMCTEf algorithm, the first version of GDL with function filtering, as presented in [Sánchez et al., 2005]. The algorithm runs multiple iterations of the cost propagation phase. Specifically, each iteration runs with an increasing r bound, which limits the arity of the computed cost messages to r. Each iteration increases the value of the r bound employed by the last iteration, thus yielding better solutions. IMCTEf keeps on running iterations until the problem is solved optimally. The IMCTEf algorithm was the first one to introduce the function filtering operation, which filters out assignments that are known to be suboptimal.

Figure 4.9 depicts the message computation scheme implemented by IMCTEf. Notice that the filtering operation is performed after the projection over the constraints resulting from the combination of the input constraints. This is because filtering prior to projecting would yield the same results at the expense of a larger computational cost (more assignments would have to be considered). Notice also that the filtering operation employs a set of constraints $u_1, \ldots, u_k$ received during the previous iteration, and that contain information on bounds for the different assignments over V.

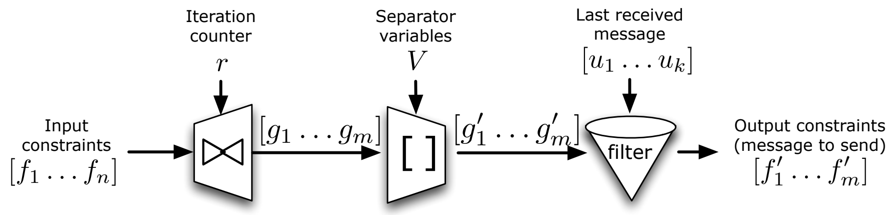


Figure 4.9: IMCTEf message computation method. Parameter r is a computation bound.

Recall from Section 3.3.4 that function filtering was initially intended to reduce the computational effort required to solve constraint optimization problems (COPs) in a centralized manner. Later on, the work in [Brito and Meseguer, 2010a] applied function filtering to DCOPs with the aim of reducing the amount of communication required to solve them. As presented, these works employ the one-sided filtering strategy in Equation (3.2), but our two-sided improvement in Equation (3.5) can be used interchangeably in all cases.

As shown in Figure 4.10, DIMCTEf uses almost the same method than IMCTEf to compute messages. However, it changes the semantics of the r bound. While IMCTEf uses r as a computation bound, DIMCTEf employs it as a communication bound. Likewise IMCTEf, DIMCTEf also partitions the input constraints into disjoint groups, but this partitioning considers only the separator variables. Notice that, after combining the functions in each group, the projection operation removes all variables that are not in the separator. Therefore, by considering only the separator variables in the combination operation, the method is limiting the maximum number of variables that each output constraint can contain.

In Section 4.2.2 we introduced the idea of decomposing a constraint into lower-arity ones to further reduce the communication requirements of DIMCTEf. That is, we *split* a large n-ary constraint into a set of smaller r-arity ones, where $n > r$, which represent it as accurately as possible. This operation allows us to define a new message computation method, the so-called Top-down GDL, as outlined in Figure 4.11. Top-down GDL first computes the combination of input messages to subsequently *split* the resulting function into a set of r-arity

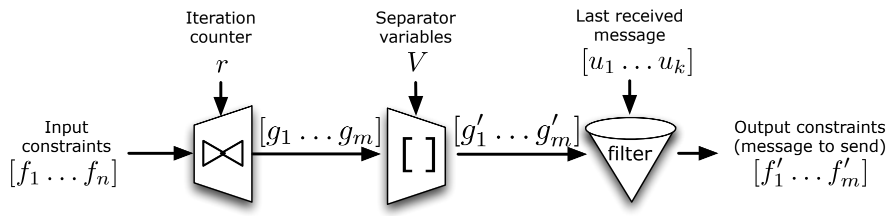


Figure 4.10: DIMCTEf message computation method. Parameter r is a communication bound.

functions that satisfy the communication bound requirement.

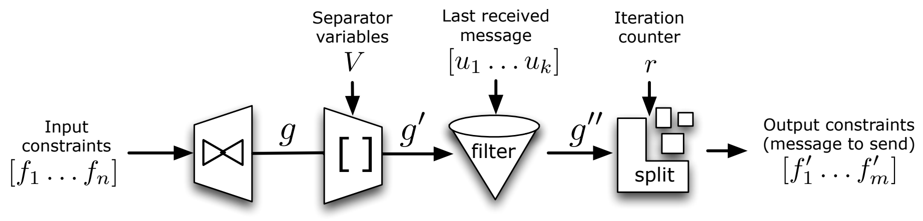


Figure 4.11: Top-down GDL message computation method. Parameter r is a communication bound.

4.3.4 Analysis

The purpose of this section is to yield a scheme for message computation that generalizes and encompasses the message computation schemes introduced above. Before that, we turn our attention to the way the algorithms analyzed so far operate. We observe that while MCTE(r) and IMCTEf focus on minimizing the computational effort, DIMCTEf and Top-down GDL focus on reducing communication even at the expense of an increase in computation. With this aim, the r parameter is employed as either a computation or communication bound over different operations of the message computation task. On the one hand, MCTE(r) and IMCTEf employ the r parameter as a computation bound over the combination of input functions. On the other hand, DIMCTEf and Top-down GDL employ the r parameter as a communication bound over different operations: DIMCTEf uses it as a bound over the combination operation of the input constraints (r is the maximum number of separator variables that can appear in the combination of any input constraints); Top-down GDL uses it as a limit on the output of the split operation. [10] Table 4.1 summarizes how the bound parameter r affects the message computation task in the algorithms analyzed above.

[10]Observe that these two interpretations of parameter r are not independent because the computation bound limits the communication bound.

Algorithm	Parameter r	Maximum size of any constraint computed by i	sent from i to j
CTE, DCTE, Action-GDL	no parameter	$d^{\|J\|}$	$d^{\|V\|}$
MCTE(r), IMCTEf	computation bound	$d^{min(\|J\|,r)}$	$d^{min(\|V\|,r)}$
DMCTE(r), DIMCTEf	communication bound	$d^{\|J\|-max(0,\|V\|-r)}$	$d^{min(\|V\|,r)}$
Top-down GDL	communication bound	$d^{\|J\|}$	$d^{min(\|V\|,r)}$

Table 4.1: Effect of r parameter on the message computation task of GDL-based algorithms (J is union of scopes of all constraints in F, d is the maximum domain size of the variables in J, and $|V| \leq |J|$).

Now we are ready to offer a scheme for the message computation task that generalizes the message computation methods analyzed so far, as shown in Figure 4.12. According to this scheme, the input functions are first combined in groups, similarly to what MCTE(r) and DIMCTEf do. However, the combination operation is performed by a new combine operator that can use two bounds at the same time: (i) the computation bound employed by MCTE(r) over the total number of variables; and (ii) the communication bound employed by DIMCTEf over the number of variables of the separator. Henceforth, we shall refer to these two bounds as *merge computation bound* (B_{MC}) and *merge transmission bound* (B_{MT}) respectively. As a result, this new combination operator allows us to limit both computation and communication. After the combination step, the resulting list of constraints is projected over the separator's variables and filtered according to the last received message. Finally, the filtered constraints are split into smaller ones, whose size is limited by yet another communication bound, the so-called *split transmission bound* (B_{ST}).

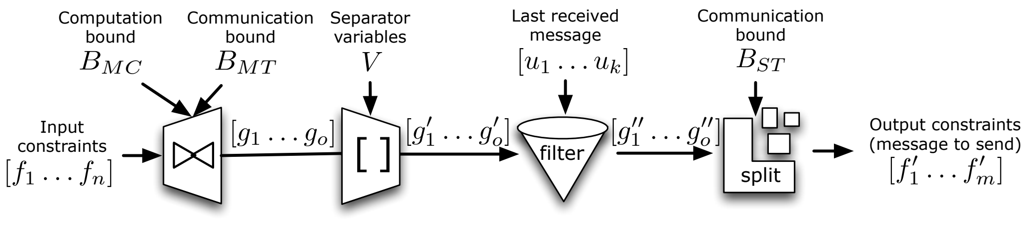


Figure 4.12: General scheme for message computation.

Notice that this general message computation scheme incorporates three bounds (B_{MC}, B_{MT}, and B_{ST}) to limit the amount of computation and communication. Table 4.2 shows how these bounds have to be set to yield the message computation methods analyzed above. By setting a bound to infinity, we are

Algorithm	B_{MC}	B_{MT}	B_{ST}
CTE, DCTE, Action-GDL	∞	∞	∞
MCTE(r), IMCTEf	r	$\geq r$	$\geq r$
DMCTE(r), DIMCTEf	∞	r	$\geq r$
Top-down GDL	∞	∞	r

Table 4.2: Settings of the general message computation scheme to implement message computation methods of GDL-based algorithms.

putting no limit to that dimension. Thus, if B_{MC} and B_{MT} are both set to infinity, there is no limit on the maximum amount of variables (and separator variables) to combine. In contrast, setting B_{ST} to infinity means that there is no limit on the number of variables of the constraints going through split, and hence any input constraint will be output unchanged.

In general, notice that each setting of the general scheme's bounds results in a new message computation method. However, for design purposes, we must first understand the relationships that hold between the three bounds.

- First, consider that the merge transmission bound is set to a larger value than that of the merge computation bound, namely $B_{MT} > B_{MC}$. Now, recall that both B_{MT} and B_{MC} represent limits on the maximum number of variables that an output function can have. However, B_{MT} refers to the maximum amount of *separator* variables, whereas B_{MC} refers to the maximum amount of variables *of any kind*. Therefore, the output constraints will never contain more than B_{MC} separator variables disregarding B_{MT}'s value.

- Second, something similar happens between B_{MT} and B_{ST}. As we have just seen, B_{MT} is the maximum number of separator variables. Also, remember that the combined constraints are then projected over those separator variables. This operation eliminates all the non-separator variables. Therefore, the constraints received by the split operation will never contain more than B_{MT} variables. As a consequence, if $B_{MT} < B_{ST}$, the constraints received by the split operation are necessarily smaller than the limit, and the split operation will simply output them unchanged.

For instance, consider the method resulting from setting $\langle B_{MC} = r, B_{MT} = r, B_{ST} = r\rangle$ and the method resulting from setting $\langle B_{MC} = r, B_{MT} = \infty, B_{ST} = \infty\rangle$. Both message computation methods are equivalent. In general, it makes no sense to consider bound settings such that $B_{MT} > B_{MC}$ or $B_{MT} < B_{ST}$. Therefore, we can safely impose the following relationship between bounds when designing a new message computation method: $B_{MC} \geq B_{MT} \geq B_{ST}$.

In this section we have presented a general scheme for message computation. As Table 4.2 shows, the general scheme can be set to implement all message computation methods in the literature. But more importantly, the general scheme in Figure 4.12 also opens the possibility of designing new message computation methods. In fact, designing a new message computation method amounts to deciding which operators to employ and how to set their computation and communication bounds. Hence, in the following section we explore these possibilities, presenting multiple methods designed to cater the needs of different application domains.

4.4 Trading off computation and communication

As explained above, we can derive new message computation methods by choosing different policies to set the bounds in the general scheme in Figure 4.12. Furthermore, we have discussed the relationships between bounds that a setting must fulfill. In this section we aim at designing new message computation methods that trade off computation and communication. More precisely, in Sections 4.4.2 to 4.4.4 we identify three "families" of message computation methods that are expected to yield different trade-offs. Before that, in Section 4.4.1 we introduce a way of parametrizing the bounds that we must set when designing a new message computation method. Notice that in this section we solely focus on the design of message computation methods, putting off their empirical evaluation until Section 4.4.5.

4.4.1 Parametrizing bounds

The purpose of introducing a parametrization is twofold. First, we aim at lowering the number of parameters to set, which is currently three (namely, the number of bounds). Second, we aim at reducing the space of values for bound settings so that we avoid settings that lead to the very same message computation method.

We start by identifying the lowest value that the least dominant bound (B_{ST}) can take on. Since the message computation methods involving bounds are all iterative, and r is the iteration counter, r is precisely the lowest value B_{ST} can take. Next, to set B_{MT} we must recall from Section 4.3.4 that the relationship $B_{MT} \geq B_{ST}$ must hold: if B_{MT}'s value is smaller than B_{ST}'s, the resulting message computation method is the same than when they are exactly equal; otherwise, if B_{MT}'s value is larger than B_{ST}'s, the bound setting does define different message computation methods. Hence, we define $B_{MT} = B_{ST} + \delta_1 = r + \delta_1$, where $\delta_1 \geq 0$ is the first parameter of the new parametrization. Analogously, since $B_{MC} \geq B_{MT}$ must hold, using the same line of reasoning we define $B_{MC} = B_{MT} + \delta_2 = r + \delta_1 + \delta_2$, where $\delta_2 \geq 0$ is also a new parameter. To summarize, we define:

$$\begin{aligned} B_{MC} &= r + \delta_1 + \delta_2, \\ B_{MT} &= r + \delta_1, \\ B_{ST} &= r \end{aligned}$$

Algorithm	δ_1	δ_2	Expected resource usage	
			Computation	Communication
MCTE(r), IMCTEf	0	0	Low	*High*[11]
DMCTE(r), DIMCTEf	0	∞	Low-Medium	Medium-High
Top-down GDL	∞	—	High	Low

Table 4.3: Parameter settings to obtain the message computation methods in the literature from our general message computation scheme in Figure 4.12. When $\delta_1 = \infty$, parameter δ_2 can take any value.

where δ_1 and δ_2 are non-negative integers. Setting values for $\langle \delta_1, \delta_2 \rangle$ defines a bound setting, and hence a new message computation method. Table 4.3 shows how to set these parameters to yield the algorithms analyzed from the literature. Notice that we do not consider the message computation methods of CTE, DCTE, and Action-GDL. However, we are not interested in those algorithms because they are not iterative, and hence unable to consider any computation versus communication trade-off.

Now we are ready to yield different message computation methods by providing the appropriate settings for $\langle \delta_1, \delta_2 \rangle$.

4.4.2 Bounded Bottom-up message computations

Based on the classification in Section 4.2, bottom-up methods are those that avoid computing any constraint that can not be sent without splitting. In terms of our general scheme, bottom-up methods are those that do not employ the split operation. Looking back at the scheme in Figure 4.12, this means that the merge transmission bound B_{MT} must be lower than or equal to the split transmission bound B_{ST}. According to our new parametrization, this whole family of methods can be characterized by the tuple $\langle 0, \delta_2 \rangle$. In other words, this family includes all the methods for which $\delta_1 = 0$.

Notice that this family includes both IMCTEf and DIMCTEf, as shown in Table 4.3. Furthermore, these methods represent the two most extreme message computation methods within the bounded bottom-up message computation family. Now, recall that IMCTEf and DIMCTEf use a very similar scheme to compute their messages. In fact, the only difference between them lays in the semantics associated to the r bound. Essentially, IMCTEf focuses on limiting the amount of computation performed by the nodes. In contrast, DIMCTEf concentrates on bounding the amount of communication. Therefore, this family of algorithms bounds the amount of both computation and communication. Furthermore, they do so by avoiding to combine constraints when the result is expected to be large. Hence, in general they should not incur in large computation costs.

[11]Note that MCTE(r) and IMCTEf are centralized algorithms and have no communication costs. Hence, this is the expected communication cost of our distributed implementation, not of the original algorithms.

Consequently, these computation methods can be expected to produce messages that require a small amount of computation, but are relatively inaccurate. Because of this inaccuracy, we can expect the algorithm to perform more iterations of the cost propagation phase, resulting in an increase in communication. As a result, we expect methods from this family to end up being reasonably good when the communication costs are relatively low with respect to the computational costs.

4.4.3 Bounded Top-down message computations

Recall that top-down methods are those that begin by computing constraints larger than what can be sent, and then proceed to compute an approximation in terms of lower arity constraints. In our general scheme, top-down methods are those that do not employ a merge transmission limit (B_{MT}) in the combination operator. Without this bound, the responsibility of complying with the iteration limit r is delegated to the split operation. Using our parametrization in Section 4.4.1, this whole family of message computation methods can be characterized by the tuple $\langle \delta_1, 0 \rangle$. In other words, this family includes all the methods for which $\delta_2 = 0$.

Obviously, this family includes both IMCTEf and Top-Down GDL algorithms, as shown in Table 4.3. In this case, we combine the computational bound offered by IMCTEf with the communication bound given by the Top-Down GDL algorithm. Notice that this is similar to what bottom-up message approximations did, but this time the communication bound is enforced by the split operation rather than during the combination. Furthermore, observe that avoiding to combine constraints is computationally much cheaper than computing the combination to subsequently split it. Hence, top-down message approximations are expected to produce messages that require more computation than bottom-up methods, but which at the same time are more accurate. As a consequence, the algorithm employing a bounded top-down message computation method is expected to require less cost propagation iterations, resulting in fewer communication costs. Therefore, we expect methods in this family to achieve their best results when the communication costs are large in comparison with computation costs.

4.4.4 Bounded Mixed message computations

The message computation families above included those methods that combined the computational bound B_{MC} with one of the communication bounds, be it either B_{MT} or B_{ST}. Hence, those methods do not employ all the computation and communication limits available in the general scheme in Figure 4.12. In this section we propose to produce new methods by providing settings that enable all bounds at the same time. According to our parametrization in Section 4.4.1, this amounts to set δ_1 and δ_2 to values greater than 0.

Employing two different communication limits at the same time may seem redundant at first. Nonetheless, there is a clear reason to do it if we consider how message computation methods differently handle these limits. As explained above, the merge transmission bound B_{MT} avoids combining some constraints. This is computationally cheap, but yields inaccurate results. Conversely, the split transmission bound B_{ST} is employed when computing a combination of constraints (namely, the exact constraint) to subsequently split it into smaller ones. Hence, complying with B_{ST} is computationally expensive but produces more accurate results. As a result, it is possible that combining both mechanisms to limit the communication actually delivers better results than employing only one of them.

Due to the larger variability among methods from this family, the performance of mixed message computations is difficult to predict as a whole. On the one hand, we expect these methods to perform similarly to the bottom up ones when δ_1 is close to zero. On the other hand, their results should be close to those of the top-down methods when δ_2 approaches zero. However, it is hard to predict the performance of a message computation method when both δ_1 and δ_2 grow larger than zero.

4.4.5 Empirical evaluation

In the previous section we have identified a number of families of message computation methods, and we have set some expectations on their results. Here we evaluate these methods to discover the parameter settings for which they offer the best computation-communication tradeoffs.

Empirical Settings

Recall that our objective is to empirically determine the best method given some ratio of communication costs against computation costs. With this aim, we measure both the computation and communication efforts that each method requires to optimally solve the problems. Specifically, the amount of computation is measured using non-concurrent constraint checks (NCCCs) [Meisels et al., 2002], whereas the total amount of bytes sent between the nodes is used as the communication cost metric.[12] With these measures we can compute a combined effort metric

$$e = \text{NCCCs} + \alpha \cdot \text{bytes} \ ,$$

where α is a ratio between the cost of sending a byte and the cost of performing an NCCC. For instance, a large α value indicates a domain that is heavily communication constrained (*e.g.* a sensor network). Conversely, a small α value represents a domain where communication is relatively inexpensive. In fact, notice that we can characterize the desired tradeoff for an application domain by specifying its α value because, given that value, the method that obtains the lowest combined effort e is the one that is expected to perform best in that domain.

[12]Effort measures combining computation and communication have already been proposed in DCOP literature [Chechetka and Sycara, 2006; Yeoh et al., 2008].

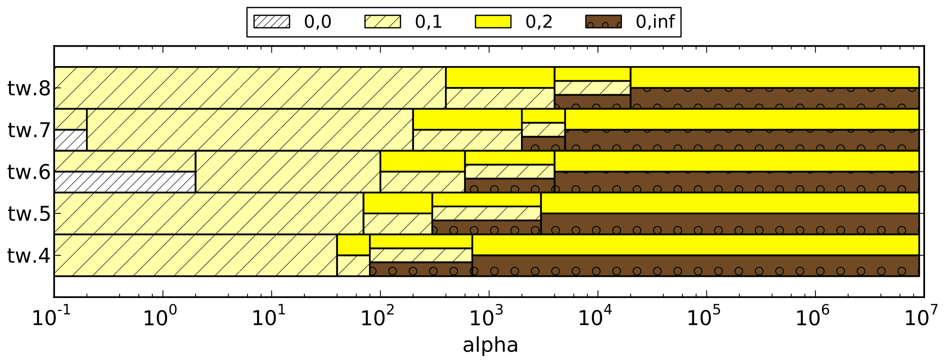
Application	Memory access time	Network latency	Time per byte	Estimated α
Meeting Scheduling (LAN)	100ns - 1ns	1μs	10ns	10^1 - 10^3
Meeting Scheduling (WAN)	100ns - 1ns	100μs	1μs	10^3 - 10^5
Sensor networks	100ns - 1ns	1ms	10μs	10^4 - 10^6

Table 4.4: Alpha values depending on application domain.

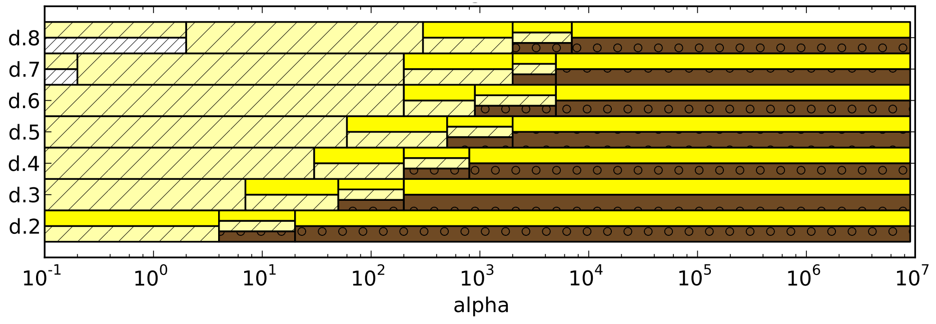
Now there is the issue of selecting values for α depending on an application's features. Although finding an accurate α value is difficult, we can still make reasonable estimations. For instance, consider again the meeting scheduling and sensor networks application domains, which are widely used in the DCOP literature [Petcu and Faltings, 2005b; Brito and Meseguer, 2010b; Vinyals et al., 2010b]. Regarding computation costs, the time of an NCCC is essentially the time of a main memory access, which is around 10^{-7}s with current technology. If we consider CPU caches, this number changes: the first access to main memory is "expensive" but subsequent accesses are "cheap", and hence the access time can drop down to 10^{-9}s. Regarding communication costs, the time to send a message is usually dominated by the network latency. Hence, sending a packet on a LAN takes around 10^{-4}s, whereas on a WAN it takes around 10^{-2}s. Likewise, the time to send a packet through a wireless link (such as those of a sensor network) hovers around 10^{-1}s. However, messages contain cost tables, which vary in size. Assuming an average of around 100 bytes per message, the actual time taken to send each byte is about two orders of magnitude lower than the network's latency. Table 4.4 summarizes these costs, grouped by three example application domains. Using these costs, we compute each α value by dividing the memory access time by the time required to send a byte. As a result, the value of α for these domains ranges from 10^1 to 10^6. In our experiments, we widened this range a bit to account for the fact that these are all approximations.

Likewise in Section 3.4.1, we segmented our experiments by the treewidth of the JT constructed by the DJTG algorithm. This allows us to observe the methods' performance when the JT's treewidth increases, as an indicator of their scalability. Similarly, we experiment on problems with an increasing variable domain size, which is also a key hardness and scalability measure. As a consequence, our experiments are characterized by just two parameters: treewidth and variable's domain size.

Each experiment consists of 100 DCOP problems, to ensure that we obtain significant results. Each of these problems has 40 variables, and a structure generated by a random graph generator with densities ranging from p=0.1 to p=0.3. Notice that, like in Section 3.4.1, these randomly generated structures do not necessarily lead to JTs with the desired treewidth. Hence, we generated as many structures as needed until we obtained 100 for which we were able to construct a JT with the intended target treewidth. Finally, as in our previous experiments, the constraints' costs are taken from a normal distribution $\mathcal{N}(0, 1)$, and then made positive by adding its minimum value to each constraint.



(a) increasing treewidth, fixed domain size 7



(b) increasing domain, fixed treewidth 7

Figure 4.13: Evaluation of bounded bottom-up message computation methods ($\delta_1 = 0$, increasing δ_2).

Notice that, even if we restrict the evaluation to methods with small δ_1 and δ_2 values, there are a wide range of methods to evaluate. Therefore, we start by comparing methods within each family separately. Thereafter, we will select only the best methods of each family and compare them to obtain the best ones overall.

Empirical Results

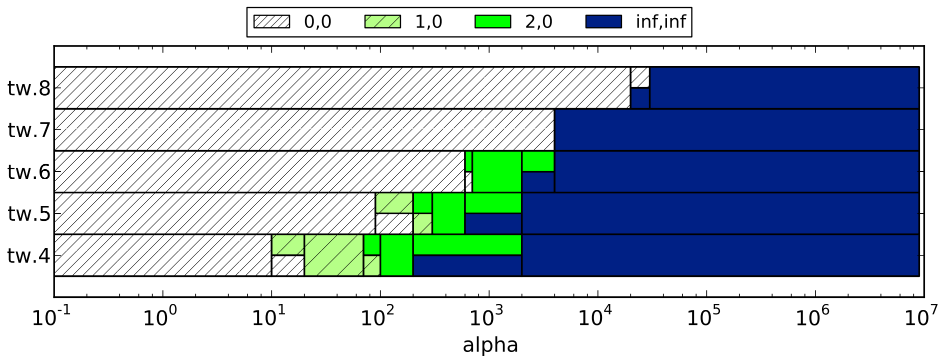
Here we discuss the results obtained from 13200 experiment executions (100 instances $\times$ 12 algorithms $\times$ 11 scenarios). As described before, each instance has 40 variables, which is unusually large in the optimal DCOP literature. Each execution considers 80 different α values, ranging from 10^{-1} (meaning that sending ten bytes is as expensive as performing a single NCCC) to 10^7 (meaning that sending a single byte is ten million times costlier than computing one NCCC). This captures a wide range of scenarios, from those where communication is extremely cheap such as LAN or inter-processor communication to those where communication is very expensive such as in sensor network inter-communication.

Figure 4.13 shows the results obtained by methods in the bounded bottom-up family. Specifically, Figure 4.13a shows the best performing algorithms depending on the α value for the experiments with fixed variable domain size (7) and an increasing treewidth. For instance, the uppermost horizontal bar (tw.8) shows the method obtained by setting $\langle\delta_1 = 0, \delta_2 = 1\rangle$ is one of the best methods for α values from 10^{-1} up to $10^{4.2}$. However, the method obtained by setting $\langle\delta_1 = 0, \delta_2 = \infty\rangle$ is equally as good when α lies between approximately $10^{3.4}$ and $10^{4.2}$. This is, the horizontal bar corresponding to an experiment is divided between multiple methods when their combined effort e is not 5% worse than the best one.

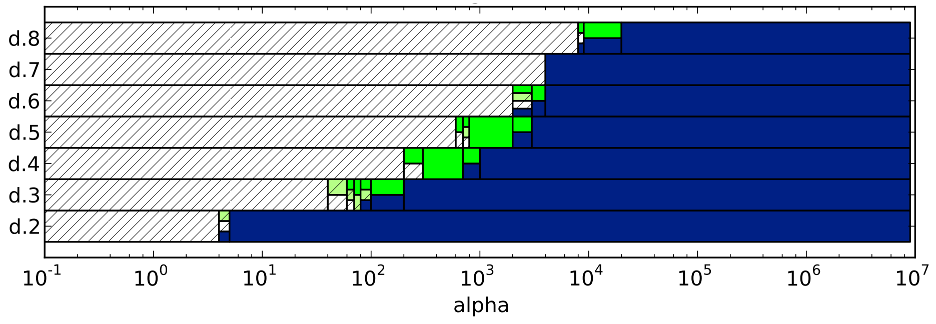
We draw three main conclusions from Figures 4.13a and 4.13b:

- It is beneficial to increase δ_2 when the communication cost rises, as expected from the results in Section 4.2.3.
- Message computation methods with a larger δ_2 scale worse in communication than in computation. This can be seen because, as either the treewidth or the domain size increases, methods with larger δ_2 values do not outperform the others until larger α alpha values.
- In all cases, the novel message computation method obtained by setting $\langle\delta_1 = 0, \delta_2 = 1\rangle$ performs equally as well or even better than the one obtained by setting $\langle\delta_1 = 0, \delta_2 = 0\rangle$. This is because the latter one was designed specifically for the centralized case, where communication is irrelevant (the lowest possible cost).

Why we do not report higher values of δ_2? Remember that the whole process is an iterative loop where parameter r is incremented at each iteration, performing function filtering. When r reaches the treewidth of the instance, it computes the optimum cost. Also, function filtering may save some of the last iterations if it filters out all suboptimal solutions before r reaches the treewidth. The algorithm always begins with $r = 2$ (because this is the size of the problems' constraints). At the first iteration, if $\delta_2 = 3$, the algorithm would compute functions of size d^5. As a result, there is no difference between $\delta_2 = 3$ and $\delta_2 = \infty$ for all problems of treewidth lower than or equal to 4. Likewise, there would be no difference between $\delta_2 = 7$ and $\delta_2 = \infty$ for problems of treewidth ≤ 8. That is, as we increase the value of δ_2, the algorithm behaves more and more like $\delta_2 = \infty$. As r grows with iterations, real bounds in the size of intermediate join functions also grow (in this case the exponent bound is $r + 0 + \delta_2$). Moreover, notice that the treewidth identifies the maximum number of variables on a node of the JT, but most nodes do not have as many variables. As a result, the effect of larger δ_2 values is less noticeable as we approach the treewidth bound, because most nodes already compute the exact same messages with lower δ_2 values. In our tests, algorithms with δ_2 values larger than 2 were not significantly different than the results of $\delta_2 = \infty$ in any occasion.



(a) increasing treewidth, fixed domain size 7

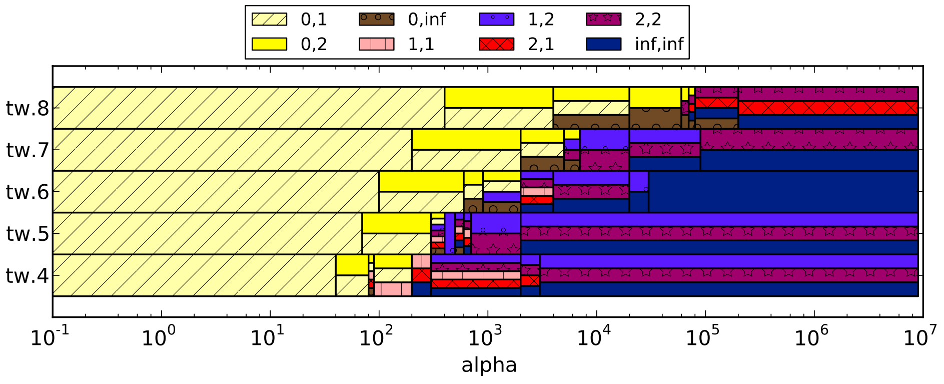


(b) increasing domain, fixed treewidth 7

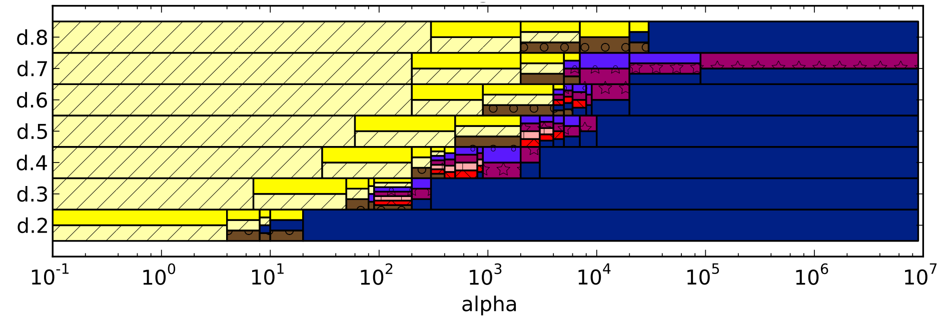
Figure 4.14: Evaluation of bounded top-down message computation methods (increasing δ_1, $\delta_2 = 0$).

Next we compare the message computation methods in the top-down message computation family. Figure 4.14 displays the results obtained by methods in this family on the same experiments and using the same format as before. In fact, the results are also similar, but this time regarding an increase of δ_1 instead of an increase of δ_2. That is, methods with higher δ_2 value are better when the communication costs are high, but they also scale worse when either the number of variables or the treewidth increase. However, a closer inspection reveals an important difference: choosing δ_2 at one of the extremes (either 0 or ∞) almost always yields better results than choosing any other δ_2 value. Therefore, unlike the bottom-up family, the method obtained by setting $\langle\delta_1 = 1, \delta_2 = 0\rangle$ is consistently worse than the one obtained by setting $\langle\delta_1 = 0, \delta_2 = 0\rangle$.

Finally, we compare the results obtained by message computation methods in the mixed family with the best ones obtained above. First of all, the novel bottom-up method obtained above by setting $\langle\delta_1 = 0, \delta_2 = 1\rangle$ outperforms all other methods when communication is cheap. Furthermore, Figure 4.15 shows that this algorithm's results improves as problems get harder, be them in the tree



(a) increasing treewidth, fixed domain size 7



(b) increasing domain, fixed treewidth 7

Figure 4.15: Evaluation of best overall algorithms.

width or in the variables' domain size. However, as communication increases, there is a range where no algorithm is a clear winner (considering also the ones in the mixed family). For instance, there are up to six different message computation methods obtaining very similar values when α is in the $[10^3, 10^4]$ range. Finally, when communication becomes very expensive ($\alpha \geq 10^5$), the results obtained by setting $\langle \delta_1 = \infty, \delta_2 = \infty \rangle$ are always equally as good or better than those of any other method. In fact, these results are in line with those reported in Section 4.2.3, and reflect the fact that such method was specifically designed for communication constrained scenarios.

4.5 Conclusions

Questions addressed in this chapter:

Q. 2. Can we improve the resource scalability of DCOP algorithms for which this scalability is a limitation?

After identifying the family of GDL algorithms as a promising approach for optimal DCOP solving, the previous chapter developed purely algorithmic improvements that allowed us to solve larger scale problems whatever the agents' capabilities are. In contrast, in this chapter we focused on improving the scalability of GDL-based algorithms by taking into account the specific capabilities of agents from different application domains.

First we noted that the key task defining the requirements of GDL-based algorithms is that of computing messages during the cost propagation phase. Furthermore, we observed that most message computation approaches in the literature were designed to minimize the computational cost of the algorithm, disregarding its communication requirements. Hence, we first unified these approaches in what we named the *bottom-up approximation* scheme.

Next we have taken the completely opposite stance, and developed the *top-down approximation* scheme. This scheme is specifically designed to reduce the algorithm's communication requirements, disregarding the computational ones. Moreover, we presented two particular realizations of this scheme: (i) the *brute-force decomposition* strategy, a naive implementation with high computational cost; and (2) the *zero-tracking decomposition* strategy, which greatly reduces the amount of computation. We empirically evaluated the performance of all these strategies, showing that top-down approximations always achieve larger communication savings than bottom-up ones. In fact, zero-tracking decomposition does so while keeping the computational cost at bay. As a result, we increased the scalability of GDL with function filtering algorithms for heavily communication constrained application domains.

This was a significant step towards answering Question 2 posed in the introductory chapter, but the issue of adapting the algorithms' requirements for applications where agents have more balanced resource capabilities still remained. Hence, we have then introduced a general scheme for cost messages computation that encompasses all current techniques used by the different GDL-based algorithms. Actually, we have shown that all the existing approaches in the literature can be obtained as specific instances of our general schema by conveniently setting the values for computation and communication bounds. Furthermore, this scheme allowed us to design novel message computation methods, representing novel combinations of previously existing strategies plus our novel *top-down* decompositions.

We have also extensively evaluated both previously existing message computation methods and the new message computation methods obtained from our general scheme by considering a continuum of communication to computation costs. Such evaluation provides guidelines to select the most appropriate message computation method depending on an application's available resources. From this analysis, we observed that the novel method obtained by setting $\langle \delta_1 = 0, \delta_2 = 1 \rangle$ is the best one when communication costs are cheap with respect to computational costs (e.g. meeting scheduling on a LAN). In contrast, the method obtained by setting $\langle \delta_1 = \infty, \delta_2 = \infty \rangle$ appears as the best method for heavily communication-constrained domains (e.g. wireless sensor networks).

More importantly, our empirical guidelines can be used to find out the most appropriate message computation method for applications whose resource availability is not heavily skewed towards neither communication nor computation (e.g. meeting scheduling on a WAN). Therefore, by allowing the designer to adapt the algorithm to the specific capabilities of the agents in her application, we effectively increased the scalability of GDL-based algorithms.

Part II

Approximate solving

Chapter 5

Scaling on dynamic applications

5.1 Introduction

Chapters 3 and 4 focused on DCOP applications where it was feasible to find optimal solutions. We identified the GDL-based class of algorithms as a family that scales well with certain types of problems, and further extended their applicability. However, at some point the problems grow large enough that they cannot be optimally solved anymore. Additionally, there exist other, densely interconnected problems (with large treewidths) to which our previous results do not apply. In those cases, the scale at which optimal solving becomes impractical is even smaller. The DCOP community realized these limitations long ago, and researchers have tried to tackle the issue by introducing approximate algorithms. These algorithms provide hopefully good solutions in shorter time and using fewer resources than optimal algorithms.

Nonetheless, as introduced in Section 1.1, scale is typically not the only issue in larger and more complex application domains. Particularly, there is another issue which has been less studied in the DCOP literature: dynamism. That is, most larger-scale coordination problems arise in dynamic situations, where both the world's conditions and the agents themselves are continuously evolving. In the second part of this dissertation (Chapters 5 and 6) we focus precisely on those dynamic applications of larger scale where optimal solving is not an option.

As noticed in [Scerri et al., 2005; Khanna et al., 2009], the bulk of DCOP-related research focuses on static problems that are completely specified at the beginning of the solving process. However, there exist several DCOP works that do consider dynamism. Some of those aim to minimize the overhead of optimally re-solving the problem after each change and/or at fixed time intervals. That is, they strive to maintain as much state as possible between subsequent optimal solving steps [Petcu and Faltings, 2007b; Khanna et al., 2009; Yeoh et al., 2011]. For instance, in [Yeoh et al., 2011] the authors define how the pseudotree and the bounds computed by ADOPT should be updated (instead of entirely recomputed) given the changes between steps. Unfortunately, these approaches suffer the scale limitation of optimal solving algorithms. Hence, they are not fit for the applications we want to explore in the remainder of this work.

Additionally, other works build on DCOPs, but specialize the model and the algorithms to handle particular application domains [Zivan et al., 2009]. Moreover, they do not provide the means (neither datasets nor simulation platforms) to let other researchers develop and compare new algorithms for their application. Furthermore, a few researchers have employed DCOPs to deal with the RoboCup Rescue Simulation challenge [Scerri et al., 2005; Kleiner et al., 2013]. However, the RoboCup challenge represents a fairly complex application domain involving heterogeneous teams of agents that must coordinate between them. Due to the lack of basic understanding about DCOPs in dynamic settings, we argue that it is better to start by tackling simpler but realistic enough problems where it is easier to build such understanding. Hence, we defer dealing with such complicated applications to Chapter 6 and focus on a simpler but realistic problem here.

Consequently, in this chapter we begin by introducing a novel realistic application, the Limited-range Online Routing Problem (LORP). In the LORP, a number of UAVs have to coordinate to service requests as quickly as possible. New requests can be introduced by human operators at any time, and the UAVs must adapt accordingly. What makes this problem particularly challenging is that the UAVs employ limited-range wireless radios to communicate, and hence a UAV can only communicate with those that are relatively near at that point in time.

Afterwards we situate this problem within the multi-agent coordination literature at large, without restricting ourselves to DCOP-related works. Then we propose an approach to tackle the LORP based on iteratively taking snapshots of the current situation and making decisions based on them. Two fundamental aspects for such approach to be successful are that: (i) the snapshots must be built and represented in a decentralized manner; and (ii) the decision making process must operate within strict time constraints. This allows for quick loops of assessment, decision and action that are particularly suited to highly dynamic application domains.

Then we realize this approach by formalizing the snapshots as DCOPs and solving them using the approximate, GDL-based Max-Sum algorithm introduced in Section 2.2.2. This technique provides two significant advantages: (i) using DCOPs means that the problem is represented as a single global utility function, and hence the collective behavior of the UAVs is easier to predict and reason about than when it emerges from the definition of individual behaviors; and (ii) we can draw results from the extensive Max-Sum literature, including its theoretical properties (e.g.: convergence and quality guarantees [Weiss and Freeman, 2001; Vinyals, 2011]), and encouraging experimental results [Rogers et al., 2011].

Notice that, as explained in Section 2.3, Max-Sum's major caveat is that it incurs on some exponential costs. For instance, [Delle Fave et al., 2012b] introduces a Max-Sum based solution for a problem similar to the LORP. However, that solution has two major drawbacks. Firstly, UAVs require an operator to facilitate their coordination, meaning that no coordination is possible outside of an operator's communication range. Secondly, the algorithm's requirements scale exponentially on the number of UAVs. In contrast, our solutions employ DCOPs with binary variables and Tractable Higher-Order Potentials (THOPS), a recent development from the graphical models community [Tarlow et al., 2010] that enables us to avoid the algorithm's exponential costs.

We present an initial development of this solution assuming that the cost of servicing each request is independent of other requests assigned to the UAV. Thereafter, we introduce a second solution where UAVs adjust their estimations on the cost of servicing a task depending on their workload, with a slight increment in complexity. As a result, UAVs using this solution are better equipped to dynamically capture and exploit the distribution of incoming requests.

To summarize, the contributions of this chapter are the following:

- We introduce the Limited-range Online Routing Problem as a highly-dynamic test-bed for the development of multi-agent coordination mechanisms and relate it to current state-of-the-art literature.
- We propose an approach to solve the LORP based on repeatedly taking distributed snapshots of the problem and making quick decisions based on that information.
- We show how the LORP can be cast as a sequence of DCOPs, solved using Binary Max-Sum, thus providing great flexibility to introduce new heuristics without modifying the solving algorithm and avoiding any exponential costs.
- We capitalize on this advantage by introducing the workload heuristic, which exploits the dynamic characteristics of the problem to make better decisions.
- We present an easy-to-use simulation framework for the development and testing of LORP solving algorithms.
- We empirically evaluate the proposed algorithms, showing that: (i) our approach allows for effective request allocation in a highly dynamic, communication constrained domain; and (ii) our workload-based solution achieves between 6% and 16% lower service times than current state-of-the-art methods, and its actual performance comes very close to that of centralized solutions.

The rest of the chapter is organized as follows. First, in Section 5.2 we describe the LORP in detail, and review current state-of-the-art approaches to tackle it. Thereafter we present our DCOP-based solutions in Section 5.3, explaining how the LORP can be encoded as a sequence of DCOPs, and how these DCOPs can be efficiently solved using Max-Sum. Next, in Section 5.4 we introduce the MASPlanes simulation platform and the benchmark algorithms we implemented. Afterwards, we empirically evaluate our proposed solutions in Section 5.5. Finally, we draw this chapter's conclusions in Section 5.6 and relate them to our research questions.

5.2 The Limited-range Online Routing Problem

In this section we present the Limited-range Online Routing Problem. First we discuss the reasons why the novel LORP is an interesting and realistic application given the current (and near-future) available technology. Then we intuitively describe the problem and present a scenario that will serve as a running example for the remainder of the chapter.

5.2.1 Problem motivation

UAVs are an attractive technology for large-area surveillance [Kingston et al., 2008]. Today, there are readily available UAVs that are reasonably cheap, have many sensing abilities, exhibit a long endurance and can communicate using radios. Several applications can be efficiently tackled with a team of such UAVs: power line monitoring, fire detection, and disaster response among others [Cox et al., 2005].

UAVs have traditionally been controlled either remotely or by following externally-designed flight plans. Requiring human operators for each UAV implies a large, specialized and expensive human workforce. Likewise, letting UAVs follow externally prepared plans introduces a single point of failure (the planner) and requires UAVs with expensive (satellite) radios to maintain continuous communication with a central station. While these constraints are acceptable in some application domains such as military operations, other applications require more flexible techniques. For instance, consider a force of park rangers tasked with the surveillance of a large natural park. Upon reception of an emergency notification, the rangers must assess the situation as quickly as possible. With this aim, they could deploy a team of UAVs to continuously fly throughout the park. Thereafter, they could issue requests for their UAVs to check certain locations. To maintain the cost-effectiveness of the approach, such UAVs cannot employ expensive communication devices. Thus, the UAVs would have limited communication ranges, oftentimes significantly smaller than the park's extension. Notice that, in this setting, neither human remote control nor centralized planning is feasible due to such communication constraints.

In this scenario, a possibility would be to deploy the planes with a fixed mission, so that they just go to the desired location(s), perform any required checks and fly back to some base. However, this approach is grossly unequipped to handle the dynamism of the problem. That is, during a critical situation (e.g., when there is an actual fire in the park), the rangers will receive many reports, and will quickly run out of UAVs to handle them. A possibly better approach involves UAVs that can act autonomously and can coordinate between them.

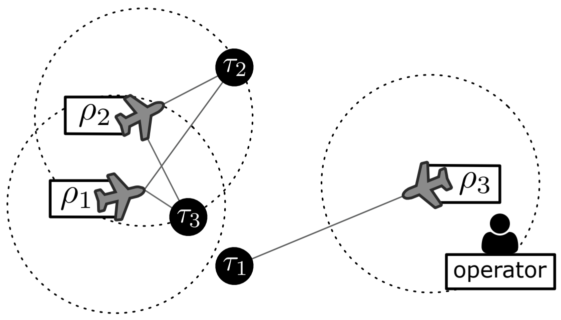


Figure 5.1: Example Limited-range Online Routing Problem scenario.

Such UAVs may keep flying around the park and coordinate to attend requests as the rangers introduce them, thus hopefully improving the UAVs' effectiveness. However, the autonomous operation and coordination of UAVs is an open research question receiving increasing attention. It involves challenges ranging from low-level operational details of flight control to high level coordination between UAVs [National Research Council, 2006]. In this work we assume that low level control can be handled entirely by the UAVs' auto-pilot systems, and focus on high-level coordination challenges instead.

There are two main approaches to enable such coordination in the literature, each best suited for a different kind of UAV missions. On the one hand, the objective in exploratory missions is to collect accurate information and keep it updated. Hence, the first approach focuses on shared information collection, fusion and maintenance techniques to fulfill these missions [Zlot et al., 2002]. On the other hand, some applications involve specific tasks that the UAVs should carry. For instance, requests introduced by the rangers can be considered as tasks to be performed by the UAVs. Hence, coordination between UAVs in this context implies making decisions about which UAV should conduct each task. Unfortunately, as explained in Section 5.2.3, most state-of-the-art multi-agent task allocation mechanisms cannot be employed in such settings, particularly due to the communication range limitation. Hence, we must identify the characteristics and particular challenges of these applications and define the Limited-range Online Routing Problem (LORP) to capture them.

5.2.2 Example scenario

Recall our example scenario above, where a force of park rangers have acquired a fleet of unmanned UAVs to help them monitor a large natural park. The force intends to request its UAVs to check certain locations when alerts are received, or as part of their routine surveillance plans. In turn, UAVs are expected to fulfill these requests as quickly as possible. Specifically, UAVs should try to minimize the average request service time, where service time is the amount of time passed between the issue of a request by an operator and the arrival of a

UAV to the request's location.[1] A common approach to this kind of problems is to adopt a centralized strategy: UAVs' routes are planned at a central station, which is in charge of commanding them. There, the central station guarantees cooperation between UAVs. However, if the natural park is significantly larger than the UAVs' communication ranges, performing centralized planning is unfeasible because the resulting plan cannot be effectively transmitted to the UAVs. Figure 5.1 represents a snapshot of what the rangers' scenario may look like at some point in time. In this example, the force owns three UAVs labeled ρ_1, ρ_2, and ρ_3. The communication range of each UAV is drawn as a dotted circle around it. By inspecting these circles, we observe that UAVs ρ_1 and ρ_2 can communicate between them. Moreover UAV ρ_3 can communicate with the rangers' operator. However, there is no way for UAVs ρ_1 and ρ_2 to communicate neither with UAV ρ_3 nor with the operator. The scenario contains three targets represented by the black circles labelled with τ_1, τ_2, and τ_3. A solid line linking a UAV with a target indicates that the UAV is aware of that target. For instance, UAV ρ_1 and UAV ρ_2 know that targets τ_2 and τ_3 exist, but they are not aware of target τ_1. Given this scenario, a centralized planner would probably send UAV ρ_1 to targets τ_3 and τ_1, UAV ρ_2 to target τ_2, and leave ρ_3 idle. However, this plan of action can never be computed nor implemented when assuming that UAVs have limited communication range, because there is simply no way for ρ_1 to discover τ_1 at this particular point in time.

As illustrated by this example, the only reasonable strategy to achieve UAV cooperation with limited communication range is to make the UAVs directly coordinate between themselves in a decentralized manner. The LORP itself does not specify how agents should represent and relay information about which requests are pending and/or completed, nor which messages can be exchanged. Hence, it is part of a LORP solution to define a specific model and an algorithm that allows agents to make specific decisions. Therefore, in the next section we explore a broad range of multi-agent coordination approaches from the literature and why those methods can or cannot be employed to deal with the LORP.

5.2.3 Related work

Although there is no extensive literature about solving highly dynamic problems using DCOPs, there exists a vast body of research related to multi-agent (and/or multi-robot) task allocation. Due to its similarity with the LORP, we focus mainly on the multi-robot routing problem [Lagoudakis et al., 2005]. In this problem a number of robots (or agents) have to visit a number of waypoints in the minimum possible time.

By far, the largest amount of such works focuses on market-based allocation mechanisms (*e.g.*, [Dias and Stentz, 2000; Gerkey and Mataric, 2002; Dias et al., 2006]), where tasks are allocated to agents using auctions. The precursor in this area is the auction algorithm [Bertsekas, 1988], an iterative distributed algorithm to optimally solve the classic *assignment problem* [Kuhn, 1955]. The assignment

[1]We assume that a request is serviced as soon as a UAV reaches its location.

problem is a static task allocation problem where several tasks can be allocated to a number of agents. Each agent has a different cost for performing each task, and can only be assigned to a single task. Then, the objective is to find the allocation that minimizes the total cost incurred after assigning all tasks.

Nonetheless, on most actual-world applications there may be more tasks than agents. If the cost for a robot to perform two tasks is simply the addition of the costs of performing each task separately, then the above algorithm still yields an optimal allocation. However, in routing problems the cost of visiting multiple waypoints depends on the relative positions between those waypoints. Hence, the auction algorithm is not optimal in this case, and can lead to arbitrarily bad solutions [Gerkey and Matarić, 2004].

A first step to mitigate this problem is introduced in [Gerkey and Mataric, 2002]. Instead of performing an iterative auction, the idea here is to perform a parallel single-item auction for each task. These auctions can be resolved in a single round. Afterwards, tasks are continuously re-auctioned, but with agents' bids updated to account for the new state of the world. This approach, described as the Parallel Single-Item (PSI) auctions mechanism in [Koenig et al., 2010], improves the robustness of the system and equips it to handle changing conditions and/or the introduction of new tasks. Although PSI auctions capture some task inter-dependencies by reallocating tasks when conditions change, another possibility is to explicitly consider them. In the Sequential Single-Item auctions [Koenig et al., 2006] method, agents perform an iterative auction where a single task is assigned at each iteration. After each round, agents update their bids according to the tasks they already got in previous iterations, hence taking into account the synergies between tasks. This approach even has quality guarantees in static domains [Lagoudakis et al., 2005], but is not well-suited for dynamic domains because it requires both an arbitrary number of communication cycles and global communication.

An entirely different approach to task allocation is to employ consensus algorithms [Alighanbari and How, 2005; Li et al., 2010] to let agents maintain consistent information about the system's state. After reaching consensus, each agent can independently compute its plan using a deterministic algorithm, and the resulting plans are guaranteed to be consistent with each other. The main advantage of these algorithms is that they have been shown to converge even on time-varying network topologies. However, the time required to reach such convergence is unbounded and they cannot converge during network partitions. Hence, consensus-based works typically strive to either guarantee network connectedness [Mosteo et al., 2008] or minimize the impact of such a potentially large decision making time. For instance, [Ren and Beard, 2008] presents a UAV coordination model for fire perimeter tracking, where UAVs reach a consensus on which section of the perimeter is to be monitored by each UAV. When the conditions change, the UAVs keep monitoring their section until a new consensus is eventually reached. Because the fire perimeter is unlikely to evolve very rapidly, the previous assignment works relatively well even if it takes a long time to reach a new consensus.

Finally, some researchers have tried to combine the advantages of market-based mechanisms (quick decisions) with those of the consensus-based ones (eventual consistency). Along these lines, Choi et al. [2009] introduced the Consensus Based Bundle Algorithm (CBBA), which directly reaches a consensus on (an auction-based) solution. Nonetheless, this approach still suffers from the low resilience to network disconnections.

As explained in the previous section, LORP scenarios are highly dynamic and involve very rapid changes in the communication network topology, including frequent disconnections. Hence, consensus-based approaches are not particularly well suited for this domain. Moreover, among all the aforementioned market-based mechanisms, it turns out that only the simpler auction-based approaches (such as PSI) can be employed for the LORP due to the lack of global communication. Therefore, PSI auctions serves as the state-of-the-art benchmark any new algorithms for the LORP should outperform.

5.3 Coordinating UAVs in the LORP

After studying the LORP's characteristics, we advocate for a solution approach where UAVs make quick decisions based solely on local information, where the neighbors of a UAV are those UAVs with which it can directly communicate at a given point in time. Thus, in this section we begin by introducing an approach that allows us to treat a LORP as a distributed request allocation problem. This approach can be realized by any task allocation model and algorithm that meets the approach's constraints.

Consequently, we then introduce a first realization of this approach that models the task allocation problem as a binary DCOP, assuming independence between requests. Thereafter we show how Max-Sum can be used to solve such DCOP. Furthermore, we show that, thanks to the DCOP being binary, we can actually avoid the exponential costs typically attached to the Max-Sum algorithm. Finally we exploit the results in [Tarlow et al., 2010] to develop a more sophisticated model where UAVs take into account their own workload when evaluating the cost to service a request with only a slight increase in computational costs.

5.3.1 An aproach based on task ownership transfers

Notice that, unlike our example scenario in Figure 5.1, the LORP is a highly dynamic problem where UAVs constantly move and new requests can be introduced at any time. As a result, any approaches to deal with the LORP must try to make quick decisions. On the one hand, a long decision-making process would lead to arbitrarily bad decisions being made. That is, a decision that was good when the decision making process started may end up being arbitrarily bad if the process itself took too long (during which the scenario's conditions kept evolving). On the other hand, making complex decisions (such as long term plans for each UAV) is futile, because there is a high probability for such plans to eventually become invalid. Likewise, operating in a decentralized manner introduces an additional challenge: how to spread and maintain a consistent view of the system between agents. For instance, a LORP solution must specify how UAVs notify others of newly introduced requests, as well as of already serviced ones.

In our approach we deal with these issues by defining three simultaneous main processes:

1. At any time, an operator may introduce a new request by notifying a single UAV in its range, which becomes the request's *owner*. So long as a UAV is the owner of a request, it is responsible for the eventual servicing of that request. However, ownership of a request may be transferred to another UAV as explained next. Since there is exactly one owner of each request at a particular point in time, this guarantees a consistent (yet distributed) view of the system by the UAVs.

2. Concurrently, UAVs run cycles of a request reallocation process. The reallocation process is a sequential process with three phases. First, the UAVs construct a snapshot of the current situation by broadcasting their location and the requests they currently own. Next, the UAVs run some decentralized request allocation algorithm based on the collected information. This phase may take several communication rounds between neighboring UAVs, and ends with specific decisions on which plane should own each request. Finally, the decisions are executed by exchanging the requests' ownership between planes. When a full reallocation cycle finishes, the UAVs start a new one with updated information.

3. Meanwhile, each UAV flies towards the nearest task it owns or tries to get in range of the closest operator if it does not own any request. When a UAV reaches the location of a request it owns, the request is considered serviced and removed from the system.

We argue that this approach is particularly well suited for the LORP. First, the ownership concept frees the allocation algorithms from having to maintain consistent situational awareness because only the current owner of a request may decide to allocate it to another UAV. Additionally, by enforcing short allocation cycles the UAVs can make decisions quickly and reconsider them as the situation evolves.

5.3.2 Coordination using Independent Valuations

In this section we show how a snapshot of a LORP problem can be encoded as a binary DCOP, assuming that requests are independent between them (i.e, that the cost for a UAV to service two requests is the same as the sum of costs of servicing them separately). Thereafter, we use Max-Sum to let UAVs compute an allocation in a distributed and efficient manner.

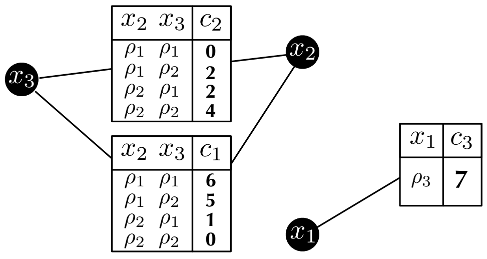


Figure 5.2: Naive DCOP encoding of the example in Figure 5.1.

Problem Encoding as a binary DCOP

Prior to encoding our problem, some notation is in place. Henceforth, Let $R = \{\tau_1, \dots, \tau_m\}$ be a set of requests, $P = \{\rho_1, \dots, \rho_n\}$ be a set of UAVs, r and p be indexes for requests and UAVs respectively, $R_p \subseteq R$ be the set of requests that UAV ρ_p can service, and $P_r \subseteq P$ be the set of UAVs that can service request τ_r. A straightforward encoding of the requests-to-UAVs allocation as a DCOP $\Omega = \langle A, X, D, C, \mathfrak{m} \rangle$ is:

- A contains one agent per UAV in P, namely $A = \{\rho_1, \dots, \rho_n\}$.
- X contains one variable x_r per request τ_r. The domain of each variable $x_r \in X$ is the set of UAVs that can service the request, namely P_r.
- $D = \{P_r \mid x_r \in X\}$ is the set of all domains of all variables.
- C contains one constraint c_p per UAV ρ_p. This constraint evaluates the cost of servicing all combinations of requests ρ_p can attend. Hence, the scope of each constraint c_p, namely $X_p = sc(c_p)$, is the set of all variables that have ρ_p in their domain.
- $\mathfrak{m}$ maps each variable x_r to the current owner of the corresponding request τ_r.

Solving the DCOP problem Ω amounts to finding the combination of request-to-UAV assignments $\mathbf{x}^*$ that satisfies

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{p \in P} c_p(\mathbf{x}[X_p]) \ .$$

Figure 5.2 shows an encoding of the motivating example in Figure 5.1. There is a variable for each request. The domain of x_1 is the set of UAVs that can service request τ_1. This is effectively the set of all UAVs that are in communication range of the owner of τ_1. Hence, the domain of x_1 is just $\{\rho_3\}$. Likewise, the domain of x_2 and x_3 is $\{\rho_1, \rho_2\}$ because both UAVs can fulfill the corresponding requests. Next, we introduce a constraint c_p for each UAV ρ_p. Because ρ_3 can only service τ_1, the scope of the constraint c_3 is x_1. As a result, c_3 is a unary

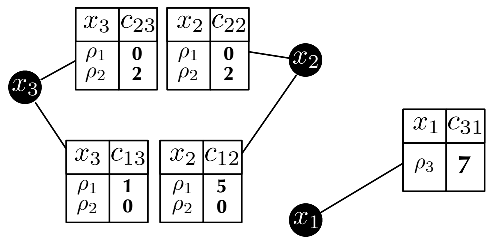


Figure 5.3: Independent valuations encoding of the example scenario.

constraint that specifies the cost for UAV ρ_3 to service τ_1, namely the distance between ρ_3 and τ_1 (hereafter δ_{pr} will be employed as a shorthand for the distance between ρ_p and τ_r). The scope of c_2 is $\{x_2, x_3\}$, because UAV ρ_2 can service both τ_2 and τ_3. Hence, c_2 specifies four costs for ρ_2:

1. 0 if both requests are allocated to ρ_1.
2. $2 = \delta_{23}$ if τ_2 is allocated to ρ_1 and τ_3 is allocated to ρ_2.
3. $2 = \delta_{22}$ if τ_2 is allocated to ρ_2 and τ_3 is allocated to ρ_1.
4. $4 = \delta_{22} + \delta_{23}$ if both requests are allocated to ρ_2.

Finally, the costs of c_1 are similarly computed. From the costs shown in Figure 5.2, we can now compute the optimal assignment $\mathbf{x}^* = \langle x_1 = \rho_3, x_2 = \rho_2, x_3 = \rho_1\rangle$.

The problem of this encoding is that it scales poorly. First, notice that the c_p constraints do not exploit the fact that we assume independence between requests. Therefore, the number of assignments in c_p is the product of the domain sizes of each of the variables in its scope. As a result, c_p grows exponentially larger with respect to the number of requests that UAV ρ_p can service.

However, we can exploit the independence between requests by decomposing each cost constraint c_p into smaller cost constraints, each one evaluating the cost of servicing a single request. That is, thanks to that independence between requests, we can represent c_p as a combination of cost constraints c_{pr}, one per variable in the scope of c_p, such that

$$c_p(\mathbf{x_p}) = \sum_{x_r \in X_p} c_{pr}(\mathbf{x_r}) \ .$$

With this transformation, the number of values to specify the cost of servicing a set of requests scales linearly with respect to the number of requests. Moreover, the new encoding represents exactly the same problem than before.

Figure 5.3 represents the same example in Figure 5.2, but using this new encoding. Notice that for each UAV we specify the cost of servicing a given request when the request is assigned to that UAV, or 0 when it is allocated to another UAV.

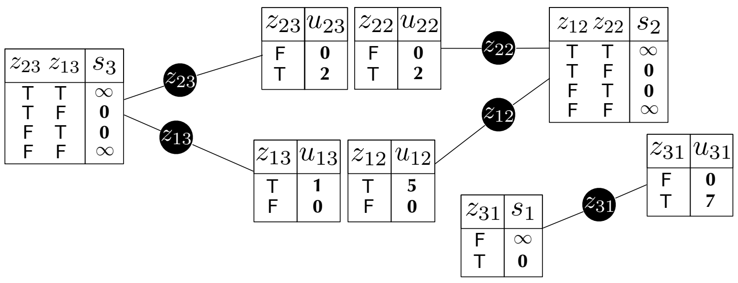


Figure 5.4: Binary independent valuations encoding of the example scenario.

However, even this improved encoding suffers from redundancy. To see how, consider another UAV ρ_4 is in the communication range of both ρ_1 and ρ_2. Since this UAV would be eligible to serve requests τ_2 and τ_3, the domain of x_2 and x_3 would become $\{\rho_1, \rho_2, \rho_3\}$. As a result, UAV ρ_1 would have to extend its cost constraint c_{12} to include a new assignment where τ_2 is assigned to ρ_4, whose cost is obviously 0.

In practice, the cost for UAV ρ_1 depends only on whether a request is allocated to it or not, disregarding which other UAV got it in the latter case. Therefore, we must aim at an encoding such that each cost constraint c_{pr} contains only two values: δ_{pr} if τ_r is allocated to ρ_p, or 0 otherwise. With this aim, we now convert all variables in X to binary variables. That is, we replace each original variable $x_r \in X$ by a set of binary variables z_{pr}, one per UAV in P_r. Previous c_{pr} constraints now generate u_{pr} constraints on these binary variables. In addition, for each request r, the variables z_{pr} are linked through a selection constraint s_r to ensure that a request can be only serviced by a single UAV. Formally, we re-encode the naive DCOP above into a new binary DCOP $\Omega' = \langle A, Z, D', C', \mathfrak{m}' \rangle$, where

- Z is the set of all binary variables z_{pr}, one for each UAV ρ_p and request that UAV can service $\tau_r \in R_p$. The domain of each variable z_{pr} is the set $D_{pr} = \{true, false\}$, where $true$ means that request τ_r is allocated to UAV ρ_p and $false$ means that it is not.

- D' is the set containing all $D_{pr} = \{\mathsf{T}, \mathsf{F}\}$ binary domains.

- $C' = U + S$ is the set containing all *cost* constraints $u_{pr} \in U$ and all *selection* constraints $s_r \in S$.

- Function $\mathfrak{m}'$ maps each variable z_{pr} to UAV ρ_p.

For instance, consider variable x_2 from our example, whose domain is $\{\rho_1, \rho_2\}$. We create two binary variables z_{12} and z_{22}. Intuitively, z_{12} taking value T (also known as z_{12} being *active*) means that request τ_2 is assigned to UAV ρ_1, whereas z_{12} taking value F (being *inactive*) means that τ_2 is not assigned to ρ_1. Then, a selection constraint s_2 guarantees that request τ_2 is assigned to one and only one UAV. In our example, the constraint s_2, with scope $\{z_{12}, z_{22}\}$, introduces an infinite cost unless exactly one of those two variables is active. After applying these transformations, Figure 5.4 shows the binary encoding of the example in Figure 5.3.

Solving the Problem with Max-Sum

Now we optimize the Max-Sum algorithm to run on the encoding shown in Figure 5.4. As explained in Section 2.2.2, Max-Sum sends messages between the nodes in the factor graph representation of the problem. Being an algorithm of the GDL family, the messages these nodes must compute are analogous to those shown in Equation (4.1) from Chapter 4. However, those messages can be specialized into the expressions below as shown in Appendix A.

On the one hand, the message from a variable z to a constraint f is simply the combination of all messages z received from its other neighbors. Namely,

$$\mu_{z\to f}(\mathbf{z}) = \sum_{f' \in N(z)\setminus\{f\}} \mu_{f'\to z}(\mathbf{z}) \tag{5.1}$$

where $N(z) \setminus \{f\}$ is the set of neighbors of variable z except from f.

However, our factor graph allows for some simplifications. First, notice that each z_{pr} is only linked to the corresponding cost constraint u_{pr} and the selector constraint s_r. That is, each variable has exactly two neighbors. Hence, it is direct to observe from Equation (5.1) that the message that z_{pr} must send to u_{pr} is exactly the one received from s_r, while the message that it must send to s_r is exactly the one received from u_{pr}. Because each variable simply relays messages between the cost constraint and the selection constraint it is linked to, henceforth we disregard the variables' messages and instead consider that constraints directly exchange messages.

On the other hand, the message from a constraint f to a variable z is

$$\mu_{f\to z}(\mathbf{z}) = \min_{\substack{\mathbf{t}\text{ extension of }\mathbf{z}\\ \text{to } sc(f)}} \left(f(\mathbf{t}) + \sum_{z' \in N(f)\setminus\{z\}} \mu_{z'\to f}(\mathbf{t}[z']) \right) . \tag{5.2}$$

Observe that the messages sent by a constraint are always defined over a single variable. That is, in our case, the messages exchanged between a u_{pr} cost constraint and an s_r selection constraint refer to assignments on binary variable z_{pr}. Therefore, one such message must contain two values, one per possible assignment to z_{pr}. At this point, we can make a further simplification and consider sending the difference between the two values. Intuitively, a constraint sending a message with a single value for a binary variable transmits the difference between the cost when that variable is active and the cost when that variable is inactive. In general, we define the single-valued message exchanged between two constraints as

$$\nu_{f\to g} = \mu_{f\to g}(\mathsf{T}) - \mu_{f\to g}(\mathsf{F}) . \tag{5.3}$$

At this point we can compute the messages sent between our cost and selection constraints using Equations (5.2) and (5.3).

- *From cost constraint to selection constraint.* This message expresses the difference for a UAV ρ_p between serving request τ_r or not, therefore

$$\nu_{u_{pr}\rightarrow s_r} = u_{pr}(\mathsf{T}) - u_{pr}(\mathsf{F}) = \delta_{pr} - 0 = \delta_{pr}\ . \tag{5.4}$$

- *From selection constraint to cost constraint.* Consider the selection constraint s_r and the cost constraint u_{pr}. From Equation (5.2), we obtain

$$\mu_{s_r\rightarrow u_{pr}}(\mathsf{T}) = 0, \quad \text{and} \quad \mu_{s_r\rightarrow u_{pr}}(\mathsf{F}) = \min_{\rho_{p'}\in P_r\setminus\{\rho_p\}} \delta_{p'r}\ .$$

  Then we can apply Equation (5.3) to obtain the single-valued message

$$\nu_{s_r\rightarrow u_{pr}} = -\min_{\rho_{p'}\in P_r\setminus\{\rho_p\}} \delta_{p'r}\ .$$

  Moreover, this message can be computed efficiently. Consider the pair $\langle\nu^*, \nu^{**}\rangle$ as the two lowest values received by the selection constraint s_r. Then, the message that this constraint s_r must send to each cost constraint u_{pr} is

$$\nu_{s_r\rightarrow u_{pr}} = \begin{cases} -\nu^* & \nu_{u_{pr}\rightarrow s_r} \neq \nu^* \\ -\nu^{**} & \nu_{u_{pr}\rightarrow s_r} = \nu^* \end{cases}. \tag{5.5}$$

To summarize, each cost constraint computes and sends messages using Equation (5.4), whereas each selection constraint computes and sends messages using Equation (5.5).

Max-Sum operation. Max-Sum is an approximate algorithm in the general case, but it is provably optimal on trees. Due to how we encode the LORP, the resulting factor graph always contains a disconnected, tree-shaped component around each selector constraint s_r. For instance, Figure 5.5 shows the optimized factor graph of our example, and identifies each disconnected component. Moreover, the algorithm is guaranteed to converge after traversing the tree from the leaves to the root and then back to the leaves again. In our case, the tree-shaped component for each request is actually a star-like tree, with the selection constraint s_r at the center, and the corresponding cost constraints u_{pr} connected to it. Hence, we are guaranteed to compute the optimal solution in two steps.

Typically, Max-Sum's decisions are made according to the costs seen by the variable nodes after running the algorithm. However, we have no variables in our graph anymore because we eliminated them. As a consequence, we have to make the decisions at either the selector nodes or at the cost nodes. The best option is to let the selectors choose, because it guarantees that the same task is never simultaneously assigned to two different UAVs. Furthermore, because the decisions are made by the selector nodes, there is no need for the second Max-Sum iteration (messages from selector to cost constraints) anymore.

Notice that, after our refinements, the only remaining Max-Sum nodes are the cost constraints u_{pr} and a selection constraint s_r for each request. Hence,

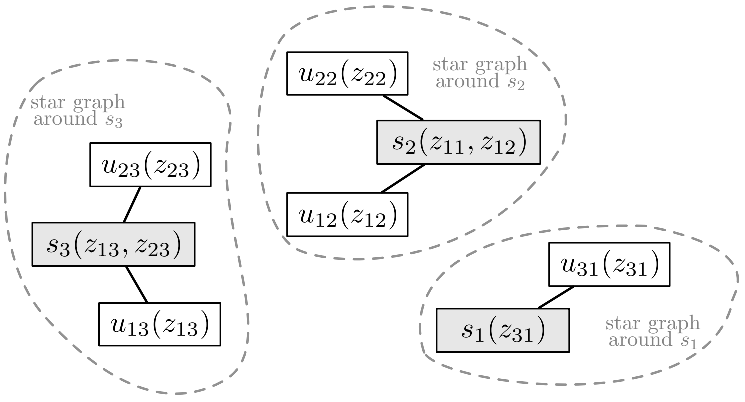


Figure 5.5: Optimized graph for Max-Sum execution assuming independent task valuations.

the UAVs can execute the algorithm distributedly by letting each plane ρ_p run the cost constraints u_{pr} as well as the selection constraints of those requests it currently owns. Moreover, a UAV only needs to know its neighbors' requests to build the logical nodes it must run. Thus, the whole graph can also be built distributedly based only on each UAVs' local information.

Max-Sum runs on our example as follows. First, each leaf cost constraint u_{pr} must send its cost to the root of its tree, s_r. That is, UAV ρ_1 sends 1 to s_3 (within UAV ρ_2), and 5 to s_2 (within itself). Likewise, UAV ρ_2 sends 2 to s_2 and 2 to s_3, whereas UAV ρ_3 sends 7 to s_1. Thereafter, the selector nodes s_r decide by choosing the UAV whose message had a lower cost. Hence, s_3 (running within UAV ρ_2) decides to allocate τ_3 to ρ_1, s_2 allocates τ_2 to ρ_2, and s_1 allocates τ_1 to ρ_3. At this point the owner of each request knows whether a request should be reallocated or not and to whom, so we can use it to implement the reallocation phase of our approach described in Section 5.3.1.

5.3.3 Coordination using Workload-based Valuations

In realistic scenarios, requests do not appear uniformly across time and space. Instead, when some emergency occurs, requests should be concentrated around the problematic area, hereafter referred to as *hot spot*. Our hypothesis is that the assumption of independence between requests may be too strong in this case. That is, such assumption can lead to unbalanced allocations, where some UAVs (the ones closer to the hot spot) are overloaded while others remain idle. Therefore, we now show that it is possible to relax this assumption while keeping an acceptable time complexity for Max-Sum. With this aim, we introduce a new factor for each UAV: a penalty that grows as the number of requests assigned to that UAV increases. Formally, let $Z_p = \{z_{pr} \text{ such that } \tau_r \in R_p\}$ be the set of variables encoding the assignment to UAV ρ_p. Let $\boldsymbol{\eta}_p$ be the number of requests assigned to UAV ρ_p, namely $\boldsymbol{\eta}_p = |\{z_{pr} \in Z_p \text{ such that } z_{pr} \text{ is active}\}|$. The workload factor for UAV ρ_p is then

$$w_p(\mathbf{Z}_p) = k \cdot (\boldsymbol{\eta}_p)^\alpha \quad , \tag{5.6}$$

where $k \geq 0$ and $\alpha \geq 1$ are parameters that can be used to control the fairness in the distribution of requests (in terms of how many requests are assigned to each UAV). Thus, the larger the α and the k, the fairer the request distribution.

The direct assessment of Max-Sum messages going out of the workload factor takes $O(N \cdot 2^{N-1})$ time, where $N = |Z_p|$. Interestingly, the workload factor is a particular case of a *cardinality potential* as defined by Tarlow *et.* al. [Tarlow et al., 2010]. A cardinality potential is a factor defined over a set of binary variables (Z_p in this case) that does only depend on the number of active variables. That is, it does not depend on which variables are active, but only on how many of them are active. As described in [Tarlow et al., 2010], the computation of the Max-Sum messages for these potentials can be done in $O(N \cdot \log N)$ time. Thus, using Tarlow's result we can reduce the time to assess the messages for the workload factors from exponential in the number of variables to linearithmic.

To further speedup the algorithm and reduce message exchanges, we can add the workload factor and the cost factors that describe the cost for UAV ρ_p to service each of the requests. In Lemma B.2 we prove that if we have a procedure for determining the Max-Sum messages going out of a factor over binary variables, say f, we can reuse it to determine the messages going out of a factor h which is the sum of f with a set of independent costs, one for each variable.

Algorithm 4 process(w'_p) ▷ All undefined values are ∞

```
1: for $r \in R_p$ do                                   ▷ Incorporate incoming messages
2:     $I[r] = \nu_{s_r \to w'_p} + c_{pr}(1)$
3: end for
4: $Pos, I' = \text{sorted}(I)$        ▷ $Pos$ are the reverse indices, and I' the sorted list
5: $cs = 0$
6: for $i = 0$ to $|I'|$ do                              ▷ Compute cumulative sums
7:     $CS^0[i] = cs + w_p(i)$
8:     $CS^-[i] = cs + w_p(i-1)$
9:     $CS^+[i] = cs + w_p(i+1)$
10:    $cs = I'[i] + cs$
11: end for
12: for $i = 0$ to $|I'|$ do                             ▷ Compute cumulative mins
13:    $M^+[i] = min(CS^+[i], M^+[i-1])$
14:    $M^L[i] = min(CS^0[i], M^L[i-1])$
15:    $M^R[i] = min(CS^0[|I'|-i], M^R[|I'|-i+1])$
16:    $M^-[i] = min(CS^-[|I'|-i], M^-[|I'|-i+1])$
17: end for
18: for $r \in R_p$ do                                  ▷ Compute and send messages
19:    $i = Pos[r]$
20:    $\xi_0 = min(M^L[i-1], M^-[i+1] - I'[i])$
21:    $\xi_1 = min(M^+[i-1], M^R[i+1] - I'[i])$
22:    $\nu_{w_p \to s_r} = \xi_1 - \xi_0 + c_{pr}(1)$
23:    send($\nu_{w_p \to s_r}$)
24: end for
```

Thus, we can define a single factor that expresses the complete cost of a UAV when assigned a set of requests, which amounts to the sum of the independent costs for each of the requests assigned plus the workload cost for accepting that number of requests. Formally the cost factor for UAV ρ_p is

$$w'_p(\mathbf{Z}_p) = w_p(\mathbf{Z}_p) + \sum_{\tau_r \in R_p} u_{pr}(z_{pr}) \quad . \tag{5.7}$$

Algorithm 4 defines how to assess the messages flowing out of the cost factor w'_p. It is obtained by composing Lemma B.2 with the algorithm described in [Tarlow et al., 2010] for cardinality potentials.[2] The algorithm is basically a dynamic programming procedure for computing minimizations over cumulative sums, which can be done in a few linear passes. The complexity is dominated by the initial sort operation, and hence the $O(N \cdot \log N)$ runtime.

Summarizing, by introducing workload valuations that do not only depend on each individual request, but also on the number of requests, we have shown that it is possible to relax the assumption of independence between valuations with a very minor impact on the computational effort required to assess the messages (from linear to linearithmic). In the next section we show that this relaxation provides significantly better allocations in terms of minimizing the service time across requests.

5.4 The MASPlanes toolkit

In the remainder of this chapter we want to empirically evaluate our DCOP-based solutions for the LORP. Being a novel problem, there exist no development and testing tools for LORP solving approaches and algorithms. Hence, in this section we first describe MASPlanes, the simulation environment we developed and open-sourced to compare the different coordination mechanisms identified and proposed in this chapter. Thereafter we introduce the state-of-the-art methods provided by MASPlanes, which serve as a baseline to test new approaches for the LORP.

5.4.1 Simulation environment

Most current UAV simulators are mainly designed for the development of low-level flight control and coordination techniques [Jang et al., 2005; Rasmussen et al., 2005; Garcia and Barnes, 2010]. They employ highly accurate physics and component models to produce as accurately as possible simulations. These are very complex simulators that require significant specific knowledge and domain expertise. Furthermore, being built for such a specialized purpose they do not emphasize the decoupling of their different components. As a consequence,

[2]The algorithm description provided in [Tarlow et al., 2010] is inaccurate. However, the authors own implementation of the algorithm is correct. It does not match the description in [Tarlow et al., 2010] and is closer to the description provided by Algorithm 4.

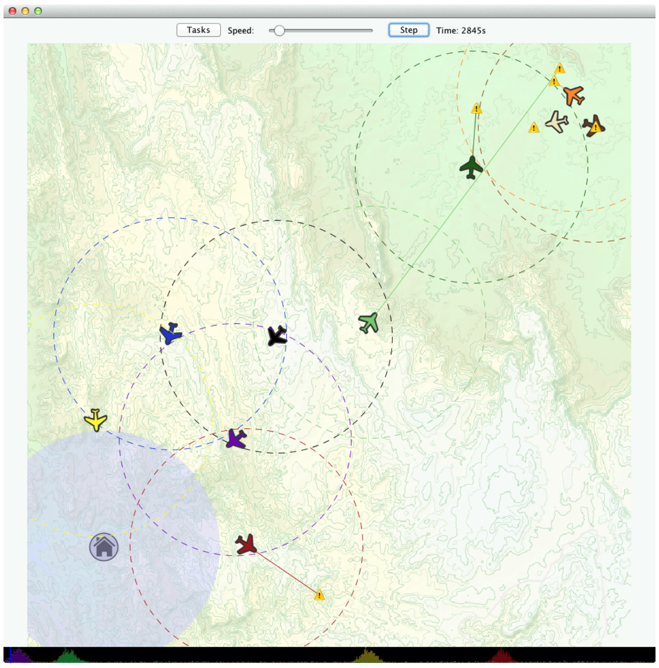


Figure 5.6: MASPlanes graphical user interface.

it is largely impractical to adapt them to perform more specialized problem-specific experiments under varying assumptions. Additionally, they are very computationally demanding, making their usage impractical to conduct the large number of experiments required to asses the behavior of a novel coordination algorithm. Happe and Berger [Happe and Berger, 2010] identified these issues and developed the CoUAV simulator. However, they focused on the problem of cooperative search and exploration instead of dynamic task allocation, so their simulator is not applicable to our problem domain.

Due to the aforementioned limitations, we developed MASPlanes [Pujol-Gonzalez et al., 2014b], an easy-to-use simulation environment specifically geared towards testing coordination methods for the LORP. The MASPlanes environment has two main components: the problem generator and the simulator.

The problem generator allows to randomly generate problem instances based on a wide number of parameters (e.g.: number, speed and communication range of the UAVs, frequency and spatial distribution of the incoming requests among others). Users define a scenario by fixing these parameters, and then they can generate as many problem instances of that scenario as required. A problem instance completely defines the characteristics of each participating UAV, as well as all requests that will be introduced during the simulation. As a consequence, we can compare the performance of different UAV coordination mechanisms on the very same dynamic situations.

MASPlanes performs step-based simulations, a common simulation technique in the multi-agent community [Railsback et al., 2006]. Currently, the simulator assumes that UAVs move at a constant speed and that they fly at different heights (so there is no need to actively avoid collisions). The main loop of the simulator is straightforward. First, the kernel initializes all participating agents. Thereafter, a *step* is performed every 100 ms, where each agent is given the opportunity to receive, compute, act and send messages. Messages sent at some step are not delivered until the next one. The simulator keeps performing steps until all requests have been introduced into the system and serviced by the UAVs. After that, the simulator reports statistics about the requests' servicing time, distances travelled by the UAVs and total run time.

There are two main agent types within the system: operators and UAVs. Operators are in charge of requesting UAVs to check some locations. Hence, during the initialization procedure, operator agents load the list of requests they will introduce from the problem definition. This list specifies each location that must be visited and when to introduce the request. When the time comes, an operator introduces a request into the system by sending a message to one of the UAVs within its communication range. If no UAV is reachable by the operator at that particular point in time, the operator keeps the request until some UAV enters its range and then delivers it.

The basic UAV agents provided by the platform implement our approach from Section 5.3.1, but without performing any request reallocation procedure. Thus, users can extend these basic UAV agents to implement the coordination algorithm of their choice. To illustrate that, the framework includes a collection of state-of-the-art coordination algorithms, both centralized and distributed as detailed below.

At run-time, the user selects which coordination mechanism to test and can monitor the resulting behavior of the UAVs through the simulator's GUI shown in Figure 5.6. This allows the algorithm designer to observe the emergent behaviors resulting from her algorithm, providing valuable insight on how it operates in practice. Finally, the simulator can also run in *batch* mode, where it simply runs the simulation until the end and reports statistics about that particular problem instance solved with the specified coordination mechanism and settings.

5.4.2 Benchmark algorithms

Comparing the performance of LORP coordination methods against the current state-of-the-art is tricky because, as explained in Section 5.2.3, most current methods cannot cope with the requirements of our problem domain. Therefore, MASPlanes also implements a relaxed version of the problem to compare against those methods that cannot actually be employed to solve the LORP. In this relaxation, the allocation is performed by a central agent that has no communication restrictions. Furthermore, the centralized allocation procedure used by the centralized agent is considered to be instantaneous. That is, there is no delay at all introduced by the allocation procedure, whatever its computational cost may be. The only constraint that such central planner enforces is that a

Algorithm 5 GreedyAllocation(P, R, V, A)

```
1: while $R \neq \{\}$ do
2:     // Choose the request-to-UAV allocation with minimum cost
3:     $\langle \tau^*, \rho^* \rangle = \arg\min_{\tau \in T, \rho \in R, \tau \in V[\rho]}$ evaluate($r, A[\rho]$)
4:     $A[\rho^*] = $ insert($\tau^*, A[\rho^*]$)
5:     $R = R \setminus \{\rho\}$
6: end while
```

UAV can not be assigned to a request whose existence is unknown by that UAV. Additionally, MASPlanes also provides two readily-available request allocation algorithms to make decisions using this relaxed problem that we describe next.

First, the *c-hungarian* algorithm runs the Hungarian method [Kuhn, 1955] to compute allocations. This method optimally solves the assignment problem in $O(n^3)$ time. However, recall that the LORP is not an assignment problem because: (i) unassigned requests have to be performed later on; (ii) the cost of servicing a request changes depending on the order in which they are serviced; and (iii) new requests may be introduced at any future time. Also notice that the solutions obtained using this method correspond to those that would be found by the auction algorithm [Bertsekas, 1988]. Aside from the actual procedure, the only difference between these methods is that the auction algorithm may be distributed, at the expense of additional delays caused by communications. Hence, the results obtained by this method represent an upper bound on the best results that a distributed auction algorithm would obtain.

Second, the *c-greedy* method represents a straightforward sequential greedy algorithm as presented in Algorithm 5. Initially, the central agent considers that no request is allocated to any UAV. Then it computes the best allocation (the allocation with minimum cost) of a single task to a single UAV. Subsequently, the algorithm keeps finding the best possible allocation of a single request to some UAV, but taking into account the previously allocated requests. The process ends when all requests have been assigned to some UAV. This procedure can be seen as a centralized version of the Sequential Single Item [Koenig et al., 2010] auctions mechanism. Hence, this solution represents an upper bound of the results we would be able to achieve while using SSI auctions if we were somehow able to use them. Because we want to minimize the average service time, the *c-greedy* algorithm employs the recommended *BidMinPath* bidding rule from [Lagoudakis et al., 2005] as the *evaluate()* function in Algorithm 5. Finally, notice that this method is also equivalent to the *SGA* method presented in [Choi et al., 2009]. This implies that, as proven in that work, the decisions obtained by *c-greedy* are equivalent to the allocation that would be obtained by the Consensus-Based Bundle Algorithm (CBBA) in the ideal case of a static network structure. Thus, *c-greedy* also represents an upper bound on the quality of the CBBA if we were able to employ it.

For completeness, we also implemented centralized versions of our algorithms: *c-independent*, which represents our solution using independent valuations; and *c-workload*, which uses workload valuations as detailed in Section 5.3.3.

Method	**is the ideal implementation of**	**if we disregard**
c-hungarian	Auction algorithm	CD, SA
c-greedy	CBBA	CD, ND, SA
	SSI auctions	CD, ND, SA
c-independent	PSI auctions	CD, SA
	Independent valuations	CD, SA
c-workload	Workload valuations	CD, SA
d-independent	PSI auctions	-
	Independent valuations	-
d-workload	Workload valuations	-

Table 5.1: Relationship of the implemented algorithms with other state-of-the-art approaches. CD, DN and SA stand for Communication Delays, Network Dynamicity (changes in the communication network) and Situational Awareness (knowledge mismatches between agents) respectively.

All these centralized algorithms serve as baselines to assess the performance of the actually distributed solutions. Obviously, the distributed solutions implemented in MASPlanes include our DCOP-based solutions *d-independent* (using independent valuations) and *d-workload* (using workload valuations). Additionally, we implemented the distributed state-of-the-art PSI method, and discovered an interesting connection: the PSI method and our *d-independent* method are functionally equivalent. This can be seen by following the example execution in Section 5.3.2, but interpreting it as if the selector constraints opened an auction for their task, and the cost constraints bid for the corresponding requests. In other words, we have found out that running Max-Sum over an optimized model that assumes independence between requests is exactly the same as running the well-known PSI auctions method. Due to this surprising match, in the following section we only report results about *c-independent* and *d-independent*, but these represent also the results obtained by the centralized and distributed PSI methods respectively.

To summarize, Table 5.1 presents the algorithms we implemented in MASPlanes and their relationship with other state-of-the-art approaches. Under the assumptions presented in that table, the algorithm's decisions would match those that we obtain in our evaluation. However, this does not necessary imply that the algorithms cannot work without such assumptions. For instance, CBBA can operate on a dynamic network, but would obtain worse results than those of *c-greedy*.

5.5 Empirical evaluation

In the last section we introduced the MASPlanes simulation toolkit and the algorithms we implemented to benchmark our proposed DCOP-based approaches. Therefore, in this section we explore the behavior of our novel approaches under

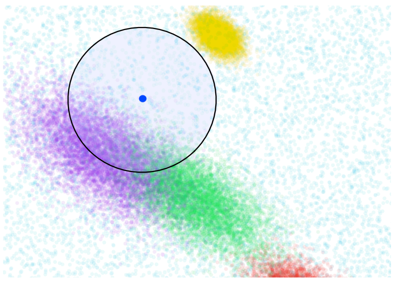


Figure 5.7: Example of *hot spots* distribution of requests.

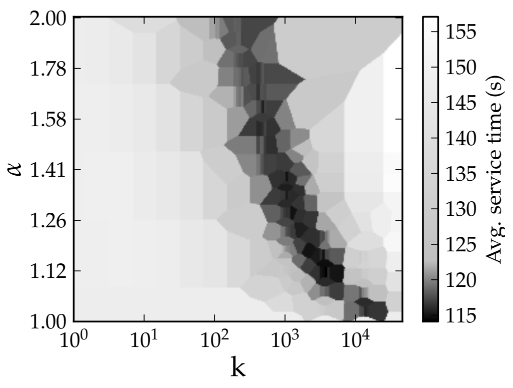


Figure 5.8: Parameter exploration in the *hot spots* scenarios.

different conditions to asses how they fare against the current state-of-the-art solutions. First, we aim to validate the hypothesis that different spatial distributions of the incoming requests may favor some methods over others. Thereafter, in Section 5.5.2 we thoroughly explore the behavior of our proposed algorithms on a wide range of scenarios.

5.5.1 Effects of the spatial distribution of requests

Because our hypothesis about hot spots is independent of whether the algorithms can or cannot work in a distributed manner, we start by comparing all algorithms using their idealistic (centralized) implementation. Thereafter, we compare the results of *c-independent* and *c-workload* with their actually applicable *d-independent* and *d-workload* distributed counterparts.

Empirical settings

We prepared two scenarios that represent a time-span (T) of one month. During that time, 10 UAVs with a communication range of 2 km survey a square field of 100 km^2. We assume that the UAVs always travel at a cruise speed of 50 km/h. In these scenarios, a single operator submits requests at a mean rate of one request per minute. However, we introduce four crisis periods during which the rate of requests is much higher. To accomplish this, the requests submission times are sampled from a mixture of distributions. The mixture contains four normal distributions $\mathcal{N}_i(\mu_i, 7.2\,\text{h})$ (one per crisis period) and a uniform distribution for the non-crisis period. The μ_i means themselves are sampled from a uniform distribution $\mathcal{U}(T)$.

The two scenarios differ on the spatial distribution of requests. In the *uniform* scenario, requests are uniformly distributed along space, whereas the *hot spot* scenario models a more realistic setting where crisis requests are localized around hot spots. These spatial hot spots are defined as bivariate Gaussian distributions

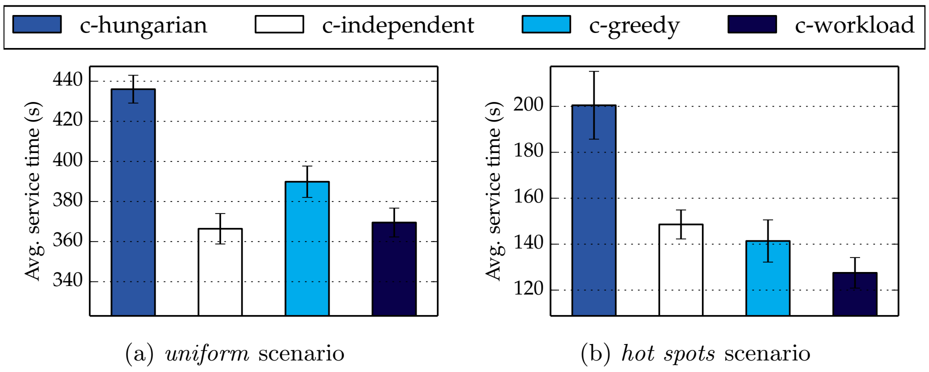


Figure 5.9: Effects of the spatial distribution of requests. Error bars show the standard error of the mean.

with randomly generated parameters. Figure 5.7 depicts an example of such scenario, where colored dots stand for requests. The scattered dots correspond to non-related requests, whereas related requests form dot clouds around their hot spot. Finally, the strong dot represents the operator, and the circle surrounding it shows the operator's communication range.

To use the *c-workload* method we have to set values for the k and α parameters. Hence, we performed an exploration on the space of these parameters to determine which values are suitable to the *hot spot* scenarios. Figure 5.8 shows the results we obtained after this exploration. The colors correspond to the median of the average service time that we obtained after running the algorithm in 30 different scenarios for each (k, α) pair. For instance, when $k = 10^2$ and $\alpha = 1.12$ the algorithm achieved a median average service time of 137 s. Observe that the algorithm exhibits a smooth gradient for any fixed value of α or k. Hence, good combinations of k and α can be found by fixing one parameter to a reasonable value and performing a descent search on the other one. For instance, we chose $k = 1000$, and found the best corresponding α to be 1.36 with 0.01 precision. We also conducted a similar exploration in the uniform scenario. However, the results were similar for k values between 10^2 and 10^5, and α values between 1.01 and 1.68. Therefore, we use the same values than in the hot spot scenario.

Results

Next we ran all centralized algorithms on a new set of 30 new problems of each scenario to ensure that the parameters were not overfitted. Figure 5.9a shows the average service time (time since the request is generated until it is serviced) achieved by the different algorithms in the *uniform* scenario. The best performing algorithm is *c-independent*, with *c-workload* being just 1% worse. Despite the small difference, all results are found significant by the (paired, nonparametric) Wilcoxon signed rank test [Wilcoxon, 1945] with $p = 0.01$. The

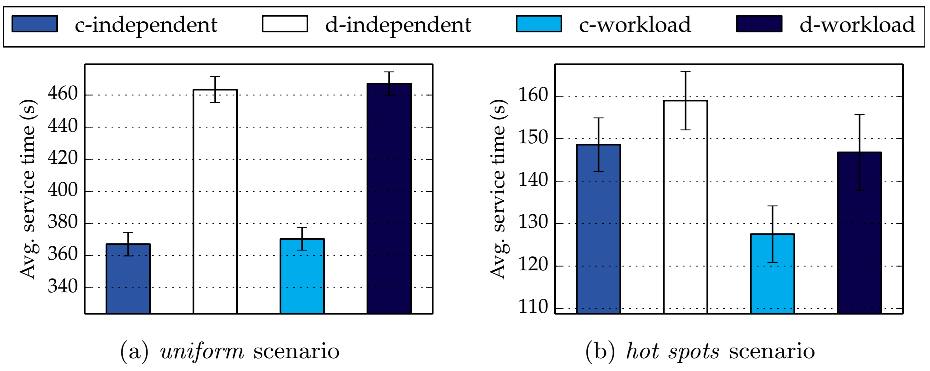


Figure 5.10: Impact of distributed solving.

worst algorithm is clearly *c-hungarian*, taking nearly 16% more time on average.

These results contrast with those of the *hot spot* scenario. As shown in Figure 5.9b, *c-workload* rises as the best method in this case, taking 10% less time than *c-greedy* and 14% less time than *c-independent*. Interestingly, this means that both *c-workload* and *c-greedy* are able to capture and exploit the underlying spatial distribution of requests, whereas *c-independent* works very well when the requests are uniformly distributed but is not well equipped to handle other, more realistic distributions. Finally, *c-hungarian*'s results show that completely disregarding the dynamism of the problem leads to overall bad decisions.

At this point, it is clear that the underlying spatial distribution of tasks has different effects on the varying methods. However, we must check whether such differences still hold in the actually applicable distributed versions. Hence, we ran the same scenarios, but this time using *d-independent* and *d-workload* to compare their results with the respective idealized cases. Figure 5.10 shows the results obtained on both scenarios. Notice that the differences persist, with *d-independent* achieving insignificantly ($p = 0.01$) better results than *d-workload* in the uniform scenario while being significantly worse in the *hot spots* one. In fact, *d-workload* turns out to be equally as good as *c-independent* in the uniform scenario according to the Wilcoxon test.

Aside from confirming our hypothesis about the spatial distribution of requests, this experiment also provides an idea about the cost of distributed solving. Specifically, it shows that there's a 20% time increase between the distributed methods and their idealized counterparts in the *uniform* scenario, and between 6% ({c,d}-independent) and 13% ({c,d}-workload) in the *hot spots* one. The lower overhead in the *hot spots* scenario is due to requests being closer between them, giving the planes more opportunities to coordinate.

5.5.2 Exploring d-workload's behavior

In the previous section we presented results on two scenarios that differed only on the spatial distribution of requests. However, it is still unclear how *d-workload* will perform in other scenarios. Hence, in this section we aim at providing a better insight on the behavior of *d-workload* under varying scenario's characteristics.

Empirical settings

To this end, we chose a number of scenario's characteristics and prepared a full factorial experiment to evaluate their impact on the different algorithms. To capture a wide range of scenarios yet keep the number of experiments within a reasonable number, we selected the following four characteristics:

- **Load.** Defines the overall amount of requests per UAV. To keep things simple, we always introduce a total of $43,200$ requests (1 req/min on average), so the load is controlled solely by varying the number of available UAVs.
- **Spread.** Captures how spatially spread the requests are between them. In scenarios with low spread, the requests tend to be clustered together around hotspots. In high-spread scenarios, the tasks are scattered around a larger area. We control the spread of the scenario by defining a *hostspot-radius*. In the problems of that scenario, approximately 90% of the tasks of each hotspot will be within hotspot-radius distance from the hotspot center. Such distribution is achieved by sampling from an Inverse Wishart distribution with 2.5 degrees of freedom and a scale matrix experimentally adjusted to produce locations within the desired hotspot radius.
- **Communication range.** Represents the UAVs' (and operator) communication ranges.
- **Time distribution sharpness.** Specifies how constant is the rate at which requests are introduced. Because we keep a constant number of tasks for all scenarios, sharpness can be defined fully in terms of the number of crises. In high sharpness scenarios, there is a single short and highly active crisis period. In contrast, low sharpness scenarios contain nine different crisis periods of moderate activity, but no highly active period.

The chosen values for each feature are detailed in Table 5.2. For the characteristics not described here we employed the values reported in Section 5.5.1. Namely, the scenarios represent a full month of simulated time, with planes that move at a constant speed of 50 km/h on a square area of $100\,\text{km}^2$.

Once again, we must decide the k and α values to use in our *d-workload* method. From the previous results, we know that a good value for α can be found for any reasonable k value. Therefore, we fixed $k = 1000$ and explored on the α parameter. We generated an example problem from each of the above

Feature	**Parameter**	**(Low, Mid, High)**
Load	number of planes	20, 10, 5
Spread	hotspot-radius	1 km, 3 km, 6 km
Communication range	communication range	1 km, 2 km, 3 km
Time distribution sharpness	number of crises	9, 3, 1

Table 5.2: Problem dimensions explored and their values

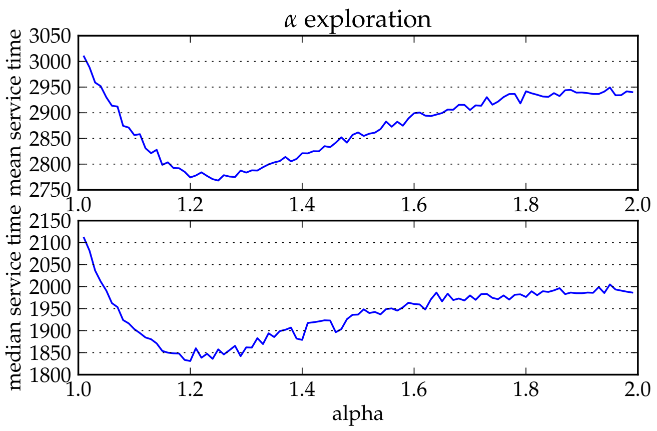


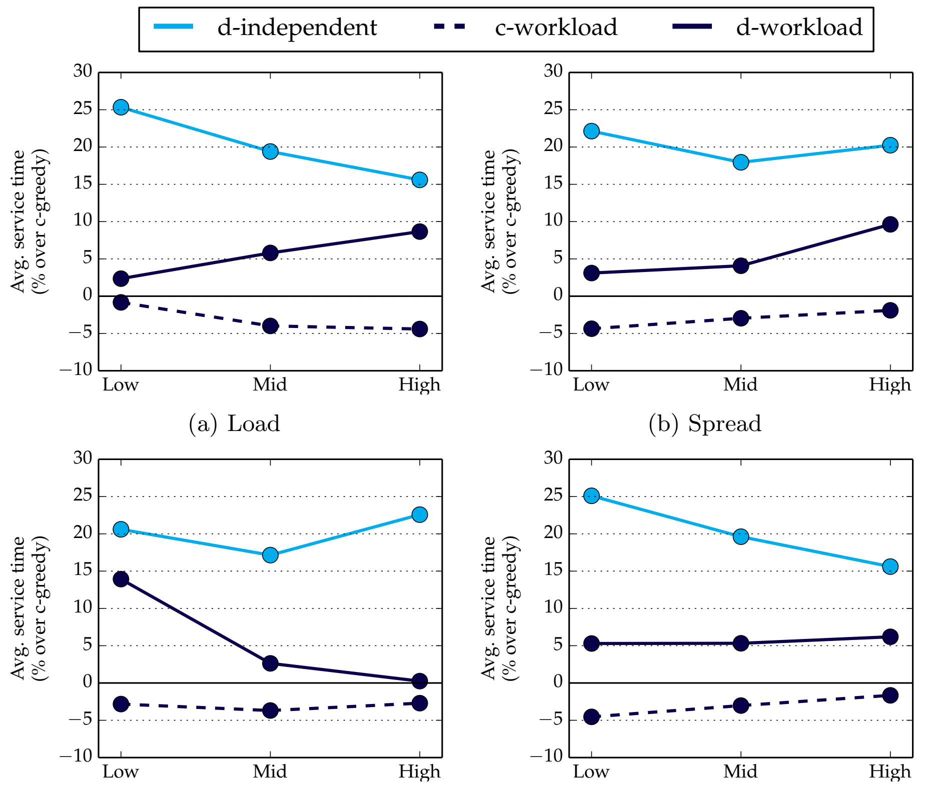
Figure 5.11: Exploration of α values.

81 scenarios, and ran *d-workload* with $k = 1000$ and $\alpha = [1.01, 1.02, .., 2]$ on each of them. Figure 5.11 shows the mean and median average request service times obtained for each α value. In those graphs we observe a (somewhat rough) gradient, as expected by the previous results. Overall, the best α value is 1.25, with a median service time of 276.9 s, and a mean service time of 183.1 s.

Results

After choosing the k and α values, we are ready to compare all algorithms in the different scenarios. With this aim, we generated 30 problems of each scenario (for a total of 2430 problems). On each of these problems we evaluate the performance of: (i) *d-independent*, which represents both the state-of-the-art PSI auctions method as well as our DCOP-based solution with independent valuations; (ii) *c-greedy*, that came up as the best existing centralized method; (iii) *d-workload*, which is our novel algorithm; and (iv) *c-workload*, which represents a bound on the best performance achievable by our novel *d-workload* mechanism.

Figure 5.12 shows the main effects plots of each characteristic, with the results of each algorithm represented as a percentage of the average service time with respect to that of *c-greedy*. Albeit sometimes small, the differences between all methods are deemed significant by the Wilcoxon signed-rank test with $p = 0.01$. Each plot shows the effect of the studied characteristic averaged across



(a) Load

(b) Spread

(c) Communication range

(d) Time distribution sharpness

Figure 5.12: Main effects plots (percentage *w.r.t. c-greedy*).

the levels of all other characteristics. For instance, the *Low* point for *d-workload* in Figure 5.12a represents the average service time of *d-workload* between all problems with a *Low* load, whatever their spread, communication range and sharpness levels are.

Figure 5.12a shows that the higher the load, the worse *d-workload* compares to *c-greedy*. However, notice that *c-workload* displays an opposite behavior, becoming better than *c-greedy* when there are more tasks per plane. This may seem counterintuitive at first, but has an easy explanation: recall that we control the load by varying the number of available planes. Hence, the higher the load, the less planes involved in the experiment. As a consequence, *d-workload*'s performance degrades not because there are more tasks, but because the planes are more spread and have less opportunities to coordinate. Moreover, *d-workload* approaches *c-greedy*'s results when the number of planes increases (reading the graph from right to left), almost catching up to it despite operating in a distributed manner.

Figure 5.12b shows that, unsurprisingly, *d-workload* achieves comparatively worse results than *c-greedy* as the spatial distribution of requests becomes sparser. This is due to two reasons. First, because the more spread requests are, the more UAVs will fly apart, and the lower chances to coordinate because of the communication range limit. Second, because workload methods exploit the spatial correlation between requests: if requests are more spread, this means that there is a lower spatial correlation between them, and hence the workload heuristic becomes worse. Finally, the previous section showed that *d-independent* works comparatively better when the requests are uniformly distributed. This works at its favor when the spread increases. However, being distributed mechanism it is also negatively affected by the first reason above. As a consequence, the relative performance of *d-independent* varies unpredictably when the spread changes.

In Figure 5.12c we can observe that, as the communication range increases, the distributed methods' results approach those of the centralized ones. This is because, as the communication range grows, the UAVs have more opportunities to coordinate. There is an exception in the case of *d-independent* and a high communication range. In this case, a larger communication range actually worsens the results, because it causes the planes to stay very spread out, thus dealing much worse especially in the low spread scenarios. In contrast, *d-workload* ends up achieving very close (less than 1% worse) results to those of *c-greedy* in the largest range we tested, even though it is not large enough to allow for global communication.

Finally, Figure 5.12d also shows some interesting behaviors. On the one hand, *d-independent* becomes better as the sharpness increases. This is because UAVs that coordinate with this method tend to evenly split the covered space between them. Hence, the more requests there are at the same time, the better its outcome. In contrast, *c-greedy* and *d-workload* try to be more clever than that, which pays off better when there are less requests than when the system is overloaded. This is even more noticeable with the workload-based methods. Recall that the main strength of such methods is to prepare for future requests. However, high sharpness scenarios reward better plans for the current requests than better predictions about the future.

Given that the only state-of-the-art method that satisfies all our problem's constrains is *d-independent*, these graphs show a very interesting figure: *d-workload* outperforms *d-independent* on all scenarios, lowering the average service time between 36 s (16%) and 18 s (6%). Hence *d-workload* comes up as the method of choice for the LORP. Moreover, in our experiments *d-workload* closes between 25% and 100% of the gap existing between *d-workload* and the state-of-the-art centralized *c-greedy* algorithm.

5.6 Conclusions

Questions addressed in this chapter:

Q. 3. Can the DCOP framework handle dynamic problems? And if so, how?

Q. 4. Can we develop modeling techniques that benefit certain DCOP algorithms?

In this chapter we turned our attention to larger-scale, dynamic applications where optimal solving is not an option anymore. While the literature about approximate DCOP solving is pretty extensive, their application to dynamic application domains has been much less studied. In fact, although there exist a few works about DCOPs for dynamic applications, we noticed that there is a significant lack of tools and testbeds for their development.

Hence, we first introduced the Limited-range Online Routing Problem as a relatively simple but realistic and highly dynamic application domain for the development of multi-agent coordination mechanisms. Then we studied the intrinsic issues that methods pursuing to deal with the LORP must overcome. As a result, we identified the need for quick loops of assessment, decision and action to deal with highly-dynamic problems such as the LORP. Additionally, we identified the robotics community as the most significant source of studies about dynamic multi-agent coordination applications. Therefore, we surveyed their current state-of-the-art and identified which methods can and cannot be employed for the LORP, thus setting the benchmarks for our work on it.

Next we introduced a LORP solving approach based on repeatedly taking distributed snapshots of the situation and making quick decisions based on them. Moreover, we argue that DCOPs are particularly well suited for this approach because both the model and the solving algorithms are inherently distributed. This answers our Question 3 in the introductory chapter, but also introduces some limitations. Namely, the approach is only suitable for local approximate algorithms that do not require any global state and can make decisions quickly enough.

Among the local approximate algorithms introduced in Section 2.2, we chose to employ Max-Sum because of its theoretical guarantees and encouraging experimental results. The major caveat of Max-Sum is that, in its standard form, agents take exponential time to compute their messages. As a consequence, we presented an initial solution assuming independence between requests. Using a clever encoding of this model as a binary DCOP, we showed that it is possible to overcome the Max-Sum exponentiality in some cases. Moreover, we discovered that this solution functionally mimics the operation of the state-of-the-art decentralized parallel single-auctions approach.

One of the advantages of using a DCOP model is that it is easily extensible by simply modifying and/or adding more constraints. Furthermore, we showed that by favoring the use of Tractable Higher Order Potentials, we heavily benefit the efficiency of the Max-Sum algorithm, hence addressing Question 4. Specifically, here we demonstrated that it is possible to introduce new constraints to represent the workload of each UAV with only a slight increase in complexity. As a result, we obtained a more refined model that maintains the large computational benefits.

To evaluate our novel approaches we also introduced the MASPlanes simulation toolkit, a simulation environment designed to compare coordination mechanisms for the LORP. MASPlanes fills the need for tools and testbeds for the development of DCOP-based solutions for dynamic applications by providing:

(i) a problem generator able to generate problem instances of varying scenario conditions; (ii) a simulator that enables researches to visualize and evaluate their algorithms; and (iii) a number of readily-implemented state-of-the-art centralized and distributed methods to serve as benchmarks.

Finally, we empirically evaluated our novel DCOP-based methods using MASPlanes. The evaluation shows that this improved version is always as good or better than the state-of-the-art single-item auctions approach, achieving an average of 12% (and up to 17%) lower service times. Moreover, its performance comes close (just 5% worse service times on average) to that of the state-of-the-art centralized approaches, which can not be implemented in the real-world because of the communication range limit. Therefore, our DCOP-based approach using Max-Sum and workload valuations is the method of choice for decentralized coordination in the LORP, where global communication is not possible.

Chapter 6

Scaling on the design front

6.1 Introduction

In Chapter 5 we tackled the highly-dynamic Limited-range Online Routing Problem using DCOPs. In this regard, we have shown that DCOP-based solutions are competitive with the current state-of-the-art multi-agent dynamic coordination literature. Additionally, we determined that local approximate DCOP algorithms are the only ones that can be employed in such dynamic application domains. In particular, we identified the Max-Sum algorithm as a promising approach to deal with dynamic task allocation problems. Nevertheless, the LORP is a relatively simple problem where all agents are homogeneous and perform the same tasks.

In contrast, many practical applications involve agents with different capabilities that must still cooperate in dynamic and unpredictable environments [Schurr et al., 2005]. Therefore, our goal in this chapter is to develop methodologies allowing us to transfer the techniques and results from Chapter 5 to more complex problems. Furthermore, we aim to put those in practice by developing models for a particular application domain: the RoboCup Rescue Simulation (RCS) [Skinner and Ramchurn, 2010], where heterogeneous teams of agents (*i.e.*, police patrols, fire brigades and ambulances) must join forces to mitigate damages to a city after a natural disaster has taken place.

The coordination problem faced by teams of rescue agents has been successfully addressed in the literature from various perspectives and with a wide variety of solution techniques. Likewise in applications with homogeneous agents, the most common approaches here are also based on task allocation (e.g. [Nair et al., 2002; Scerri et al., 2005]). A standard model for the task allocation problem in the context of rescue agent teams is the Extended Generalized Assignment Problem [Scerri et al., 2005]. However, such model cannot properly encode synergies and interferences among agents working on related tasks. For instance, EGAP cannot express that it is possible for two agents to perform the same task, but less desirable than letting them perform separate tasks (because otherwise they could interfere with each other). Since capturing such synergies is essential for effective cooperation in rescue missions, other works model the problem as a coalition formation problem [Ramchurn et al., 2010a,b]. Nonetheless, those approaches do not scale well in scenarios where the number of possible coalitions is large.

An interesting alternative is to model the task allocation problem using a DCOP, as proposed in [Kleiner et al., 2013]. This approach has two significant advantages. On the one hand, DCOPs are expressive enough to model relationships that EGAP is not able to capture. On the other hand, we can employ any of the several readily available local approximate algorithms to find good solutions (allocations). Along the line of this dissertation, we observe that the GDL-based Max-Sum algorithm has been applied to a significant variety of application domains with successful results. Some examples of succesful Max-Sum applications include UAVs task assignment [Delle Fave et al., 2012b], radar coordination [Kim et al., 2010], and the RoboCup Rescue [Ramchurn et al., 2010b; Kleiner et al., 2013] itself.

However, as introduced in Section 2.2.2, Max-Sum has a significant caveat: in its basic form, the algorithm exhibits an exponential complexity in the number of agents involved in the same factor. Nonetheless, in Section 5.3 we have shown that it is possible to reduce the computation costs to polynomial time (between $O(n)$ and $O(n \log(n))$) for some specific types of factors, known as *Tractable Higher Order Potentials* (THOPs) [Tarlow et al., 2010]. Notice that not all DCOP functions can be represented using THOPs. Despite that, our own work on the LORP as well as works on different domains [Penya-Alba et al., 2012] indicate that THOPS are expressive enough for several applications. Because a distinctive feature of THOPs is that they are defined over binary variables, from now on we refer to using Max-Sum with THOPs as Binary Max-Sum (BMS) to differentiate it from standard Max-Sum.

While BMS is a promising approach for multi-agent coordination, modeling problems using only THOPs is not straightforward. This is not a major issue when tackling simpler problems such as the LORP, but may become a significant obstacle for more complex problems involving heterogeneous agents. Hence, it is crucial to devise effective design methodologies for BMS to become more widespread. In this chapter we take an important step in this direction by using BMS to enable effective multi-team coordination in the rescue scenario.

Against this background, the main contributions in this chapter are the following:

- We develop a THOP-only model that mimics the single-team firefighters task allocation model in [Kleiner et al., 2013], hence reducing Max-Sum's complexity from exponential to polynomial time.

- We present a methodology that eases the design of THOP-only models for complex problems involving heterogeneous but interrelated tasks, such as those faced by rescue teams. Following this methodology, we develop a THOP-only multi-team coordination model for the police and firefighters RCS problem.

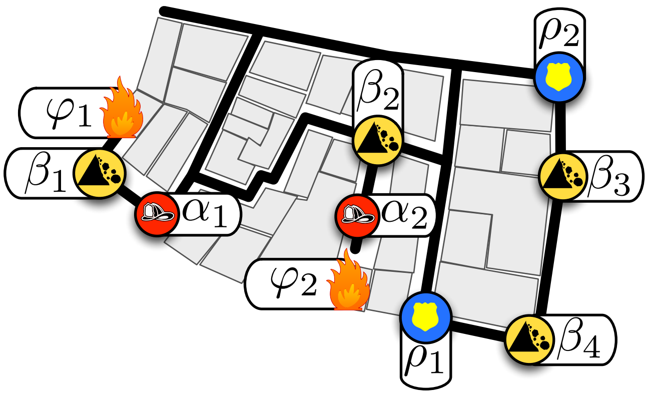


Figure 6.1: Example RoboCup Rescue Simulation scenario.

- We empirically show that BMS obtains better results than other state-of-the-art DCOP algorithms (operating on a standard DCOP model), preventing more than twice as much damage to the city.

The remainder of this chapter is organized as follows. Section 6.2 presents introduces the RoboCup Rescue Simulation problem that we tackle. Section 6.3 describes how the DCOP model to coordinate firefighters from [Kleiner et al., 2013] can be mapped to a THOP-only model. Section 6.4 presents our methodology to handle complex problems involving heterogeneous agents, and develops a complete model for the police and firefighter forces. Next, Section 6.5 reports our empirical findings. Finally, Section 6.6 draws this chapter's key conclusions.

6.2 Problem description

The RoboCup Rescue Challenge Platform [Skinner and Ramchurn, 2010] is a benchmarking environment that simulates a urban search and rescue scenario where rescue forces (police patrols, ambulances and fire brigades) must coordinate their actions.

Specifically, police patrols can unblock roads, fire brigades can extinguish fires, and ambulance agents can rescue trapped civilians. RCS creates a realistic simulation environment that presents significant aspects of dynamism (e.g., fires spread across a city), uncertainty (e.g., the behavior of fires is determined by a number of factors that may not be perfectly sensed or modeled), and issues of scale (e.g., tens of rescue agents and possibly hundreds of fires, blockades and civilians in a large urban area) [Kitano et al., 1999]. In this chapter we focus on the coordination problem of fire brigades, police patrols and their interactions. Hence, we now identify the main insights on the operation and objectives of these teams.

Regarding fire brigades, a first element to consider is travel time: the closer a fire brigade is to a fire, the sooner they will be able to work on it. Moreover, the more fire brigades acting on one fire, the faster they will contain it. However, beyond a certain number of fire brigades (which depends on the fire size), the contribution of each additional brigade is less significant. In fact, given a large enough number of fire brigades, they will completely overpower that fire, making it useless to assign any new units there.

Another key issue for the fire fighting activity is that fires evolve and spread over time. A crucial insight on the RCS's rules of fire spreading is that new fires are more likely to spread than older ones, whereas older fires are fierier and hence harder to extinguish. Therefore, fire brigades should prioritize new fires to prevent them from spreading as much as possible, and only then focus on older ones. Overall, fire brigades must cooperate to ensure that an adequate number of agents is allocated to each fire considering fire fieriness and travel time.

Regarding police patrols, the travel time is also the first issue to consider. However, unlike fires, road blockades do not evolve over time unless some agent is acting on them. Moreover, in the version of the simulator we employed, multiple police agents cannot act on the same blockade at the same time. That is, even if multiple police patrols are assigned to the same blockade, only one of those patrols can actually work on clearing it. As a result, police patrols should spread out as much as possible to free roads as fast as they can.

Considering the whole picture, police patrols and fire brigades must coordinate their actions to improve their effectiveness. Namely, police patrols should focus on blockades that might be far away but are crucial for the fire fighting activity. In this manner, fire brigades would be able to tackle important fires even when those are not immediately reachable due to road blockades.

For example, consider the scenario depicted in Figure 6.1, which will serve as a running example throughout the chapter. In this scenario we have:

- Two police patrols $P = \{\rho_1, \rho_2\}$.
- Two fire brigades $A = \{\alpha_1, \alpha_2\}$.
- Four blockades $B = \{\beta_1, \beta_2, \beta_3, \beta_4\}$ that prevent agents from transiting the road they are blocking.
- Two ignition points $F = \{\varphi_1, \varphi_2\}$, which are buildings that are on fire at the beginning of the simulation.

Now, if we consider the police patrol coordination problem without taking the fire fighting activity into account, a good allocation for this scenario would be $(\rho_1 \to \beta_4), (\rho_2 \to \beta_3)$ because both agents would work on their closest blockade, minimizing their travel time and hence maximizing their performance. However, if we consider the overall goal of the rescue agents including fire fighters, a better allocation is $(\rho_1 \to \beta_2), (\rho_2 \to \beta_1)$. This way the both fire brigade agents α_1 and α_2 can choose to work on either fire φ_1 or φ_2, something they could not do otherwise.

6.3 Single-team coordination

In this section we develop a THOP-only model for the coordination of the firefighters team, disregarding police forces entirely. With this aim, we first present an improved version of the general DCOP model in [Kleiner et al., 2013]. Thereafter we show how this model can be converted to a binary DCOP with THOPs. This allows us to run BMS instead of Max-Sum, and hence to reduce the complexity from exponential to linearithmic time.

6.3.1 Firefighters DCOP model

Solving the firefighters problem amounts to choosing which fire should each fire brigade attend next. These decisions can be encoded by a set of decision variables $Y = \{y_a \mid a \in A\}$, where each variable y_a takes some value in F (*e.g.*, $y_{\alpha_1} = \varphi_2$ means that brigade α_1 is assigned to fire φ_2). Thus, our objective is to assess a complete assignment $\mathbf{y}$ that maximizes the team utility $u(\mathbf{y})$.

In a DCOP model, the team's utility is expected to be decomposed as a sum of constraints. In our case, a natural decomposition is to introduce two kinds of constraints: (i) *fire constraints*, that specify the gain obtained when firefighters are allocated to some fire; and (ii) *cost constraints*, that specify the cost for agents to reach different fires. Thus, we define the firefighters team utility $u(\mathbf{y})$ as

$$u(\mathbf{y}) = \sum_{f \in F} e_f(\mathbf{y}) - \sum_{f \in F} \sum_{a \in A} r_{af}(\mathbf{y}[y_a]),^1 \tag{6.1}$$

where e_f are fire constraints and r_{af} are cost constraints as defined next.

Fire constraints. As explained in Section 6.2, some fires are more relevant than others. Hence, to asses $e_f(\mathbf{y})$, we first compute a value v_f for each fire f, corresponding to the utility obtained by assigning a single brigade to put it out. Next, notice that more than one brigade can be assigned to the same fire. A simple model for e_f is to multiply v_f by the number of brigades that are assigned to fire f, namely $n_f(\mathbf{y})$. Nevertheless, depending on the fieriness and size of a fire, there is a threshold t_f in the number of fire brigades that can successfully cooperate on extinguishing it. Thus, we consider that an assignment of fire brigades to a fire f is penalized when more than t_f fire brigades are assigned to fire f. Moreover, this penalty increases with the number of additional agents beyond the threshold. Combining all these assessments, $e_f(\mathbf{y})$ is calculated as

$$e_f(\mathbf{y}) = v_f \cdot n_f(\mathbf{y}) - \kappa \cdot [\max(0, n_f(\mathbf{y}) - t_f)]^\gamma, \tag{6.2}$$

where $\kappa > 0$ and $\gamma \geq 1$ are arbitrary coefficients that control the harshness of the penalty.

[1] Recall from Definition 2.5 that $\mathbf{y}[y_a]$ is the projection of assignment $\mathbf{y}$ to y_a. That is, an assignment of the single variable y_a to the value that $\mathbf{y}$ assigns to variable y_a.

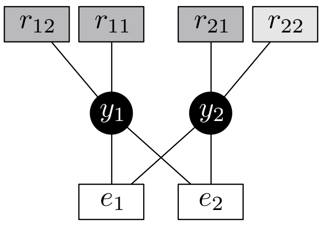


Figure 6.2: Example factor graph of the firefighters DCOP model. For clarity reasons we write indices instead of the (proper) elements in the subscripts. For instance, we write r_{12} instead of $r_{\alpha_1\varphi_2}$. Cost constraints constraints are depicted with a grey background, which is darker for blocked fires and lighter for accessible ones.

Cost constraints. All fire brigades are equally capable in RCS. Thus, the cost for an agent to reach each a fire only depends on the distance between them and on whether that fire is reachable or not. As a result, we evaluate the cost of assigning a brigade to a fire as proportional to the square of the distance between them.[2] Additionally, a blockade may prevent fire brigade a from reaching fire f. Therefore, we discourage choosing blocked fires by introducing an additional cost M when a cannot reach f. Consequently, r_{af} is assessed as

$$r_{af}(\mathbf{y_a}) = \begin{cases} \nu d_{af}^2 + M \cdot o_{af} & \text{if } \mathbf{y_a} = f \\ 0 & \text{otherwise} \end{cases}, \tag{6.3}$$

where d_{af} is the normalized distance between brigade a and fire f, $\nu \geq 0$ is an arbitrary coefficient, and o_{af} is a constant with value 1 when agent a cannot reach fire f or 0 otherwise.

At this point we can readily formalize the DCOP model as a tuple $\Omega_{\text{fire}} = \langle A, Y, D_{\text{fire}}, C_{\text{fire}}, \mathfrak{m}_{\text{fire}} \rangle$ where:

- $A = \{\alpha_1, \ldots, \alpha_n\}$ is the set of fire brigade agents involved in the problem;
- $Y = \{y_1, \ldots, y_n\}$ is the set of variables, one for each agent.
- $D_{\text{fire}} = \{D_1, \ldots, D_n\}$ is the set of domains of the variables in Y. Each domain D_i is a set containing all possible fires to which agent α_i may be assigned.
- C_{fire} is the set containing all fire constraints e_f and cost constraints r_{af}.[3]
- $\mathfrak{m}_{\text{fire}}$ maps each variable $y_a \in Y$ to the corresponding agent $\alpha_a \in A$.

[2]We normalize distances so that the largest distance between any two points in a scenario is 1.

[3]To ease the explanation we depict the cost constraints using positive costs, whereas the actual values in the DCOP are negative utilities (the same value but negated).

Figure 6.2 shows the factor graph of the firefighters DCOP model for our example in Figure 6.1. In this case we have two variables because there are two firefighters, two fire constraints because there are two fires, and four (2×2) cost constraints specifying the cost for each firefighter to reach each fire.

Now we compute allocations using Max-Sum by instantiating the e_f and r_{af} factors and exchanging messages between them and the variables in Y. However, recall that computing a factor's messages in Max-Sum takes exponential time on the number of variables involved in that factor. Because the e_f factors depend on all of the problem's variables, running Max-Sum on this model takes an exponential time on the number of fire brigades. In practice this means that for most scenarios Max-Sum cannot compute a solution within the time limits enforced by the RoboCup simulator (1 s).

6.3.2 THOP-only firefighters model

Next we show how to exactly encode the previous model in a binary form and only using THOPs. As a result, we will be able to run BMS and hence require polynomial instead of exponential time. The process detailed here is very similar to how we binarized the LORP model in Section 5.3.2. In general, the procedure to binarize a standard DCOP model involves two simple steps:

1. **Binarize variables.** Any non-binary variable with d possible values (where $d > 2$) is replaced by d new boolean variables. Then, these replacement variables are connected by a new selection constraint whose purpose is to ensure that one and only one of these boolean variables must be active.

2. **Re-encode constraints.** Each original constraint is now connected to all binary variables corresponding to its original d-ary ones. Notice that a standard constraint involving k non-binary variables has d^k entries. With boolean variables, the number of entries increases up to $2^{d \times k}$. These new entries include all the original ones, which get the same costs, plus some invalid combinations that get ∞ cost.

Therefore, to binarize our model, we first split each decision variable y_a into a set $Z_{a\cdot} = \{z_{af} \mid f \in F\}$ of $|F|$ binary variables. Intuitively, variable z_{af} is active in a solution whenever agent a is assigned to fire f, and it is inactive means that agent a is assigned elsewhere. Next, we add a constraint $s_a(Z_{a\cdot})$ for each brigade a to ensure that one and only one of its variables is active at once. Namely,

$$s_a(\mathbf{z}_{a\cdot}) = \begin{cases} 0 & \text{if exactly one } \mathbf{z}_{af} \in \mathbf{z}_{a\cdot} \text{ is active} \\ -\infty & \text{otherwise} \end{cases} . \tag{6.4}$$

With a slight abuse of notation, we now redefine the factors of the DCOP model to operate over the binary variables in Z instead of the n-ary variables in Y. Likewise Equation (6.1), the utility function $u(\mathbf{z})$ is split into fire factors e_f and

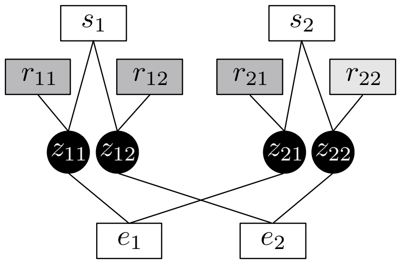


Figure 6.3: Example factor graph of the firefighters binary DCOP model.

cost factors r_{af}. The binarized versions of the utility functions in Equations (6.2) and (6.3) are the following:

$$e_f(\mathbf{z}_{\cdot f}) = v_f \cdot n_f(\mathbf{z}_{\cdot f}) - \kappa \cdot [\max(0, n_f(\mathbf{z}_{\cdot f}) - t_f)]^{\gamma} \tag{6.5}$$

$$r_{af}(\mathbf{z}_{af}) = \begin{cases} \nu d_{af}^2 + M \cdot o_{af} & \text{if } \mathbf{z}_{af} \text{ is active} \\ 0 & \text{otherwise} \end{cases} \tag{6.6}$$

where $Z_{\cdot f} = \{z_{af} \mid a \in A\}$ is the set of variables that relate to fire f and $\mathbf{z}_{\cdot f}$ is an assignment to those variables.

At this point we can represent the entire firefighters coordination problem as a binary DCOP $\Omega_{\text{fire}}^{\text{bin}} = \langle A, Z, D_{\text{fire}}^{\text{bin}}, C_{\text{fire}}^{\text{bin}}, \mathfrak{m}_{\text{fire}}^{\text{bin}} \rangle$ where:

- $A = \{\alpha_1, \ldots, \alpha_n\}$ is the set of fire brigade agents above;
- $Z = \{z_{af} \mid a \in A, f \in F\}$ is a set of binary variables, one for each pair of agent and fire.
- $D_{\text{fire}}^{\text{bin}} = \{D_1, \ldots, D_{|A| \times |F|}\}$ is the set of domains of the variables in Z, where each D_i is simply $\{\mathsf{T}, \mathsf{F}\}$.
- $C_{\text{fire}}^{\text{bin}}$ is a set including all fire constraints e_f, cost constraints r_{af}, and selection constraints s_a.
- $\mathfrak{m}_{\text{fire}}^{\text{bin}}$ maps each variable $z_{af} \in Z$ to the corresponding agent $\alpha_a \in A$.

Figure 6.3 shows the factor graph of the binary version of the DCOP model in Figure 6.2. Obviously, this conversion by itself does not provide any benefit. However, we can now analyze the constraints and check whether they are THOPs or not. With this aim, notice that this model is very similar to the workload valuations model in Section 5.3.3. The only significant difference is that in Section 5.3.3 we assigned tasks (requests) to agents (UAVs) whereas here we assign agents (fire brigades) to tasks (fires). Therefore, we can directly match our constraints here to those in the previous chapter as follows:

- The fire constraints do not depend on the specific brigades attending them, but only on how many. Hence, a fire constraint fulfills the condition to be a cardinality potential, and its BMS messages can be computed in $O(N \log(N))$ time using the procedure in Algorithm 4 (Page 114).

- The selection constraints in Equation (6.4) are exactly like the selection constraints in the previous chapter, and hence their messages can be computed using Equation (5.5).
- The cost constraints depend only on one variable and their messages are trivial to compute. Furthermore, we can also combine them with either the fire or selection constraints using Lemma B.2.

As a result, it is now possible to assess an approximate solution to our problem by applying BMS to the THOP-only model. Furthermore, the complexity of each iteration of the algorithm is reduced from the $O(|F||F|^{|A|})$ time of Max-Sum over the DCOP model to $O(|F||A|\log|A|)$ time of BMS on our THOP-only model.

6.4 Inter-team coordination

In the RoboCup Simulation it is rather unrealistic to think that a single-team can make its own decisions. In general, teams depend on each other to accomplish their tasks. Consider again our example in Figure 6.1. We know that fire brigades alone will try to avoid blocked fires. However, this completely disregards the fact that police agents will be removing blockades in the meantime, and hence the fire brigades may be able to reach blocked fires in the near future. Therefore, to capture the interdependencies between the decisions of the different teams, we argue that it is necessary to perform inter-team coordination.

Here we present a methodology to enact inter-team coordination that enables multiple teams to make their decisions considering a shared goal. Our methodology is intended to help the designer build a representation of the complete inter-team coordination problem as a single utility function. This is not an easy endeavor because, as the number of teams increases, the global utility function becomes more and more complex, possibly becoming unmanageable by the designer. To overcome this challenge, our methodology proposes a modular construction of the global utility function by following the next steps:

1. *Define independent coordination models* for each team involved in the inter-team coordination.
2. *Identify the coordination objects* that capture the interdependencies between teams. Such objects will serve to create coordination variables, which are meant to act as interfaces between single-team coordination models.
3. *Extend single-team coordination models* to connect them to the coordination variables.

At the end of this process, the global utility function is readily obtained by simply adding up the extended single-team coordination models into a single function. Since, as we show below, the resulting global utility function decomposes additively as a sum of functions, the teams involved will be able to apply max-sum to assess their decisions. A distinctive advantage of our methodology is that, once coordination variables are defined, the designer does not need to consider the whole inter-team coordination problem anymore. That is, each team independently connects its intra-team coordination model with the coordination variables. Therefore, our methodology avoids the design complexity explosion.

Next we exemplify the application of our methodology to the coordination of a team of fire brigades and a team of police forces. To avoid redundancy, we directly introduce the THOP model in this part.

6.4.1 Define independent coordination models

The first step in our methodology consists in separately defining the coordination models for each individual team involved in inter-team coordination. In Section 6.3 we already introduced a coordination model for a team of fire brigades. Hence, we just need to develop a coordination model for a team of police forces.

The Police Team Model

Recall from Section 6.2 that police patrols can remove blockades from roads, freeing the paths for other types of agents to move along. Hence, it is critical that policemen coordinate between them to remove blockades as quickly as possible. Therefore, the coordination problem faced by the policemen team is to decide the assignment of patrols to blockade removal tasks. As in the case of fire brigades and fires, we encode an allocation of patrols to blockades using a set of binary variables $X = \{x_{pb} \mid p \in P, b \in B\}$ where x_{pb} is active (takes value T) if patrol p is assigned to blockade b and inactive otherwise. Obviously, a patrol can not remove more than one blockade at a time. Also, in the version of RCS we used multiple patrols cannot work on the same blockade at the same time. As a consequence, the goal of the police team coordination problem is to compute the best allocation of patrols to blockades where each patrol is assigned to at most one blockade and each blockade is not assigned to more than one policemen.[4]

Similar to fire brigades, the utility of a complete allocation $u(\mathbf{x})$ can be decomposed in *blockade constraints* e_b and *cost constraints* r_{pb}, namely

$$u(\mathbf{x}) = \sum_{b \in B} e_b(\mathbf{x}[X_{\cdot b}]) - \sum_{b \in B} \sum_{p \in P} r_{pb}(\mathbf{x}[x_{pb}]), \tag{6.7}$$

where $X_{\cdot b} = \{x_{pb} \mid p \in P\}$ is the set of all variables related to blockade b. Since all patrols are assumed equally capable in RCS and blockades do not have distinguishing characteristics, we assign a positive utility v_B to attending any blockade. This utility is obtained whenever some patrol is assigned to remove that blockade. However, we must also ensure that no more than one patrol

[4]Notice that we cannot enforce each patrol to be allocated to some blockade. This is because, after some time, there will be more patrols than blockades in the scenario. As a consequence, no solution would be feasible at this point if we enforced such constraint.

is assigned to each blockade. Therefore, we define a blockade constraint for blockade b as

$$e_b(\mathbf{x}_{\cdot b}) = \begin{cases} -\infty & \text{if } n_b(\mathbf{x}_{\cdot b}) > 1 \\ v_B & \text{if } n_b(\mathbf{x}_{\cdot b}) = 1 \\ 0 & \text{otherwise} \end{cases}, \tag{6.8}$$

where $n_b(\mathbf{x}_{\cdot b})$ is the number of patrols assigned to blockade b.

Next, the cost of assigning a patrol p to service blockade b is analogous to what we did for the fire brigades. First, we introduce a constant o_{pb} that is 0 if the blockade is directly accessible to the patrol (no other blockade appears in the path between p and b), or 1 if the path from p to b is obstructed by some other blockade. Then the cost is proportional to the square of the distance d_{pb} between the patrol and the target blockade, with an additional penalty Q if the path is obstructed:

$$r_{pb}(\mathbf{x}_{pb}) = \begin{cases} d_{pb}^2 + Q \cdot o_{pb} & \text{if } \mathbf{x}_{pb} \text{ is active} \\ 0 & \text{otherwise} \end{cases}. \tag{6.9}$$

Finally, we still need to prevent the same patrol from being assigned to several blockades at once. Hence, we add a consistency constraint cp_p (with scope $X_{p\cdot} = \{x_{pb} \mid b \in B\}$) for each police patrol p to guarantee that it does not get assigned to multiple blockades:

$$cp_p(\mathbf{x}_{p\cdot}) = \begin{cases} 0 & \text{if at most one } \mathbf{x}_{pb} \in \mathbf{x}_{p\cdot} \text{ is active} \\ -\infty & \text{otherwise} \end{cases}. \tag{6.10}$$

With all the necessary constraints properly defined, we can now represent the police forces coordination problem as the binary DCOP model $\Omega^{\text{bin}}_{\text{police}} = \langle P, X, D^{\text{bin}}_{\text{police}}, C^{\text{bin}}_{\text{police}}, \mathfrak{m}^{\text{bin}}_{\text{police}} \rangle$ where:

- $P = \{\rho_1, \dots, \rho_n\}$ is the set of police patrols above;
- $X = \{x_{pb} \mid p \in P, b \in B\}$ is a set of binary variables, one for each pair of patrol and blockade.
- $D^{\text{bin}}_{\text{police}} = \{D_1, \dots, D_{|P| \times |B|}\}$ is the set of domains of the variables in X, where each D_i is simply $\{\mathsf{T}, \mathsf{F}\}$.
- $C^{\text{bin}}_{\text{police}}$ is the set containing all blockade constraints e_b, cost constraints r_{pb}, and consistency constraints cp_p.
- $\mathfrak{m}^{\text{bin}}_{\text{police}}$ maps each variable $x_{pb} \in X$ to the corresponding agent $\rho_p \in P$.

Figure 6.4 shows the factor graph of the police forces single-team model for our running example. Notice that it follows exactly the same structure than the binary single-team model for the firefighters. The only difference lays on the actual constraints, which we now identify as being THOPs. First, the reasoning for the blockade and cost factors is analogous to the fire and cost factors in

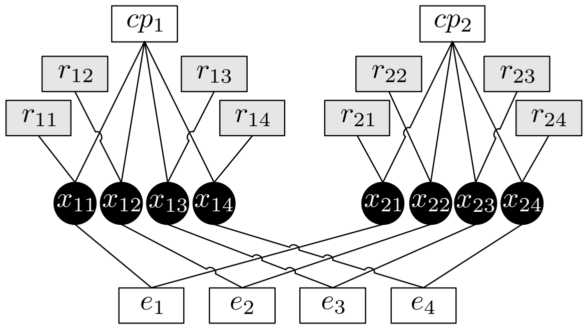


Figure 6.4: Example factor graph of the police forces binary DCOP model.

the previous section. To the best of our knowledge, the consistency constraints cp_p in Equation (6.10) have not been described as THOPs before. Therefore, in Appendix C we derive an efficient procedure to compute these constraints' messages in $O(|B|)$ time. As a result, we can now confirm that the binary single-team model for the firefighters is actually a THOP-only model too.

6.4.2 Identify the coordination objects

At this point we have single team models for both our teams. Hence, we proceed to the second step in our methodology: to identify the coordination objects. In our RoboCup example the coordination objects between fire brigades and policemen are simply the blockades. On the one hand, police forces should prioritize blockades that are actually preventing fire brigades from performing their duties. On the other hand, fire brigades would like to know which blockades will be removed in the near future to make better decisions. Thus, we create a binary coordination variable c_b for each blockade b as a means of representing the coordination objects relating police patrols and fire brigades. The coordination variable for a blockade b must become active whenever the blockade is to be removed in the near future, or inactive otherwise.

In the RCS domain these variables represent everything our police forces and fire brigades need to know to coordinate with each other. In other words, the coordination variables can be understood as representing the *common language* between our individual teams. Such language is intended to enable the fire brigades team and the policemen team to exchange information about their interdependencies regarding blockades.

6.4.3 Extending single-team models

The third step in our methodology is to extend the independent team models by connecting them to the coordination variables. Hereafter we extend both the fire brigades team model and the police patrols team model to take coordination variables into account.

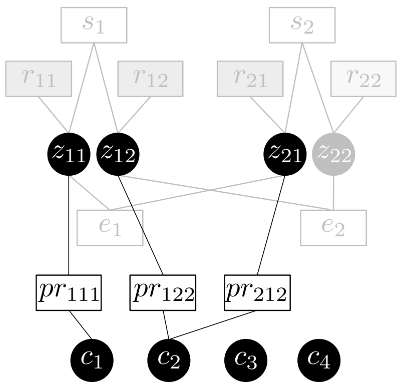


Figure 6.5: Example factor graph of the firefighters interface. For clarity reasons we write indices instead of the (proper) elements in the subscripts. For instance, we write pr_{122} instead of $pr_{\alpha_1\varphi_2\beta_2}$.

Extending the fire brigades team model

Fire brigades can modify their utility function provided that they know which blockades will be removed by police patrols. In particular, the penalty associated to a blocked fire should be removed whenever police patrols are planning to remove the blockade that prevents the fire brigade from accessing it. The interface between the fire brigades model and the coordination variables can be done by simply adding an additional factor pr_{afb} whenever fire brigade a is being prevented from reaching fire f by blockade b.

$$pr_{afb}(\mathbf{z}_{af}, \mathbf{c}_b) = \begin{cases} M & \text{if } \mathbf{c}_b \text{ is active and } \mathbf{z}_{af} \text{ is active} \\ 0 & \text{otherwise} \end{cases} . \tag{6.11}$$

The role of this factor is simply to cancel out the penalty in Equation (6.3) when the blockade b that is preventing fire fighter a to reach fire f is being attended by some police agent. For instance, in the example in Figure 6.1, blockade β_1 is preventing brigade α_1 from reaching fire φ_1. Therefore, a new factor $pr_{\alpha_1\varphi_1\beta_1}$ is required to cancel out the penalty in r_{11} if blockade β_1 is being attended (and thus c_{β_1} is active). The full interface for our example problem is shown in Figure 6.5. Notice that there is no pr constraint connecting variables c_{β_3} and c_{β_4} with the firefighters model, because those blockades are not blocking any fire brigade from reaching any fire.

Extending the police team model

The internal variables of the police team should be consistent with the semantics of the coordination variables above. Specifically, a coordination variable c_b must only be active if some police agent is attending blockade b. That is, variable c_b is an indicator of whether any of the variables in $X_{\cdot b}$ are active. We can enforce

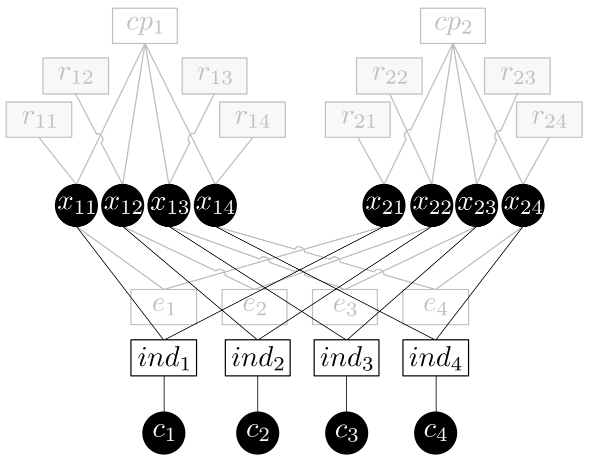


Figure 6.6: Example factor graph of the police forces interface.

this by adding a new *indicator* factor $ind_b(\mathbf{c}_b, \mathbf{x}_{.b})$ for each blockade, defined as

$$ind_b(\mathbf{c}_b, \mathbf{x}_{.b}) = \begin{cases} 0 & \text{if } \mathbf{c_b} = \mathsf{T} \text{ and some } \mathbf{x}_{pb} \in \mathbf{x}_{.b} \text{ is active} \\ 0 & \text{if } \mathbf{c_b} = \mathsf{F} \text{ and all } \mathbf{x}_{pb} \in \mathbf{x}_{.b} \text{ are inactive} \\ -\infty & \text{otherwise} \end{cases} . \tag{6.12}$$

The resulting interface for our example problem is shown in Figure 6.6. Notice that the coordination variables shown here are exactly the same ones from the firefighters interface in Figure 6.5.

The new indicator is not known to be a THOP, but we can derive expressions for its messages by noticing that it is a composite potential [Tarlow et al., 2010], where c_b defines two partitions depending on whether it is active or not. When a coordination variable c_b is active, the constraint becomes a selector constraint that yields a utility of 0 when exactly one of the $X_{.b}$ variables is active, or $-\infty$ otherwise. When c_b is inactive, the constraint becomes an *AllInactive*[5] constraint between the variables in $X_{.b}$. Since the indicator constraint is a composite potential, and in each of the partitions defined by c_b we have a THOP whose messages can be assessed in linear time, we can efficiently assess the BMS messages out of this constraint in time $O(2 \cdot |B|)$.

At this point we can easily construct a single DCOP model that represents the whole problem and allows for inter-team coordination. This results from combining the single team DCOPs defined above with the coordination variables and the interface constraints. Namely, the composed DCOP for the full problem is $\Omega^{\text{bin}}_{\text{full}} = \langle A^{\text{bin}}_{\text{full}}, V, D^{\text{bin}}_{\text{full}}, C^{\text{bin}}_{\text{full}}, \mathfrak{m}^{\text{bin}}_{\text{full}} \rangle$ where:

- $A^{\text{bin}}_{\text{full}} = A \cup P$ is the set of all fire brigades and police patrols above.

[5]The messages for the *AllInactive* constraint are trivial to derive. The messages to all its variables are simply $-\infty$.

- $V = Z \cup X \cup \{c_1, \ldots, c_{|B|}\}$ is the set of all variables of the single-team models $\Omega^{\text{bin}}_{\text{fire}}$ and $\Omega^{\text{bin}}_{\text{police}}$ plus the coordination variables.
- $D^{\text{bin}}_{\text{full}} = \{D_1, \ldots, D_{|V|}\}$ is the set of domains of the variables in V, where each D_i is simply $\{\mathsf{T}, \mathsf{F}\}$.
- $C^{\text{bin}}_{\text{full}} = C^{\text{bin}}_{\text{fire}} \cup C^{\text{bin}}_{\text{police}} \cup PR \cup IND$ is the set containing the single-team constraints plus all the interface constraints (*i.e.*, all penalty removal constraints PR and all indicator constraints IND).
- $\mathfrak{m}^{\text{bin}}_{\text{full}}$ maps each variable to the corresponding agent.

Since this DCOP has been built as an additive composition of THOP-only models and interfaces, we can readily apply BMS to solve the inter-team coordination problem. The execution of BMS yields an exchange of information from team to team regarding coordination variables. Messages from brigades to patrols represent how much interested brigades are in police forces removing a blockade, whereas messages from patrols to brigades convey the police team's cost of removing a blockade.

6.5 Empirical evaluation

In this section we empirically evaluate the performance of our DCOP-based task allocation model for the fire and police teams. With this aim, we compare our Binary Max-Sum (BMS) method with the methods implemented in the RMASBench [Kleiner et al., 2013] platform, namely the *DSA* and the *Greedy* methods.

DSA is the local-state search approximate DCOP algorithm we introduced in Section 2.2.1. Being a standard DCOP algorithm, we run DSA over the standard (non-binary) DCOP models.[6] In contrast, the *Greedy* method represents a simple greedy allocation where each agent chooses the target that maximizes its individual utility, without coordinating at all.

We run experiments on the standard scenarios used in the 2013 RoboCup competition, namely Paris and Kobe. However, we discard the scenarios' elements that are irrelevant for our evaluation, i.e., everything but ignition points, police forces and fire brigades. The simulator works by time steps, where each step represents a minute of real time. The algorithms have one second to compute an allocation at each step, and the simulation finishes either when all fires are extinguished or after 300 simulation steps. Additionally, we randomly block 5% of the roads at the beginning of the simulation. Hence, the order in which police forces remove these blockades may have a noticeable impact on the results, depending on how well the coordination mechanism works.

After an extensive empirical evaluation, we fixed the utility function's parameters to κ=2, γ=1.4, and ν=10, because these provided the overall best

[6]Although not detailed here, we also developed standard versions for both single-team police forces and the multi-team models.

Algorithm	*Greedy*	*DSA*	BMS
Score	5.29±0.79 %	2.94±0.43 %	1.13±0.18 %
NCCCs	0.00±0.00 k	7.51±0.11 k	79.62±0.81 k
Num. Msgs	0.00±0.00 k	91.08±0.92 k	536.31±8.84 k
Total bytes	0.00±0.00 Kb	711.60±7.16 Kb	4189.92±69.03 Kb
CPU time	16.14±0.45 ms	220.91±7.02 ms	726.37±15.09 ms

Table 6.1: Statistics for *Greedy*, *DSA* and BMS averaged over 30 runs in the Paris scenario (agents start acting after 25 iterations).

results for all algorithms. Moreover, the utility of each fire is $v_f = 4 - I_f$, where $I_f \in \{1, 2, 3\}$ is the fieriness of fire f as reported by the simulator, and t_f is the area of fire f divided by 100. Intuitively, the fierier a fire is, the longer a building has been burning, and the less valuable it is to contain. Additionally, we scale all the utilities of the police forces model by 10^{-3} to give more relevance to the fire brigades team than to the police agents team. With the same objective, we set $M = 100$, and $Q = 50$, so that blockades preventing fire brigades from reaching fires are more important than blockades in the path of police agents.

Regarding the parameters of the algorithms we set the number of maximum iterations for *DSA* and BMS to 100. Lowering the number of iterations to 50 slightly decreased the quality of both algorithms, whereas increasing them to 1000 did not improve the results, wasting resources in both cases. Also, we experimented with multiple values for the stochastic probability parameter of the *DSA* algorithm. The values we employed ranged from 0.1 to 0.9 (in 0.1 increments), and we chose 0.1 because it yielded the best results overall. Likewise, we used a damping factor [Frey and Dueck, 2007] of 0.9 in the BMS implementation. Finally, we employed the Anytime framework from [Zivan, 2008] to keep track of the best global solution (assignment) each algorithm found during all the iterations, and used that as the final result.

The performance of each algorithm is evaluated using the following metrics:

- **Score**. The score is the main performance metric used by the RoboCup simulator. It evaluates the percentage of damage suffered by the city, with 100% meaning that it has been completely destroyed.

- **NCCCs** [Meisels et al., 2002]. NCCCs capture the per-iteration average amount of non-parallelizable computation performed by the agents.

- **Num msgs**. Tracks average number of messages sent between all agents in a single iteration.

- **Total bytes**. Counts the average number of bytes per iteration sent between all agents.

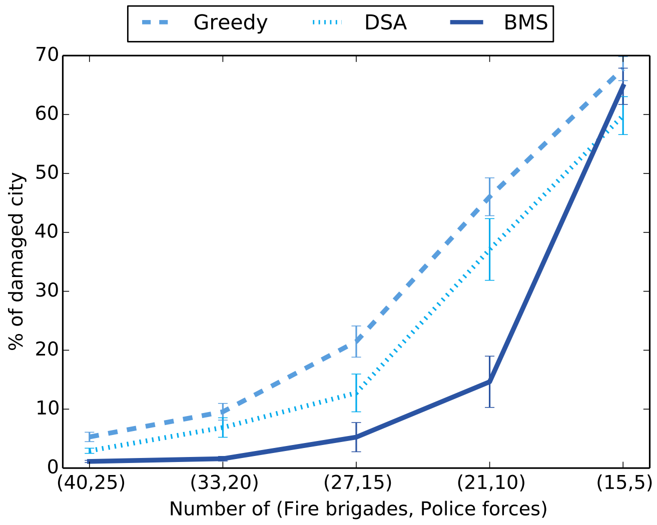
Results

Table 6.1 shows the results we obtained on the Paris map with a start time of 25 simulation steps[7] averaged over 30 simulations. BMS achieves the best score, with only 1.13% of the city damaged. In contrast, more than twice as many buildings burn down when using *DSA*, and more than four times with the *Greedy* algorithm. This gain in quality comes at a cost though. *Greedy* agents obtain the worst results in quality but require no coordination resources. *DSA* requires few computational resources and relatively low communication, whereas BMS computes an order of magnitude more than *DSA*, and requires substantially more bandwidth. Nonetheless, taking into account that an iteration of the RCS represents one minute of real time, all these costs are within an acceptable range. We also experimented with the closest allocation method described in [Parker and Gini, 2013]. However, with that method fire agents spend too much time watering down old fires (which are unlikely to spread), and an average of 49% of the city gets damaged. This result indicates that our utility function is properly capturing the characteristics of the problem.

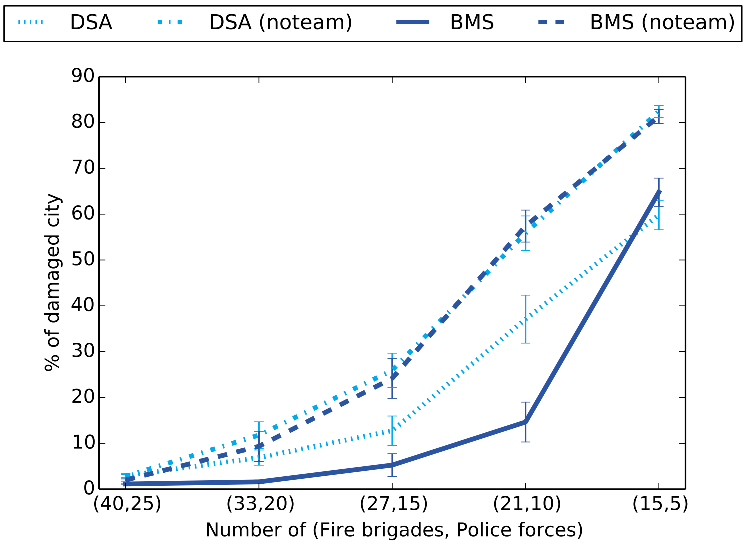
Next we assess the behavior of the different algorithms when the amount of fire and police agents decreases. Overall, Figure 6.7a shows that BMS coordinated agents achieve significantly better results than *DSA* and *Greedy* in all but the worst conditions. Namely, BMS prevents around 2.5 times more damages than *DSA*, and around 4 times more damages than *Greedy*. This trend continues when the amount of available police and fire brigades decreases. For instance, BMS saves 85.4% of the city when there are only 21 fire brigades and 10 police patrols, whereas *DSA* saves only 62.9% of the buildings and *Greedy* an even lower 54%. However, there is a point where there are so few agents that the situation is helpless and most of the city gets damaged. In this scenario, we observe this effect when we reduce the resources to 15 fire brigades and 5 police agents. At this point all algorithms obtain fairly similar results, and *DSA* becomes the best strategy thanks to its greedy but still coordinated nature.

While Figure 6.7a shows that both *DSA* and BMS outperform the *Greedy* strategy given a sensible number of rescue agents, we cannot be sure wether this is purely because of the algorithm or thanks to the inter-team coordination. Therefore, we repeated the above experiments using BMS and *DSA*, but now without the inter-team coordination constraints. Figure 6.7b shows the results we obtained, where "DSA (noteam)" and "BMS (noteam)" represent the corresponding algorithms but without the inter-team constraints. The results are particularly revealing. On the one hand, the algorithms without inter-team coordination perform similarly. BMS provides slightly better results when there are more agents, but the differences become insignificant (according to a Wilcoxon signed-rank test with $p = 0.01$) when the number of agents decreases past 27 fire brigades and 15 police patrols. On the other hand, both algorithms perform notably better when employing our inter-team coordination constraints, thus validating our inter-team methodology and the resulting model.

[7]This means we prevent agents from executing any action before 25 time steps, so that fires have time to spread.



(a) Comparison of coordinated teams using different algorithms



(b) Coordinated teams vs Independent teams (ITs)

Figure 6.7: Performance comparison when decreasing the agent resources. Results are averaged over 30 runs and error bars represent the standard error of the mean.

Furthermore, the gains from inter-team coordination are clearly larger for BMS than for *DSA*. This difference has a simple explanation. Notice that the inter-team constraints require an agent (especially police patrols) to temporarily worsen its own individual outcome (by attending a farther blockade) to realize the inter-team gains (to allow a firefighter to reach a more important fire). Now recall that *DSA*, like all other local-state search approximate algorithms detailed in Section 2.2, is essentially a greedy algorithm. In contrast, Max-Sum is the only known local-state DCOP algorithm that does not operate in a greedy manner. Therefore, the Max-Sum algorithm (and BMS by extension) is intrinsically better equipped to exploit these coordination situations where some temporary individual sacrifice must be made to achieve a greater outcome.

Finally, we conducted similar experiments in the 2013's Kobe scenario. The observed trends are the same, but the differences between algorithms are tighter because the problem is easier: in Kobe there is only one fire focus and the map is small and much easier to navigate.

6.6 Conclusions

Questions addressed in this chapter:
Q. 1. Can we identify application characteristics that provide cues as to which solving algorithm is better for that application?
Q. 5. Can we ease the modeling of complex scenarios as DCOPs?

In the previous chapter we showed that DCOP-based solutions are a competitive approach to tackle multi-agent coordination in dynamic problems. Moreover, we identified local-state approximate algorithms as the algorithms of choice in such settings. In particular, we identified Max-Sum (and more precisely BMS) as a promising algorithm to deal with dynamic multi-agent task allocation problems. However, there are a number of concerns that were not answered in that chapter.

First and foremost, a notable caveate of BMS is that it significantly increases the modeling complexity. This poses a significant barrier to the adoption of BMS, especially for more complex application domains involving heterogeneous agents, emphasizing our Question 5 in the introduction. Therefore, in this chapter we provided actionable methodologies to help in designing binary DCOP (and hopefully THOP-only) full-problem models without incurring on any design complexity explosion. Specifically, our methodology allows for a compositional approach to the modeling task: first, the designer develops standard and isolated models for the different functional areas (*i.e.*, teams of homogeneous agents) of the application. Then we showed how any standard DCOP models can be easily binarized. Finally, the individual models are combined into a full-problem model by identifying the application's inter-team coordination points as coordination variables and independently interfacing each model with those variables.

Another outstanding issue in Chapter 5 is that, although we showed that BMS is competitive with other state-of-the-art non-DCOP approaches, we did not compare BMS with other local-state approximate DCOP algorithms. Using the methodology presented in this chapter we built an inter-team coordination model for the RoboCup Rescue Challenge, which allowed for such comparison. The experiments with fire brigades and police agents show that BMS teams employing inter-team coordination are significantly more effective than uncoordinated teams. Moreover, the evaluation shows that BMS achieves up to 2.5 times better results than other state-of-the-art DCOP algorithms.

Finally, our work on the LORP suggested that dynamic task allocation problems are a good fit for the BMS algorithm. Nonetheless, we needed more applications to solidify this hypothesis. Hence, the inter-team coordination model presented here and the good empirical results we obtained are an important support for this idea. Furthermore, there are two key observations that further back up this conjecture: (i) Recall that BMS's gains are only realized when the model's constraints are THOPs. However, we have seen that some typical task allocation restrictions (such as "two agents should attend this task simultaneously" or "this task should be performed before that other task") can actually be modeled using THOPs; and (ii) being the only local-state DCOP algorithm that is not intrinsically greedy, the algorithm is better equipped to handle typical task allocation situations where an agent must temporarily sacrifice its own utility to eventually achieve a greater outcome. As a result, we argue that this answers our Question 1 by identifying dynamic task-allocation problems as a good fit for the BMS algorithm.

Chapter 7

Conclusions and future work

In this chapter we first summarize the work developed during this dissertation. Then we draw the most notable conclusions attained from it, and finally present some lines for future research.

7.1 Summary

This work revolved around enabling and scaling effective multi-agent coordination techniques. Among the several approaches to multi-agent coordination, we centered on the DCOP framework for two main reasons. Firstly, because despite being one of the simplest approaches to multi-agent coordination, the DCOP model can represent a vast number of coordination situations. Secondly, because the framework supports the development of generic algorithms to solve those problems. Moreover, the algorithms may ensure that the optimal solution is found, but they may also trade-off solution quality for execution speed, providing the flexibility required to fit many different practical applications.

Given the flexibility of this approach, there have been both a large number of proposed applications with varying requirements (*e.g.,* meeting scheduling, wireless sensor networks, traffic light control) and an equally vast array of solving algorithms of different characteristics (*e.g.,* DPOP, ADOPT, DSA, Max-Sum).

Therefore, we started in Chapter 2 by reviewing the DCOP framework and showing how it can be used to model an example multi-agent coordination problem. Next we introduced the Generalized Distributive Law and showed how its fundamental ideas can help solve DCOPs. Finally, we surveyed the DCOP literature, identifying the major algorithms and classifying them according to both their solving approach and especially by their scalability/quality characteristics. As a result, we identified three broad classes of algorithms: (i) optimal algorithms, that guarantee the maximum solution quality but incur on exponential costs and hence scale poorly; (ii) global-state approximate algorithms, that have lower costs and scale better but cannot guarantee optimality. Furthermore, those algorithms can oftentimes provide quality guarantees on the solutions found, at the expense of maintaining some logical global state among all agents. As a consequence, their scalability is still limited and tend to not handle failures gracefully; and (iii) local-state approximate algorithms, that provide fast solutions and are more robust to message and/or agent failures, but provide weak or no quality guarantees at all.

This classification readily provides a solid guideline to assess which algorithms are better suited for the different kinds of applications where DCOPs may be employed. Furthermore, it allowed us to identify the Generalized Distributive Law as the core idea shared by most inference algorithms, even by those that were developed before the GDL idea itself was widely known. Against this background, we worked on both complete and approximate DCOP solving, with three common underlying focal points:

- To exploit GDL to efficiently solve DCOPs.
- To improve the scalability of current algorithms and thus allow for larger-scale DCOP solving.
- To adapt the solving models and/or algorithms to the application's characteristics.

Due to the fundamental differences on their objective and requirements, we divided the remainder of the dissertation in two major blocks: a first one dedicated to optimal solving using GDL-based algorithms, and a second part dedicated to approximate solving using the local-state approximate version of GDL, commonly known as Max-Sum.

Optimal solving

The first GDL-based algorithm was introduced a relatively long time ago [Petcu and Faltings, 2005b], and researchers have since been proposing new techniques to improve their efficiency [Petcu and Faltings, 2005a, 2007a; Vinyals et al., 2009; Brito and Meseguer, 2010a]. One of the most promising among those techniques is known as function filtering [Sánchez et al., 2005; Brito and Meseguer, 2010b]. Therefore, this part of the dissertation focused on improving the efficiency and scalability of the GDL with function filtering algorithm.

In Chapter 3 we first introduced the GDL with function filtering algorithm itself. Recall that the central idea behind function filtering is to iteratively build more accurate representations of the problem in the form of lower-bound constraints, while detecting and removing suboptimal assignments in the process. Initially the problem is represented with coarse constraints over few variables, that are small and provide loose lower bounds on the problem. Then, at each subsequent iteration agents exchange larger constraints defined over more variables, representing tighter lower bounds. Meanwhile, the lower bounds collected in the previous iteration are combined with the best known solution to detect and prune out suboptimal assignments. This pruning reduces the size of the constraints required to represent the problem at the current level of accuracy, and hence the number of solutions to consider in the subsequent iterations.

This simple description of the algorithm already points out the three key operations that influence the algorithm's performance, namely:

- How to assess which assignments can or cannot be filtered.
- How to compute the "best known solution".
- How to compute the algorithm's messages (lower bound constraints) at each iteration.

In Chapter 3 we worked on the first two of these operations. Namely, we identified an opportunity to improve the tightness of the computed lower bounds with only a minimal increase in computational cost. As a result, the filtering operation can now prune more assignments and the algorithm operates equally as well or even better in all the experiments we conducted. Next we turned our attention to the computation of the best known solution, that also influences the filtering operation because it serves as the cutoff upper bound. In this regard, we introduced a scheme to compute multiple upper bounds simultaneously and presented a couple of techniques that exploit such scheme.

Next, Chapter 4 focused on the last operation: message computation. First we showed how the message computation during the algorithm's execution can be adapted to improve the algorithm's performance on either heavily communication-constrained or heavily computation-constrained applications. Building on those results, we then thoroughly analyzed and combined all techniques for message computation from the different GDL-based algorithm variants. As a result, we presented a framework for message computation that generalizes all these algorithms through a few parameters. Furthermore, these parameters allowed us to trade-off between computation and communication costs, and thus to adapt the algorithm to the different resources available on different applications.

Approximate solving

Recall that DCOPs are generally NP-Hard. Hence, although in the first part of the dissertation we were able to increase the scalability of GDL-based algorithms, at some point the problems become large enough that they simply cannot be optimally solved anymore. Thus, the second part of this work involved application domains that require approximate solving techniques. Particularly, we identified highly dynamic multi-agent coordination challenges to be amongst the most promising practical applications of the DCOP framework. In contrast, most literature about approximate DCOP solving focuses on synthetic (*i.e.,* randomly generated) and static problems. As a result, we observed that there is a significant lack of tools and testbeds for the development of DCOP algorithms that can cope with these promising applications.

Consequently, in Chapter 5 we began by introducing the novel Limited-range Online Routing Problem (LORP) as an example of large-scale, highly dynamic scenario. Next we surveyed the multi-agent literature at large, identifying an extensive body of related work from the robotics community. This allowed us to establish the state-of-the-art techniques available for this kind of problem, which served as the baseline to improve upon. Thereafter we introduced MASPlanes, an open-source simulation environment for the development and testing of LORP solution techniques, that allows for easy implementation and comparison of the developed algorithms. Then we showed how the LORP problem can be modeled as a sequence of DCOP instances, and that this approach can cope with and exploit the dynamism of the problem. Additionally, we introduced the Tractable Higher-Order Constraints (THOPs), that combined with a clever encoding of the problem allowed us to reduce the computation cost of the Max-Sum algorithm from exponential to polynomial time.

Finally, in Chapter 6 we tackled the RoboCup Rescue Simulation challenge, where heterogeneous teams of agents (police patrols and fire brigades) must coordinate to prevent damages to a city after a natural disaster has taken place. We observed that modeling becomes an issue on this kind of problems involving heterogeneous agents, and focused on easing that endeavor. With this aim, we introduced a methodology that exploits the compositional nature of DCOPs to avoid an explosion in design complexity. Essentially, our approach allows the designer to focus on the different functional areas of the problem, one at a time. Then we demonstrated this methodology by developing an inter-team coordination model for police forces and firefighters. Thereafter we implemented both standard DCOP and THOP-only variants of the proposed model. Lastly, we empirically evaluated both models in the RoboCup simulator using several algorithms, showing the benefits of our inter-team model paired with the Max-Sum algorithm.

7.2 Lessons learned

In this section we recall our research questions from Section 1.2. At the same time, we digest the major results we obtained, explaining how these results allowed us to address our questions.

Optimal solving

Our first research question was intimately related to the intrinsic characteristics of different DCOP solving approaches. Specifically, we posed the following question:

Q. 1. Can we identify application characteristics that provide cues as to which solving algorithm is better for that application?

Our survey in Section 2.2 proved to be a valuable tool to answer this question. Namely, it allowed us to identify inference-based complete algorithms in general, and GDL-based algorithms in particular, as the only optimal algorithms with a theoretical advantage: their complexity is exponential on the treewidth of the Junction Tree instead of on the number of variables in the problem. As a result, these algorithms can scale to large problems so long as their treewidth remains low.

Nonetheless, such advantage comes at a cost in the form of resource requirements. That is, agents using GDL-based algorithms typically require large amounts of memory and communication. Therefore, we posed a follow-up question to the previous one:

Q. 2. Can we improve the resource scalability of DCOP algorithms where this scalability is a limitation?

Our work on the GDL with function filtering algorithm is specifically geared towards answering this question. Recall that function filtering reduces the size of the constraints computed and sent during the algorithm's execution. Therefore, in Chapter 3 we focused on improving the effectiveness of the filtering technique.

With this aim, we first presented the so-called two-sided filtering bound, that improves the quality of the lower bounds computed during the filtering process. Our experiments showed that in the worst case, two-sided filtering does not provide significant advantages but it does not worsen the algorithm's performance either. In contrast, experiments with harder problems showed that two-sided function filtering can lead to significant reductions in the amount of resources required to optimally solve DCOPs.

Next we introduced several techniques to compute better upper bounds by exploring multiple solutions instead of a single one. We empirically evaluated these novel techniques, showing that they can further improve the efficiency of GDL with function filtering. Used in conjunction with two-sided filtering, our novel distributed stochastic exploration method significantly reduces the communication and computation costs of the algorithm, especially on the hardest problems. Unfortunately, we have also learned that in easier problems, such as the meeting scheduling instances from the USC DCOP Repository, the overhead of exploring multiple solutions is not compensated by the marginal benefits obtained in the filtering process.

Finally, our experiments showed that these algorithmic improvements to the filtering process also obtain a significant memory reduction. As a result, we were able to extend the range of solvable problems using a limited amount of memory. That is, our novel techniques enable agents with limited memory to solve problem instances that they could not solve before.

Devising techniques that lower the amount of resources required by the algorithm directly answers our Question 2. However, there is also a more indirect approach. In actual-world settings, the agents' computation and communication capabilities vary greatly between different application domains. Hence, another way to improve the applicability of a DCOP algorithm is to adapt the algorithm to the particular capabilities of the agents in the application at hand. In Chapter 4 we took this indirect route, and strived to tailor the GDL with function filtering algorithm for applications with varying characteristics.

Firstly, we noticed that most message approximation techniques in the literature are designed to minimize the computational cost of the algorithm, disregarding its communication requirements. Hence, we took the completely opposite stance, and developed the top-down approximation scheme. This scheme is specifically designed to reduce the algorithm's communication requirements, disregarding the computational ones. Moreover, we presented two particular realizations of the scheme: (i) the brute-force decomposition strategy, which is a naive implementation with high computational cost; and (2) the zero-tracking decomposition strategy, which greatly reduces the amount of computation. Our experiments show that top-down approximations achieve large communication savings with respect to previous state-of-the-art methods. Therefore, we argue that top-down approximations in general, and specifically zero-tracking decompositions increase the scalability of GDL with function filtering algorithm for heavily communication constrained application domains.

Nonetheless, the issue of adapting the algorithms' requirements for applications where agents have more balanced resource capabilities still remained. Hence, we then introduced a general scheme for cost message computation that combines all techniques employed by the different GDL-based algorithms in the literature. This scheme provided us with the necessary flexibility to design novel methods that can actually tradeoff between computation and communication requirements. In fact, our experimental evaluation showed that the one of these novel methods is the best one when communication costs are cheap with respect to computational costs (e.g. meeting scheduling on a LAN). In contrast, the method using only zero-tracking decomposition rose as the best method for heavily communication-constrained domains (e.g. wireless sensor networks).

More importantly, our empirical guidelines can be used to find out the most appropriate message computation method for applications whose resource availability is not heavily skewed towards neither communication nor computation (e.g. meeting scheduling on a WAN). Therefore, by allowing the designer to adapt the algorithm to the specific capabilities of the agents in her application, we effectively increased the scalability of GDL-based algorithms.

Approximate solving

In the second part of the thesis we turned our attention to larger-scale problems where optimal solving is not possible anymore. Nonetheless, we observed that scale is usually not the only issue on such domains. Particularly, larger scale coordination applications oftentimes involve agents that can move and operate within a constantly evolving dynamic environment. In contrast, most DCOP works assume that the problems to solve are static. As a result, we argued that an important question to answer before considering the scalability issues in such scenarios is:

Q. 3. Can the DCOP framework handle dynamic problems? And if so, how?

Our work on the Limited-range Online Routing Problem in Chapter 5 identified the need for quick loops of assessment, decision and action to deal with highly-dynamic problems. While we do not claim this is the only way to approach such problems using DCOPs, we proposed a solution based on iteratively taking snapshots of the current situation and making decisions based on them. Furthermore, we identified two basic requirements for this approach to be successful: (i) the snapshots must be built and represented in a decentralized manner; and (ii) the decision making process must operate within strict time constraints. Moreover, DCOPs are particularly well suited for this approach because both the model and the solving algorithms are inherently distributed. However, the second requirement implies that only local-state approximate algorithms that can make decisions quickly enough are suitable for this approach.

Consequently, we planned to implement our approach using the Max-Sum algorithm, which is the local-state approximate version of GDL. Nonetheless, in its standard form agents take exponential time to compute their messages. This made us re-consider Question 2 again, asking whether we can improve the performance of Max-Sum because these exponential costs are a significant computational burden. To answer this question we developed an initial solution assuming independence between requests. Using a clever encoding of this model as a binary DCOP, we showed that it is possible to overcome Max-Sum's exponentiality in some cases. This result illustrates that, along with the actual algorithms, modeling plays a crucial role to tackle actual-world applications. Likewise, it lead us to our next research question:

Q. 4. Can we develop modeling techniques that benefit certain DCOP algorithms?

Fortunately, some recent work from the machine learning community demonstrated that the computation of Max-Sum messages can be simplified for several classes of constraints, known as Tractable Higher-Order Potentials (THOPs). Therefore, we can favor the use of THOPs while modeling and thus guarantee that Max-Sum will solve our problems efficiently. On this account, we then built a more refined model using THOPs, that takes the UAV's workload into account when making decisions.

Our experiments showed that this improved version is always equally as good or better than the best previously available state-of-the-art approach. On top of that, the performance of Max-Sum using this new model comes close to that of the best centralized approaches, which can only be employed in simulation (and not in the actual-world) because of the communication range limit. This confirmed that we found a compelling approach to deal with dynamic problems using DCOPs, and that modeling the problem using THOPs allowed us to employ the Max-Sum algorithm to obtain very convincing results.

Nonetheless, developing THOP-only models is significantly more complex than designing standard DCOP models. Our work showed that this is not an issue for relatively simple problems such as the LORP. However, it becomes a palpable setback on more complex application domains, such as those involving agents with different aptitudes. This resulted in our next question, namely

Q. 5. Can we ease the modeling of complex scenarios as DCOPs?

Accordingly, in Chapter 6 we presented guidelines to help in designing binary DCOP models. On the one hand, we explained how standard DCOP models can be transformed into binary ones. On the other hand, we introduced a methodology to model complex scenarios. The key insight of our proposal is to exploit the additive nature of DCOPs (recall that a DCOP's utility function is a sum of constraints). Hence, we proposed to develop full-scale models in three steps. First, the designer models the different functional areas of the application (*e.g.,* the different teams in a rescue setting) independently. Second, she identifies the coordination objects that form the basis of the necessary interactions between these areas. Third, she extends each independent model to interface with the coordination variables.

We also demonstrated our methodology by developing an inter-team coordination model for the RoboCup Rescue Challenge. Notice that we emphasized the benefits of this methodology to build THOP-only models (where reducing the design complexity is most needed), but it can also be employed to build standard full-problem DCOP models. In fact, we experimented with both coordinated and uncoordinated fire brigades and police agents, running both Max-Sum over the THOP-only models and the state-of-the-art DSA over equivalent standard models. The results validated our methodology, showing that coordinated teams are vastly more effective than uncoordinated teams.

Finally, notice that we approached both the LORP and the RoboCup Rescue as dynamic multi-agent task allocation problems and obtained competitive results for both cases. Hence, we claim that a complementary answer to Question 1 above is that multi-agent dynamic task-allocation problems are a good fit for the Max-Sum algorithm. Such claim is further supported by two observations. Firstly, notice that Max-Sum's efficiency gains introduced in this dissertation apply only when the applications are modeled using THOP constraints. However, we have seen that some typical task allocation restrictions (such as "two agents should attend this task simultaneously" or "this task should be performed before that other task") can actually be modeled using THOPs. Secondly, recall from Section 2.2 that Max-Sum is the only local-state DCOP algorithm that is not intrinsically greedy. Therefore, the algorithm is inherently better equipped to handle situations where an agent must temporarily sacrifice its own utility to eventually achieve a greater global outcome, which is a typical situation in task allocation problems.

7.3 Future work

While this dissertation has realized significant contributions on scaling GDL-based algorithms for both optimal and approximate solving, it also opens several paths for future work. In the following we present some areas where further research is due.

On GDL with function filtering

Our work on inference-based optimal DCOP solving has produced some interesting results, especially on accounting for the varying resources available to agents on different application domains. However, there are some paths we did not explore in this work that might significantly impact the efficiency of these algorithms.

GDL with function filtering as an approximate algorithm. In this dissertation we focused on the GDL with function filtering algorithm as a complete solving algorithm. However, nothing precludes the usage of this algorithm (along with all our proposed techniques) as a global-state approximate algorithm. Moreover, although the basics of the algorithm would be the same, there are some open issues to investigate.

For instance, the direct approach would be to simply stop the algorithm once: (i) agents run out of resources; or (ii) a certain quality of the solution is guaranteed (the upper bound is within some approximation ratio of the lower bound). However, a smarter approach would be to also modify the filtering operation to prune out assignments that may be better than the current solution (upper bound), but not by more than the required approximation ratio. That is, consider a user that requires a solution whose cost is not larger than twice that of the optimal solution. Then, the filtering process could filter out any assignment whose lower bound is larger than half the cost of the current solution. As a result, we expect that significantly more assignments would be pruned, and hence the algorithm would operate faster and require less resources.

Finer-grained communication and computation bounds. Notice that the computation and communication bounds employed in our message computation scheme (in Section 4.3.4) are defined as maximum numbers of variables. The reason is historical, because filtering was built on top of the MCTE(r) algorithm where the number of variables (and their domain) exactly determines the size of the exchanged constraints. However, the introduction of function filtering makes the number of variables an inappropriate ruler: after the filtering process, the exchanged constraints become sparser. Therefore, the number of variables is not a good estimation on the size of these constraints anymore. As a consequence, a better approach would be to define the aforementioned bounds based on the actual number of assignments within the constraints instead. Nonetheless, such change would affect the approximation and decomposition methods, and hence significant further work is required to materialize this idea.

Combine GDL and filtering with search-based algorithms. While we expect the above improvements to significantly foster the GDL with function filtering efficiency and scalability, a more ambitious endeavor is to combine these inference techniques with search-based algorithms. Ideally, this combination would provide the theoretical guarantees of GDL-based algorithms and their proven efficiency on low treewidth problems with the efficacy of search heuristics.

In this regard, the work by Kim *et al.* [Kim and Lesser, 2014] is a promising initial step. However, their DJAO algorithm is more a chaining of techniques than a combination: inference is used as a preprocessor to obtain some initial bounds for the search, and then a regular search algorithm is executed. In contrast, a true combination would allow both algorithms to operate in an interleaved manner. More specifically, we envision a search-based algorithm that would, in an online fashion, employ an approximate version of GDL with function filtering to compute bounds on the subproblems defined by an and-or search tree [Marinescu and Dechter, 2007].

On dynamic problems, Max-Sum and THOPs

While the first part of the thesis focused on detail-oriented improvements to proven techniques, the second part has been more explorative. We introduced a novel problem, we compared DCOP approaches to state-of-the-art methods from the robotics community, and we imported results from the machine learning literature. As a result, the potential for future extensions is broader on this front. In the following we describe three major directions of research that we expect to produce significant and fruitful research.

More complex variants of the LORP. The Limited-range Online Routing Problem as presented in this dissertation represents a realistic fundamental coordination challenge to tackle. Furthermore, we consider the MASPlanes toolkit to greatly ease the barrier to entry to conduct research on it. However, in its current incarnation, the LORP is admittedly simple on the coordination aspect.

In contrast, the domain represented by the LORP admits many variations that would make it a further challenge. For instance, it would be reasonable for some requests to have higher priority than others, or even consider requests having associated deadlines. Likewise, it would be interesting to consider that UAVs may perform different kinds of tasks (*e.g.* taking a picture, transmitting an alert message, streaming a video). Also, there could be different kinds of UAVs with varying capabilities, or even terrestrial autonomous vehicles to coordinate.

Some may argue that the RoboCup Rescue Challenge already fits that role. However, the RCS is actually two challenges in one. Obviously, one challenge is the coordination part. However, and more importantly, RCS is a significant modeling challenge. That is, the simulation is so complex and the uncertainty so high, that the main issue in RCS is the development of actual models of the problem more than the coordination part. Additionally, the simulator and its operation do not help the cause. On the one hand, the whole system (which is a composition of several applications) is huge, and hence it is notably hard to master. On the other hand, running simulations is slow and requires lots of resources, making the modeling task even harder to handle.

Deal with Max-Sum convergence and communication issues. In the latter part of thesis we have successfully applied Max-Sum (and Max-Sum with THOPs, also known as BMS) to two different domains. Despite our eventual success, in the process we have had to deal with two known issues of the algorithm:

- *Lack of convergence.* Unlike some other algorithms such as DSA, Max-Sum is not guaranteed to converge unless the factor graph is a tree. On some problems, such as in the LORP, this is not a major issue. Even though the algorithm may not converge, it ends up cycling through several good states. Thus, in these cases it is enough to stop the algorithm after some time and extract a solution. However, in other cases the algorithm ends up cycling through notoriously bad states [Weiss and Freeman, 2001]. Several works, including our own in Section 6.5, employ rather tricky approaches to deal with this issue (*e.g.*, damping the messages, adding noise to the inputs) [Tarlow et al., 2011]. Although these devices do help, there are many cases where they do not solve the issue. Thus, other researchers have proposed heuristic post-processing techniques that to try to recover good solutions even when the algorithm does not converge [Zivan and Peled, 2012; Mostafa et al., 2014]. Unfortunately, such heuristics may end up finding arbitrarily bad solutions. As a result, there is an interesting research space on coming up with ways to avoid this problem that are sound and robust unlike all these current approaches.

- *Communication overhead.* Another issue of Max-Sum is that it sends large amounts of (very small) messages, as evidenced in Table 6.1. However, during our work above we noticed that many of those messages carry essentially duplicated information. That is, in many occasions, an agent sends either the same or very slightly modified messages to those it sent in the previous iteration of the algorithm. Therefore, we expect a careful study of this fact to reveal opportunities for large reductions in the number of messages required by Max-Sum. Furthermore, it is likely that even some computation can be avoided too.

Explore THOPs as a coordination language. In this dissertation we have shown the large impact that THOPs have on the Max-Sum algorithm. Furthermore, we demonstrated that some typical coordination interactions (*e.g.*, "two agents must collaborate to perform this task") can be modeled using THOPs. However, we only presented a few cases that arose in our example application domains. In contrast, it is likely that a large variety of interactions can actually be modeled as THOPs and thus exploit the benefits of BMS. Regrettably, the current path to realize these benefits is notably tortuous, and involves several challenges. First, the researcher must identify those opportunities. Then, she must find the appropriate constraints to represent the interaction. Finally, she needs to demonstrate that those constraints are THOPs and devise simplified computation algorithms for their messages.

As a result, we consider future research on easing the above path to have a large potential. In particular, we envision a catalog of coordination interactions. Such a catalog would provide readily available THOP-only representations for each interaction, and the corresponding algorithms to efficiently compute their Max-Sum messages. Furthermore, this inventory would then allow us to create a language to specify coordination situations. In combination with the methodologies presented in this work, such a language would considerably simplify the modeling of many applications. Additionally, it would enable us to provide readily implemented, efficient Max-Sum solutions straight from the problem specification. In fact, we already made the first steps in this direction by detailing a few THOPs [Pujol-Gonzalez et al., 2013a], and open sourcing a reference BMS implementation including them [Pujol-Gonzalez and Penya-Alba, 2014].

Appendix A

Max-Sum as a GDL algorithm

In this appendix we show the equivalence of the messages computed by the GDL algorithm and the messages typically employed to describe the Max-Sum algorithm.

First we recall the context for the GDL message computation formula in Equation (4.1). In this context, the agent/node i has a set of constraints C_i that represent its stake in the problem. Additionally, the separator S_{ij} is the set of variables shared between node i and node j. Then, in the canonical GDL algorithm from [Aji and McEliece, 2000], the message sent from node i to node j is computed as shown in Equation (4.1). Namely,

$$m_{i \to j} = (m_{\widehat{j} \to i} \bowtie C_i)[S_{ij}] \ , \tag{A.1}$$

where $m_{\widehat{j} \to i}$ is the combination of messages received by node i from all its neighbors in the JT except from j itself. That is

$$m_{\widehat{j} \to i} = \mathop{\bowtie}_{k \in N(i) \setminus \{j\}} m_{k \to i} \ ,$$

where $N(i)$ is the set of neighbors of node i.

From this equation we want to derive the expressions for the Max-Sum messages as shown in Equations (5.1) and (5.2). Remember that the major difference between the canonical GDL algorithm and Max-Sum is that the former operates on a junction tree whereas the latter operates on a factor graph.[1] Therefore, we make the following observations.

Observation A.1. The factor graph is a bipartite graph of factor and variable nodes. Thus, we can differentiate between messages sent from variable nodes to factor nodes and vice versa.

[1]In an abuse of notation, we write f to represent both a factor node of the factor graph and the constraint it represents. Likewise, we write v to refer to both a variable node and the variable it represents.

Observation A.2. In Max-Sum, variable nodes in the factor graph do not contain any constraint. Therefore, the stake in the problem C_v of a variable node v is always an empty set. In contrast, the stake of the problem $C_f = \{f\}$ of a factor f is a single element set containing the constraint it represents.

Observation A.3. Because variable nodes always involve exactly one variable, the separators $S_{vf} = S_{fv} = \{v\}$ between a variable node and a factor node is always a set of exactly one variable: the variable represented by the variable node.

Given Observation A.1, we now proceed by independently deriving the messages from factor nodes to variable nodes and the messages from variable nodes to factor nodes.

From variable nodes to factor nodes. Because of Observation A.2, we know that C_i in Equation (A.1) is always empty for these messages. Hence, we can simplify the message to

$$m_{v\to f} = m_{\widehat{f}\to v}[S_{vf}] \ .$$

By Observation A.3 and Definition 3.5, this is the same as

$$m_{v\to f}(\mathbf{v}) = \min_{\mathbf{t} \text{ extension of } \mathbf{v}} m_{\widehat{f}\to v}(\mathbf{t}) \ . \tag{A.2}$$

At this point we focus on the scope of $m_{\widehat{f}\to v}$. Notice that this constraint is simply an aggregation of the constraints received by v:

$$m_{\widehat{f}\to v} = \bowtie_{f'\in N(v)\setminus\{f\}} m_{f'\to v} \ .$$

From Equation (A.1), each message $m_{f'\to v}$ within this aggregation must have been projected over the corresponding separator $S_{f'v}$. However, from Observation A.3 we know that all separators $S_{f'v}$ are $\{v\}$, independently of the specific factor f'. As a consequence,

$$sc(m_{\widehat{f}\to v}) = \bigcup_{f'\in N(v)\setminus\{f\}} sc(f') = \{v\} \ .$$

Now, given that the scope of $m_{\widehat{f}\to v}$ is $\{v\}$, the only possible extension of an assignment $\mathbf{v}$ to the scope of $m_{\widehat{f}\to v}$ is $\mathbf{v}$ itself. Therefore, Equation (A.2) can be rewritten as

$$m_{v\to f}(\mathbf{v}) = \min_{\mathbf{v}} m_{\widehat{f}\to v}(\mathbf{v}) \ .$$

Furthermore, it is obvious that the min operator does nothing at this point, and hence can be safely removed. Additionally, we can expand $m_{\widehat{f}\to v}$ and apply Definitions 3.1 and 3.2 to obtain the canonical expression of the Max-Sum messages sent from variables to factors, namely

$$m_{v\to f}(\mathbf{v}) = \sum_{f'\in N(v)\setminus\{f\}} m_{f'\to v}(\mathbf{v}) = \mu_{v\to f}(\mathbf{v}) \ .$$

From factor nodes to variable nodes. These messages are easier to derive because there are fewer optimizations we can make. First, by **??** A.2**??** A.3, and expanding $m_{\widehat{f}\to v}$ in Equation (A.1) we get

$$m_{v\to f} = \left(f \bowtie \left(\bowtie_{v' \in N(f)\setminus\{v\}} m_{v'\to f} \right) \right)[v] \ .$$

Likewise above, we now focus on the scope of the combination. As introduced in Definition 3.1, the scope of a combination is the union of the scopes of each combined constraint. Hence,

$$sc\left(f \bowtie \left(\bowtie_{v' \in N(f)\setminus\{v\}} m_{v'\to f} \right) \right) = sc(f) \cup \bigcup_{v' \in N(f)\setminus\{v\}} sc(m_{v'\to f})$$

Now, by Observation A.3, we know that the scope of a message from any variable node v' to the factor node f is the variable itself, namely $sc(m_{v'\to f}) = \{v'\}$. Moreover, due to how the the factor graph is constructed, a variable node v' only sends messages to factor nodes whose constraint depends on that variable. Therefore, the scope of the incoming messages must be a subset of the scope of the factor node's constraint. Formally, $sc(m_{v'\to f}) \subseteq sc(f)$ for any variable v'. As a consequence, the scope of the combination is actually the scope of the factor node's constraint,

$$sc\left(f \bowtie \left(\bowtie_{v' \in N(f)\setminus\{v\}} m_{v'\to f} \right) \right) = sc(f) \ .$$

Finally, we can expand the combination and projection above to obtain the classical expression of the Max-Sum message sent by factor nodes to variables nodes:

$$m_{f\to v}(\mathbf{v}) = \min_{\substack{\mathbf{t} \text{ extension of } \mathbf{v} \\ \text{to } sc(f)}} \left(f(\mathbf{t}) + \sum_{v' \in N(f)\setminus\{v\}} m_{v'\to f}(\mathbf{t}[v']) \right) = \mu_{f\to v}(\mathbf{v}) \ .$$

Appendix B

Addition of independent valuations

Lemma B.1. Let $f(x_1,\ldots,x_n)$ be a factor over binary variables $x_1,\ldots,x_n$. To simplify the notation, let the variables' domain be $\{0,1\}$, where $x_i = 1$ means that the variable is active. Let $g(\mathbf{x}_i) = \gamma_i \cdot \mathbf{x}_i$ be another factor defined only over variable x_i. Let $h(x_1,\ldots,x_n) = f(x_1,\ldots,x_n) + g(x_i)$ be the factor obtained by adding factors f and g. Let ν_k be the real-value message received from neighbor k. Then, from Equation (4.1) we get

$$\mu_{f\to x_j}(\mathbf{x}_j,\nu_1,\ldots,\nu_n) = \min_{\substack{\mathbf{t} \text{ extension of } \mathbf{x_j}\\ \text{to } X}} \left(f(\mathbf{t}) + \sum_{x_k\in X\setminus\{x_j\}} \nu_k \cdot \mathbf{t}[x_k] \right) \tag{B.1}$$

and the corresponding real-valued message

$$\nu_{f\to x_j}(\nu_1,\ldots,\nu_n) = \mu_{f\to x_j}(1,\nu_1,\ldots,\nu_n) - \mu_{f\to x_j}(0,\nu_1,\ldots,\nu_n) \tag{B.2}$$

Then we have that

$$\nu_{h\to x_j}(\nu_1,\ldots,\nu_n) = \begin{cases} \nu_{f\to x_j}(\nu_1,\ldots,\nu_i+\gamma_i,\ldots,\nu_n) & j \neq i \\ \nu_{f\to x_j}(\nu_1,\ldots,\nu_i+\gamma_i,\ldots,\nu_n) + \gamma_i & j = i \end{cases} \tag{B.3}$$

Proof. by the definition in Equation (B.1), we observe that

$$\begin{aligned}\mu_{h\to x_j}(\mathbf{x}_j,\nu_1,\ldots,\nu_n) &= \min_{\substack{\mathbf{t} \text{ extension of } \mathbf{x_j}\\ \text{to } X}} \left(h(\mathbf{t}) + \sum_{x_k\in X\setminus\{x_j\}} \nu_k \cdot \mathbf{t}[x_k] \right) = \\ &= \min_{\substack{\mathbf{t} \text{ extension of } \mathbf{x_j}\\ \text{to } X}} \left(f(\mathbf{t}) + \gamma_i \mathbf{x}_i + \sum_{x_k\in X\setminus\{x_j\}} \nu_k \cdot \mathbf{t}[x_k] \right)\end{aligned}$$

First we take the case where $i \neq j$, and obtain

$$\begin{aligned}
&\mu_{h\to x_j}(\mathbf{x}_j, \nu_1, \ldots, \nu_n) = \\
&= \min_{\substack{\mathbf{t}\text{ extension of }\mathbf{x_j}\\ \text{to } X}} \left(f(\mathbf{t}) + (\nu_i + \gamma_i)\mathbf{x}_i + \sum_{x_k \in X\setminus\{x_i, x_j\}} \nu_k \cdot \mathbf{t}[x_k] \right) = \\
&= \mu_{f\to x_j}(\mathbf{x}_j, \nu_1, \ldots, \nu_i + \gamma_i, \ldots, \nu_n) \quad .
\end{aligned}$$

From here, the case where $j \neq i$ of Equation (B.3) follows from directly applying Equation (B.2).

Now we assume that $i = j$, and obtain

$$\begin{aligned}
\mu_{h\to x_j}(\mathbf{x}_j, \nu_1, \ldots, \nu_n) &= \min_{\substack{\mathbf{t}\text{ extension of }\mathbf{x_j}\\ \text{to } X}} \left(f(\mathbf{t}) + \gamma_j \mathbf{x}_j + \sum_{x_k \in X\setminus\{x_j\}} \nu_k \cdot \mathbf{t}[x_k] \right) = \\
&= \gamma_j \mathbf{x}_j + \min_{\substack{\mathbf{t}\text{ extension of }\mathbf{x_j}\\ \text{to } X}} \left(f(\mathbf{t}) + \sum_{x_k \in X\setminus\{x_j\}} \nu_k \cdot \mathbf{t}[x_k] \right) = \\
&= \gamma_j \mathbf{x}_j + \mu_{f\to x_j}(\mathbf{x}_j, \nu_1, \ldots, \nu_n) = \\
&= \gamma_j \mathbf{x}_j + \mu_{f\to x_j}(\mathbf{x}_j, \nu_1, \ldots, \nu_j + \gamma_j, \ldots, \nu_n)
\end{aligned}$$

where the last equality comes from the fact that ν_j is not taken into account when assessing $\mu_{f\to x_j}$. Now, the expression in Equation (B.3) for the case $i = j$ follows directly from applying the definition in Equation (B.2). □

In terms of the Max-Sum algorithm, this means that if we can efficiently compute the messages flowing out of f, we can also efficiently compute the messages flowing out of h.

Lemma B.2. Let f be a factor over binary variables $Y = \{y_1, \ldots, y_n\}$. Let $g(\mathbf{y}) = \sum_{i=1}^{n} \gamma_i \cdot \mathbf{y}[y_i]$ be another factor defined as the addition of a set of n independent factors, one over each variable y_i. Let $h(\mathbf{y}) = f(\mathbf{y}) + g(\mathbf{y})$ be the factor obtained by adding f and g. From Equation (4.1), the outgoing message from f to some variable y_j is assessed as

$$\mu_{f\to y_j}(\mathbf{y}_j, \nu_1, \ldots, \nu_n) = \min_{\substack{\mathbf{t}\text{ extension of }\mathbf{y_j}\\ \text{to } Y}} \left(f(\mathbf{t}) + \sum_{y_k \in Y\setminus\{y_j\}} \nu_k \cdot \mathbf{t}[y_k] \right),$$

and the corresponding real-valued message is

$$\nu_{f\to y_j}(\nu_1, \ldots, \nu_n) = \mu_{f\to y_j}(1, \nu_1, \ldots, \nu_n) - \mu_{f\to y_j}(0, \nu_1, \ldots, \nu_n) \ .$$

Then, we can assess the outgoing message from h to y_j as

$$\nu_{h\to y_j}(\nu_1, \ldots, \nu_n) = \nu_{f\to y_j}(\nu_1 + \gamma_1, \ldots, \nu_n + \gamma_n) + \gamma_j.$$

Proof. Direct by iterative application of Lemma B.1 to each of the independent factors that compose g. □

Appendix C

The AtMostOne constraint is a THOP

In this appendix we derive an efficient procedure to compute Max-Sum's messages of an *AtMostOne* binary constraint. An *AtMostOne* constraint f is simply a constraint ensuring that no more than one variable among those in its scope is active at the same time. Namely, the constraint has scope $X = \{x_1, \ldots, x_n\}$ and (for maximization problems) it is defined as

$$f(\mathbf{x}) = \begin{cases} 0 & \text{if at most one } \mathbf{x}_i \in \mathbf{x} \text{ is active} \\ -\infty & \text{otherwise} \end{cases} .$$

From Equation (5.2), we know that the generic Max-Sum message from a constraint f to some variable x_i is

$$\mu_{f \to x_i}(\mathbf{x}) = \max_{\substack{\mathbf{t} \text{ extension of } \mathbf{x} \\ \text{to } X}} \left(f(\mathbf{t}) + \sum_{x_j \in X \setminus \{x_i\}} \mu_{x_j \to f}(\mathbf{t}[x_j]) \right) .$$

There are two key insights to efficiently compute the messages of the *AtMostOne* factor above. On the one hand, given the definition of the constraints, notice that the only valid extensions $\mathbf{t}$ are those that have no more than 1 active variables. On the other hand, because BMS nodes exchange single-valued messages (as explained in the Chapter 5), the inactive utility of all incoming messages is 0. Hence, the outgoing messages for each possible value of x are:

- For $\mathbf{x}_i = \mathsf{T}$, no more variables can be active. That is, the only valid extension t is the one where all other variables $x' \in X \setminus \{x_i\}$ are inactive. Therefore, by directly considering the only valid extension we obtain

$$\mu_{f \to x_i}(\mathsf{T}) = 0 + \sum_{x_j \in X \setminus \{x_i\}} \mu_{x_j \to f}(\mathsf{F}) = 0 + 0 = 0.$$

- For $\mathbf{x}_i = \mathsf{F}$, the only valid extensions t are: (i) t_0, where all variables x_j are inactive; or (ii) $t_j \in T$, where each t_j is an extension where the single variable $x_j \in X \setminus \{x_i\}$ is active. Hence, we consider these two options separately:

$$(t_0) : 0 + \sum_{x_j \in X \setminus \{x_i\}} \mu_{x_j \to f}(\mathsf{F}) = 0 + 0 = 0 \; ;$$
$$(t_i) : \max_{t_j \in T} \left(0 + \mu_{x_j \to f}(\mathsf{T})\right) = \max_{t_j \in T} \nu_{x_i \to f} = \max_{x_j \in X \setminus \{x_i\}} \nu_{x_j \to f} \; ,$$

  And now, using the associativity of the max operation, we combine both these options to obtain

$$\mu_{x \to f}(\mathsf{F}) = \max_{x_j \in X \setminus \{x_i\}} \left(\max\left(0, \nu_{x_j \to f}\right)\right)$$

As a result, the real valued outgoing message from this constraint f to each of its neighboring variables $x_i \in X$ is

$$\nu_{f \to x_i} = \mu_{f \to x_i}(\mathsf{T}) - \mu_{f \to x_i}(\mathsf{F}) = - \max_{x_j \in X \setminus \{x_i\}} \left(\max\left(0, \nu_{x_j \to f}\right)\right) \; .$$

Finally, we observe that all outgoing messages can be computed in two simple linear passes. In the first pass, we compute the two maximum values ν^* and ν^{**} among all incoming messages $\nu_{x_i \to f}$. Thereafter, in the second pass we assess the outgoing real-valued message for each variable x_i as:

$$\nu_{f \to x_i} = \begin{cases} -\max(0, \nu^*) & \text{if } \nu_{x_i \to f} \neq \nu^* \\ -\max(0, \nu^{**}) & \text{if } \nu_{x_i \to f} = \nu^* \end{cases} .$$

Hence, the BMS messages for an *AtMostOne* constraint can be computed in linear time, and this constraint is a THOP.

Bibliography

Aji, S. M. and McEliece, R. J. (2000). The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343.

Ali, S., Koenig, S., and Tambe, M. (2005). Preprocessing techniques for accelerating the dcop algorithm adopt. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1041–1048. ACM.

Alighanbari, M. and How, J. P. (2005). Decentralized task assignment for unmanned aerial vehicles. In *IEEE Conference on Decision and Control*, pages 5668–5673. IEEE.

Atlas, J. and Decker, K. (2007). A complete distributed constraint optimization method for non-traditional pseudotree arrangements. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 111. ACM.

Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. (2002). The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840.

Bertsekas, D. P. (1988). The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of operations research*, 14(1):105–123.

Bollobas, B. (2001). *Random Graphs.* Cambridge University Press.

Bowring, E., Pearce, J. P., Portway, C., Jain, M., and Tambe, M. (2008). On k-optimal distributed constraint optimization algorithms: New bounds and algorithms. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 607–614. International Foundation for Autonomous Agents and Multiagent Systems.

Brito, I. and Meseguer, P. (2010a). Cluster tree elimination for distributed constraint optimization with quality guarantees. *Fund. Informaticae*, 102:263–286.

Brito, I. and Meseguer, P. (2010b). Improving dpop with function filtering. In *AAMAS*, pages 141–148.

Chapman, A. C., Rogers, A., and Jennings, N. R. (2011). Benchmarking hybrid algorithms for distributed constraint optimisation games. *Autonomous Agents and Multi-Agent Systems*, 22(3):385–414.

Chechetka, A. and Sycara, K. (2006). No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1427–1429. ACM.

Choi, H.-L., Brunet, L., and How, J. P. (2009). Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25(4):912–926.

Cleary, J. and Witten, I. (1984). Data compression using adaptive coding and partial string matching. *Communications, IEEE Transactions on*, 32(4):396 – 402.

Cox, T. H., Nagy, C. J., Skoog, M. A., Somers, I. A., and Warner, R. (2005). Overview of the civil UAV assessment document. `http://www.nasa.gov/centers/dryden/pdf/111760main_UAV_Assessment_Report_Overview.pdf` (accessed September 24, 2014).

Dechter, R. (1997). Mini-buckets: A general scheme for generating approximations in automated reasoning. In *Proc. IJCAI-97*, pages 1297–1303.

Dechter, R. (2003). *Constraint processing*. Morgan Kaufmann.

Dechter, R., Kask, K., and Larrosa, J. (2001). A general scheme for multiple lower bound computation in constraint optimization. In *Proc. CP-01*, pages 346–360.

Dechter, R. and Rish, I. (1997). A scheme for approximating probabilistic inference. *Proceedings of Uncertainty in Artificial Intelligence (UAI'97)*, pages 132–141.

Delle Fave, F., Farinelli, A., Rogers, A., and Jennings, N. R. (2012a). A methodology for deploying the max-sum algorithm and a case study on unmanned aerial vehicles. In *IAAI 2012: The Twenty-Fourth Innovative Applications of Artificial Intelligence Conference*.

Delle Fave, F. M., Rogers, A., Xu, Z., Sukkarieh, S., and Jennings, N. R. (2012b). Deploying the max-sum algorithm for decentralised coordination and task allocation of unmanned aerial vehicles for live aerial imagery collection. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 469–476. IEEE.

Delle Fave, F. M., Stranders, R., Rogers, A., and Jennings, N. R. (2011). Bounded decentralised coordination over multiple objectives. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 371–378. International Foundation for Autonomous Agents and Multiagent Systems.

Dias, M. B. and Stentz, A. (2000). A free market architecture for distributed control of a multirobot system. In *6th International Conference on Intelligent Autonomous Systems (IAS-6)*, pages 115–122.

Dias, M. B., Zlot, R., Kalra, N., and Stentz, A. (2006). Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270.

Diestel, R. (2000). *Graph Theory* {*Graduate Texts in Mathematics; 173*}. Springer-Verlag Berlin and Heidelberg GmbH & Company KG.

Estrin, D., Govindan, R., Heidemann, J., and Kumar, S. (1999). Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 263–270. ACM.

Farinelli, A., Rogers, A., Petcu, A., and Jennings, N. R. (2008). Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 639–646. International Foundation for Autonomous Agents and Multiagent Systems.

Fioretto, F., Campeotto, F., Da Rin Fioretto, L., Yeoh, W., and Pontelli, E. (2014). GD-GIBBS: a GPU-based sampling algorithm for solving distributed constraint optimization problems. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1339–1340. International Foundation for Autonomous Agents and Multiagent Systems.

Fitzpatrick, S. and Meertens, L. (2003). Distributed coordination through anarchic optimization. In *Distributed Sensor Networks*, pages 257–295. Springer.

Frey, B. and Dueck, D. (2007). Clustering by passing messages between data points. *Science*, 315(5814):972–976.

Garcia, R. and Barnes, L. (2010). Multi-uav simulator utilizing x-plane. In *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009*, pages 393–406. Springer.

Gerkey, B. P. and Mataric, M. J. (2002). Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768.

Gerkey, B. P. and Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954.

Gershman, A., Meisels, A., and Zivan, R. (2009). Asynchronous forward bounding for distributed cops. *Journal of Artificial Intelligence Research*, 34(1):61.

Gutierrez, P., Lee, J. H., Lei, K. M., Mak, T. W., and Meseguer, P. (2013). Maintaining soft arc consistencies in bnb-adopt+ during search. In *Principles and Practice of Constraint Programming*, pages 365–380. Springer.

Gutierrez, P. and Meseguer, P. (2010). Saving redundant messages in bnb-adopt. *American Conference on Artificial Intelligence (AAAI-10)*, pages 1259–1260.

Gutierrez, P., Meseguer, P., and Yeoh, W. (2011). Generalizing adopt and bnb-adopt. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume One*, pages 554–559. AAAI Press.

Happe, J. and Berger, J. (2010). CoUAV: a multi-UAV cooperative search path planning simulation environment. In *Proceedings of the 2010 Summer Computer Simulation Conference*, pages 86–93. Society for Computer Simulation International.

Hatano, D. and Hirayama, K. (2013). Deqed: an efficient divide-and-coordinate algorithm for dcop. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 566–572. AAAI Press.

Hirayama, K. and Yokoo, M. (1997). Distributed partial constraint satisfaction problem. In *Principles and Practice of Constraint Programming-CP97*, pages 222–236. Springer.

Jang, M.-W., Reddy, S., Tosic, P., Chen, L., and Agha, G. (2005). An actor-based simulation for studying UAV coordination. *A Parametric Model for Large Scale Agent Systems*, page 323.

Jensen, F. V. and Jensen, F. (1994). Optimal junction trees. In *Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*, pages 360–366. Morgan Kaufmann Publishers Inc.

Junges, R. and Bazzan, A. L. (2008). Evaluating the performance of dcop algorithms in a real world, dynamic problem. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 599–606. International Foundation for Autonomous Agents and Multiagent Systems.

Katagishi, H. and Pearce, J. P. (2007). Kopt: Distributed dcop algorithm for arbitrary k-optima with monotonically increasing utility. In *Ninth DCR Workshop*.

Khanna, S., Sattar, A., Hansen, D., and Stantic, B. (2009). An efficient algorithm for solving dynamic complex dcop problems. In *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT'09. IEEE/WIC/ACM International Joint Conferences on*, volume 2, pages 339–346. IET.

Kiekintveld, C., Yin, Z., Kumar, A., and Tambe, M. (2010). Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 133–140. International Foundation for Autonomous Agents and Multiagent Systems.

Kim, Y., Krainin, M., and Lesser, V. (2010). Application of max-sum algorithm to radar coordination and scheduling. In *Workshop on Distributed Constraint Reasoning*.

Kim, Y. and Lesser, V. (2014). DJAO: A Communication-Constrained DCOP algorithm that combines features of ADOPT and Action-GDL. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. AAAI Press.

Kingston, D., Beard, R. W., and Holt, R. S. (2008). Decentralized perimeter surveillance using a team of UAVs. *IEEE Transactions on Robotics*, 24(6):1394–1404.

Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., and Shimada, S. (1999). Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings.*, volume 6, pages 739–743. IEEE.

Kleiner, A., Farinelli, A., Ramchurn, S., Shi, B., Maffioletti, F., and Reffato, R. (2013). Rmasbench: benchmarking dynamic multi-agent coordination in urban search and rescue. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1195–1196. International Foundation for Autonomous Agents and Multiagent Systems.

Koenig, S., Keskinocak, P., and Tovey, C. A. (2010). Progress on agent coordination with cooperative auctions. In Fox, M. and Poole, D., editors, *AAAI*, volume 10, pages 1713–1717.

Koenig, S., Tovey, C., Lagoudakis, M., Markakis, V., Kempe, D., Keskinocak, P., Kleywegt, A., Meyerson, A., and Jain, S. (2006). The power of sequential single-item auctions for agent coordination. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1625. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Kschischang, F. R., Frey, B. J., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498–519.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Kumar, A., Petcu, A., and Faltings, B. (2007). H-dpop: Using hard constraints to prune the search space. In *IJCAI'07-Distributed Constraint Reasoning workshop, DCR'07*, pages 40–55.

Lagoudakis, M. G., Markakis, E., Kempe, D., Keskinocak, P., Kleywegt, A. J., Koenig, S., Tovey, C. A., Meyerson, A., and Jain, S. (2005). Auction-based multi-robot routing. In *Robotics: Science and Systems*.

Li, Z., Duan, Z., Chen, G., and Huang, L. (2010). Consensus of multiagent systems and synchronization of complex networks: a unified viewpoint. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(1):213–224.

Maheswaran, R. T., Pearce, J. P., and Tambe, M. (2004a). Distributed algorithms for dcop: A graphical-game-based approach. In *ISCA PDCS*, pages 432–439. Citeseer.

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 *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 310–317. IEEE Computer Society.

Marinescu, R. and Dechter, R. (2007). Best-first and/or search for graphical models. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 1171. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Meisels, A., Kaplansky, E., Razgon, I., and Zivan, R. (2002). Comparing performance of distributed constraint processing algorithms. In *AAMAS-02 DCR workshop*, pages 86–93, Bologna, Italy.

Modi, P. J., Shen, W.-M., Tambe, M., and Yokoo, M. (2005). Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180.

Mostafa, H., Pal, P., and Hurley, P. (2014). Message passing for distributed qos-security tradeoffs. *The Computer Journal*, 57(6):840–855.

Mosteo, A. R., Montano, L., and Lagoudakis, M. G. (2008). Multi-robot routing under limited communication range. In *IEEE International Conference on Robotics and Automation*, pages 1531–1536. IEEE.

Nair, R., Ito, T., Tambe, M., and Marsella, S. (2002). Task allocation in the robocup rescue simulation domain: A short note. In *RoboCup 2001: Robot Soccer World Cup V*, pages 751–754. Springer.

National Research Council, S. (2006). *Decadal Survey of Civil Aeronautics: Foundation for the Future*. The National Academies Press.

Nguyen, D. T., Yeoh, W., and Lau, H. C. (2013). Distributed Gibbs: a memory-bounded sampling-based DCOP algorithm. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 167–174. International Foundation for Autonomous Agents and Multiagent Systems.

Okimoto, T., Joe, Y., Iwasaki, A., Yokoo, M., and Faltings, B. (2011). Pseudo-tree-based incomplete algorithm for distributed constraint optimization with quality bounds. In *Principles and Practice of Constraint Programming–CP 2011*, pages 660–674. Springer.

Ottens, B., Dimitrakakis, C., and Faltings, B. (2012). DUCT: An Upper Confidence Bound Approach to Distributed Constraint Optimization Problems. In *AAAI*.

Parker, J. and Gini, M. (2013). Teams to exploit spatial locality among agents. *Spatial Computing 2013 colocated with AAMAS*, page 73.

Paskin, M., Guestrin, C., and McFadden, J. (2005). A robust architecture for distributed inference in sensor networks. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 8. IEEE Press.

Pearce, J. P. and Tambe, M. (2007). Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1446–1451. Morgan Kaufmann Publishers Inc.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

Pecora, F. and Cesta, A. (2007). Dcop for smart homes: A case study. *Computational Intelligence*, 23(4):395–419.

Penya-Alba, T., Cerquides, J., Rodriguez-Aguilar, J. A., and Vinyals, M. (2012). A Scalable Message-Passing Algorithm for Supply Chain Formation. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 1436–1442.

Petcu, A. and Faltings, B. (2005a). Approximations in distributed optimization. In *Principles and Practice of Constraint Programming-CP 2005*, pages 802–806. Springer.

Petcu, A. and Faltings, B. (2005b). A scalable method for multiagent constraint optimization. In *IJCAI'05 Proceedings of the 19th International Joint Conference on Artificial intelligence*, pages 266–271.

Petcu, A. and Faltings, B. (2006). ODPOP: an algorithm for open/distributed constraint optimization. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 703. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Petcu, A. and Faltings, B. (2007a). Mb-dpop: A new memory-bounded algorithm for distributed optimization. In *IJCAI*, pages 1452–1457.

Petcu, A. and Faltings, B. (2007b). Optimal solution stability in dynamic, distributed constraint optimization. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 321–327. IEEE Computer Society.

Pujol-Gonzalez, M. (2010–2014a). GDLFiltering: a GDL with function filtering DCOP solver. `http://github.com/kilburn/GDLFiltering`.

Pujol-Gonzalez, M. (2013–2014b). MASPlanes simulator for the development of distributed coordination algorithms. `https://github.com/MASPlanes/MASPlanes`.

Pujol-Gonzalez, M., Cerquides, J., Escalada-Imaz, G., Meseguer, P., and Rodriguez-Aguilar, J. A. (2013a). On binary max-sum and tractable hops. *11th European Workshop on Multi-agent Systems (EUMAS 2013)*, 1113.

Pujol-Gonzalez, M., Cerquides, J., Farinelli, A., Meseguer, P., and Rodriguez-Aguilar, J. A. (2014a). Binary max-sum for multi-team task allocation in robocup rescue. *International Joint Workshop on Optimisation in Multi-Agent Systems and Distributed Constraint Reasoning (OptMAS-DCR).*

Pujol-Gonzalez, M., Cerquides, J., and Meseguer, P. (2014b). MASPlanes: A multi-agent simulation environment to investigate decentralised coordination for teams of UAVs (demonstration). In *The 13th International Conference on Autonomous Agents and Multiagent Systems*, pages 1695–1696. International Foundation for Autonomous Agents and Multiagent Systems.

Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., and Rodriguez-Aguilar, J. A. (2011a). Communication-constrained dcops: Message approximation in gdl with function filtering. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 379–386. International Foundation for Autonomous Agents and Multiagent Systems.

Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., and Rodriguez-Aguilar, J. A. (2011b). Improving function filtering for computationally demanding dcops. *Workshop on Distributed Constraint Reasoning at IJCAI 2011*, pages 99–111.

Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., and Rodriguez-Aguilar, J. A. (2011c). Two-sided function filtering. *11th Workshop on Preferences and Soft Constraints*, pages 104–112.

Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., Rodríguez-Aguilar, J. A., and Tambe, M. (2013b). Engineering the decentralized coordination of UAVs with limited communication range. In *Advances in Artificial Intelligence - 15th Conference of the Spanish Association for Artificial Intelligence, CAEPIA*, volume 8109 of *Lecture Notes in Computer Science*, pages 199–208. Springer.

Pujol-Gonzalez, M., Kleiner, A., Farinelli, A., Ramchurn, S., Shi, B., Maffioletti, F., and Reffato, R. (2012–2014c). RMASBench: Multi-agent coordination benchmark. `https://github.com/MASPlanes/MASPlanes`.

Pujol-Gonzalez, M. and Penya-Alba, T. (2013–2014). Binary max-sum java library. `http://binarymaxsum.github.io/`.

Railsback, S. F., Lytinen, S. L., and Jackson, S. K. (2006). Agent-based simulation platforms: Review and development recommendations. *Simulation*, 82(9):609–623.

Ramchurn, S. D., Farinelli, A., Macarthur, K. S., and Jennings, N. R. (2010a). Decentralized coordination in robocup rescue. *The Computer Journal*, 53(9):1447–1461.

Ramchurn, S. D., Polukarov, M., Farinelli, A., Truong, C., and Jennings, N. R. (2010b). Coalition formation with spatial and temporal constraints. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 3-Volume 3*, pages 1181–1188. International Foundation for Autonomous Agents and Multiagent Systems.

Rao, A. S., Georgeff, M. P., et al. (1995). Bdi agents: From theory to practice. In *ICMAS*, volume 95, pages 312–319.

Rasmussen, S. J., Mitchell, J. W., Chandler, P. R., Schumacher, C. J., and Smith, A. L. (2005). Introduction to the multiuav2 simulation and its application to cooperative control research. In *American Control Conference, 2005. Proceedings of the 2005*, pages 4490–4501. IEEE.

Ren, W. and Beard, R. (2008). Cooperative fire monitoring with multiple UAVs. In *Distributed Consensus in Multi-vehicle Cooperative Control*, Communications and Control Engineering, pages 247–264. Springer London.

Rogers, A., Farinelli, A., Stranders, R., and Jennings, N. R. (2011). Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, 175(2):730–759.

Rollon, E. and Dechter, R. (2010). New mini-bucket partitioning heuristics for bounding the probability of evidence. In *AAAI*, pages 1199–1204.

Sánchez, M., Larrosa, J., and Meseguer, P. (2005). Improving tree decomposition methods with function filtering. In *Proc. IJCAI-05*, pages 1537–1538.

Scerri, P., Farinelli, A., Okamoto, S., and Tambe, M. (2005). Allocating tasks in extreme teams. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 727–734. ACM.

Schurr, N., Marecki, J., Scerri, P., Lewi, J. P., and Tambe, M. (2005). *Programming Multiagent Systems*, chapter The DEFACTO System: Coordinating Human-Agent Teams for the Future of Disaster Response, page 296. Springer.

Silaghi, M. C. and Yokoo, M. (2006). Nogood based asynchronous distributed optimization (adopt ng). In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1389–1396. ACM.

Silaghi, M.-C. and Yokoo, M. (2007). Dynamic dfs tree in adopt-ing. In *Proceedings of the 22nd national conference on Artificial intelligence*, volume 1, pages 763–769. AAAI Press.

Skinner, C. and Ramchurn, S. (2010). The robocup rescue simulation platform. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1647–1648. International Foundation for Autonomous Agents and Multiagent Systems.

Stefanovitch, N., Farinelli, A., Rogers, A., and Jennings, N. R. (2011). Resource-aware junction trees for efficient multi-agent coordination. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 363–370. International Foundation for Autonomous Agents and Multiagent Systems.

Stranders, R., Delle Fave, F. M., Rogers, A., and Jennings, N. (2011). U-gdl: A decentralised algorithm for dcops with uncertainty. In *Proceeding of the AAMAS workshop on Optimization of Multi-Agent Systems.*

Stranders, R., Tran-Thanh, L., Fave, F. M. D., Rogers, A., and Jennings, N. R. (2012). Dcops and bandits: exploration and exploitation in decentralised coordination. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 289–296. International Foundation for Autonomous Agents and Multiagent Systems.

Sultanik, E., Modi, P. J., and Regli, W. C. (2007). On modeling multiagent task scheduling as a distributed constraint optimization problem. In *IJCAI*, pages 1531–1536.

Tambe, M., Bowring, E., Jung, H., Kaminka, G., Maheswaran, R., Marecki, J., Modi, P. J., Nair, R., Okamoto, S., Pearce, J. P., et al. (2005). Conflicts in teamwork: Hybrids to the rescue. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 3–10. ACM.

Tarlow, D., Givoni, I. E., and Zemel, R. S. (2010). HOP-MAP : Efficient Message Passing with High Order Potentials. In *International Conference on Artificial Intelligence and Statistics*, volume 9, pages 812–819.

Tarlow, D., Givoni, I. E., Zemel, R. S., and Frey, B. J. (2011). Interpreting graph cuts as a max-product algorithm. *arXiv preprint arXiv:1105.1178.*

Vinyals, M. (2011). *Exploiting the structure of Distributed Constraint Optimization Problems to assess and bound coordinated actions in Multi-Agent Systems.* Ph.D. Thesis.

Vinyals, M., Pujol, M., Rodriguez-Aguilar, J. A., and Cerquides, J. (2010a). Divide-and-coordinate: Dcops by agreement. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 149–156. International Foundation for Autonomous Agents and Multiagent Systems.

Vinyals, M., Rodriguez-Aguilar, J. A., and Cerquides, J. (2009). Generalizing dpop: Action-gdl, a new complete algorithm for dcops. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1239–1240. International Foundation for Autonomous Agents and Multiagent Systems.

Vinyals, M., Rodríguez-Aguilar, J. A., and Cerquides, J. (2010b). Constructing a unifying theory of dynamic programming dcop algorithms via the generalized distributive law. *JAAMAS*, pages 1–26.

Vinyals, M., Rodriguez-Aguilar, J. A., and Cerquides, J. (2010c). Divide-and-coordinate by egalitarian utilities: Turning dcops into egalitarian worlds. In *Workshop 25: Optimisation in Multi-agent Systems*, page 33.

Vinyals, M., Rodríguez-Aguilar, J. A., and Cerquides, J. (2010d). Egalitarian utilities divide-and-coordinate: Stop arguing about decisions, let's share rewards!. In *ECAI*, pages 1025–1026.

Vinyals, M., Shieh, E., Cerquides, J., Rodriguez-Aguilar, J. A., Yin, Z., Tambe, M., and Bowring, E. (2011). Quality guarantees for region optimal dcop algorithms. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 133–140. International Foundation for Autonomous Agents and Multiagent Systems.

Weiss, Y. and Freeman, W. T. (2001). On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):736–744.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics*, 1:80–83.

Yeoh, W., Felner, A., and Koenig, S. (2008). BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 591–598. International Foundation for Autonomous Agents and Multiagent Systems.

Yeoh, W., Varakantham, P., and Koenig, S. (2009). Caching schemes for dcop search algorithms. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 609–616. International Foundation for Autonomous Agents and Multiagent Systems.

Yeoh, W., Varakantham, P., Sun, X., and Koenig, S. (2011). Incremental dcop search algorithms for solving dynamic dcops. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1069–1070. International Foundation for Autonomous Agents and Multiagent Systems.

Yin, Z. (2008). USC dcop repository.

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):55–87.

Zivan, R. (2008). Anytime local search for distributed constraint optimization. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1449–1452. International Foundation for Autonomous Agents and Multiagent Systems.

Zivan, R., Glinton, R., and Sycara, K. (2009). Distributed constraint optimization for large teams of mobile sensing agents. In *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT'09. IEEE/WIC/ACM International Joint Conferences on*, volume 2, pages 347–354. IET.

Zivan, R. and Peled, H. (2012). Max/min-sum distributed constraint optimization through value propagation on an alternating dag. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 265–272. International Foundation for Autonomous Agents and Multiagent Systems.

Zlot, R., Stentz, A., Dias, M. B., and Thayer, S. (2002). Multi-robot exploration controlled by a market economy. In *IEEE International Conference on Robotics and Automation.*, volume 3, pages 3016–3023.